Imperial College of Science and Technology and Medicine

The Management School, OR Section

University of London

Metastrategy
Simulated Annealing and Tabu Search
~~Algorithms~~ for Combinatorial Optimization Problems

by

Ibrahim Hassan Osman

A thesis submitted for the

Degree of Doctor of Philosophy of the University of London

and for the

Diploma of Imperial College

June 1991

In the name of God, Most Gracious, Most Merciful.

"..., O my Lord! Advance me in knowledge."

(QUR-AN Surra XX. 114.)

Dedicated with love

to my parents

and to my

sister

Abstract

This thesis deals with algorithms for combinatorial optimization problems related to location, resource allocation, routing and distribution systems. The capacitated clustering problem (CCP) is the problem in which a given set of weighted points is to be partitioned into subsets ("clusters") so that the total weight of points in each cluster is less than a given cluster capacity. The objective is to find the "centre" of the cluster, and to minimize the total scatter (distance) of points from the centre of the cluster to which they have been allocated. The generalised assignment problem (GAP) is the problem of finding a minimum cost assignment of a set of "jobs" to a set of "agents" such that each job is assigned to exactly one agent and the total resource of each agent is not exceeded by the demands of the jobs assigned to him. The vehicle routing problem (VRP) involves the design of a set of minimum cost routes, originating and terminating at a central depot, so that the demands of all customers are supplied by the vehicles operating these routes. The total demand of the customers on each trip must not exceed the vehicle capacity. These problems are of practical importance in business applications, logistics and industrial management. Exact algorithms can solve only small size problems of this type. With this situation, it is important to have approximate algorithms which can provide near optimal solutions for large-sized problems in reasonable amount of computation time.

In this thesis, we design, develop and analyse empirically, metastrategy approximate (MA) algorithms for the above mentioned problems. A metastrategy algorithm guides and directs the operations of a subordinate method (such as local search) in order to enhance its performance and to avoid poor local optima. Recently, two MA algorithms have been proposed to solve difficult combinatorial optimization problems: Simulated annealing (SA), which is based on ideas from statistical mechanics; and tabu search (TS), which is based on the general tenets of intelligent problem solving.

A survey of exact and approximate methods for each of the above problems together with a review of the metastrategy TS and SA methodologies are given. The main contribution of this thesis is the development of the MA algorithms for solving combinatorial optimization problems. We investigate their performance and compare them against other available methods with respect to solution quality and computation time. We have developed the concept of a λ–interchange neighbourhood mechanism and used it in local search algorithms. A new cooling schedule for SA is established, which has out-performed other cooling schedules in the literature. A new dynamic search strategy for (TS) produces better results than the classical approach for all the above problems. Also, we have designed a special data structure for the TS algorithm with which the computation time is more than halved. Computational results are presented for all the algorithms that have been developed. In cases where results for test problems are available (for example, the VRP), the new methods improve on all results reported in the literature.

Acknowledgements

Firstly, I would like to express my gratitude to God, He who taught man who knew not. Thanks also to my family, for their love, their support and the feeling that they are always there across the miles to turn my fears, and anxieties into a higher degree.

I am grateful to Professor Nicos Christofides, who introduced me to the field of Combinatorial Optimization Theory while I was at Imperial College. I am also indebted to the Harriri Foundation for the financial support I received, without which, this thesis would not have been possible. I am also grateful to the Harriri Foundation and the Management School for supporting my trips to Beograd and Athens. Many thanks to Professor Laurence Wolsey for his hospitality and help during two visits I made to Belgium as a participant of the "European Doctoral Program in Quantitative Methods in Management".

The professional support which I received from many scholars has clarified my research task. I am very grateful to Dr. Chris Potts who introduced me to the Simulated Annealing approach, and encouraged me to investigate the Tabu Search technique. I wish to thank Professor Fred Glover for the long discussions we had on top of the hills of Hydra, and also for his continuous encouragement and support. The help of Professor Luk Van Wassenhove is also appreciated for sending me his exact Fortran code for the Generalised Assignment Problem. Many thanks to Dr. Dirk Cattrysse who sent me a set of test problems.

The internal help at Imperial College must not be forgotten, especially, Dr. John Beasley for his friendly advice, and Dr. Norman Murray for patiently signing extra computer units whenever I required. This thesis would have taken longer without the moral support of Miss Ghada Al-Tajir, and Yazid Sharaiha for their help in reading parts of my thesis. Many thanks to Dr. Mohan Krishnamoorthy, for his critical reading of the final draft of this manuscript; sharing offices with him was a good experience.

I would like to thank my friends and fellow students at the Operational Research section for making life in the college a little more bearable. The help I received from Merza Hassan in a variety of ways is deeply appreciated. Many thanks to: Ali Salem El-Sabah, Andreas Johannsen, Asma Bahormoz, George Magalokonomos, Hala Badra, Niki Niktari, and Yang Guang for the friendship they have shown and the pleasant environment they created throughout the course of this work.

Finally, I would like to express my deepest gratitude to my cousin Dr. Ahmad Osman and to many friends in Lebanon for the moral support they have expressed. Last but not least, the friendship of Miss Mona Al-Qatari, Gary Hutton and Marwan Naghi is also appreciated.

Table of Contents

Chapter 1

INTRODUCTION

1.0 Introduction

During the past two decades, a great deal of effort has been invested in the field of *combinatorial optimization.*

Combinatorial optimization is defined (Lawler [1976]) as follows:

Combinatorial optimization is the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects.

Typical problems in combinatorial optimization are: an optimal location of facilities at given positions, the best grouping of customers, optimal ordering of jobs on machines, optimal assignment of jobs (customers) to agents (vehicle routes), optimal selection among various investment possibilities. Major references on aspects of combinatorial optimization are Lawler [1976], Christofides *et al.* [1979], Papadimitriou & Steiglitz [1982], Nemhauser & Wolsey [1988].

An instance of a combinatorial optimization problem can be seen as an implicit description of a finite set of feasible solutions. An objective (weight) function assigns to each of these solutions a value. The optimal solution is then that feasible solution with minimum (or maximum, depending on the prob em) value. These problems are mostly well-defined in the sense that an optimal solution always exists if the set of feasible solutions is non-empty. Thus, it is not so much the existence of the set of feasible solutions or, indeed the existence of an optimal solution, but rather the computational effort required to obtain the optima which is of central interest in the design of algorithms for combinatorial optimization problems.

In most of these problems, the optimal solution is computationally difficult to obtain. Hence, it is important to have *approximate algorithms* (*heuristics*) which can provide near optimal solutions for large-sized problems in a reasonable amount of computational time.

Of late, a lot of research attention has been focussed on the development of intelligent and effective heuristic approaches that solve large problems of practical size. These approaches have evolved through interactions and analogies derived from biological, physical, computer and decision making sciences. New approaches to the approximate solution of difficult combinatorial problems include: simulated annealing, tabu search, genetic algorithms and neural networks. The first derives from physical science - more specifically, from statistical mechanics. The second stems from the general tenets of intelligent problem solving. The last two are inspired by principles derived from biological sciences. See Glover, *et al.* [1989] for a further exposé.

In this thesis, we use the first two of the above-mentioned approximate approaches and describe them as *metastrategy algorithms*. Briefly stated:

> A **metastrategy algorithm** is a computer method for solving approximately, difficult, large and/or complex combinatorial optimization problems for which optimal solutions can not be found in a realistic amount of computing time using even the most powerful computer. Such an algorithm is superimposed on other subordinate methods, and enhances their performance by using different perational and organizational strategies.

In this chapter we discuss the following :

- What is OR in the past and today ?
- Combinatorial optimization problems and their complexity.
- Why metastrategy algorithms are used ?
- Research objectives and contributions.
- The thesis outline

1.1 What is OR in the past and today ?

In the literature, two definitions can be found for the term OR. It is known as Operational Research in Britain and as Operations Research in American.

The Operational Research Society (UK) has adopted the following short definition:

> "The Science of Decision Making in Business, Industry, Government and Society."

The American definition of OR is as follows:

> "Operations Research is the professional discipline that deals with the application of scientific methods to decision making, especially to the allocation of resources. Operations researchers aim to provide rational bases for decision making; they seek to understand and structure complex situations and to use this understanding to predict system behaviour and improve system performance. Much of this work is done using analytical and numerical techniques to develop and manipulate mathematical and computer models of organizational systems composed of people, machines, and procedures."

Operational research began during World War II when the British Government drafted a team of scientists from various disciplines to assist with military operational problems. The goal was to combine purely technical research and operational research by bringing in ideas from Engineering, Management, Mathematics, Computer Science, and Psychology. Because this team conducted research on (military) operations, its activities were called **operational research.** By the early 1970's, the researchers efforts had resulted in the development of large bodies of theory and methods such as: linear, integer and dynamic programming; queueing theory; network analysis; inventory and replacement theories; scheduling; simulation and others. Practitioners have successfully used OR methods to contribute to many areas including the military, manufacturing, transportation, communications, health care, banking and other public sectors. Today, OR scientists and practitioners work on methodological subjects such as optimization, probabilistic models, decision analysis, stochastic processes, and also on the combination of OR with other disciplines like artificial intelligence, physical and biological sciences. The majority of operational researchers work in areas of both public (such as energy and urban issues) and private concern. Private industrial applications include, for example marketing, operations management, production, scheduling, finance and decision support systems.

1.2 Combinatorial optimization problems and their complexity.

The name *combinatorial optimization* is composed of **combinatorial** and **optimization.** The word Combinatorial is sometimes replaced by **discrete**, and optimization by **programming.** A **combinatorial optimization problem** (COP) asks to find a solution of minimizing (or maximizing) an objective function over a combinatorial (or discrete) set of feasible solutions.

An *optimization problem*, P, is generally defined as follows:

P: Minimize $f(x)$

subject to $x \in S$

where $S \subseteq X$ denotes a *feasible region* in the space X. In other words, S is the set of feasible solutions satisfying the imposed constraints. The function $f{:}S \rightarrow \Re$ is called the *objective function,* where $\Re$ is the set of reals. A feasible solution, $x \in S$, is *optimal* if no other feasible solution y satisfies $f(y) < f(x)$.

The sets X and S take a variety of forms according to specific applications. We call an optimization problem P as a *combinatorial optimization problem,* (COP), if X and S are combinatorial or discrete (i.e. discrete sets of finite elements). For example, the family of a finite set of n objects contains 2^n finite elements.

In this thesis, we consider three combinatorial optimization problems related to location, resource allocation, routing and distribution problems:

The capacitated clustering problem (CCP)

The CCP is the problem in which a given set of weighted points is to be *partitioned* into *clusters* so that the total weight of points in each cluster is less than a given cluster capacity. The objective is to *define* a *centre* for each cluster and to find that partition which minimizes the total *scatter* (distance) of points from the centre of the cluster to which they have been allocated.

The generalised assignment problem (GAP)

The GAP is the problem of finding a minimum cost *assignment* of a set of *jobs* to a set of *agents* such that each job is assigned to exactly one agent, and the total *resource* of each agent is not exceeded by the demands of the jobs assigned to him.

The vehicle routing problem (VRP)

The VRP involves the design of a set of minimum cost delivery *routes*, originating and terminating at a central depot, for *a fleet of vehicles of variable* (or *fixed*) *size*, which services a set of *customers*. Each customer is supplied by a route operated by one vehicle. The total demand of the customers on each route must not exceed the *vehicle capacity*, and the total length (travel) of each vehicle route must not exceed a pre-specified *maximum bound* value.

These problems have been extensively studied, especially in the last decade. They have attracted both theoretical as well as practical attention. The theoretical interest arises mainly from the intriguing nature of their underlying combinatorial optimization models. Moreover, these problems are interesting because of their close relation with other difficult combinatorial problems.

From the above definitions, it can be seen that the above problems share one common characteristic: They are **easy** to describe but **difficult** to solve. The development of *computational complexity* theory has led, in the last fifteen years, to a fascinating insight into the inherent difficulty of these and other combinatorial optimization problems. This theory has also contributed by providing rigourous methods for evaluating algorithms and for classifying problems as *hard* or *easy*. According to Johnson *et al.* [1985], "This theory is deeply indebted to the field of combinatorial optimization, which has provided it with invaluable motivation, insight, paradigms".

What is meant by an easy (or difficult) problem?

To answer this question, we need to introduce an important complexity measure of algorithms: the *time complexity*, which is the number of arithmetic computation steps required to solve a given problem instance. Given an instance of a problem of size N, an algorithm is said to have a complexity $O(g(N))$ if $f(N)$, the maximum time required to execute the algorithm is such that: $|f(N)| < c \times |g(N)|$. Here, c is a constant and $g(N)$ is a real function of N.

A problem is *easy* if it has an algorithm with time complexity $O(N^k)$ for a constant k, where N is the size of the problem. Here, $O(N^k)$ is a function that depicts: $a_k \times N^k + a_{k-1} \times N^{k-1} + \ldots + a_1 \times N + a_0$ where, a_i are constants. Such complexity is said to be of *polynomial order,* and the algorithm is called a *polynomial time* algorithm. On the other hand, if any algorithm for the problem requires a complexity not bounded by a polynomial function in N, it is considered to be difficult or *intractable.* Typical orders that are not polynomial are $O(N^{\log N})$, $O(k^N)$ and $O(N!)$. Table 1.1 illustrates the estimated computer time necessary to execute the number of steps defined by some functions.

Table 1.1. Computation time to execute $f(N)$ steps (one step is assumed to take one microsecond)

	Size N		
Number of steps $f(N)$	20	50	100
$1000 \times N$	0.02 sec	0.05 sec	0.1 sec
$100 \times N^3$	0.8 sec	12.5 sec	100 sec
2^N	1 sec	35 years	3×10^4 centuries
3^N	5.8 min	2×10^9 centuries	...

What is NP-completness?

As the feasible region of many COP's are finite sets, an optimal solution for a minimization problem P can be obtained by a straight forward method that enumerates all feasible solutions in the space S and outputs the one with the minimum objective value. This method is called complete *enumeration*. Then, the class of NP problems is characterised by:

1. Every problem P $\in$ NP can be solved by complete enumeration.
2. Each enumerated case can be solved in polynomial time.

A problem P_1 is said to be *polynomially reducible* to a problem Q_1 if any instance of P_1 can be transformed to an instance of Q_1 in polynomial time. This suggests that the complexity of P_1 is not greater than Q_1 if the complexity of the transformation is negligible. Also, a problem Q_2 is said to be $NP-hard$ if any problem P_2 in class NP is reducible to Q_2. Furthermore, if an $NP-hard$ problem Q_2 belongs to the class NP then Q_2 is said to be $NP-complete$. Thus, an $NP-complete$ problem is the most difficult type of problem in NP. This is a brief description of essential result in computational complexity. Further reading on the subject can be found in Ibraki [1987], and a comprehensive book on the $NP-$completeness theory and problems by Garey and Johnson [1979].

Computational complexity theory has provided strong evidence that the solution for some optimization problems belonging to the class NP is likely to require a computation time that grows exponentially with a problem size. Hence the attention paid to the study of exact algorithms for NP problems in general, and the problems considered in this thesis in particular, is likely to diminish as the size of the problem instance increases.

Our interests in the CCP, GAP and VRP are due to both their theoretical and practical importance. They belong to the class of $NP-complete$ problems. Exact algorithms for them can only solve problems of small size. The largest problems, that have been solved optimally, are of sizes: 60 jobs and 10 agents for the GAP - Cattrysse [1990]; 50 customers and 8 vehicles for the VRP - Christofides [1985]; and up to 60 customers for special cases of the VRP - Laporte *et al.* [1985]. There are no exact methods reported for the CCP. Practical applications of these problems are numerous, and many organizations have achieved major economic benefit by implementing effective modelling, efficient innovative CCP, GAP, VRP algorithms for handling their strategic, tactical and operational problems. See Mulvey *et al.* [1984], Fisher *et al.* [1986], Cattrysse [1990] and Golden *et al.* [1988] for further details on the application of these problems.

1.3 Why metastrategy approximate algorithms.

In practice, algorithms are required to solve difficult problems of large size. It may not be possible to solve these problem instances exactly using even the most sophisticated algorithms. With this situation, a less ambitious approach to handle such difficult problems would be to relax the goal from one of obtaining exact (optimal) solutions to one of obtaining good sub-optimal solutions. Although these practical problems are difficult to solve in an optimizing sense, they can be solved in an operational sense. Operational researchers have recognized early, the importance of approximate algorithms (heuristics). These algorithms can provide near optimal solutions for large-sized problems in reasonable amounts of computation time and with reasonable storage space requirements. Research efforts have resulted in the development of a vast number of heuristics for different combinatorial optimization problems, see survey in Zanakis *et al.* [1989].

One group of approximate algorithms for combinatorial optimization problems is known as *iterative improvement* (*local search*) methods. An iterative improvement algorithm starts from a feasible solution and repeatedly seeks to improve it by altering the solution, through the application of *search mechanisms* until no further improvements are possible. At this stage, the algorithm stops at a *local optimum* solution. The local optima depends heavily on the starting feasible solution, and the search mechanisms. These local optima are often of low quality.

Recent advances in the design of sophisticated approximate methods have resulted in the development of innovative approaches called metastrategy algorithms that overcome some of the disadvantages and limitations of local search methods. We mention briefly two metastrategy algorithms that have been proposed to solve difficult combinatorial optimization problems:

Simulated annealing (SA), which is based on randomized search and acceptance strategies, emerged from the analogy between the annealing process of solids and the problem of solving COP's. It was developed by Kirkpatrick, Gelatt and Vecchi [1983], and independently, by Cerny [1985].

Tabu search (TS), which is based on constraining and freeing search strategies, is drawn from the general tenets of intelligent problem solving. It was originated by Glover [1986]. A similar approach was independently developed by Hansen [1986].

Both SA and TS approaches are devised so as to avoid being trapped in poor local optima. They allow the local search method to be continued after a local minimum is detected where no further search for a global minimum can be performed. Hence, they can be viewed as enhanced versions of local search techniques. They also increase the chances of obtaining near optimal solutions when applied to new problem areas. There have been a few successful applications of the metastrategy SA and TS algorithms. However, apart from the unsuccessful use of SA to solve the GAP, by Cattrysse [1990], we are not aware of any reports on the use of these novel approaches to solve the CCP, GAP and the VRP.

1.4 Research objectives and contributions

The goal of this thesis is to fully develop, design, and empirically analyse metastrategy algorithms for the CCP, GAP and VRP. We also provide some general guide-lines on how these approaches can be used for other similar combinatorial optimization problems. These guide-lines indicate the best choice of strategies and parameters for particular problem types. The main drawback of these approaches is that, although they produce near optimal solutions for COP's, they do not guarantee optimality. Asymptotic convergence to the set of optimal solutions has been proven for SA under conditions which are mainly of theoretical interest. They also provide little hope in achieving such convergence in less than exponential time.

The main contributions of this thesis are the development of the metastrategy approximate (MA) algorithms for the above mentioned problems. For each problem, we start by giving a brief discussion on the mathematical model. We then develop approximate local search descent and metastrategy SA and TS algorithms. We analyse and investigate their performances and compare them against other available methods with respect to solution quality and computation time. We have developed the concept of a λ–interchange neighbourhood mechanism and used it in local search and MA algorithms. A new *cooling schedule* for SA is established. This has out-performed other cooling schedules in the literature. A *new dynamic search strategy* and a *data structure* for (TS) are also developed. The TS algorithm produces better results than the classical TS approach for all the above problems. In addition to proposing new different search strategies for both TS and SA algorithms, we show through extensive computational experiments that our algorithms out-perform the best of available heuristic algorithms in the literature.

We compare the performance of SA and TS methods against one another, and also against other existing solution methods on the three COP's. To our knowledge, this is the first attempt of its kind that has been carried out. Although there exist single comparisons of either SA or TS approaches on different problems, few make such a cross-comparison. In the case of Skorin-Kapov [1990], both MA's were applied to the quadratic assignment problem, and compared. The SA algorithm, however, was borrowed. As a consequence, biased results might have been produced. Since SA is so parametric in nature, we have to be cautious with such comparison since better SA applications than the one used in the above comparison were found. The only other comparisons to our knowledge were reported by Malek *et al.* [1989] for the TSP. In this comparison, sequential and parallel TS and SA algorithms were developed and compared, using a classical cooling schedule in the SA algorithms.

1.5 The thesis outline

This thesis is organised as follows:

Chapter 2 is composed of two parts. The first is concerned with heuristic methods, classification, evaluation criteria and local search methods. The second deals with two metastrategy algorithms, namely simulated annealing (SA) and tabu search (TS). We first review the theory of SA, classify its cooling schedules and survey most of its applications. Next, the TS methodology and principles are introduced together with an explanation of their components. This is followed by a survey of TS applications. Surveys of the CCP, GAP, and VRP are provided in their respective Chapters and, hence, are not considered in Chapter 2.

Chapter 3 to Chapter 5 have similar structures: First, the relevant problem is reviewed and then algorithms to solve it are developed and analysed. Although, the general features of the algorithms are similar, they are tailored to meet the particular problem's characteristics (for example objectives and constraints). The similarity between the CCP, GAP and VRP is that they have the concept of assignment (points to clusters, jobs to agents, and customers to vehicle routes, respectively). The differences are many. For example, the CCP has the additional requirement of locating centres. Although the order of assignment is not crucial for the CCP and GAP, it is in the case of the VRP. Thus, apart from catering for the differences, similar discussions and algorithmic principles are maintained. The reason is to make each chapter as self-contained as possible.

Chapter 3 considers the capacitated clustering problem. We introduce the new neighbourhood generation mechanism and develop the concept of λ–interchange iterative descent methods. The new proposed cooling schedule for SA is compared with the best existing cooling schedules in the literature. The SA algorithm is then compared to the 1-interchange and 2-interchange iterative improvement descent methods both with a random and heuristically initial solution. This constructive heuristic is developed to give a computationally fast initial feasible solution. Two versions of TS are introduced.

The first is inspired from the classical TS principles. The second is our modified and improved version. These TS versions are compared with the best SA results. TS techniques are described in greater depth in Chapter 4 and Chapter 5.

Chapter 4 studies the generalised assignment problem. Here, we develop further, different neighbourhood selection strategies such as *first-improve* (*best-improve*) approaches. Moreover, a long-term strategy is used which gathers information during the search to regenerate different heuristic initial starting solutions. All strategies that have been developed are tested on the iterative descent methods. The best combination of these strategies are identified and, then embedded into the SA and TS approaches. TS methodologies are explained in greater detail. These include definitions of a *short-term memory* function and its elements such as *aspiration criterion, tabu list data structure, tabu conditions, and tabu list sizes.* Computational results are reported on the hardest type of test problems in the literature (problems of type C, See Martello *et al.* [1981]). We compare our SA results with that of Cattrysse [1990]. Finally, experiments with descent, SA and TS algorithms are carried out and the results are compared those from exact solution methods.

Chapter 5 tackles the vehicle routing problem. This chapter summarises the best of all the descent and metastrategy algorithms, that have been developed. Some of the suggestions that were put forwards in the foregoing Chapters are implemented. A new data structure is developed with which the computation time of the TS algorithm is more than halved. Regression analysis is used to find good fits for the tabu list sizes and the number of iterations. Both values depend on the characteristics of the problem. Computational results are reported on 17 test problems taken from the literature and for 9 new randomly generated problems.

Finally, in Chapter 6, we present conclusions and propose avenues for future research. In the appendices, we record the best results that were found along with the data sets used. These are stored on the floppy diskette included with the thesis.

Chapter 2

LITERATURE REVIEW

Heuristic! Patient rules of thumb, So often scorned: Sloppy! Dumb! Yet, slowly, common sense become.
(Pearl)

2.0 Introduction.

In this Chapter, we present a classification of heuristic methods, the analysis of heuristics, their advantages and disadvantages. We also describe the basic search strategies used in the simulated annealing (SA) and the tabu search (TS) metastrategy algorithms with special attention to the way in which they incorporate heuristic information. Applications which highlight the main contributions and findings of both (SA) and (TS) are reviewed. The material presented in this Chapter is intended to be a supplement, not a replication to some excellent surveys that are available. Therefore, only relevant information is included. This work, together with the given references, should present a complete literature review on heuristics and metastrategy approximate algorithms for many combinatorial problems.

2.1 Heuristics (Approximate algorithms).

Heuristics are rules of thumb, common sense and educated guesses. They are criteria, or computer methods for deciding the most effective among several alternative courses of action in order to achieve some goal. They do not necessarily identify the most effective course of action. *Algorithms* are procedures for solving a problem stated in mathematical terms. Algorithms without proven convergence to the optimum are called *heuristics*. *Approximate algorithms* have convergence properties but do not guarantee the precise solution, only an approximation of it. In the following discussion, we use the terms heuristic and approximate algorithm interchangeably.

There are numerous definitions of heuristics. We will adopt the definition of Nicholson [1971] who defines a heuristic as a procedure '*...for solving problems by an approach in which the problem can be interpreted and exploited intelligently to obtain a reasonable solution*'. This definition is preferred for two main reasons: Firstly, it emphasizes the importance of exploiting the structure of a problem in designing a heuristic. Secondly, it refers to *a reasonable solution* to the problem. The definition therefore encompasses methods which produce good, though not necessarily optimal solutions, to well defined problems as well as methods for ill-structured problems where all the objectives and constraints may not even have been defined explicitly.

Heuristics were recognized in the ancient world, and used as early as 300 A.D. (Michael [1972]). Modern interest in heuristic starts with Polya [1948]. The 1950s were a flourishing period for Management Science and Operational Research. This period saw many practical problems successfully solved through the application of inelegant but effective heuristics. In the 1960s, attention turned to optimization. This led to the development of more sophisticated exact algorithms. While these algorithms represented a significant research achievement, they failed to provide reliable solutions to many practical sized problems. In the 1970s, computational complexity results were discovered. These results provided evidence that, since most combinatorial optimization problems are intractable, attention paid to heuristics should not be discouraged. As a result, intellectual efforts were directed to the study of heuristics. However, the efforts began from an advanced perspective that emphasized sophisticated design, and (theoretical and empirical) performance analysis. The science of heuristics aimed toward understanding the workings of heuristic knowledge and current research is focussed on closely related issues: *design, analysis, implementation* and *reasons for success and failure.*

A large number of $NP-hard$ combinatorial problems of practical size can only be solved efficiently by heuristic methods, rather than by exact methods. Almost all large-sized instances of combinatorial optimization problems (e.g., the travelling salesman problem (TSP) and other routing problems, the quadratic assignment, many kinds of flow and job shop scheduling problems as well as most of the integer programming problems) can be solved effectively by heuristic methods. A recent survey of 442 articles on heuristic methods and applications covering more than 12 classes of heuristic approaches and 144 areas of applications can be found in Zanakis *et al.* [1989]. For more information on heuristic and heuristic design, we refer to Fisher *et al.* [1989], Eglese [1986], Muller-Merbach [1981, 1984], Silver *et al.* [1980] and Lin [1975].

2.1.1 Why should we use a heuristic method ?

Most complex problems require the evaluation of an immense number of alternatives to determine an exact solution. Heuristics play an effective role in such problems by indicating a way forward to reduce the number of evaluations and to obtain solutions within reasonable time constraints. There are also many other reasons for using a heuristic method. These include:

(a) The combinatorial optimization problem is of such a nature that an exact solution procedure is unknown. Sometimes, although an exact procedure may exist, it may be computationally prohibitive to use or unrealistic in its data requirement. Also, the time and resources required to develop a suitable exact algorithm may far exceed what is needed to implement a heuristic method.

(b) Heuristic methods, by design, may be simpler for the decision maker to understand. This understanding would increase their chances of implementation.

(c) Heuristic methods can be used for learning purposes and as part of any exact procedure. In exact branch and bound procedures, they are used to get an initial feasible solution to provide a bound on an optimal solution and to decide on branching strategies.

(d) In mathematical formulations of *real* world problems, Some of the most difficult aspects are often ignored (what objectives and what constraints to include, what alternatives to test and how problem data is collected and approximated). Poor quality of the data used to estimate model parameters may induce much larger errors than the sub-optimality of a heuristic.

With all these possible reasons for using heuristics, an optimal solution to a problem may sometimes be unnecessary. This is closely related to whether the problem involves a model of a strategic or tactical nature. A strategic problem requires a one-time major decision for which an elaborate analysis may be justified. Tactical decisions are more minor and repetitive in nature; decisions for which opportunities exist for correcting errors made earlier. Heuristic methods are more suitable for vaguely defined problems where exact methods would fail to be effective. Thus, we believe that approximate methods are of increasing importance to operational research practitioners, decision analysts and managers.

2.1.2 Classification of approximate algorithms.

Heuristics are algorithms based on the developer's ingenuity, experience and skillful art of design. The process of designing heuristics is a process of taking decisions, and the process of taking decisions is a process of choice between alternatives. Therefore, it is not surprising to have a large variety of them. In the following, some general principles of heuristic design that are widely applicable are classified according to Ball *et al.* [1981] and Zanakis *et al.* [1989], with some modifications of our own.

A. ***Construction.*** (a single pass heuristic)

Construction algorithms generate a feasible solution after a single *pass* through the data. *Greedy* methods are an important class of single pass heuristics in which successive steps are taken so as to maximize the immediate gain. Furthermore, a feasible solution is not found until the end of the heuristic procedure. Examples of construction heuristics are given in Section 3.2 for the CCP, the heuristic of Martello *et al.* [1981] for the GAP, and in Section 5.2.1 for the VRP.

B. ***Improvement.***

Improvement heuristics begin with a feasible solution and successively improve it by a sequence of exchanges or mergers in a local search. Generally, a feasible solution is maintained throughout the procedure. All local (neighbourhood) search (or iterative improvement) heuristics fall into this category. In these methods, the search proceeds by moving from one feasible solution to one of its *neighbours* while improving the objective function and maintaining feasibility. If no such new solution is found, a local optimum is obtained. The λ-interchange method is one such procedure and is implemented in Section 3.3 for the CCP, in Section 4.2 for the GAP and in Section 5.1 for the VRP.

C. *Mathematical programming*

Mathematical programming approaches use optimization models and a suitably truncated version of an exact solution procedure to obtain an efficient heuristic algorithm for the problem. More precisely, when it is not computationally feasible to implicitly enumerate all possible cases, only a small fraction are systematically enumerated. The rest are enumerated in a heuristic manner. This idea can be incorporated naturally in branch and bound algorithms or in dynamic programming approaches. The two-phase method of Fisher *et al.* [1981] is an example of this approach to the VRP. The general use of branch and bound algorithms for the purpose of obtaining approximate algorithms was also studied for the VRP by Christofides *et al.* [1979].

D. *Partitioning*

Partitioning algorithms break the problem into a set of similar subproblems, each of which is solved independently. These sub-solutions are then merged or patched into a solution for the original problem. Karp [1977] uses this approach to solve the TSP in the plane by partitioning the plane into small regions. A TSP is solved within each region, and the resulting sub-solutions are patched together to form the final tour.

E. *Solution space restriction*

In this approach, the set of solutions to a problem is restricted, so that it becomes easier to solve by an efficient algorithm. In some sense, all heuristics are restrictive methods. In particular, iterative improvement methods, which allow only feasible solutions to be accepted during the search, belong to this class. Some scheduling problems with time windows are *NP-complete* problems. If the time windows are fixed, the problems may be solved using a minimum cost flow algorithm (see Orloff [1976]). However, an optimal solution to a restricted problem need not necessarily be a global optimum to the original problem. In some respects, tabu search algorithms which free and constrain the search to solutions of special properties belong to this category as the solution is always feasible but obtained through a restricted search procedure.

F. ***Solution Space Relaxation.***

Algorithms in this category work in contrast to the restriction principle. The feasible solution space is expanded to obtain a tractable relaxed problem. A solution to the relaxed problem might be infeasible to the original one. However, it is possible that a feasible solution can be easily obtained. All linear programming relaxations of an integer program belong to this category. In addition, lagrangean heuristics have been widely used to obtain good solutions. Barcelo *et al.* [1984] report such an approach to the capacitated plant location problem. Klincewicz *et al.* [1986] solve the same problem, but with single source constraints.

G. ***Composite algorithms.***

This category is based on the idea that two heuristics can be combined in order to have a significantly better overall performance. It is not uncommon to see heuristics which combine construction (A) and improvement (B) heuristics to search for local optima. In this approach, (A) is used to obtain an initial starting solution. This is then passed into (B) to be further improved. This combined approach forms the basis of the descent methods we have developed in this thesis. This approach is explained in greater detail in the subsequent chapters.

H. ***Metastrategy approximate algorithms.***

Metastrategy approximate algorithms organise and direct the search of subordinate methods such as local search and employ different strategies in order to enhance their performances. These strategies include:

- A *long term strategy*, which is a learning procedure that gathers information during the algorithm run to improve the choice of initial constructive solutions.
- A *high evaluation strategy* in that the alternate solutions are selected by the best or the least disimprovement criterion.
- A *random* strategy for selection and generation of alternative solutions.
- A *probabilistic* strategy for acceptance alternative solutions to replace the current one.
- *Constraining andfreeing* strategies in order to guide the exploration of the solution space.

Examples of these metastrategy algorithms are *tabu search* (Glover [1986]) and *simulated annealing* (Kirkpatrick *et al.* [1983]). This category of algorithms also includes newly developed heuristics such as *Genetic* algorithms and *Neural Networks*. Genetic algorithms have their origin in biological science (Holland [1985]). They employ an evolutionary learning strategy that encourages the survival of the fittest (Goldberg [1989]). Neural networks are based on ideas from artificial intelligence techniques. The algorithmic approach consists of a number of small primitive processing units linked together via weighted, directed connections. Each unit receives input signals via weighted incoming connections, and responds by sending a signal to all the units it has outgoings to. For further details refer to Masson *et al.* [1990] and Korts *et al.* [1988].

It has to be pointed out that the simple and obvious techniques appear to have been explored. Metastrategy algorithms form a growing area of research that is developing rapidly and gaining major popularity among OR researchers. These algorithms are now able to obtain near optimal solutions to many hard combinatorial problems of sizes that were not solved satisfactorily before.

2.1.3 Performance analysis of approximate algorithms.

In recent years, there has been a growing interest in developing sophisticated methods to evaluate the performance of heuristics. Essentially, the performance of a heuristic can be evaluated in three different ways: worst-case analysis, probabilistic (or average-case) analysis and empirical analysis. These approaches provide different advantages in achieving the objective of measuring the performance of a heuristic. Thus they should be treated as complementary rather than competitive.

Worst-case analysis.

Worst case analysis establishes the maximum deviation form optimality that can occur when a specified heuristic (H) is applied to problem instances of a problem P. A worst-case study also involves the construction of examples for which performance of the heuristic is as bad as its guarantee. In addition, it also aims to provide upper bounds on the number of steps that a given algorithm can take to solve any problem instance. This type of analysis has the advantage of providing *worst-case performance guarantees*. However, the main disadvantage of the analysis is that the worst-case performance of an algorithm is usually not predictive of the average performance. For further details, refer to Fisher [1980].

The performance guarantee is normally expressed in terms of a *worst-case ratio*, r, such that $C_h(I) \leq r \times C(I)$ for some $r > 1$, where $C_h(I)$ and $C(I)$ are, respectively the heuristic and the optimal solution of a minimization problem instance $I \in P$. Equivalently, we can express the heuristic performance in terms of a *worst-case relative error*.

trivial in comparison with the previous two methods. Many researchers and practitioners argue that worst-case analysis bounds are too loose to be useful in justifying solutions, and that probabilistic analysis tends to assume unrealistic probability space of the input data. However, they believe that both methods of analysis are useful in providing a better scientific understanding of the problem characteristics and the heuristic properties. Furthermore, the principal advantages of empirical testing are accuracy and certainty for the problems that are run. However, the empirical analysis is often performed in a subjective way rather than a scientific manner like in the previous analyses. There is also a lack of uniformity and no widely accepted guide-lines for conducting experimental analysis. In particular, there is a definite need for a standard set of easily obtainable test problems. For further information on empirical analysis, we refer to the work of Ball *et al.* [1981], and Golden *et al.* [1985].

In the thesis, only empirical methods are used to analyse the performance of the developed algorithms. This is based on the relative percentage error of the solutions, and on some evaluation criteria which will be explained in the next section.

2.1.4 Evaluation criteria for approximate algorithms.

(i) ***Quality of solution and computation time.***

Solution quality and computation time of an algorithm are important criteria to assess the effectiveness of an algorithm as discussed in Section 1.2 and Section 2.1.3. A reasonable running time is a very important element to algorithm evaluation and implementation. Therefore, a very desirable algorithm would be one that is equipped with a set of adjustable parameters, that would allow the user to meet changes in emphasis between cost and performance through controlling the trade-off between the quality of the solution and the amount of computational effort.

(ii) ***Code difficulty and ease of implementation.***

It is difficult to measure the intricacy and simplicity of coding of a particular algorithm. Algorithm principles must be simple, not cumbersome; generally applicable, not problem specific. This generality would enable an easy implementation of the algorithm to new domain areas with little apriori knowledge of the problem structure.

(iii) ***Flexibility.***

Since heuristic algorithms are typically involved in the solution of real world problems, it is important that they should be flexible. In particular, they should easily handle changes in the model, constraints and objective function.

(iv) ***Robustness.***

This class of algorithms have a number of desirable characteristics including: the ability to perform parametric analysis; a good characterisation that would enable a user to prove that the solution is within a certain percentage error (deviation) from the optimal solution; the ability to generate high quality feasible solutions - which do not depend strongly on the choice of the initial solution - whenever such a solution exists.

(v) ***Simplicity and analyzability.***

There is a significant appeal to algorithms that can be simply stated and that lend themselves more readily to analysis. Extremely complex algorithms are less likely to be analyzed in terms of flexibility, and quality of solution than a simple algorithm.

(vi) ***Interactive computing and technology changes.***

The idea of using man-machine interaction within the algorithm has been widely implemented in many systems. The seminal work in this respect is that of Krolak *et al.* [1971, 1972] for the TSP. It is now common knowledge that a good user interface makes a computerized system more appealing. An important factor in the effectiveness of an interactive system is the ability to portray solutions graphically. The user of the computerized system who may have some *feel* for the problem, could then avail the opportunity to make this experience known to the system. With the rapid advance in computer technology and the increase in micro-computer power, design of good user interfaces and the incorporation of the users experience through user interactions and inputs, would enhance the applicability and benefit of such algorithms. Algorithm designers should therefore take into account these new environments.

2.1.5 Disadvantages of approximate algorithms.

Some approximate algorithms produce *local optima* that usually depend on the initial starting solution. This is particularly so in the case of iterative improvement methods. These local optima can be very far from optimality. In addition, for most combinatorial problems there are no guide-lines available for an appropriate choice of initial solution. Furthermore, heuristic algorithms are not elegant from the mathematical point of view and the worst-case analysis is not known for most problems. For instance, the worst-case time complexity of Lin's well-known local search algorithm for the TSP that uses k-opt arc exchanges is still an open question.

In order to avoid some of the disadvantages of the local search methods, while maintaining its simplicity and generality, the following possibilities can be considered:

- Execution of the improvement algorithm for a large number of initial starting solutions. Performing a finite number of different random starts is computationally expensive and would never guarantee that an optimal solution is found (Papadimitriou *et al.* [1982]).

- Introduction of a more complex neighbourhood structure in order to generate better neighbouring solutions.

- The use of sophisticated learning strategies that gather information during the algorithm execution. This information can be used to penalise certain positions by adding penalty costs to the original data at the end of each run. The modified data is then passed into a construction procedure to generate different (possibly better) starting solutions for another execution of a composite algorithm.

- Accepting, in a limited way, some *disimproving* moves so as to escape from local optimality. Note that in local search methods only those moves that decrease the objective value are accepted.

2.1.6 Local search descent methods.

To understand simulated annealing and tabu search, we first need to understand the working of local search optimization methods. A combinatorial optimization problem P can be specified by identifying the set of its feasible solutions together with an objective function, C, that assigns a numerical value $C(S)$ to each solution S. An optimal solution is a solution with the minimum possible objective value.

Given an arbitrary starting solution to P, a local search descent method attempts to improve on that by a series of local impro ng changes. To define a local search descent algorithm, we define first a step-by-step method for perturbating solutions to obtain different ones. The set of solutions that can be obtained in one such step (or move) from a given solution S is called a *neighbourhood of* S, $N(S)$. The method that generates $N(S)$ is called *a neighbourhood generation mechanism.* In a narrower sense, we will refer to a *move* as a particular instance of mapping function, e.g., speaking of a move "from S to $n(S)$ in $N(S)$ ". A move, whose identity depends on the solution details of S as well as the neighbourhood mechanism N, will be called a *solution specific move*. In addition to the generation mechanism, the algorithm must determine the *search and selection criteria* of alternatives among the neighbours. There are two such obvious criteria or strategies: *a first improve* , FI, strategy which chooses the first neighbour S' that reduces the objective value upon detection; and *a best improve*, BI, strategy which selects the best S' that produces the largest decrease in the objective value in the whole neighbourhood of the current solution. The algorithm then performs the loop steps are depicted in Figure 2.1, and returns a solution S.

```
1.    Get an initial solution S.
2.    While there is an untested neighbour $S' \in N(S)$, execute the following.
      2.1   Let $S'$ be the trial neighbour of $S$.
      2.2   If $C(S') < C(S)$, then, set $S \leftarrow S'$.
            Otherwise, repeat Step 2.
3.    Stop, and return the local optimum $S$.
```

Figure 2.1. Local search descent method for a minimization problem.

Although, S need not be an optimal solution when the loop is finally exited, S will be *locally optimal* in that none of its neighbours has a lower objective value. Figure 2.2 illustrates the conceptual difference in local and global optimal solutions.

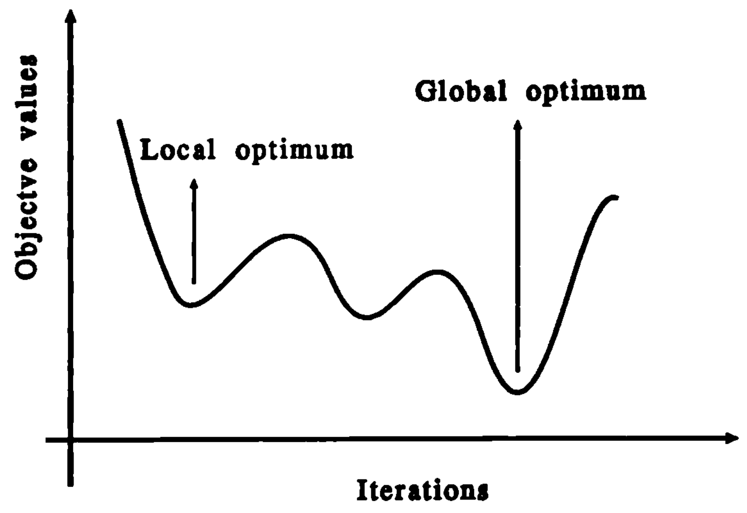


Figure 2.2 Global and local optimum.

2.1.7 Metastrategy algorithms and local search.

This section illustrates the general concepts that the metastrategy algorithms use in guiding local search methods. We first define what a *search strategy* means. A search strategy is a procedure that determines the order in which neighbouring solutions are searched, accepted or generated. We distinguish between a *blind* search and *guided* search. In the **blind** search the order in which solutions are generated depends only on the information gathered by the search during the execution of the algorithm. This order is unaffected by either the character of the unexplored portion of the neighbourhood or by the goal criterion. The goal is to avoid poor local optima that are often produced by the local search method. Local search descent methods are examples of this blind procedure, since they only accept a solution that generates a reduction in the objective function value. The ***guided*** search uses partial information about the problem domain and about the nature of the goal to direct the search toward the more promising regions. Metastrategy simulated annealing (SA) and tabu search (TS) algorithms combine different strategies such as learning, and disimproving, together with a good neighbourhood generation mechanism in order to escape poor local minima.

The Figure 2.3 shows a schematic representation of the computational search progress of the descent and the metastrategy SA and TS algorithms. In this figure, we start with an initial solution S_1 (either at random or by construction) with an objective function value $C(S_1)$. A neighbour of this solution $S_2 \in N(S_1)$ is generated by a suitable neighbourhood mechanism N and the change in the objective function values, $\Delta = C(S_2) - C(S_1)$, is calculated. If $\Delta < 0$ then all three procedures (descent, SA and TS) accept the generated neighbour as the new current solution; **otherwise** if $\Delta \geq 0$ S_1 is kept as the current solution in the descent method and the search continues to find other neighbours. For any *i*, if S_i is the *best* in its own neighbourhood $N(S_i)$ then, the descent method stops and declares S_i as its local minimum. However, the SA and TS procedures treat the case of $\Delta > 0$ differently from the descent method. Both simulated annealing (SA) and tabu search (TS) approaches allow the continuation of the search beyond the local optimality of descent methods. This implies that these methods accept *disimproving moves* (solutions that increase rather than decrease the objective value). However, each approach uses different strategies to achieve the common goal of escaping from local optimality.

In Figure 2.3, S_4 represents a local optimum. Let us assume that the SA procedure accepts a move to S_5 with an objective function value $C(S_5)$. This move has an associated increase $\delta = C(S_5) - C(S_4)$ in the objective value with an *acceptance probability,* which is not absolutely zero, but that depends on δ. Normally this acceptance probability is set to $e^{-\delta/T}$ where T is a *control parameter* which corresponds to *temperature* in the analogy with physical annealing. This acceptance probability function implies that small increases are more likely to be accepted than large increases. Moreover, when T is high most moves will be accepted, but as T approaches zero most disimprovement moves will be rejected. The SA algorithm starts with a high temperature and proceeds, by attempting a given number of neighbourhood moves at each temperature. The temperature parameter is controlled and is gradually lowered during the algorithm, according to a ***deterministic cooling schedule*** that also defines a stopping criterion (Kirkpatrick *et al.* [1983]).

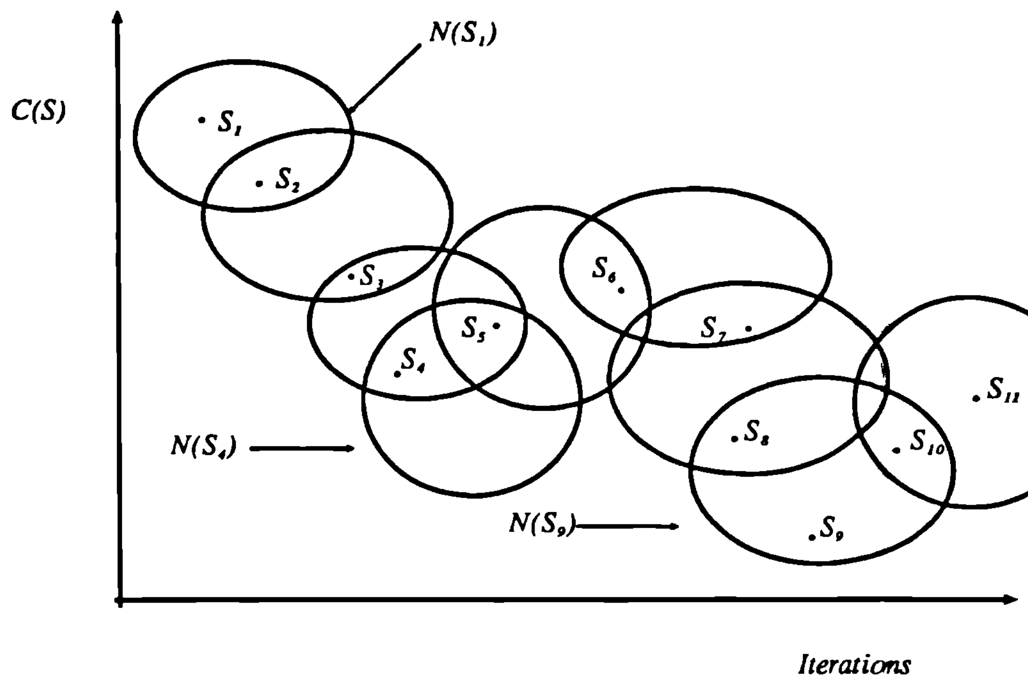


Figure 2.3. Progress of the local search and metastrategy search computations. S_1, S_4, S_9 are the initial, local and optimum solutions respectively. $N(S_i)$ depicts the neighbourhood of S_i.

The TS accepts a move to S_5 if it results in the ***highest evaluation*** move in the neighbourhood $N(S_4)$ of the current solution S_4. The TS algorithm assumes the use of the highest evaluation strategy. This strategy selects the best feasible solution in the neighbourhood that reduces the objective function value the most, or disimproves it the least. With this strategy, the pursuit of such moves could induce reversion to the same local optimum and, could, hence, result in *cycling*. Thus, a means to avoid cycling is necessary. This is accomplished by *forbidding* moves with certain *attributes* (making them tabu), and choosing moves from those remaining with the highest evaluation criterion. Tabu search structures the operation of descent methods employing *tabu conditions* that have the goal of preventing cycling and inducing the exploration of new regions (Glover [1989a, 1989b]).

Both SA and TS *may* obtain the optimal solution (S_9 in Figure 2.3) through a series of uphill and downhill moves and continues until a stopping rule is satisfied.

2.2 Simulated annealing.

2.2.1 Simulated annealing background

The simulated annealing metastrategy has its origins in statistical mechanics. The interest in SA began with the work of Kirkpatrick *et al.* [1983], and Cerny [1985]. They proposed a simulated annealing algorithm which 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 path* is melted by increasing the temperature of the heat path 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 each temperature value, widespread irregularities and defects can be locked into the crystal structure. This is known as rapid *quenching* which results in meta-stable structures.

In this analogy, the *states* of the solid correspond to the feasible solutions of a combinatorial optimization problem; the *energy* of the states correspond to the objective function value of the solutions; the minimum energy or *ground state* corresponds to an optimal solution; the *rapid quenching* process can be viewed as analogous to local optimization via steepest descent. When the external temperature is zero, no state transition can lead to a state of higher energy. Thus, as in local optimization, uphill moves are prohibited and the algorithm is trapped in a local minimum. When crystals are grown in practice, the bad local optima are avoided by a process of *careful annealing*. In this process, the temperature (T) descends slowly through a series of levels, each held long enough for the crystal melt to reach equilibrium at that temperature. As long as the temperature is not zero, uphill 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 $\Delta \geq 0$, a random number $\theta \in [0, 1]$ from the uniform distribution is drawn, and if $\theta \leq e^{-\Delta/T}$ then S' is accepted; 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 declining temperatures.

SA is more of an approach than a specific algorithm. In any application to a particular combinatorial optimization problem, we must take a number of decisions on *choices*. These choices fall into two classes according to (Johnson *et al.* [1989]): problem specific choices and generic choices for cooling schedules. These classes are further explained in the following sections.

2.2.2 Problem specific choices.

The list of choices are presumably specified in the optimization problem we are trying to solve. Improved performance can often be obtained by modifying the definition of these choices. This will be demonstrated in the applications covered in the thesis. Typical problem choices can be summarised as follows:

(i) A concise representation of the set of feasible solutions, an objective function value to be minimised and an initial starting solution.

(ii) A neighbourhood generation mechanism that generates neighbouring solutions must be defined. The efficiency of SA depends on the neighbourhood structure used. This is theoretically proven by the asymptotic convergence to the set of optimal solutions (Hajek [1988]).

(iii) Strategies for move selections and search order inside the neighbourhood must be defined. Also, an easy and an efficient method must be used for calculating the changes in the objective value of alternate solutions.

2.2.3 Generic choices.

Generic choices define the components of the cooling schedule. A cooling schedule must give specific answers to the following questions on how to determine:

(i) The initial starting value of the temperature parameter T.

(ii) The cooling rate and the temperature update rule.

(iii) The number of iterations to be performed at each temperature.

(iv) The termination of the algorithm (stopping criterion).

2.2.4 Cooling schedules.

The performance of the SA algorithm depends strongly on the chosen cooling schedule. With a proper cooling schedule, near optimal solutions can be reached for many combinatorial problems. A great variety of theoretical and practical cooling schedules have been suggested by many authors in the literature. In this section we distinguish between two types of cooling schedules which are further classified later.

***(i) Theoretical cooling schedules*:**

This type of cooling schedules provides asymptotical convergence of the SA algorithm to the set of globally optimal solutions with probability equal to one. The SA algorithm has been modelled using the theory of Markov chains with respect to the temperature parameter T, and a transition matrix P_{ij} which represents the transitional probability of moving from the solution state i to the solution state j. Two formulation models have emerged:

A *homogeneous* algorithm, in which T is reduced in an infinite sequence of *stages* until it reaches zero. A stage is normally defined as a Markov chain in which the temperature T is kept constant, Aarts *et al.*[1985]. The Markov chain *length*, L, (temperature duration) is the number of iteration moves performed at a given temperature value. It can be shown that, if T is kept constant for an infinite number of iteration moves L, and if any two solution states i and j can be reached from each other in a finite number of moves with positive probabilities (i.e. the solution space is completely connected), then this Markov chain has a unique stationary (steady-state) distribution independent of the starting state. Thus, as $T \rightarrow 0$ the corresponding *stationary distribution* is uniform distribution over the set of optimal solutions. These type of models have been studied by many authors including Aarts *et al.* [1985] and Lundy *et al.* [1986].

An *inhomogeneous* algorithm is also one in which T is reduced to zero in an infinite sequence of stages. However, each stage represents *only one* iteration move. More precisely, one iteration (attempted move) is performed at each temperature before the temperature is reduced (a Markov chain of length, L=1). The transitional probabilities, P_{ij} are now dependent on the number of iterations. For this class of algorithms, several results have been derived giving sufficient conditions for asymptotic convergence to the set of optimal solutions (see Gidas [1985], Mitra *et al.* [1986], and Hajek [1988]). The important results of Hajek [1988] offer a necessary and sufficient condition for the asymptotic convergence provided that the sequence of temperatures $T_k \rightarrow 0$, as $k \rightarrow \infty$ but not faster than $O\left(\frac{1}{\ln(1+k)}\right)$ at each iteration k. More precisely, The cooling schedule has the update rule $T_k = \frac{d}{\ln(1+k)}$, the condition for convergence is that d be greater than or equal to the *depth*, suitably defined, of the deepest local minimum which is not a global minimum. The depth of a local minimum (say S) is the smallest number E, E > 0 such that some solution S' with its objective value $C(S') < C(S)$ can be reached from S at *height* $C(S)+\text{E}$. A solution S' is said to be reachable at height E from S if $S = S'$ and $C(S) \leq \text{E}$ or if there is a sequence of solutions $S = S_0, S_1, \ldots, S_p$ for some $p \geq 1$ such that $S_{i+1} \in N(S_i)$ for $0 \leq i < p$ and $C(S_i) \leq \text{E}$ for $0 \leq i \leq 0$.

***(ii) Simple cooling schedules*:**

In addition to theoretically based cooling schedules, there are many other practically based, simple cooling schedules, which are similar in nature to that of Kirkpatrick *et al.* [1983]. These schedules are in practice vastly superior to those based on the strict adherence to the purely theoretical principles, since the latter are motivated primarily by the need to be able to *prove* convergence - however slow that may be.

2.2.5 Classification of cooling schedules.

In any implementation, the conditions that guarantee asymptotic convergence to the set of global minima can only be approximated. Due to these approximations, identification of the global optimum in less than exponential time by the SA algorithm can not be guaranteed with probability one. However, asymptotic convergence results are of great help when trying to approximate the corresponding cooling parameters. In the following, we will classify the cooling schedules from the literature into three categories (Osman & Christofides [1989]). Each category includes both the simple theoretical and practical cooling schedules.

It has to be pointed out that all implemented annealing schemes (practical or theoretical) use a random selection of neighbours during the algorithm search. Thus, in the following, we assume this random recourse throughout, unless otherwise stated. Furthermore, we assume throughout the thesis that an iteration is an attempted move, whether it is accepted or not.

2.2.5.1 Stepwise temperature reduction schemes.

This category includes implementations in which a fixed Markov chain length, L, is performed at a given value of the temperature T, before T is reduced in stages according to some specific rules. This reduction is illustrated in Figure 2.4. In this category, there are two types of cooling schedules:

1- The Simple Cooling Schedules.

- *Initial and final temperature values* (T_s and T_f)

Kirkpatrick *et al.* [1983] propose the following empirical rule to choose a value for T_s. They started with an initial large value for T_s. A few hundred moves are then attempted in order to determine the fraction of the accepted moves to the total number of moves attempted at the given value of T_s. If this fraction is less than an given *acceptance ratio* χ (say $\chi = 0.8$), then the initial temperature value is doubled. This procedure continues until the observed fraction exceeds this threshold. The SA algorithm continues with the temperature reduced in a stepwise fashion, using expression (2.2) for its temperature update rule, until the system *freezes* at a value of T_f (less than 1, if the cost coefficients in the problem are integers) at which the algorithm terminates.

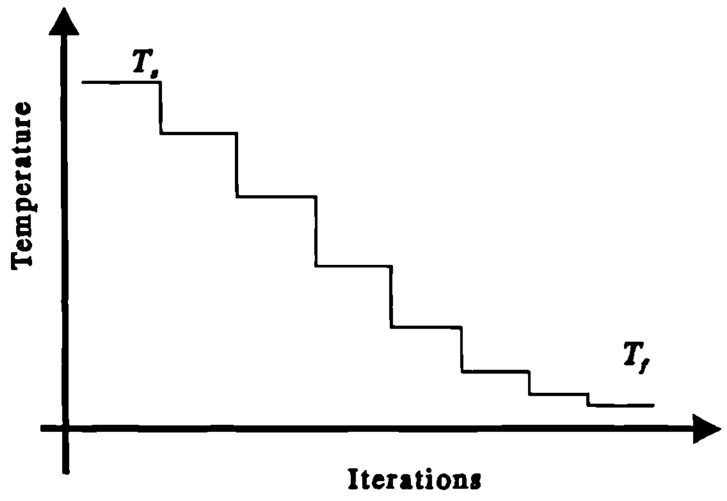


Figure 2.4. Stepwise temperature reduction schemes.

This initial rule was further refined by Johnson *et al.* [1989]. The initial T_s value was determined so that the probability of accepting the average change in the objective values, $\overline{\Delta}$, of a number of initial random moves was equal to a given acceptance ratio χ_s (say $\chi_s = 0.8$). More precisely, they solved the following expressions to obtain a value for T_s:

$$\chi_s = e^{\left(\frac{-\bar{\Delta}}{T_s}\right)}$$

hence, $$T_s = \frac{-\bar{\Delta}}{\ln(\chi_s)} \tag{2.1}$$

Note that $\bar{\Delta}$, the average was taken only for those moves which produced a positive increase in the objective function.

Bonomi *et al.*'s [1984] stopping criterion uses a fixed number of temperature reduction stages for which the algorithm is to be executed. Kirkpatrick *et al.*'s [1983] termination criterion is based on the solution changes. If the last solutions for a number of consecutive Markov chains length are identical, the algorithm is stopped. This latter stopping criterion is further refined by Johnson *et al.* [1989]: If a number of consecutive chains (say 5) have a percentage of accepted moves which is less than or equal to a given percentage value χ_f (say 2 %), then the annealing process is declared to be *frozen* and is stopped.

• *The decrement ratio and the temperature update rule.*

The value of the decrement ratio is chosen, such that the successive Markov chain lengths, *L*, can be kept small, but long enough to achieve quasi equilibrium before the temperature is updated. A frequently used rule is known as *the geometric cooling* rule. This rule updates the temperature sequence in the following way:

$$T_{k+1} = c \times T_k, \qquad k = 0, 1, \ldots \tag{2.2}$$

where *c* is a temperature factor- a constant value smaller than but close to 1 - generally taken to be in the interval [0.50, 0.99]. Kirkpatrick *et al.* [1983] propose this rule first with c= 0.90 and the temperature duration $L = n$, where *n* is the number of variables in the problem. This rule is also adopted by Johnson *et al.* [1989] who use L=$m \times |N|$, where m is a constant and $| N |$ is the expected size of the neighbourhood. The advantage of this rule is that the temperature duration will remain proportional to the number of neighbours no matter what the problem size. Bonomi *et al.* [1984] used a fixed number of temperature reductions, and the same rule for the temperature updates. The temperatures are reduced in a sequence 50 steps to the final temperature $T_f < 1$, and at each temperature value, the temperature duration (L) is given an arbitrary value of 9000 iterations for a TSP instance of size 400 cities.

2- The theoretically approximated cooling schedules.

In this class, we consider the cooling schedule of Aarts *et al.* [1985] and discuss, briefly, a few other schedules. The temperature cooling pattern was shown earlier in Figure 2.4.

The initial temperature value T_s.

The initial temperature, T_s, is obtained by monitoring the evolution of the system during a number of moves before the actual optimization process starts. Let us define m_1 and m_2 to be the number of moves with $\Delta_{ij} \leq 0$ and $\Delta_{ij} > 0$ respectively, $m_0 = m_1 + m_2$, and let $\overline{\Delta}$ be the average value of Δ_{ij} for those moves for which $\Delta_{ij} > 0$. The value of T_s is then adjusted in such a way that a constant value of the acceptance ratio, χ, is maintained. The value of T_s is then given by the final value of T. This final value is obtained by updating T,m_0 times according to the following expression

$$T = \overline{\Delta} \times \left(\ln \frac{m_2}{m_2 \times \chi - (1 - \chi) \times m_1} \right)^{-1} \qquad (2.3)$$

The above calculation of T_s can be reliable only if the objective values for the different moves are sufficiently uniformly distributed. In cases where the assumption is false, Equation 2.3 would result in a T_s value which is too small and, therefore, will cause the SA algorithm to get stuck in a local optimum. In this case, T_s is determined according to

$$T_s = \rho \times \max_{i,j \in I_\Omega} \Delta_{ij} \tag{2.4}$$

Where $\rho \gg 1$ (e.g., $\rho = 10$), Ω is the solution space - the finite set of all possible solutions - , and I_Ω is the set of the solution labels contained in Ω.

Temperature duration and cooling temperature decrease.

The decrements of the temperature are chosen to be small, thus avoiding the necessity of long Markov chains for establishing equilibrium at each new value of the temperature. Equilibrium means the establishment of a *stationary* steady-state probability distribution of the accessible state. Aarts *et al.* [1985] argue that the stationary distributions for two succeeding Markov chains should be close to each other. Thus, a small Markov chain length (the temperature duration L) is used. The value of L is taken to be equal to the maximum size of the neighbourhood of any state $i \in \Omega$. After each performed Markov chain (say k-th Markov chain), the average value $\overline{C}(T_k)$ and the standard deviation $\sigma(T_k)$ of the objective function are computed. The temperature is then updated according to the expression

$$T_{k+1} = T_k \times \left\{ 1 + \frac{\ln(1+\delta) \times T_k}{3 \times \sigma(T_k)} \right\}^{-1} \tag{2.5}$$

where δ is a given small decrement ratio specified at the beginning of the algorithm.

Termination criterion

The SA termination criterion is based on the decrease during the optimization process of the average value of the objective function over a number of Markov chains. Let us define S^* be the optimal solution. At the k-th Markov chain, the algorithm is terminated once $\overline{C}(T_k) - C(S^*)$ is small. Since S^* is not known, Aarts *et al.* [1985] estimate that if T_s is small then, the average difference in the objective function is as follows:

$$\overline{C}(T_k) - C(S^*) \cong T_k \times \left(\frac{\partial \overline{C}(T)}{\partial T} \right)_{T=T_k} \tag{2.6}$$

The algorithm is then terminated at a value of T_f if the following is satisfied

$$\left(\frac{\partial \overline{C}(T)}{\partial T} \right)_{T=T_f} \times \frac{T_f}{\overline{C}(T_s)} < \varepsilon \tag{2.7}$$

for some small real number, ε.

Note that, this cooling scheme uses a fixed Markov chain length, a constant decrement ratio, δ, and temperature update rule which depends on the change in the objective function. The most important among these parameters is the choice of the value of the decrement ratio, δ. A small value of δ produces a near optimal solution at a high computation time, while a larger value of δ produces the reverse (a lower quality solution at a smaller computation time). Thus, the value of δ represents the trade-off between the solution quality and computation time. The next important item in this scheme is the value of the stopping parameter ,ε, which has to be reasonably chosen.

This class of cooling schedules includes other schemes such as Huang *et al.* [1986]. Huang *et al.* [1986] specify the initial temperature T_s differently, based on White [1984]. White [1984] proposes that $T_s \gg \sigma$, where σ is the standard deviation of the objective function during a random sampling. The value of σ was obtained after an initial exploration of the solution space. Assuming a normal distribution of the objective function and a given probability χ to accept moves whose objective values are up to $r \times \sigma$ worse than the initial one, Huang *et al.* then computed a value for T_s as follows:

$$T_s = l \times \sigma \tag{2.8}$$

where l is evaluated by:

$$l = -\frac{r}{\ln(\chi)}$$

The scheme of Huang *et al.* [1986] also uses a variable Markov chain length rather than a fixed Markov length like in Aarts *et al.* [1985] scheme. In this scheme, the Markov chain length is chosen based on the following requirements:

(i) The number of accepted moves whose objective value is within the range $\pm\delta$ from the average objective value $\overline{C}$.

(ii) The total number of newly accepted moves.

If the ratio of the number in (i) to the number in (ii) equals to a given a certain value, χ then, a new chain is started. A typical value for δ is $\delta = 0.5 \times \sigma$. Assuming a normal distribution for the value of the objective function, it can be shown that $\chi = erf\left(\frac{\delta}{\sigma(T_k)}\right)$ where *erf*(x) is an error function. Huang *et al.* define the ***within count*** and the ***tolerance count*** to be the number of accepted moves with an objective value inside and outside the range $(\overline{C}(T) - \delta, \overline{C}(T) + \delta)$, respectively. Equilibrium is considered maintained if the within count exceeds a target value before the tolerance count exceeds a maximum tolerance limit. Typically, the target value equals to $\zeta \times \chi$ while the tolerance limit has a value of $\zeta \times (1 - \chi)$ for some parameter ζ depending on the size of the problem.

Although, the above conditions provide Markov chains of length that are dynamically adjusted, these lengths can be long at low value of T_k. To avoid this case, some other conditions were also imposed.

Huang *et al.* [1986] decrement rule requires the difference, between the average of the objective values of the last two consecutive Markov chains, to be less than the standard deviation of the objective function. This requirement is translated, and resulted in the following decrement rule:

$$T_{k+1} = T_k \times \exp\left(\frac{-\lambda \times T_k}{\sigma(T_k)}\right) \tag{2.9}$$

where λ is a constant less than one.

Once the equilibrium is established at the end of the k-th Markov chains, a check for the stopping criterion is made. This criterion is based on, the difference between the maximum and the minimum objective values among the accepted moves at T_k is compared with the maximum change in the objective value of any accepted move during the k-th chain. If they are the same, they concluded that all the accessed solutions have about the same objective values. Hence, there is no need to continue using the simulated annealing algorithm. At this stage, the algorithm is deemed to have reached its stopping criterion.

2.2.5.2 Continuous temperature reduction schemes

This category includes the class of cooling schedules which perform only one move at each temperature value. The temperature T is reduced after each iteration move (inhomogeneous Markov chain of length L equal to one). In this class, the temperature is continuously reduced according to some rule. A temperature evolution pattern is depicted in Figure 2.5. This class includes the cooling schedules of Hajek [1988], Lundy *et al.* [1986]. We only elaborate on Lundy *et al.* [1986] as it has some common features with the scheme developed in this thesis.

Lundy *et al.* [1986] build their cooling schedule based on the asymptotic convergence properties of the simulated annealing algorithm. This scheme uses a constant decrement ratio and an exponential temperature update rule. The advantage of this scheme is that it requires less worries about the optimal size of the temperature duration, as L=1, and needs only a careful choice for the decrement ratio.

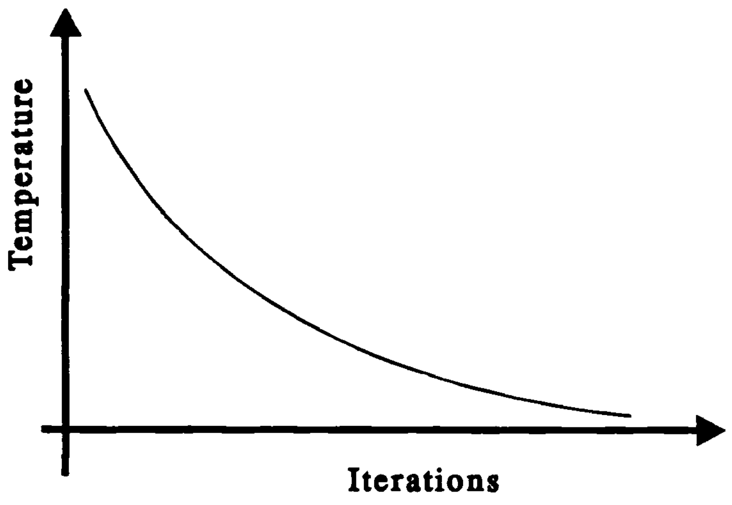


Figure 2.5. Continuous temperature reduction schemes.

The cooling scheme of Lundy *et al.* [1986] is summarised as follows:

The initial temperature value T_s.

It is proposed that the initial value T_s of T needs to be chosen so that $T_s \gg U$. U is an upper bound on the maximum change in the objective function between neighbouring solutions. This implies that in the initial stage the algorithm is likely to accept most changes in the solution. Thus, good solutions which can only be attained by a sequence of changes will not be missed. Since, such a sequence includes moves which increase the objective function.

The final temperature value T_f.

The final value of T, T_f is obtained by requiring that, the probability of the objective value of the current solution to be more than an error ε above the optimal objective value, to be less than some small real number, υ. More precisely, if S is the current solution at T_f and S^* is the optimal solution then, it is required that $Pr\{C(S) > C(S^*) + \varepsilon \mid T = T_f\} < \upsilon$. Solving this condition would lead to the following bound for the value of T_f:

$$T_f \leq \frac{\varepsilon}{\ln(|\Omega| - 1) - \ln(\upsilon)} \tag{2.10}$$

where $|\Omega|$ is the size of the solution space Ω. Equation (2.10) provides a strong condition in that T_f is always very small. More precise, the probability of accepting any uphill move is zero long before this condition is met. They suggested another practical stopping criterion such that, as soon as T_f gets small enough (say, $T_f < \frac{-1}{\ln(p_{\min})}$ where $p_{\min}$ is some small probability), the simulated annealing should be stopped. Furthermore, at this point, the optimization process may continue with an exhaustive search of the current final solution until no further improvement can be made.

The temperature update rule.

In this scheme, the temperature is reduced at each iteration by successively smaller amounts using a constant decrement ratio $\beta \ll 1\ U$, i.e. the temperature is updated after each iteration k according to the following rule:

$$T_{k+1} = T_k \times (1 + \beta \times T_k)^{-1} \tag{2.11}$$

It can shown that the total number of iterations M needed before the algorithm terminates satisfies $M < \frac{\ln|\Omega| - \ln(\upsilon)}{\beta \times \varepsilon}$. This means that the number of iterations M is essentially proportional to $\ln|\Omega|$. This value of M is used as a general guide to determine a bound on the number of iterations depending on β. Inversely, if we have definitive values for T_s, T_f and M then β can easily be evaluated as:

$$\beta = (T_s - T_f) \times (M \times T_s \times T_f)^{-1} \tag{2.12}$$

2.2.5.3 Non-monotonic Reduction Schemes

This class of non-monotonic reduction schemes reduce continuously the temperature after each attempted interchange, with occasional increases (or resets) in the temperature. The reset is an increase in the temperature to a higher value than the current one. The reset occurs whenever a complete search of the neighbourhood takes place without any accepted move. This means that there is no point in decreasing the temperature any further. A pictorial representation of the temperature pattern is illustrated in Figure 2.6.

This scheme is developed by Osman & Christofides [1989] to handle some of the disadvantages inherent in the previous two classes. *First*, the neighbourhood is often randomly searched to generate alternate solutions, at low temperature values of T, the probability of accepting worse solutions than the current one becomes small. If there are only a few neighbourhood solutions which give improved objective values then, the previous classes would miss or may take a long time to find them. Thus, a systematic search of the neighbourhood is preferred. This observation is also supported by the work of Johnson *et al.* [1989], and Connolly [1990].

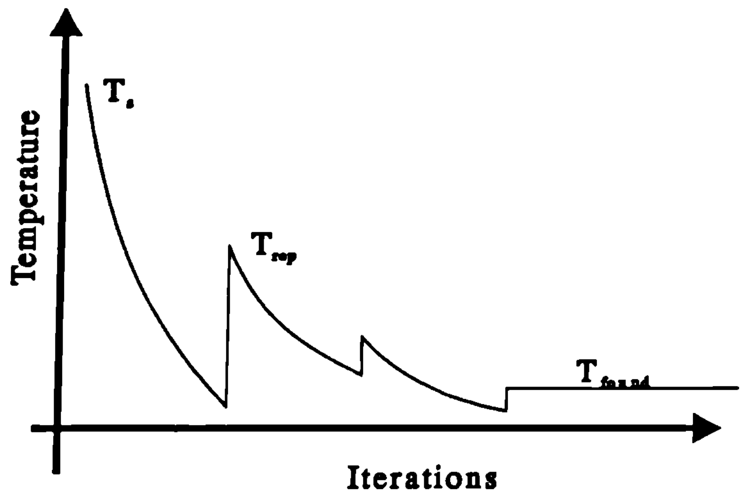


Figure 2.6. Non-monotonic temperature reduction scheme.

Second, in SA schemes worse solutions can be accepted. Therefore, It is possible that the SA solution at the end of the search is worse than the best solution found during the run. Hence, storing the best solution will be an advantage. Doing so, our SA approach would then resemble the tabu search algorithm in one aspect. Similar, approaches which store the best solution are also implemented by Matsuo *et al.* [1989] and Connolly [1990].

The SA scheme Osman & Christofides [1989] gives the user control over the trade-off between the quality of solution and the computer time through the number of temperature resets required. In this scheme, the temperature is reset initially to

$T_{rep} = \frac{T_s}{\mu}$ where $\mu \geq 2$. Subsequent resets update the value of T_{rep} to $\frac{T_{rep}}{\mu}$. If the updated value of T_{rep} is greater the temperature before the reset then, the current temperature is updated. Otherwise, the temperature is reset to the temperature at which the best solution is found, T_{found}. This scheme uses a variable decrement ratio whose values are related to the number of iterations and to the characteristic of the problem being solved. Both the multiple resets and the variable decrement ratio would allow the running time to be spread more effectively rather than being wasted at the beginning or end of the cooling schedule. A more detailed description of this cooling schedule will be found in later chapters.

Connolly's [1990] scheme uses the decrement ratio and the temperature rule of the Lundy *et al.* [1986] scheme (Section 2.2.5.2). With the exception, that the temperature is reset only once to T_{found}. The reset occurs as soon as a neighbourhood search happened without any move is accepted. After the reset, β is set to zero. Then, the search is continued for a pre-specified number of iterations with a constant temperature, T_{found}. The temperature pattern of this scheme is shown in Figure 2.7.

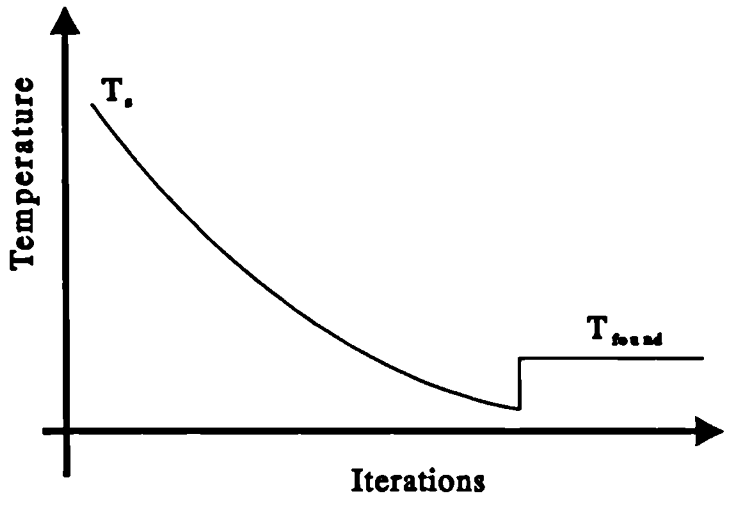


Figure 2.7. Connolly's temperature reduction scheme.

Connolly's [1989] scheme would work on problems having shallow optima (i.e., a *smooth* topology, see Figure 2.8b). However, in problems with deep local minima (i.e, a *bumpy* topology, see Figure 2.8a) this scheme may fail.

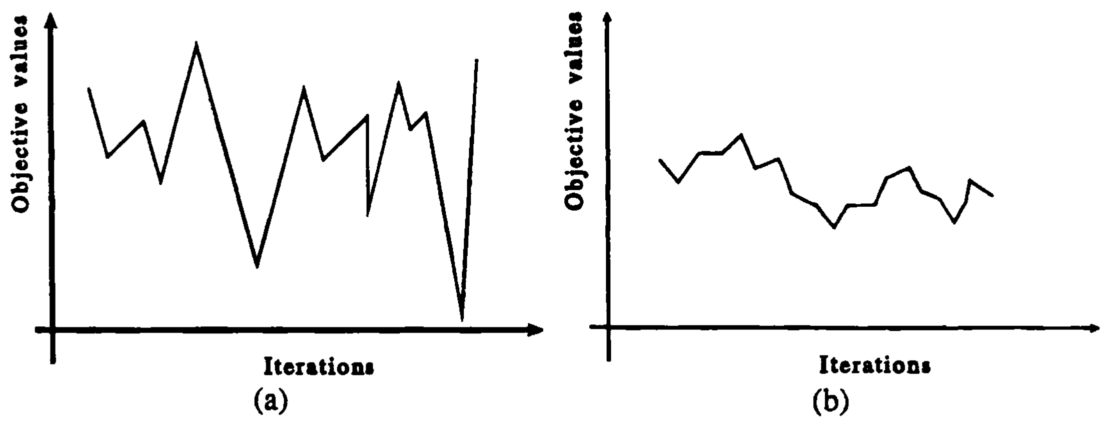


Figure 2.8 Topological representation of objective values

2.2.6 Simulated annealing procedure.

Having defined previously, all simulated annealing terminology and its analogy with statistical mechanics, SA can be seen as a generalised iterative improvement algorithm. The SA algorithm steps are summarised as follows:

```
Step 1  Generate an initial random or heuristic solution $S$.
        Set an initial temperature $T_s$ for $T$, and other generic parameters (see Section 2.2.3).
Step 2  Choose a solution $S' \in N_\lambda(S)$ and
        compute the difference in the objective values, $\Delta = C(S') - C(S)$.
Step 3  If:
        (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. $e^{\left(-\frac{\Delta}{T}\right)} > \theta$, (where $0 < \theta < 1$ is a
              random number).
        Then  replace $S$ by $S'$.
        Else  Retain the current solution $S$.
Step 4  Update the temperature T depending on a set of rules, including:
        (i)   The cooling schedule used (see Section 2.2.5),
        (ii)  Whether an improvement was obtained in step 3 above,
        (iii) Whether the neighbourhood $N_\lambda(S)$ has been completely searched.
Step 5  If a "stopping test" is successful stop, else go to step 2.
```

2.2.7 Performance of SA on combinatorial optimization problems.

Ever since its introduction in 1983, simulated annealing has been applied to a large number of different combinatorial optimization problem and in diverse areas. We confine ourselves to applications where some concluding remarks and novelties can be drawn. In addition, we review some of the more recent applications that have not been listed in the surveys mentioned here.

2.2.7.1 The travelling salesman problem (TSP).

The TSP problem acts as a test bed for almost every new algorithm. The TSP consists of finding the shortest Hamiltonian circuit in a complete graph, where the nodes represent cities. The cost of a tour is the total distance covered in traversing all the cities.

Kirkpatrick *et al.* [1983] and Cerny [1985] introduce the SA algorithm and present its application to the TSP. Cerny [1985] reports that the effectiveness of SA depends on the neighbourhood structure that is used. Kirkpatrick [1984] studies a special class of the TSP in the plane. He studies the performance of a SA algorithm which uses a simple cooling scheme (Section 2.2.5.1) incorporating repeated applications of the well-known 2-opt or 3-opt iterative procedures of Lin *et al.* [1973]. He concludes that SA out-performs solutions found by Lin's arc exchange iterative methods, Lin [1965]. Bonomi *et al.* [1984] apply SA to the TSP in the plane using a simple cooling schedule, with a decrement ratio $c= 0.925$. The neighbourhood generation mechanism is restricted through partitioning the unit square into disjoint subregions. Two cities from the same region or two adjacent subregions are chosen at random to be exchanged. Experiments on problem instances of size $n= 100$, 200, and 400 cities shows that the SA algorithm always produces better solutions than Lin's 2-opt procedures. In this application, the 2-opt procedure is rejected for large-sized problems, and it is coupled with the *convex hull* procedure of Golden *et al.* [1980] to enhance the overall composite algorithm. The performance of this composite procedure is then compared to that of their SA algorithm for $n \geq 200$. This procedure out-performs the SA algorithm in only 3 out 100 problem instances. Furthermore, probabilistic analysis of their SA on the 400 city problems and on a TSP with 10,000 cities shows that the cost of the final solutions are $0.749\sqrt{n}$ and $0.763\sqrt{n}$ for the 400-city and the 10,000-city problems, respectively. For the TSP in the plane, the optimal solution is proven to converge asymptotically to $0.749\sqrt{n}$ as $n \rightarrow \infty$. Moreover, the complexity of their SA algorithm is polynomially bounded by

$O(n^p)$ with $1 < p < 2$. Thus, their conclusion is that the SA algorithm is a powerful tool for solving large and difficult problems provided it is supplemented by an efficient move selection mechanism that favours the choice of *important* moves. Aarts *et al.* [1985] also use an $O(n^3 \ln(n))$ SA algorithm to solve this class of TSP and report that the average solution is less than 2% from the optimal solution.

Golden *et al.* [1986] use simulated annealing to solve the TSP and p-median location problems. They implement a SA scheme which reduces the temperature in 25 equal intervals waiting for equilibrium to be achieved at each temperature value before it is updated. They compare their SA scheme with a sophisticated hybrid procedure CCAO, (Golden *et al.* [1985]), on a number of well- known test TSP's from the literature. The CCAO heuristic combines *convex hull* with greatest angle and cheapest insertion criteria, followed by a branch exchange procedure. They conclude that CCAO out-performs SA in both solution quality and computation time. They find that SA is extremely sensitive to small changes in the cooling schedule. On the other hand, Van Laarhoven *et al.* [1987] use the scheme reported in Aarts *et al.*[1985] (Section 2.2.5.1) to solve well-known TSP test problems. For example, their scheme, when tested on the 318-city problem of Lin *et al.* [1973], produces a solution with a relative percentage deviation of 1.12% in 811.2 minutes. In contrast, the SA algorithm of Golden *et al.* [1986], produces a relative percentage deviation of 4.03% in 713.6 minutes. However, the results of Golden *et al.* [1986] show that their SA performs poorly in comparison to their CCAO algorithm, since for the 318-city example the relative performance deviation of the CCAO was 1.01%, and was produced in 43.55 minutes. The computation time of all the above algorithms were obtained using the same type of VAX 11/780-computer.

We conclude that it is of great importance to choose a good cooling schedule for the SA algorithm to be effective. The scheme of Aarts *et al.* [1985] out-performs the

SA scheme of Golden *et al.* [1986]. However, SA is out-performed by the tailored CCAO heuristic, both with respect to solution quality and computation time. More research is needed to analyse the merit of SA on the TSPs. Johnson *et al.* [1989] have reported applications of SA to the TSP, graph colouring and partitioning problems.

2.2.7.2 The quadratic assignment problem (QAP).

After the TSP, The QAP is one of the most studied problem using simulated annealing. The QAP is the problem of assigning inter-communicating objects to locations so as to minimise the total cost of communication between them.

Burkard *et al.* [1984] use a step-wise reduction simple cooling schedule. They adopt the temperature update rule (Equation 2.2) with a decrement ratio $c= 0.5$. The total number of iteration moves performed at each temperature, L, is increased as the search proceeds. In other words, L is initially set to be equal to the size of the neighbourhood, $| N |$. Subsequently, L is multiplied by a constant, 1.1 (i.e., $L= 1.1{\times}L$). Not many details are given regarding the initial temperature and the stopping criterion. They compare the SA scheme with sophisticated techniques and descent methods. They conclude that the SA algorithm reveals an excellent behaviour, and produces solutions which are 1 to 2% away from the best known solutions. Unlike the SA approach, the solution quality of their descent method decreases as the size of the problem increases. Furthermore, their descent method depends heavily on the choice of the starting solution.

Wilhem *et al.* [1987] report on the use of another SA scheme which involves waiting for convergence at each temperature before cooling to the next temperature values. Although, they report a large number of results for a variety of parameter settings, they were unable to indicate a consistently dominant set of parameters that would produce best results. Nevertheless, their results are better than those of Burkard *et al.* [1984].

Connolly [1990] presents a non-monotonic cooling scheme which has been discussed in the classification of cooling schedules (See Section 2.2.5.3). Comparison of the SA schemes show that both the Burkard *et al.*[1984] and Wilhem *et al.*[1987] annealing algorithms are out-performed by the SA annealing approach of Connolly [1990] in terms of quality of solutions. The compared algorithms are allowed the same amount of running time. The conclusion is that SA is an extremely efficient heuristic for the QAP.

2.2.7.3 Machine scheduling problems

1- The permutation flow-shop scheduling problem. (FSSP)

The FSSP may be stated as follows: Each of n jobs is to be processed on machines $1,..,m$ in that order. P_{ij} is the processing time of job i on machine j. At any time, each machine can process at most one job and each job can be processed on at most one machine i.e., preemption is not allowed. The sequence in which the jobs are to be processed is the same for each machine. The objective is to find a sequence of jobs which minimizes the maximum completion time C_{max}.

Osman & Potts [1989] have investigated the application of SA algorithms and descent methods to the solutions of the FSSP. The implemented SA algorithm uses a continuous temperature reduction scheme. The decrement ratio and temperature cooling rule is the same as Lundy *et al.* [1986] (Equations 2.12 and 2.11). The initial temperature is set arbitrarily to a fifth of the average processing time of all the jobs on the machines, and the final temperature value is set to the smallest change in the objective function. The number of iterations was estimated in terms of the problem characteristics. The computational time required by the algorithm was shown to be $O(nm\ln(n+m))$ and near optimal solutions were produced. Different neighbourhood search methods were tried and the best was identified. The SA algorithm is compared with two composite descent methods. The composite descent algorithms are hybrid combinations of several constructive methods and local search descent procedures, DES. The first uses the NEH procedure of Nawz *et al.* [1983] as a starting sequence, while the second uses the PCDSD procedure which is the best of $m+1$ sequences generated from the constructive methods of Palmer [1965], and Campell *et al.* [1970]. Computational results show that the hybrid NEH+DES procedure produces the same results as the PCDSD+DES procedure, in less time. The merits of SA with the NEH+DES procedure remains to be analysed. Over all test problems with sizes up to 100 jobs and 20 machines, the SA algorithm produces an average relative percentage deviation of 0.49% and an average computational run time of 8.84 seconds. The comparative figures for the NEH+DES algorithm are 1.15% and 6.62 seconds. Thus, the simulated annealing of Osman & Potts [1989] is better than others methods and has become the state of the art method for solving the FSSP. Further details on SA experiments of the FSSP can be found also in Osman [1987].

2- **The single machine weighted tardiness problem.** (SMWT)

The SMWT problem is characterised by n jobs available for processing on a single machine. Each job has an associated weight and a due date. The objective is to find the non-preemptive sequence of the n jobs on the single machine that minimizes the total weighted tardiness cost.

Matsuo *et al.* [1989] present a SA annealing scheme with a step-wise temperature reduction scheme which replaces the Metropolis (or exponential) acceptance probabilities at each temperature by a linear function that is independent of the change in the objective value. The scheme also restricts the search to a small neighbourhood, and retains the best solution obtained during the search as opposed to the solution where the annealing stops. The scheme uses a good heuristic as a starting solution along with a low initial acceptance probability. Computational results are compared to the solution produced by the SA scheme of Aarts *et al.* [1985]. The new scheme yields results that are nearly as good as the latter. Moreover, the time required to obtain this is far less-by a factor of 10. In cases where the algorithm produces worse results, the initial acceptance probability is increased and the results are improved. Although this adjustment increases the computational effort, this new SA algorithm out-performs that of Aarts *et al.* [1985] while consuming a similar amount of computation time.

2.2.7.4 The graph partitioning problem. (GPP)

Given a graph $G=(V,E)$, where V is the vertex set and E, the edge set, the GPP is one of partitioning V into two equal sized sets V_1 and V_2, $V = V_1 \cup V_2$ such that the number of edges that have end-points in different sets is minimized. The problem is of great importance in circuit design.

Johnson *et al.* [1989] report an empirical study of the simulated annealing approach to the GPP. Their annealing scheme uses a simple cooling schedule with a step-wise reduction strategy. The initial temperature is determined so that the acceptance ratio, $\chi = 0.4$, the decrement ratio, $c= 0.95$, the temperature duration $L = 16 \times | N |$ where $| N |$ is the size of neighbourhood of any solution, and the termination criterion depend on the number of consecutive Markov chains with no improved solution (see Section 2.2.5.1 for further detail). Simulated annealing was compared with the well known traditional heuristic of Kernighan & Lin [1970]. They conclude that SA seems to be a competitive approach to the problem. For certain types of random graphs, (*dense* and of large-sized problem), it appears to out-perform the traditional heuristics of Kernighan & Lin [1970] as well as the more recent improvements of it, both, with respect to solution quality as well as computation time. However, if the graph is *sparse* and or of small size, it may well be better to spend an equivalent amount of time performing multiple runs of a heuristic specially tuned to the problem at hand. Simulated annealing is out-performed by other heuristics on *geometric* graphs. The poorer performance on this class of problems may well be traceable to the fact that the topology of the solution space is *bumpy* (see Figure 2.8 (a)) and thus, local optima may be far away from each other. Thus, SA is much more likely to be trapped in deep local optima. Their experiments also provide some insight on how annealing should be implemented. Their conclusion can be summarised as follows:

1- It is better to perform a long run than to take the best of a time-equivalent collection of shorter runs.

2- Systematic sampling of the neighbourhood is better than totally random sampling. Similar conclusions are drawn in Osman & Christofides [1989] when SA is applied to the capacitated clustering problem (CCP).

3- Both quality of solution and running time may be improved by the use of a good heuristic. Heuristics which take advantage of some special structure of the problem seem to be preferred to general ones. The initial temperature must be relatively small otherwise the benefit of the good initial solution will be lost. This is in agreement with Osman & Christofides [1989], in which SA is combined with an initial constructive heuristic solution. The results are almost the same as those produced by multiple runs of SA with random starts. However, starting simulated annealing from a constructive heuristic takes less computation time than that with an initial random start.

4- Replacing the computation of the exponential acceptance probability function with a simple look-up approximation table reduces the computation time by a third without sacrificing quality of the solutions.

5- It may be possible to allow violations of some constraints so long as a penalty for the violation is included in the cost function. This would allow for a smoother solution space in which local optima are easier to escape from.

2.2.7.5 The graph colouring problem (GCP)

The GCP is the problem of finding a *k-coloring* of a graph G=(V,E) with node set V and edge set E, i.e a partition of V into *k independent sets* $V_1, \ldots, V_k$. A set V_i of node is called *independent or stable* if no two nodes in V_i are linked by an edge in G. The objective is to find the minimum possible value of *k*.

Chams *et al.* [1987] implements a simple step-wise reduction temperature scheme for their cooling schedule to solve GCP. They compare their SA approach with other approaches. They conclude that SA seems to be extremely efficient for small sized problems (up to 200 nodes). Furthermore, the best results (in terms of CPU time and the number of *k* colours used) are obtained by combining simulated annealing with other techniques. Using SA alone may not be the best alternative, since this procedure does not use any knowledge we might have of the structure of the problem. It would be advisable to use as much as possible the technical knowledge of the problem to provide a *good* starting solution and then apply SA for the crucial last steps only to improve it. This is the conclusion reached in Johnson *et al.* [1989] and Osman & Christofides [1989] who have found that a combination of simulated annealing with other techniques may improve the performance in terms of solution quality or computation time (or both).

2.2.7.6 The multi-constraint knapsack problem. (MCKP)

Drexl [1988] solves the MCKP problem using a SA approach which uses a simple geometric cooling schedule (a simple step-wise temperature reduction). The temperature duration *L* is set to be equal to the size of the problem *n* and is gradually increased by a multiplication factor of 1.2 at the end of each Markov chain. Simulated annealing is compared to a descent method and a restricted branch and bound approach by Gavish *et al.* [1985]. The latter is the most efficient algorithm for solving MCKP exactly with the number of constraints m less than 5. Simulated annealing works as fast and efficiently as the branch and bound heuristic with arbitrary sizes of m and n. Furthermore, the gap from the optimal solution never exceeds 1%, while the descent heuristic yields results with maximum deviation of 9%. However, the descent heuristic is, on average, 20 times faster than the SA algorithm.

Drexl [1988] reports the first application of the SA approach on a combinatorial optimization problem with capacity constraints. The solution is represented by a single array of length n (a permutation). Each entry is either 0 or 1. Neighbouring solutions are obtained by switching one or two entries (to 0 or 1). Thus, not all generated solutions are feasible.

For infeasible solutions obtained during any SA algorithm, three possibilities exist:

1- It may be possible to allow violations of some constraints so long as a penalty for the violation is included in the objective function, as in Johnson *et al.* [1989]. This approach is similar to lagrangean relaxation.

2- It may be possible to restrict the search to only feasible solutions and reject all infeasible ones as in Osman and Christofides [1989].

3- It may be possible to add a second *Monte-Carlo* simulation in order to decide whether each infeasible solution should be accepted or rejected. This approach is adopted in Drexl [1988]

2.2.7.7 The generalised assignment problem (GAP)

A short definition of this problem is given in Chapter 1 and it will be studied comprehensively in Chapter 4. Cattrysse [1990] reports an application of simulated annealing to the GAP. The cooling scheme is a simple step-wise reduction schedule. The SA algorithm is explained briefly here. The reason is that we are comparing its results with our SA results in Chapter 4. Cattrysse's [1990] SA algorithm is described as follows: the initial and final temperatures are evaluated so that the probability of accepting a 1% change is 90 % at the start and equals to 0.1% at the end; the temperature update rule is given by equation (2.2) with a decrement ratio c= 0.98; the total number of iterations performed is $m \times n \times 100 \times mp$ (where mp is a multiplier, n is the number of jobs, m is the number of agents); the temperature length L is a fixed value expressed in terms of the total number of iterations and the initial and the final temperatures.

The problem specific choices are as follows: the initial starting solution is random (feasible or infeasible); the solution is represented by an array of size n in which an array entry gives the index of agent to process the corresponding job. For example, a solution $S = (2,1,2,1)$ indicates that jobs 1 and 3 are assigned to agent 2 and jobs 2 and 4 assigned to agent 1, assuming that the jobs are in the order 1,2,..,n,. The neighbourhood solutions are generated by performing 30% *random shift* operations of jobs and 70% *random switch* operations of two job assignments. Monte-Carlo simulation is added in order to decide on the acceptance or rejection of infeasible solutions that may be encountered. This follows an approach similar to that of Drexl [1988].

The annealing scheme is compared with another set partitioning approach that was developed (SP). The results from simulated annealing are very discouraging and have a poorer performance. The best of SA with a multiplier mp= 50, is with an associated average relative percentage deviation of 3.90% at an average of 2660 CPU seconds of micro-computer time compared to 0.09% at an average of 1565 seconds using the SP approach. SA is tested under many control settings. However, it was not possible to select one superior set. The performance of simulated annealing is improved by starting with a pre-processing heuristic. A simple prepossessing procedure, called *a fixing* *heuristic* F, is used to reduce the problem to a smaller one. This fixing procedure solves the linear programming (LP) formulation (Section 4.1 of Chapter 4). Then, based on the solution variables, valid inequalities (that are violated) and/or facets of the polytope are added to the formulation. Again, the extended LP is then solved. At every iteration, an equality is added for every *knapsack* constraint. When no violated inequalities can be added, all variables equal to one are permanently fixed and a reduced problem is created. The reduced problem is then solved with the SA and SP approaches to produce two FSA and FSP new heuristic solutions. The solution of FSA with *mp*= 20 was improved to produce an average percentage deviating 0.72% in an average time of 520 seconds compared to FSP 0.09% in an average of 68 seconds.

Our SA implementation for the GAP in Chapter 4 shows that our SA is better than both the FSP and FSA approaches, both, in computation time as well as in the quality of solution. When tested on the same set of problems as in Cattrysse [1990] the average relative deviation of our SA is 0.04% obtained in an average of 47 seconds (adjusted time to take into account the difference in computer speed).

2.2.7.8 Other Combinatorial applications.

SA has been applied to many problems where no problem-specific algorithms were available. These include the controlled rounding problems (Kelly *et al.* [1990]), multi-level lot-sizing problems (Kuick *et al.* [1990]), scheduling a manufacturing cell (Vakharia [1990]), placement of shapeable blocks (De Bont *et al.* [1988]), conference seminar time-tabling (Eglese *et al.* [1987]), locomotive scheduling problem (Wright [1989]), construction of exact optimal designs for linear regression models (Haines [1987] and Bohachevsky *et al.* [1986]). Major books and surveys on the theory and applications of SA are Van Laarhoven *et al.* [1987], Collins *et al.* [1988], Aarts and Korts [1988], Johnson *et al.* [1989], and Eglese [1990].

2.3 Tabu search

2.3.1 Background

Tabu search (TS) is a metastrategy iterative procedure introduced independently by Glover [1986] and Hansen [1986] for solving optimization problems. It is based on the general tenets of intelligent problem solving. Tabu search shares with simulated annealing, the ability to guide the search of iterative (local search) improvement methods. In this context, tabu search provides a guiding framework for exploring the solution space beyond points where an embedded heuristic would become trapped in a local minimum. The process in which the TS method seeks to transcend local optimality is based on an *evaluation function* that chooses the highest *evaluation* move at each iteration. The steps of the TS generic procedure are depicted below:

```
Step 1.   Get an initial solution $S$.
Step 2.   Select the best admissible solution, $S_{best}$
          ($S_{best}$ is the best of all $S' \in N(S)$: $S'$ is not in the tabu-list).
Step 3.   Update the current solution $S \leftarrow S_{best}$, and update the tabu-list.
Step 4.   Repeat Step 2 and Step 3 until a stopping criterion is satisfied.
```

The core of this procedure is in Step 2 and Step 3. The *best admissible* move is the highest evaluation move in the neighbourhood of the current solution in terms of the objective value and the *tabu restrictions*. The highest evaluation function selects the move which produces the most improvement or the least *disimprovement* in the objective function. The reason for introducing a tabu list is to store in that list characteristics of accepted moves so that these characteristics can be used to classify certain moves as tabu (i.e., to be avoided) in later iterations. By accepting disimproving moves, it becomes possible to return to solutions already visited. Hence, cycling may occur and the purpose of the tabu list is to prevent such occurrence. Thus, it is necessary to constrain and restrict the search by a *forbidding* strategy whose only function is to control and update the tabu-list. The goal of the forbidding strategy is to avoid as a result of the solution selections to retrace a path previously visited and to induce the exploration of new regions.

In the following sections, we shall explain existing TS strategies and their components. More details can be found in Glover [1989a, 1989b, 1990]. However, implementation details, illustrating specific components are given in Chapter 3 to Chapter 5 . Here, we report only the basic strategies.

2.3.2 Tabu search strategies

The TS metastrategy is generally an approach rather than a strict algorithm. The most basic form of the TS approach consists of three main strategies and their components:

(1) The forbidding strategy which manages what goes into the tabu list.

(2) The *freeing* strategy which manages what goes out of the tabu list, and when.

(3) The *short term* strategy which manges the interplay between the above strategies to select trial solutions.

In addition to the above strategies, there is also a *learning* strategy that gathers information during a TS run and this information can be used to direct the tabu search in subsequent runs. This strategy consists of the use of *intermediate* and *long term memory functions*. The TS strategies have a lot of overlap and interplays among their components. These strategies will be explained individually in the next sections.

2.3.3 The forbidding strategy

One aim of this strategy is to establish a mechanism which makes certain moves forbidden (or *tabu*) thus, preventing cycling. We will define a move s to be a mapping function defined on a subset of the feasible solution space. In order to avoid cycling which involve preventing the move from a solution S to another $s(S) \equiv S' \in N(S)$. It is clearly sufficient to check that a previously visited state is not revisited. Ideally, the tabu-list should store all such previously visited states and the list would be checked prior to any new move. However, the process of checking the tabu status of a move based on the above generally requires a great deal of memory and computational effort. This is so because we have to store all attributes of a solution that are needed to check the tabu status of a future move, i.e., recording at an extreme level of detail the whole solution before and after the move. A crude but reasonably effective way of avoid cycling is to instead insist on not reverting to a state S from S' if S' was obtained form S at the last iteration. Clearly, this does not guarantee that cycling will not occur. In addition, we may well insist that we do not return form S' to any state visited during the last $|Ts|$ iterations (and these states are, therefore, stored in the tabu-list). Moreover, even the checking of only the set of $|Ts|$ "last" states stored in the tabu-list, can be itself expensive, and a further approximation to that check must be used.

For example, the CCP, GAP and VRP problems that are considered in this thesis, are formulated in terms of the union of disjoint subsets i.e., $N = \bigcup_{i \in I} S_i$ where $I = \{1,...,K\}$ set of indices and $n =| N |$ is the size of the problem. Each subset, S_i, satisfies a given set of problem constraints, and their totality minimizes a well-defined objective function. To prevent cycling, we can use an approximate implementation of a TS algorithm as follows: Assuming the availability of a computer with a binary word of length at least n, then each feasible subset S_i can be obviously represented uniquely by a single word as:

$$W(S_i) = \sum_{j \in S_i} 2^{n-j} \tag{2.13}$$

where W is a perfect hashing function from the subsets into the set of integer values. Thus, the tabu-list would consist of $|Ts|$ sets of values $W(S_i)$ for a fully avoiding the return to a state in the tabu-list. However, computational experiments showed that, although recycling is prevented even when $|Ts|$ is small (say $n/5$), the procedure is still expensive. If instead of $W(S_i)$ we use a cruder (move set) representation of the state, based on partial move attributes. We can (with the same value of tabu list size "represent" more states and check those states faster with the result that better solution can be produced in this way.

In the context of this approximation to prevent cycling, there are four key elements to consider:

(1) To identify *attributes* of a move that will be used to create the *tabu-list* of states. Indeed we will use the tabu list not only to represent states to avoid but also to represent "moves" to avoid if these moves are "likely" to lead to a previous state. Tabu lists are normally represented by matrices that store these attributes. An attribute could be, for example, the cost of the state (solution), or the index of a customer involved in a move that produced that state, or the set S_i containing the above customer.

(2) To propose a *data structure* which can be used to update the tabu status of moves easily.

(3) To identify an *aspiration level criterion* which allows the tabu status of certain moves to be overridden and instead be considered as admissible. Not that, for example, since the state "attributes" are approximations, a move may be prohibited

although its state has been previously visited. In certain situations this is obvious, (e.g., if a proposed new state has an objective function value better than the current best solution), and the aspiration level criterion is meant to correct these errors.

In addition to these elements, it is necessary to determine how long a tabu restriction will be enforced. Or in other words, determine the *tenure* for which a move remains on the tabu list. This is important because the capacity of the tabu list is small finite. It is assumed that the likelihood of cycling is inversely related to the distance of the current solution from the previous solution. Thus, by preventing the choice of moves that represent the reverse of any decision taken during a sequence of ***the last*** | *Ts* | iterations, the TS procedure moves progressively away from all solution states of the previous | *Ts* | iterations. Here, | *Ts* | is normally called the ***tabu list length or tabu list size.*** With the help of an appropriate value of | *Ts* |, the likelihood of cycling effectively vanishes - a small value would create cycling. Note however that, a very large value is also not desirable because it would drive the procedure away from good solution regions before these regions are "fully" explored.

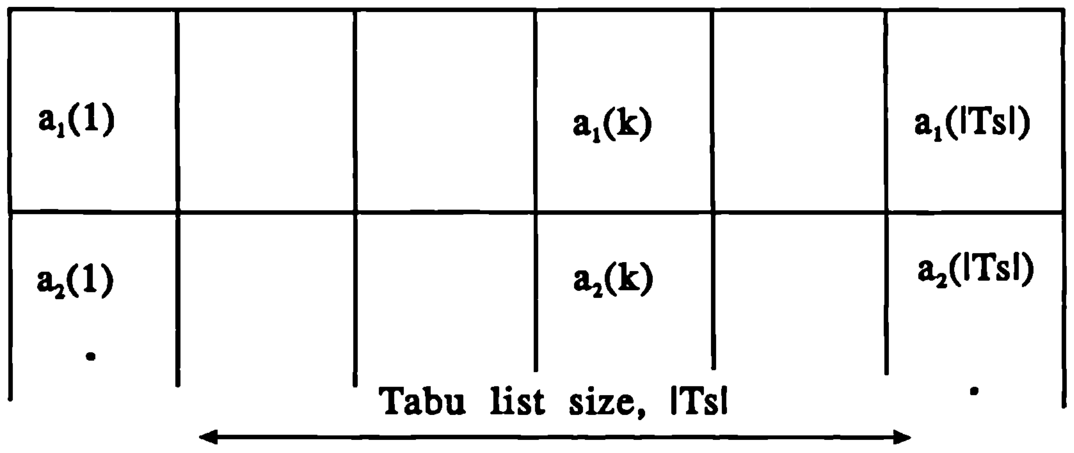


Figure 2.9. A tabu list data structure with a circular update.

Although strategies for dynamically varying the tabu list size are possible (Chapter 5), considerable success has resulted from simpler strategies that keep | *Ts* | at a fixed value (Chapter 4). We observed that the value of | *Ts* | depends on the size of the problem. This value is statistically derived and implemented. The tabu list embodies one of the primary *short-term memory functions* of any tabu search procedure. It is implemented by recording only the | *Ts* | most recent moves or their partial attributes. Once the list is full, each new move is written over the oldest move of the list. Effectively, the tabu list is processed as a circular array in first-in-first-out (FIFO) procedure. This data structure is shown in Figure 2.9 where each column represents the attributes $a_i(k)$, of each state k.

Note that this FIFO rule is one type of the many types of tabu list data structures which can be used, and an alternative is used in our TS algorithms which we will describe later. This alternative also records the iteration number at which a particular move is made tabu and requires less computational effort.

2.3.4 Aspiration criteria and tabu restriction (exception to the forbidding strategy)

Aspiration criteria are measures mainly designed to override the tabu status of a move if this move is *good enough* and sufficient to prevent cycling. The tabu restrictions and aspiration criteria play a dual role in constraining and guiding the search process (see Glover [1989a]). A move is admissible if tabu restrictions are not violated. However, a tabu move is also admissible if the aspiration criteria apply regardless of the tabu status.

Aspiration criteria are managed through the use of an aspiration function $A(C(S))$ applied to the objective function $C(S)$. Aspiration functions are either time-independent or time-dependent. In the first case, if, say, a tabu move applied to a solution *S* produces a solution whose objective value is better than the best solution found so far, it can not produce cycling. We say that the aspiration level is attained if *C(S) < A C(S))*. Another type of aspiration function, that has been found to be practical, is time-dependent and is further explained later.

2.3.5 The intermediate and long-term learning strategies.

These strategies are implemented using intermediate and long term memory functions. The intermediate function provides an element of *intensification.* It operates by recording good features of a selected number of moves (such as local attributes of good solutions) generated during the execution of the algorithm. This can be seen as a learning strategy, which seeks new solutions that exhibit similar features to those previously recorded. This is achieved by restricting or penalising moves, (encountered during a run) that do not possess favourable features.

The long term memory function is constructed in such a way as to allow the investigation of a number of alternative starting solutions for the entire TS procedure, while encouraging the selection of starting solutions not near those previously selected. It also can be viewed as an intelligent means of creating *diversification* into regions to which the procedure is applied. In other words, it avoids reliance on a blind, random starting process. The fundamental elements of intensification and diversification strategies are already present in the short term memory component of tabu search where they are applied **during the current** execution of the algorithm.

2.3.6 The freeing strategy

The freeing strategy is concerned with the management of what comes out of the tabu list. It removes their tabu restrictions so that they can be reconsidered in any future search. The attributes of a tabu move remain on the tabu list for a duration of $| Ts |$ iterations. If the tenure of such attributes has lapsed, then they are freed from their tabu status by this strategy. A move is considered admissible if none of its attributes are tabu or if it has passed an aspiration criterion test. This strategy is integrated together with others inside a ***short term*** strategy that organizes the interactions between move ***selections***, tabu restrictions and aspiration criteria. This will be explained in the following section.

2.3.7 The short term strategy (an overall strategy)

The short term memory strategy is the core of the TS algorithm. This strategy manages the interplay between all the above different strategies. It records more information about the past search behaviour than SA (which records none) and less information than branch and bound procedure (which record a full history of the search). The overall strategy is outlined and depicted in Figure 2.10.

Explanation of the Diagram in Figure 2.10.

This diagram has two key steps (highlighted by asterisks): (1) the best move in the list of admissible candidates; and (2) the update of the admissibility conditions. These two steps will be explained separately. The overall procedure starts with an initial solution (which may be ınfeasible).

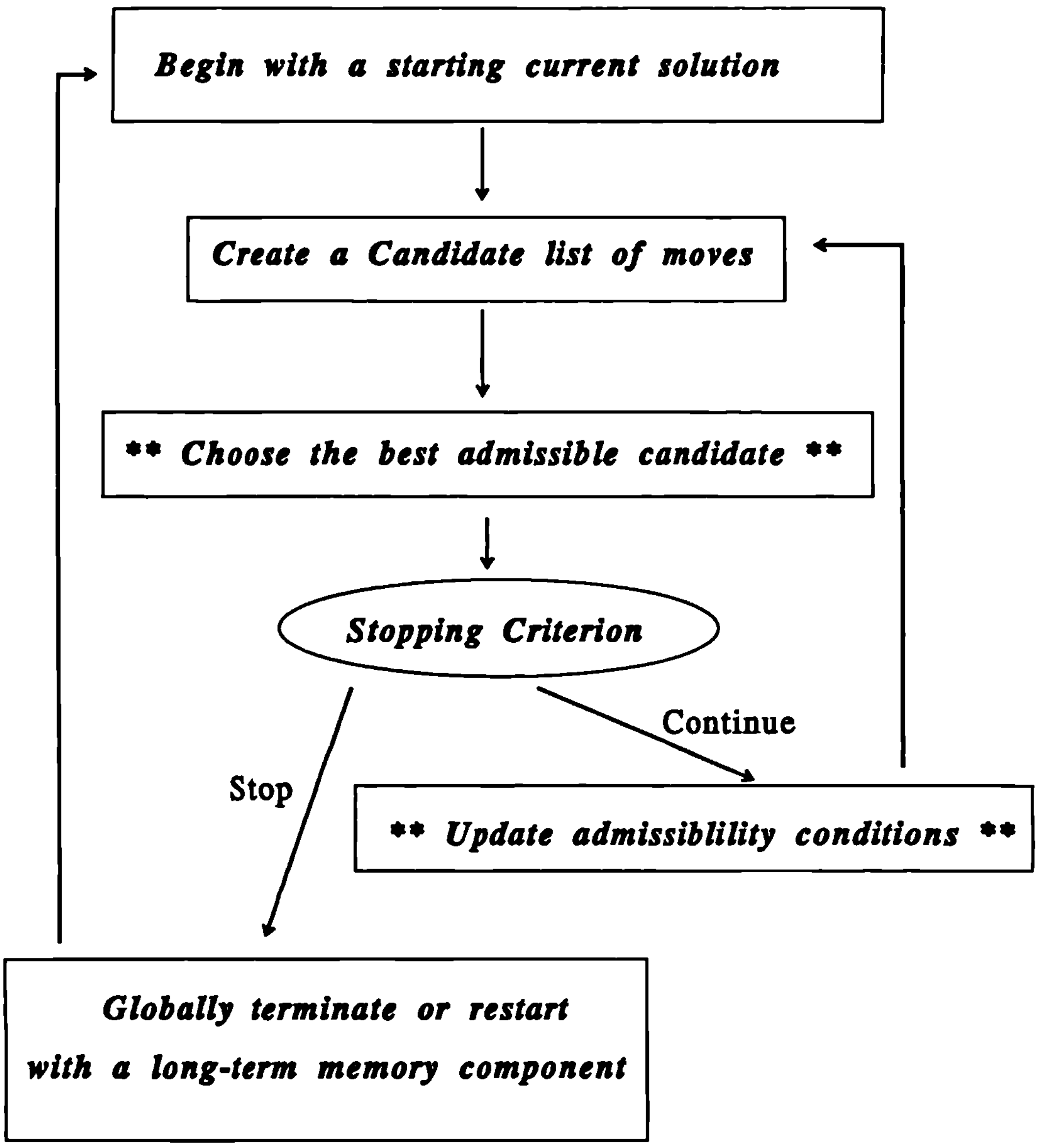

Begin with a starting current solution
Create a Candidate list of moves
** Choose the best admissible candidate **
Stopping Criterion
Continue
Stop
** Update admissiblility conditions **
Globally terminate or restart with a long-term memory component

Figure 2.10 Tabu search short term memory strategy.

(a) The candidate list of moves.

A candidate list is a sublist of the possible moves. Candidate list strategies are generally problem dependent and can be derived, for example, from the field of network optimization, move decompositions and random sampling (see Glover [1989c]). In this thesis, the TS algorithms include candidate lists of dynamic and/or fixed size; these lists contain the neighbourhoods generated by the λ–interchange generation mechanism (Section 3.3.1). If the neighbourhood is very large, a better way to sample the neighbourhood automatically is desirable. This is achieved by the dynamic search strategy which is proposed later.

(b) The best move selection strategy (or highest evaluation)

The best move selection strategy selects that admissible move from the current solution which yields the greatest improvement or the least disimprovement in the objective function, subject to the tabu restriction and aspiration criterion being satisfied. This strategy will be abbreviated as BA *best* (*admissible*) *improve* strategy. This aggressive criterion is based on the supposition that moves with higher evaluations have a higher probability of either leading to a near optimal solution, or leading to a good solution in a fewer number of steps. Figure 2.11 elaborates the step of Figure 2.10 that chooses the best admissible move. This criterion is proposed by Glover and used in some subsequent applications.

The step begins by evaluating each move in the candidate list in a sequential manner. If the number of tabu moves is small relative to those available, and if the expense of evaluating a move is not great, then it is desirable that we check first whether a given

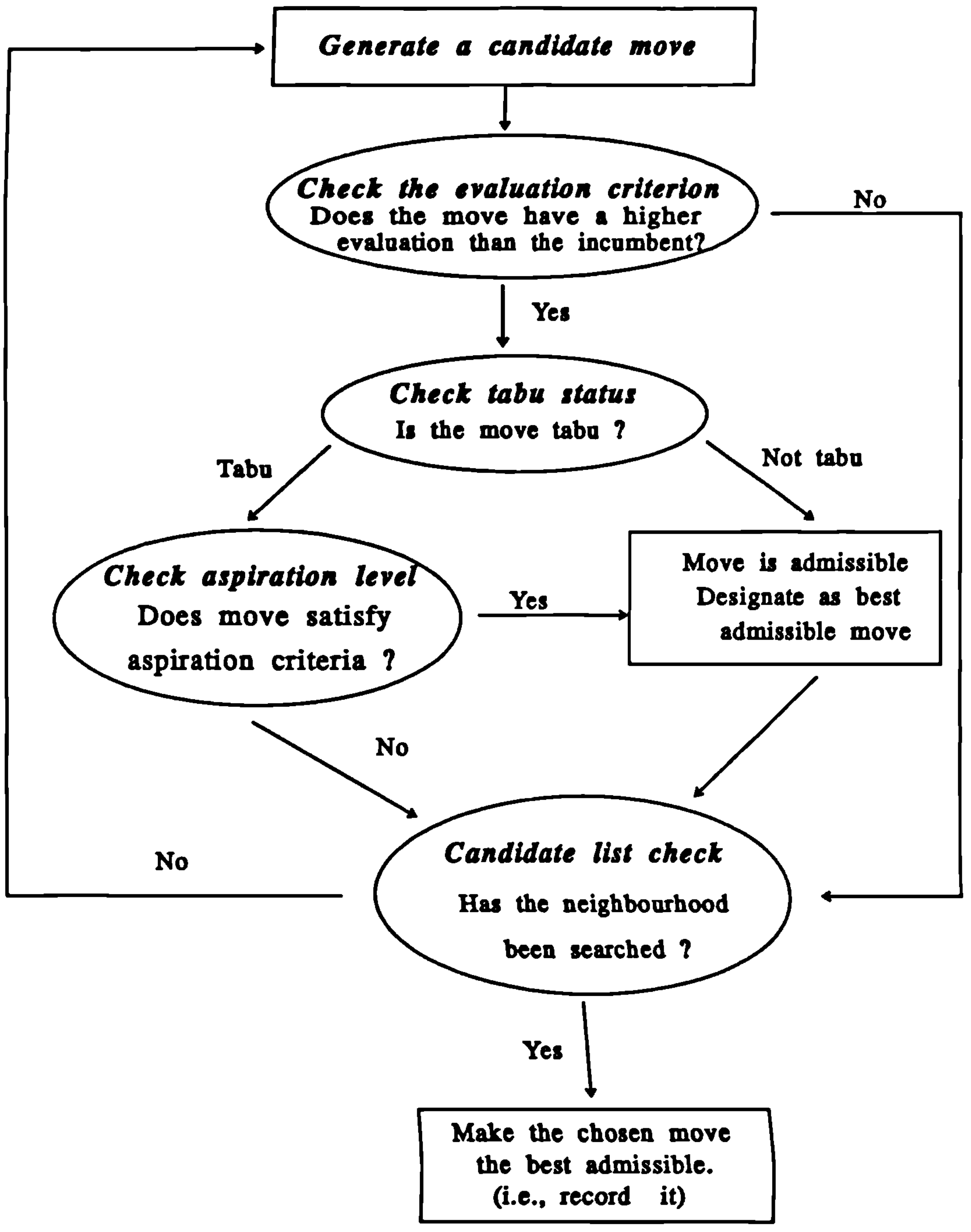

Generate a candidate move
Check the evaluation criterion
Does the move have a higher evaluation than the incumbent?
No
Yes
Check tabu status
Is the move tabu ?
Tabu
Not tabu
Check aspiration level
Does move satisfy aspiration criteria ?
Yes
Move is admissible
Designate as best admissible move
No
No
Candidate list check
Has the neighbourhood been searched ?
Yes
Make the chosen move the best admissible. (i.e., record it)

Figure 2.11. Choosing the best admissible candidate move.

move has a higher evaluation than its admissible predecessors before checking its tabu status. Otherwise, the tabu status check is made before the high evaluation check. Thereby, the computational effort is saved. If a higher evaluation move is tabu, the aspiration criteria are given the opportunity to override the tabu status. This provides a second chance to nominate the move as admissible before the next move is selected for evaluation. The inter-relationship between tabu restriction and aspiration level may, thus, be stated as follows: if a move satisfies the aspiration criteria it is admissible whether or not it is tabu, while, if it does not satisfy these criteria, then it is admissible if it is not tabu.

(c) The first and best improve selection strategy.

In the thesis, we propose the *first and best* (*admissible*) *improve* strategy, FBA, which is based of the combination of first and best improve selection strategies. This strategy uses a greedy approach which selects the first admissible move that provides an improvement over the current solution in the objective value. The corresponding solution is compared with the best recorded solution. The necessary updates are made and the selection continues for another move from the solution. If all moves in the candidate list (i.e., the neighbourhood) are tried without any improvement, the FBA strategy selects the best disimproving move. At this moment, the FBA is similar to the BA selection strategy. The FBA strategy has the advantage of *dynamic sampling* of the candidate list. The size of the search is indeterminate as it need not the whole list. This is in contrast to the BA strategy which uses a fixed candidate list size which *must* be scanned in all cases. Moreover, with FBA strategy, we are accepting more moves in good regions, hence the tabu list is updated more frequently and as a consequence a larger part of the solution space is finally searched.

(d) The stopping criterion

The *stopping criterion* terminates the TS procedure either after a specified total number of iterations have been performed in total or, since the currently best solution was found.

2.3.8 Performance of TS on combinatorial optimization problems.

At present, the TS method is not as widely studied as SA in terms of the theory and the practice of applying the method. Nevertheless, TS has been successfully applied to a number of problems. For some problems, TS has been able to find solutions superior to the best results previously obtained by any method. Tabu search has also shown advantages through its ease of implementation or in the flexibility to handle additional constraints not easily included by the original problem formulation. In the next section, we shall give a summary of the most successful applications of TS on some combinatorial optimization problems. The general feeling is that the study of the TS algorithm is an area that has to be further explored. The motivation being that TS can eventually provide a sound basis for solving COPs.

2.3.8.1 The travelling salesman problem (TSP).

Malek *et al.* [1989] implemented TS with a long-term memory strategy, and utilized tabu lists whose lengths could vary according to the tabu conditions. The size varied between one-third to one-fifth of the number of cities. The tabu status of moves is based on time-independent attributes with an aspiration level to override the tabu status. They also implemented a simulated annealing algorithm using a simple geometric cooling schedule with a step-wise temperature reduction scheme. Both "serial" and "parallel" TS and SA algorithms were tested on known problems in the literature of sizes 25, 33, 42, 57 and 100-city problems. Computational results show that the "serial" implementation of TS consistently out-performs the serial SA in terms of time and solution quality. However, the performance of "parallel" versions of these algorithms produced improved solutions of comparable quality. TS was much faster and obtained optimal solutions to the test problems with a substantially greater frequency (optimal solutions are known for these test problems). For the 50 and 75-city problems, the parallel tabu search produced better results than the best published tour lengths to date. TS computation time consumes only 1/3 to 1/25 the SA computational effort.

Another TSP study was undertaken by Fiechter [1990] who developed a parallel TS algorithm using an intermediate and long-term strategy. In addition to a restricted candidate list strategy. The algorithms were tested on a large size problems ranging from 500,...,100000 points in the plane, and a few other large-sized test problems from the literature, such as the 512-city problem of Padberg and Rinaldi [1987]. The TS solution for the 10000-point problem was 72.95, compared to 73.36 obtained by Bonomi & Lutton [1984] using SA. The parallel TS results for the 512-city problem were 0.55% away from the known optimal solution. Computational results on the large-sized problems show that the TS solutions were very close to the asymptotic estimates of the optimal solutions. Another TS application on the TSP can be found in Knox *et al.* [1989].

2.3.8.2 The Quadratic assignment problem. (QAP)

An application of TS to the QAP has been developed by Skorin-Kapov [1990], using a tabu list whose size varies both in relation to the problem size and in relation to the stage of the search. The TS approach also makes use of the long term strategy for diversification of the search process. The algorithm utilizes a time-independent tabu list and an aspiration level criterion. The TS algorithm obtained the best known solutions for all tested problems ranging from 15 to 36 facilities taken from the literature, and new problems of size up to 90 facilities. The method also succeeded in achieving such solutions at a much smaller CPU time. Furthermore, the TS solutions were equal to or better than the solutions obtained by a SA algorithm developed by Burkard *et al* [1984].

A parallel version of TS that incorporated a long term strategy was applied to the QAP by Taillard [1990]. The new feature in this application was the introduction of a random selection of the tabu list size, | *Ts* |, from a small interval about the mean of two chosen values. This size was maintained for 2 × | *Ts* | iterations before a new value was selected. The computational results showed that the parallel TS algorithms out-performed the algorithm of Skorin-kapov [1990] on all the test problems and found better solutions for the large-sized problems. Tabu search algorithm seems to be the best available approximate algorithms for the QAP (see Connolly [1991]).

2.3.8.3 Machine scheduling problems

1- **The permutation flow-shop scheduling problem.** (FSSP)
An application of TS to the FSSP was implemented by Widmer *et al.* [1989]. The algorithm was composed of two phases. The first finds an initial starting solution using a constructive heuristic method by considering the FSSP to be an open-ended TSP. Here, the distances between jobs were obtained by a weight function based on the difference between the processing times of jobs. The second phase tries to improve the initial sequence using a TS technique. Neighbourhood sequences were generated by permuting two jobs which were in two respective positions. A simple selection procedure is used in which all admissible neighbours are searched and the best is selected to be the new current solution. In this method, a circular tabu list is used to update the list of tabu moves, with tabu conditions prohibiting permutated jobs from returning to their previous positions. A small fixed tabu list size of 7 is used.

Computational results of TS on small-sized problems up to 20 jobs, were compared to the algorithm of Nawaz *et al.* [1983] - NEH - and showed that TS yields better solutions than NEH for 58% of problems, identical solutions for 14% of problems and worse solutions for 28% of problems. TS, however, consumes 4.5 times more CPU time than

NEH. There are no direct comparisons between TS and SA reported for this problem. However, the SA of Osman & Potts [1989] reports results on problems up to 100 jobs. They produced better solutions than NEH for 82.5% of problems, with identical solutions generated for the remaining 17.5%. By considering both SA and TS implementations, we conclude that the SA algorithm may be the most efficient of all the three algorithms for the FSSP. Another TS application for the problem was applied by Taillard [1990], in which neighbours were generated by shifting a job from one position and inserted in another. Different first and best improve strategies for selection of moves were tested. A parallel version of TS algorithm was presented to speed up the calculation of the best move in the neighbourhood. The conclusion, again, was reached that TS produced better results than NEH and that the first improve strategy would require less computational time than the best improve strategy at least on problems with sizes up to 50 jobs.

2- The single machine problem with linear delay penalties and set-up cost dependencies.

A tabu search study by Laguna *et al.* [1989] examined the above problem which requires minimizing a weighted combination of delay penalties and sequence-dependent set-up costs. The method utilized the concept of a long-term memory function and a best improve strategy for selection of alternate moves. Several branch and bound procedures were used to find the optimal solution for test problems. These procedures could not find the optimal solutions of 20-job problems within 150 CPU seconds on a mainframe computer. However, with a comparable time of 155 CPU seconds on a microcomputer, TS approach completed a set of 12 solution trials per problem and succeeded in obtaining an optimal solution (found by other means) to each problem. Moreover, the worst solutions obtained by TS were within 0.2% of optimality on average. In addition, the method also generated high quality solutions very quickly to larger problems.

2.3.8.4 The graph colouring problem (GCP)

A study by Hertz *et al.* [1987] compared the performance of TS & SA on the GCP. The TS approach used simple tabu components in which a first-improve selection strategy was employed with a fixed-sized sample of moves in order to reduce the computational effort. The tabu list was a small fixed size of 7. Moves were randomly selected rather than sequentially searched. Computational results on problems with sizes ranging from 100 to 1000 vertices, demonstrated that TS obtained solutions of significantly higher quality than SA at a lower computational time.

2.3.8.5 Other Combinatorial applications.

A study, comparing TS and SA for the clique partitioning problem was conducted by De Amorim *et al.* [1989]. The main conclusion was that the most commonly used heuristics were proven to perform badly in comparison with both SA and TS. The computational time for SA algorithm was generally greater than that of the TS algorithm.

Application of tabu search to the solution of the Course Scheduling Problem in a University, the independent sets problem in a graph and a few others problems can be found in Hertz *et al.* [1989]. Also, an application of TS to the Job Shop Scheduling Problem can be found in Widmer [1991]. A comprehensive review in the form of a book is currently under preparation by Glover & De Werra [1991]. Materials in this would cover the past history and development and recent research in TS. However, for surveys on successful applications so far, we refer to Glover [1989a, 1990, 1990a].

Chapter 3

SIMULATED ANNEALING AND TABU SEARCH FOR THE CAPACITATED CLUSTERING PROBLEM

3.0 Introduction.

The capacitated clustering problem, CCP, is defined as follows: A set $J=\{1,...,n\}$ of n customer points is given. The demand of a customer point j is a known value d_j. The distance (or "cost") between any two points i and j is given by the matrix $[c_{ij}]$. The problem is to assign the n points to p clusters (each point is assigned to exactly one cluster) so that the total demands of points assigned to a cluster k is not greater than Q_k, a given capacity for cluster k. The objective is to minimise some measure of clustering quality. For a given cluster, a "centre" is that point of the cluster from which the sum of the distances to all other points in the cluster is minimised. This sum is called the "scatter" of the cluster. The measure of clustering quality which is used in this chapter is the total scatter of all clusters. A pictorial representation of the CCP is given in Figure 3.1.

The applications of clustering models are diverse and appear in many other areas including:

- Depot location in distribution systems: See for example, Cullen *et al.* [1981], Klincewicz *et al.* [1988], Derby-Dowma *et al.* [1988], Bookbinder *et al.* [1988];

- Sale force territorial design: Mulvey *et al.* [1984];
- Switching centres in communication networks: Mirzaian [1985];
- Clustering of customers into different market segments in marketing studies: Chaffray *et al.* [1973];
- Clustering instances of varieties of cancer: Burbank [1972];
- Drawing inferences of scene content in pattern recognition problems: Bryant [1978].

For a comprehensive review of clustering models and their applications, refer to Brandeau *et al.* [1989].

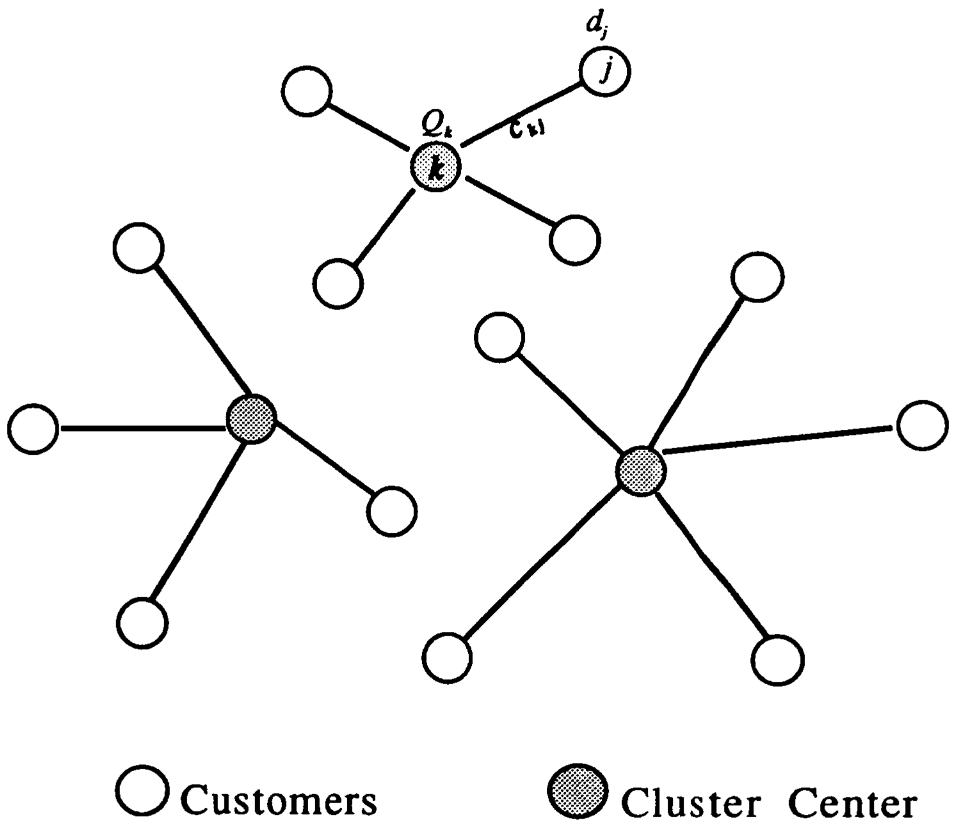


Figure 3.1. The Capacitated Clustering Problem

Recently, a great deal of attention has focussed on two new techniques: simulated annealing (SA) and tabu search (TS) for solving hard combinatorial optimization problems (COPs). Both these methods help reduce the effects of local optimality using strategies based on ideas from statistical mechanics and intelligent problem solving respectively. SA algorithms have been successfully applied to many COPs namely: the travelling salesman problem, Cerney [1985], Kirkpatrick [1984], Kirkpatrick *et al.* [1983], Bonomi *et al.* [1984]; the quadratic assignment problem, Burkard *et al.* [1984], Wilhem *et al.* [1987], Connolly [1990]; the flow shop scheduling problem Osman & Potts [1989]. Reviews of the theory and other extensive applications of SA can be found in Van Laarhoven *et al.* [1987], Eglese [1990], and Johnson *et al.* [1988]. For the theory and applications of tabu search, we refer to two fundamental papers, Glover [1989a, 1989b].

In this chapter, we investigate the applications of SA and TS to the CCP. In Section 3.1, we formulate the CCP as an integer programming problem and summarize its relationships to other combinatorial problems. In that section, we also review some of its available solution methods. In Section 3.2, we develop a constructive heuristic for the CCP. Section 3.3 introduces λ-interchange descent improvement procedures for the CCP. These procedures are similar to those introduced by Lin [1975], and Lin *et al.* [1973] for the travelling salesman problem. In Section 3.4, a new cooling schedule for simulated annealing algorithm is introduced and implemented for the CCP. In this section, we also give a brief classification of different SA algorithms. In Section 3.5, we discuss briefly tabu search and its implementation to the CCP. This technique is introduced in Section 2.3 and is explained in greater detail in Chapter 4 and Chapter 5. Computational results in Section 3.6 compare the proposed SA algorithm with other SA algorithms and with descent methods. We also compare different search and selection strategies for descent and simulated annealing methods, in addition to a comparison of SA and TS algorithms. These comparisons are based upon solution quality and running time on a set of randomly generated problems. Section 3.7 contains some concluding remarks.

3.1 Mathematical formulation

Define the input data:

c_{kj} = Cost of assigning point j to centre k,

d_j = Demand of point j,

Q_k = Capacity of centre k,

p = Desired number of clusters,

$J = \{1,\ldots,n\}$, the set of customer points,

$K \equiv J$, the set of potential cluster centers.

Define the decision variables:

$y_k = 1$, if customer k is chosen as the $k-th$ cluster centre,

$= 0$ otherwise.

$x_{kj} = 1$, if customer j is assigned to centre k,

$= 0$ otherwise.

The integer programming formulation of the CCP can then be stated as follows:

$$\text{Min} \quad \sum_{k \in K} \sum_{j \in J} c_{kj} x_{kj} \tag{3.1}$$

subject to:

$$\sum_{k \in K} x_{kj} = 1, \qquad \forall j \in J, \tag{3.2}$$

$$\sum_{k \in K} y_k = p, \tag{3.3}$$

$$x_{kj} \le y_k, \qquad \forall k \in K, \forall j \in J, \tag{3.4}$$

$$\sum_{j \in J} d_j x_{kj} \le Q_k \times y_k \qquad \forall k \in K, \tag{3.5}$$

$$x_{kj}, y_k \in \{0,1\}, \qquad \forall k \in K, \forall j \in J. \tag{3.6}$$

We assume that the costs are symmetric (i.e., $c_{kj} = c_{jk}$ for all k and j), and that $c_{kk} = 0$ for all k. In the above formulation K and J are taken to be identical, but this restriction is not necessary. Constraints (3.2) are the assignment constraints which force each customer to be assigned to one and only one cluster. Constraint (3.3) specifies the desired number of clusters. Constraints (3.4) ensure that no customer can be allocated to a point that is not a centre and constraints (3.5) specify that the total demand is less than the centre capacity. In the presence of constraint (3.5), constraints (3.4), are redundant in this formulation, however, they yield a much tighter linear programming bound than the equivalent, (but weaker) formulation without them, (Geoffrion *et al.* [1978]). The CCP can also be called the Capacitated p-median Problem.

An early integer formulation of the clustering problem may be found in Rao [1971]. A branch and bound algorithm for it is suggested by Konntz *et al.* [1975]. Stanfel [1986] provides an optimal algorithm based on Lagrangean relaxation and dynamic programming to solve clustering problems where the customers lie on a line and suggests the usage of the algorithm for the more general case. An algorithm for the Uncapacitated Clustering Problem (UCP) is developed by Mulvey & Crowder [1979]. This is based on Lagrangean relaxation and a sub-gradient procedure for improving the lower bounds, as in Held *et al.* [1974]. The UCP is the p-median problem (Christofides & Beasley [1982], Beasley [1985], Ahn *et al.* [1988]) and related to the Uncapacitated Facility Location Problem (Cornuejols *et al.* [1977]).

For the UCP, Whitaker [1983] describes a heuristic which modifies the greedy interchange algorithm of Teitz & Bart [1968], thereby, obtaining improvements on the quality of solution and running time. Mulvey & Beck [1984] present a hybrid heuristic-sub-gradient method to obtain good solutions for the CCP. The heuristic-sub-gradient procedure is repeated for a predetermined number of iterations.

At each iteration, the cluster centres are obtained by solving a relaxed problem that excludes the assignment constraints. A heuristic procedure attempts to locate a feasible assignment of points to centres. Improvements to this solution are made through the pairwise interchange of entities between clusters. Computational results on test problems of size up to 100 customer points, show that the heuristic produces results with a relative percentage deviation in the range of 0.43% to 5.4%.

There is a vast literature on problems that are mathematically related or very similar to the CCP. The 0-1 Capacitated Facility Location Problem (CFLP) has the CCP formulation without Constraint (3.3). This constraint is included in the objective function by associating a fixed cost, f_k, for each opened centre. The objective function then becomes:

$$\sum_{k \in K} \sum_{j \in J} c_{kj} x_{kj} + \sum_{k \in K} f_k y_k \tag{3.1'}$$

Neebe & Rao [1983] formulate the problem as a Set Partitioning Problem. They use a branch and bound procedure to solve it, where bounds are obtained by a linear programming relaxation. Heuristic algorithms based on lagrangean relaxation have been widely used to obtain good solutions for solving COPs. They generate feasible solutions to the original problem by modifying the solutions obtained from the relaxed problems, in order to satisfy the original constraints prior to their relaxation. For the Capacitated Plant Location Problem, CPLP, Barcelo & Casanovas [1984] report such a heuristic approach in which the assignment constraints are relaxed. Cornuejols *et al.* [1991] report computational results comparing different relaxations found in the literature for the CPLP. For the single source CPLP, Klinewicz & Luss [1986] and Darby-Dowman & Lewis [1988] present another similar approach in which the capacity constraints are relaxed. Finally, if we are given a fixed or known set of centres, the CCP becomes a special case of a Generalised Assignment Problem, GAP, Fisher *et al.* [1986], Martello & Toth [1990].

3.1.1 Solution representation.

The CCP is a combinatorial optimization problem of the following form: Given a set $J = \{1, \ldots, n\}$ of n customer points, we seek a feasible solution, which is a partition of J into p *clusters*, $S = (S_1, S_2, \ldots S_p)$, with each subset S_i having a *centre* ζ_i in such a way that $K = \{\zeta_1, \ldots, \zeta_p\}, K \subseteq J$. We assume that $p \geq 2$ and $Q_{\zeta_1} = \ldots = Q_{\zeta_p} = Q$ although this last condition is not strictly necessary. Nevertheless, we use $Q_{\zeta_k}, k = 1, \ldots, p$, to denote the capacity of cluster k.

A generic feasible solution to the CCP can be represented as:

$$J = \bigcup_{k=1}^{p} S_k; \qquad S_{k_1} \cap S_{k_2} = \varnothing, \quad \forall\, k_1, k_2 \in \{1, \ldots, p\}, k_1 \neq k_2;$$

$$\sum_{j \in S_k} d_j \leq Q_{\zeta_k}, \quad \forall\, k \in \{1, \ldots, p\}; \quad S = (S_1, \ldots, S_p) \quad \text{with} \quad C(S) = \sum_{k=1}^{p} \sum_{j \in S_k} c_{\zeta_k j} \tag{3.7}$$

where $C(S)$ is the objective value of the solution S.

3.2 A constructive heuristic.

Constructive heuristics are defined in Section 2.1.2 (A). Here, we propose a constructive heuristic for the CCP. This heuristic attempts to build a solution in three iterative stages. The first stage is to choose an initial set of centres. The second stage assigns customers to the centres found in the first stage. The final stage re-computes the new centres according to the final assignment. The heuristic description is as follows:

FIND: Find a set of p centres.

Step 1: Find the largest $c_{i^*j^*}$. Set $\zeta_1 = i^*$ and $\zeta_2 = j^*$. $K = \{\zeta_1, \zeta_2\}$.

If $p = 2$ go to ASSIGN; Else, Set $r \leftarrow 2$

Step 2: Set $r \leftarrow r + 1$. Find the next centre $\zeta_r \in J - K$ so that the product of costs from ζ_r to the previous centres is maximized, i.e., identify $\zeta_r = l \in J - K$ such that

$$\prod_{q \in K} c_{lq} = \max_{q \in K, t \in J-K} \prod c_{tq}$$

Step 3: Set $K \leftarrow K \cup \zeta_r$. If $r = p$; go to ASSIGN Else; Go to Step 2.

ASSIGN: Assign customer points to centres.

Step 4: For each point $j \notin \{\zeta_1, \cdots, \zeta_p\}$, find the distance to its nearest centre. Arrange the points in increasing order of these distances. Assign points in this ordered sequence to their corresponding centres so long as the resource capacity allows. If not, assign the point j to its immediately available nearest centre.

RE-COMPUTE: Re-compute cluster centres

Step 5: For each cluster S_k find the new centre ζ_k. In other words,

identify $\zeta_k = l \in S_k$ such that, $C(S) \equiv \sum_{j \in S_k} c_{lj} = \min_{t \in S_k} \sum_{j \in S_k} c_{tj}$. (3.8)

The above heuristic may fail to find a feasible solution when the capacities are very tight. Note, however, that in order to guarantee a feasible solution (if one exists) we would need to solve a multiple- knapsack (or bin-packing) problem exactly. This would be too time consuming to be considered for inclusion in our general heuristic. Also, note that the above three steps can be recursively repeated to improve the solution further.

3.3 Iterative improvement methods.

Iterative improvement methods have been introduced and discussed in Section 2.1.2 (B). The algorithm starts with any feasible solution and then iteratively improves upon the current objective value by amending the solution if possible. Descent and metastrategy SA and TS algorithms are examples of such methods. In any implementation of descent and metastrategy approaches, the following steps have to be followed.

(i) An initial starting solution,

(ii) A generation mechanism for generating neighbouring solutions,

(iii) An acceptance criterion, (of an alternative solution)

(iv) A stopping criterion

3.3.1 A λ-interchange mechanism.

The generation mechanism describes how a solution S can be altered to generate S' the neighbouring solution. It is a mapping function, N_λ, from S into $N_\lambda(S)$, the *neighbourhood* of S. We define a *move* to be a transition from one solution to another. This is characterised by a set of *attributes*. An *attribute* is a change made during a move. For example, in the exchange of two customers between two clusters, the attributes of the move are the indices of the clusters, the indices of the customers (from those clusters) involved, and the change in the objective function. Note that the set of attributes is an over-specification for the move, itself. It stores some "historical" information after a sequence of moves, and this "memory" is made use of in TS, as mentioned later. Clearly, the effectiveness of any iterative algorithm is partly determined by the efficiency of the generation mechanism and the way in which a neighbourhood is searched for a better solution. Lin *et al.* [1965] suggested a generation mechanism called the K-change (arcs exchange) procedure for the TSP. We adopt and introduce a similar customer-interchange procedure for the CCP.

The neighbourhood generation mechanism for the CCP can be explained as follows: Consider a solution $S = (S_1, \ldots, S_{k_1}, \ldots, S_{k_2}, \ldots, S_p)$. let ζ_{k_1} and ζ_{k_2} be the centres of the clusters S_{k_1} and S_{k_2}, respectively. A λ-interchange between two given clusters S_{k_1} and S_{k_2} is a replacement of a subset $\overline{S_{k_1}} \subset S_{k_1}$, $|\overline{S_{k_1}}| \leq \lambda$ with $\overline{S_{k_2}} \subset S_{k_2}$, where $|\overline{S_{k_2}}| \leq \lambda$, and vice versa, to get two new clusters, with possibly different new centres. Now,

$$S_{k_1} \leftarrow \left(S_{k_1} - \overline{S_{k_1}}\right) \cup \overline{S_{k_2}}, \& \quad S_{k_2} \leftarrow \left(S_{k_2} - \overline{S_{k_2}}\right) \cup \overline{S_{k_1}}; \tag{3.9}$$

and ζ_{k_1}, and ζ_{k_2} are re-computed. Hence, a new solution $S \equiv (S_1, S_2, \ldots, S_p)$ with $K = \{\zeta_1, \zeta_2, \ldots, \zeta_p\}$ is obtained. There are $p(p-1)/2$ possible combinations of pairs of sets of customers that need to be searched to generate neighbouring solutions.

Definition 3.1

A λ-interchange neighbourhood, $N_\lambda(S)$, of a solution S is the family of all possible solutions S' that can be reached from S in one λ-interchange over all combinations of a pairs of sets, (S_{k_1}, S_{k_2}), in S.

3.3.2 The order in which neighbours are generated

The effectiveness of iterative algorithms is affected by the way neighbours are generated. This generation depends on the order in which the pairs of sets are selected for λ–interchange.

Let the permutation $\sigma \equiv (1,\ldots,k_1,\ldots,k_2,\ldots,p)$ be the order of cluster indices in a given CCP solution, $S = (S_1,\ldots,S_{k_1},\ldots,S_{k_2},\ldots,S_p)$, where $\sigma(i) = i, \forall\, i = 1,\ldots,p$. We define an *ordered search* for cluster pairs. An ordered search selects all possible combinations of pairs of clusters (S_{k_1}, S_{k_2}) according to σ without repetition. A total of $p(p\text{-}1)/2$ combinations of (S_{k_1}, S_{k_2}) are examined in the following order:

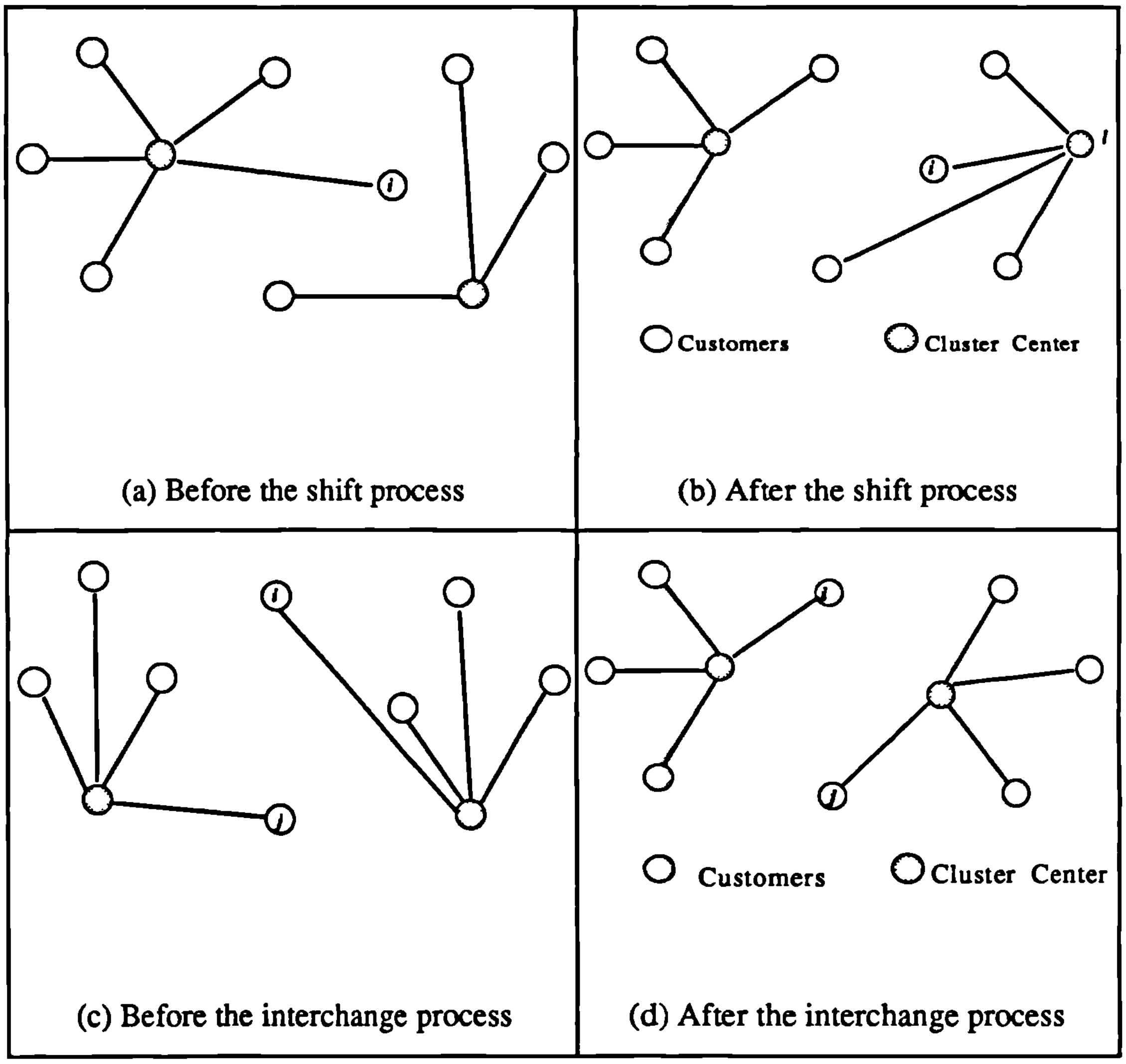
$$(S_{\sigma(1)}, S_{\sigma(2)}),\ldots,(S_{\sigma(1)}, S_{\sigma(p)}), (S_{\sigma(2)}, S_{\sigma(3)}),\ldots,(S_{\sigma(p-1)}, S_{\sigma(p)}) \qquad (3.10)$$

Definition 3.2

A *cycle* of search is a complete examination of all neighbours S' in the neighbourhood $N_\lambda(S)$ of a given solution S. These neighbours are generated by the λ–interchange mechanism which selects systematically in the order of a permutation σ all pairs of clusters (S_{k_1}, S_{k_2}) without repetition.

For the descent algorithm and tabu search, the same permutation, σ, is used after each cycle is completed (see later for details). However, in the case of the simulated annealing algorithm, a new permutation σ is generated at the end of each cycle of search. (Note that for a given S, descent and TS algorithms search the entire neighbourhood $N_\lambda(S)$ before a decision is made, whereas for SA a decision on whether to accept a change or not is made after each move, in which case the sets S_k are updated and the sequence (3.10) is affected).

Furthermore, for a given pair S_{k_1}, S_{k_2} we must define the search order for the customers to be exchanged between S_{k_1} and S_{k_2}. We consider the case of $\lambda = 1$ and a similar analogy can be followed for other values of λ. The 1-interchange mechanism uses two processes to generate neighbours.



(a) Before the shift process

(b) After the shift process

(c) Before the interchange process

(d) After the interchange process

Figure 3.2. The λ-interchange mechanism for the case of ($\lambda = 1$)

(*a*) A *shift process* which is represented by the (0,1), (1,0) operators. The (0,1) and (1,0) denote the shift of one customer from one cluster (say S_{k_1}) to another cluster (say S_{k_2}) or vice versa. Figure (3.2) illustrates an example for the shift process in which customer i is shifted from one cluster to another by the (1,0) operator. After the shift process, centres of clusters must be re-evaluated. As a result, customer l becomes the centre of the cluster to which i has been shifted, Figure (3.2.b).

(*b*) An *interchange* process which is represented by the (1,1) operator. This process exchanges a customer (say i) from one cluster with another customer (say j) from the other cluster. Figure (3.2c) shows two clusters before the interchange move, whilst Figure (3.2d) depicts the situation after the interchange takes place. In Figure (3.2d), the centres of both clusters have been changed due to the interchange move.

The customers in a given pair of clusters, S_{k_1} and S_{k_2}, are searched sequentially and systematically for improved feasible solutions by the shift and interchange processes. The order of search, we implement, uses the following order of operators (0,1), (1,0), (1,1) on any given customer pair to generate neighbouring solutions.

3.3.3 Evaluation of the cost of a move

The cost of a move is the difference between the objective function value of S and S', $\Delta = C(S') - C(S)$. S' is generated the λ-interchange mechanism applied to S. The calculation of Δ requires finding the (new) centres for each one of the involved clusters (say S_{k_1} and S_{k_2}) and computing C(.) from Equation (3.8). Here, the centre ζ_{k_1} is re-computed exactly after each move in $O\left(\left(\frac{n}{p}\right)^2\right)$ time. Finally, the cost of the move can be obtained by a simple calculation.

3.3.4 λ–interchange descent algorithm.

λ–interchange descent algorithm can be described as follows:

Step 1. Generate an initial (random or heuristic of Section 3.2) solution S .

Step 2. Choose a solution $S' \in N_\lambda(S)$ in the order indicated in Section 3.3.2.

Step 3. **If** S' is a better solution than S (a reduction in the objective value, $\Delta < 0$)

Then replace S by S' and go to Step 2.

Step 4. **If** the neighbourhood $N_\lambda(S)$ of S has been completely searched (a cycle of search is completed) without any improvement in the objective value **Then** Stop;

Else go to Step 2.

The iterative improvement descent algorithm has some peculiarities and difficulties to be overcome in any application. It can be shown that the algorithm depends strongly on the initial starting solution. A good constructive heuristic may reduce the computation time to produce a good final solution. Starting from a completely random solution would requires more iterations to find a good solution. However, there are no guide-lines on how to initiate an iterative descent method in order to obtain a good quality solution.

Clearly, a larger neighbourhood would most likely provide better local optima but will take a larger computation time to search. Hence, we will show that the nature of a neighbourhood mechanism determines whether biased or completely random starts should be used.

In classical steepest descent method, the entire neighbourhood $N_\lambda(S)$ is searched before a *best-improve* (BI) move is made. However, an obvious alternative is to accept the *first-improving* move (clearly this is not *steepest* descent) in which case the sets S_k are modified before the current search cycle is completed. After such a move is made, the search can continue in two stages. Either by restarting the cycle from the beginning (with the new definition of S_k) or by continuing from the current position to the end of the cycle and restarting a new cycle until the current position is reached again. This is similar to searching a circular list. The circular search is more efficient than the restart strategy, because of its computational advantage.

Definition 3.3. (λ-optimality)

A solution S is λ-optimal (λ-opt) if and only if, for any $S_{k_1}, S_{k_2} \in S$ no improvement can be made by a λ-interchange.

For the iterative descent algorithm, we use the first-improve (FI) search strategy incorporated with a 1-interchange or 2-interchange neighbourhood generation mechanism to produce two descent algorithms 1+FI and 2+FI respectively. The 2+FI algorithm produces 2-optimal solutions by applying the 2-interchange mechanism to the 1-optimal solution produced by 1+FI solution. Note that a λ-opt solution is not unique and depends on the order of the search.

Proposition 3.1.

The λ-optimal solutions produced by the descent methods are μ-optimal for any $\mu < \lambda$.

Proof: (immediate)

3.4 Simulated annealing algorithm.

Simulated annealing is an iterative descent algorithm modified by random ascent moves in order to escape poor quality local minima. The level of randomization is determined by a control parameter (T), called the temperature which tends to zero according to a deterministic "cooling" schedule. The theory and practical applications of the SA algorithm are introduced in Section 2.2. In most of the applications discussed, SA performed remarkably well. The sensitivity of the performance of the algorithm to the choices of the control parameter was exhibited in some of the applications, thus demonstrating how in practice the annealing algorithm can be prematurely trapped in local minima. The SA computation time can be speeded up by incorporating ideas that include: alternative methods of calculating (approximately) the change in the objective function, Δ; identification of a set of promising regions to search and the size of the neighbourhood.

In this section, we briefly classify the different types of cooling schedules which are discussed in Section 2.2.5. We also describe the SA we developed and implemented for the CCP.

3.4.1 Classification of the cooling schedules. (A summary)

Implementing a SA algorithm needs a cooling schedule that defines:

(i) An initial value of the control parameter T;

(ii) A decrement function T;

(iii) Indicates how many iterations to be performed at each temperature;

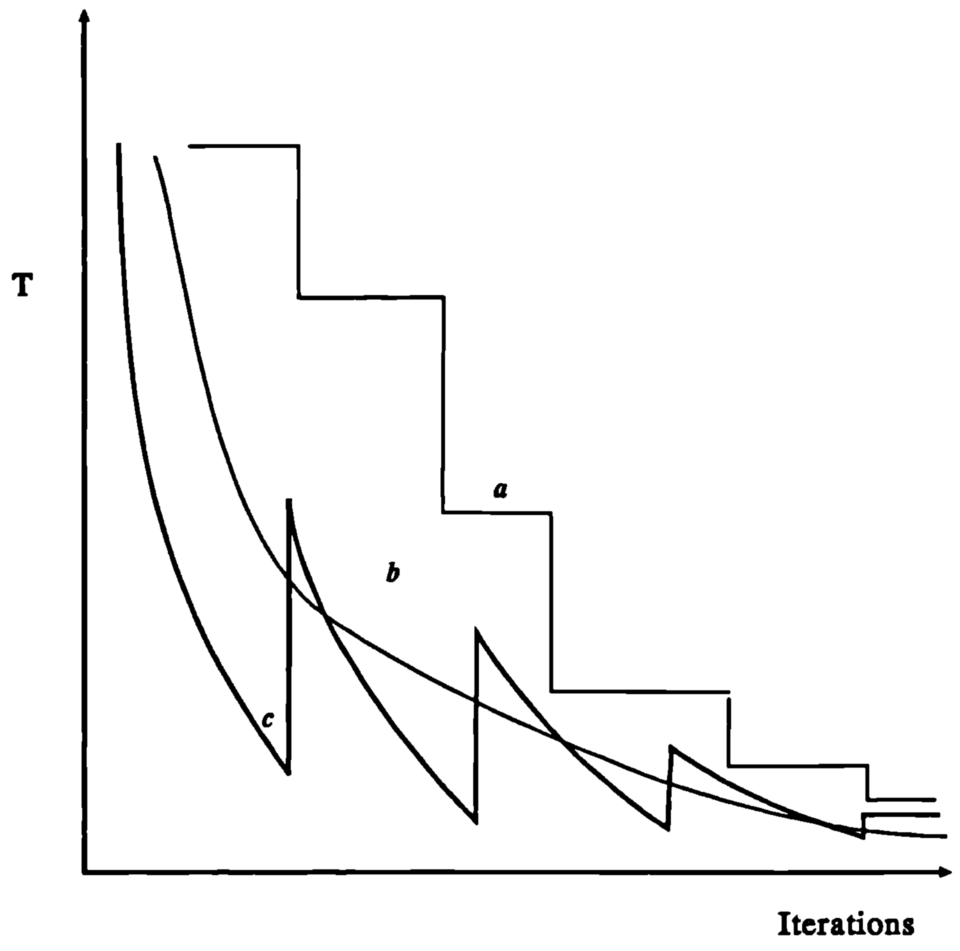
(iv) A termination criterion.

The performance of the SA algorithm depends strongly on the chosen cooling schedule. With a proper cooling schedule near-optimal solutions can be obtained for many combinatorial problems (see Section 2.2).

We classify SA algorithms according to the rules used to define cooling schedule parameters. (Another classification in a bibliography by Johnson [1988] is based on "categories" such as: general; theory; algorithm design; computational experiments;

combinatorial optimization problems and application areas). Figure 3.3 illustrates the following three different decrement rules. This figure shows a pictorial representation of the temperature evolutions as the number of iterations grow.

1. *Stepwise temperature reduction* schemes (Section 2.2.5.1): In this category, the temperature duration, *L*, is equal to either the size neighbourhood (i,e., the entire neighbourhood is searched before T is reduced), or a pre-specified number. *L* iterations are to be performed at a given value of the temperature T, before T is reduced according to some formula. Each scheme has its own stopping criterion.



a : Stepwise temperature reduction scheme
b : Continuous temperature reduction scheme
c : Non-monotonic temperature reduction scheme

Figure 3.3 Classification of cooling schedules

2. *Continuous temperature reduction* schemes (Section 2.2.5.2): This category reduces the temperature after each attempted move. Moreover, the temperature is updated after each iteration. This category is similar to (1) if we let $L = 1$. Again, some rules are used to control the temperature and stop the algorithm.

3. *Non-monotonic temperature reduction* schemes (Section 2.2.5.3): The temperature is reduced after each attempted move with occasional increases in the temperature. The philosophy of the temperature increase is based on the idea that there is no point in reducing the temperature any further when one cycle (or a pre-specified number of cycles) has been completed with no change in the current solution. The new developed SA algorithm for the CCP forms the basis of this category and will be explained next.

3.4.2 The new cooling schedule.

This section introduces our non-monotonic cooling schedule. We perform a single feasible iteration (one attempted feasible move) at each temperature. In addition, we define a *cycle* to be a complete neighbourhood search of the current solution. The generic parameters are for this cooling schedule are:

(i) The starting temperature T_s and the final temperature T_f.

(ii) The temperature decrement rule and the parameters α and γ used for decrementing T by a ratio dependent on the number of iteration k.

(iii) The condition for occasional temperature increase, and the temperature reset variables, T_{reset}, the value to which T is reset, after a complete cycle with no change. (If such a cycle is encountered we say that the "reset condition" is satisfied. T_{reset} is itself is determined by an update rule).

(iv) The total number of iterations M for a stopping criterion or, another alternative criterion based on the number of temperature resets, R.

Our cooling schedule controls the temperature decreases after each iteration and the occasional increases after the occurrence of a no-change cycle in the following ways:

(i) The initial and final temperature values.

The initial value T_s is not very important. A small value is recommended if we start from a good heuristic solution. This would also save some computation time. Although the final temperature T_f is included, its value could be set arbitrarily to 1 with no effect. It is included only to show the relationship with Lundy & Mees [1986].

(ii) The decrement rule.

After each iteration k, the temperature is decreased according to a parameter β_k which is updated as follows: let us define

$$\beta_k = \frac{(T_s - T_f)}{(\alpha + \gamma\sqrt{k})T_s T_f} \tag{3.11}$$

where α and γ are constants. Then, the temperature is updated according to the sequence:

$$T_{k+1} = \frac{T_k}{(1 + \beta_k T_k)} \tag{3.12}$$

Note that if $\gamma = 0$, β_k is a constant independent of k and is the rule used by Lundy & Mees [1986]. In our case β_k decreases as k increases and the temperature is then decreased more slowly with k.

(iii) The condition for occasional temperature increase.

If a cycle of search is made without accepting any λ-interchange, it is likely that, with the current temperatures dropping even further there will be no changes accepted. Therefore, the current temperature T_k needs to be reset to a higher value. The reset values should not be very high since what we want is only to escape the current local optimum, but not to deviate from the current solution very much as if we are starting from a totally new random solution. The temperature reset could then allow further moves to be accepted, and perhaps better solutions might be generated.

At iteration k, where the reset condition is met, the temperature at iteration $k+1$ is reset to a temperature value, T_{reset} (i.e., $T_{k+1} = T_{reset}$). We need to define how the values of T_{reset} are updated. Initially, T_{reset} is set to the value of T_s. Before the reset of the current temperature takes place, the value of T_{reset} is updated as follows:

$$
\begin{aligned}
T_{reset} &= \frac{T_{rep}}{2}, && \text{if } T_{reset} > T_k \\
&= T_{found}, && \text{otherwise} \qquad (3.13)
\end{aligned}
$$

where T_{found} is the temperature value at which the best solution is found. After the reset temperature of T_{reset} is made according to Equation (3.13), then $T_{k+1} = T_{reset}$. After this reset, the temperature updates are carried out using either Equations (3.12) until the algorithm stops.

(iv) The stopping criterion.

Our SA algorithm is designed so that it gives the user control of the trade-off between the quality of solution and the computation time. Two stopping criteria are used. The first is to execute the algorithm for a pre-specified number of iterations M. The second is the one which depends on the number of temperature resets, R, since the best solution is found.

Note that α and γ control the rate of temperature reduction with iteration number

k. Hence, there is a trade-off in the choice of the values of α & γ. A large ratio $\frac{\alpha}{\gamma}$ produces a cooling schedule similar to previous cooling schedules for small values of k. A small ratio $\frac{\alpha}{\gamma}$ produces a cooling schedule which is slow even for small k. The values of α & γ are related to the problem size.

The cooling schedule parameters are obtained from the problem-specific characteristics. A cycle of search is performed on the initial solution (with no changes done) to estimate these parameters. We assume S is an initial solution for a CCP with n as the number of customer points and p as the number of clusters. From our experimental experience, we suggest that $\alpha = p \times Nfeas$, where $Nfeas$ is the number of feasible moves found during the cycle (i.e., the feasible - w.r.t. capacity - part of the neighbourhood $N_\lambda(S)$); $\gamma = n$; T_s can take any values in Equations (2.3), (2.4), (2.8); and T_f is set to δ_{min}, the smallest change in the objective function found during the cycle; the total number of iterations M is given by $M = n \times Nfeas$. We must be careful with this value of M as it can be small for very tight capacity problems, and large for very loose problems. Therefore, an alternative stopping criterion is also used. It is based on the number of consecutive temperature resets for which there are no improvements made to the best solution.

A recent cooling scheme, by Connolly [1990] (see Section 2.2.5.3) can be seen as a special case of our proposed scheme. It shares the tenets of our scheme in the reset criterion. However, It only resets the temperature once as opposed to several times. When no changes occur in a complete search neighbourhood, the algorithm of Connolly [1990] sets the temperature to, T_{found}, the one at which the best improved solution is found and kept at that value thereafter.

3.4.3 The general annealing algorithm.

Our implementation of the SA algorithm for the CCP is presented in this section. The generic SA algorithm is denoted by SA(S, M, N, C) and abbreviated by SMN.C, where S defines the initial starting solution, M describes a method of pair cluster selection, N describes how customers in a given pair of clusters are controlled by the λ –interchange, and C refers to the cooling schedule that is applied. The solution reported is the best solution found and recorded during the search, as opposed to the one at which the algorithm is terminated.

The steps of the SA algorithm:

(i) Initialisation:

Step 1. Generate an initial *heuristic* solution S by the heuristic method of Section 3.2. or by a random procedure.

Step 2. Initialisation of the *cooling schedule parameters*:

Perform a cycle of search, $N_1(S)$ without accepting neighbours to obtain δ_{max}, δ_{min} the largest and smallest change in objective function values respectively, and the total number of feasible solutions *Nfeas* in the neighbourhood $N_1(S)$.

Set $T_s \leftarrow \delta_{max}$, $T_f \leftarrow \delta_{min}$, $T_{reset} \leftarrow T_s$, $T_{found} \leftarrow T_s$, $\alpha \leftarrow n \times Nfeas$, $\gamma \leftarrow n$, $R \leftarrow 3$, $k \leftarrow 1$, and set the counter $r_{best} \leftarrow 0$

(ii) Selection and acceptance of generated neighbours.

Step 3. Select a solution $S' \in N_1(S)$ sequentially and systematically (see Section 3.3.1, and Section 3.3.2).

Compute $\Delta = C(S') - C(S)$.

Step 4. If $\Delta \le 0$, or $e^{(-\Delta/T_k)} > \theta$, where θ is a uniform random number $0 < \theta < 1$, then accept S' and set $S \leftarrow S'$.

If S' is the best so far then keep S'. Update T_{found} and set $r_{best} \leftarrow 0$.

(iii) **Temperature updates.**

Step 5. Evaluate the temperature decrement ratio β_k as in Equation (3.11), and update temperatures according to the rule (3.12) or the conditions for occasional increase (3.13) of Section 3.4.2.

If the rule (3.12) is used, then $r_{best} \leftarrow r_{best} + 1$.

Set $k \leftarrow k + 1$.

Step 6. If k is greater than M, (or for the other stopping criterion, if $r_{best} > R$) then, stop and report the best found solution and the computation time.

Otherwise go to Step 2.

We evaluate SA algorithms by restricting our attention to those types in SA(S, M, N, C) as follows: S $\in$ {R, H}, where R is a random solution and H is a heuristic solution of Section 3.2; M $\in$ {R, S, $\overline{\text{S}}$}, where R, S and $\overline{\text{S}}$ denote a random and a systematic and a random systematic selection (Equation 3.10) of the cluster pair for the 1-interchange mechanism, respectively; N $\in$ {R, S}, where R refers to a random customer choice for the 1-interchange move while S refers to a systematic (ordered) 1-interchange of customer points; C $\in$ { AV, LM, OC } where, AV, LM and OC represent Aarts *et al.* [1985], Lundy & Mees [1986], and Osman & Christofides [1989] cooling schedules respectively. Osman & Christofides [1989] is the new cooling schedule represented in Section 3.4.2.

Proposition 3.2.

The SA algorithm with a *non-monotonic* cooling schedule produces at iteration Q a solution which is always better than or equal to that produced at iteration $M < Q$, holding constant other cooling schedule parameters (including the seed of the pseudo-random number generation).
proof: (immediate)

The importance of this property, is that it provides a guarantee of no degradation in solutions, if extra computation time is available. However, this obviously most necessary guarantee is not provided by many other cooling schedules in the literature.

3.5 Tabu search implementations.

Tabu search is a novel technique for solving combinatorial optimization problems (see Glover [1989a, 1989b]). The basic components of this approach are explained in Section 2.3. Furthermore, its relationship with the iterative improvement (local search) method is also introduced in Section 2.1.7. Generally, the TS algorithm starts with an initial feasible solution e.g., the one obtained from the constructive heuristic in Section 3.2 and makes a succession of moves to transform this solution into a better solution. TS shares with SA the ability to continue the search beyond the local optimality of the descent method. Moreover, TS uses the same 1-interchange mechanism to generate the same neighbourhood. In order to implement this technique, we also need to define the following elements (see Section 2.3.7):

(i) A tabu list and tabu list size.

(ii) An aspiration level (ignoring tabu condition).

(iii) Selection strategies (accept the *first* and *best* admissible moves).

(iv) A stopping criterion.

Let us consider an CCP instance of size n customers and p centres, then the TS elements are now explained briefly as follows:

(i) (a) Tabu conditions are based on move attributes. In this implementation, we select only one tabu condition. This prevents the return of two customers to previous clusters that they already occupied. This restriction remains during the following sequence of $|Ts|$ iterations, where $|Ts|$ is the length of tabu list size (see Section 2.3.3)

(b) In order to determine the tabu status of a move, a tabu list must be constructed. A data structure for a tabu list takes the form of a matrix, TABL, of size $n \times p$. Each entry TABL(e_i, S_t) records the iteration number at which customer, e_i, is removed from the cluster S_i by the 1-interchange. This data structure is updated automatically as the number of iterations is increased. Hence, the tabu status of a move does not require an update. Furthermore, using this data structure, the tabu status of a move can be checked easily by examining the corresponding entry in the matrix. For example, to check whether e_i is allowed to return to S_i and e_j is allowed to return to S_j at iteration k, we need the test in Equation (3.14). If the test is positive, then the move is considered tabu, otherwise it is deemed admissible.

$$k-\mathrm{TABL}(e_i,S_i) \leq |Ts|$$

and $$k-\mathrm{TABL}(e_j,S_j) \leq |Ts| \qquad (3.14)$$

(c) The length of tabu list | *Ts* | is found experimentally. If the tabu list size is small, cycling would occur. However, if it is large, the process might be driven away from the vicinity of a global optimum. The optimum tabu list size will be long enough to prevent cycling but small enough to allow for a continuous solution space exploration. For the results produced in this chapter, we use two formulae to estimate the tabu list sizes:

Let us define ρ to be the capacity ratio of the required demands to the total available cluster capacity, i.e.,

$$\rho = \frac{\sum_{i=1}^{n} d_i}{\sum_{k=1}^{p} Q_k}. \qquad (3.15)$$

For the case of the TS algorithm with the FBA selection strategy (see this section (iii) below) | *Ts* | is estimated by regression analysis of the computational results as:

$$|Ts| = 8+(0.078-0.067\times\rho)\times n\times p \qquad (3.16)$$

whereas for the case of the TS algorithm with the BA selection strategy | *Ts* | is estimated as:

$$|Ts| = \mathrm{Max}\{7, \quad -40+9.6\times\ln(n\times p) \qquad (3.17)$$

(ii) The aspiration criterion (Section 2.3.4), we have used, is defined as follows: a tabu move is admissible, if this move reduces the objective function value of the current solution to a value below the objective function value of the best solution found so far. This indicates that a new path not visited previously is to follow.

(iii) We consider, here, two move selection strategies (see Section 2.3.7b and Section 2.3.7c):

Best admissible move, BA: This strategy evaluates all neighbours and selects a move to obtain a solution S'. S' is the highest evaluation move, and satisfies the admissibility c nditions - S' is not a tabu move, or it is a tabu move but passed the aspiration criterion (see Section 2.3.4.). S', then becomes the next current solution.

First and best admissible move FBA: This selection strategy combines both the first-improve with the best-improve strategies incorporating the admissibility conditions. FBA accepts the first improving move upon its discovery; if there is no move that improves the current solution, then the best admissible move that was recorded is selected (see Section 2.3.7)

(iv) The stopping criterion: when the number of iterations for which there is no improvement over the best solution, is greater than a constant value which is fixed a priori, *MAXI*, we stop.

3.5.1 Tabu search algorithm

In this section, we list the TS algorithm steps:

(a) Initialisation.

Step 1. Get an *initial heuristic* solution S, e.g by applying the constructive heuristic of Section 3.2.

Initialise, the *tabu list size* $|T_s|$ according to Equations (3.16 or 3.17);

set the *tabu list* TABL(*i,j*)$\leftarrow -\infty\ \forall i, j$,

Set a value for *MAXI* and Set counters $k \leftarrow 1$, and $k_{best} \leftarrow 0$.

(b) Selection and acceptance of generated neighbours.

Step 2. *Choose* an admissible move $S' \in N_1(S)$ (feasible and not tabu or one whose tabu status is overridden by the aspiration criterion) according to the **BA** (best-admissible-improve) or **FBA** (first-best-admissible-improve) selection strategy.

Update as follows:

- Store the attributes of the newly accepted move in the tabu list matrix TABL;
- Update the current solution S.

Set $k \leftarrow k+1$.

If the current solution is better than the best solution found so far **then** update the latter and set $k_{best} \leftarrow k$.

(c) The stopping test.

Step 3. **If** ($k - k_{best} > MAXI$) **then** go to Step 4, otherwise go to Step 2.

Step 4. Display the TS final and initial solutions together with their computation times. Stop.

3.6 Computational experience.

3.6.1 Test problems.

We generate two sets of ten test problems each. One set contains problems of size (nxp): 50×5 and the other set contains problem of size 100×10. Customer points are located in the plane and their coordinates are randomly generated from a uniform distribution in the interval $U[1,100]$. We use the Euclidean distance metric to denote the cost, c_{ij}, between pairs of customer points i and j. The demand values, $d_j \; \forall j \in J$ are generated from $U[1,20]$. The cluster capacities $Q_k \; \forall k \in K$ are identical for a given problem. The tightness of a problem is expressed by the ratio ρ obtained using Equation (3.15). Q_k is chosen such that ρ is the interval [0.82, 0.96]. Computational results are evaluated using the relative percentage deviations RPD (and average relative percentage deviations, ARPD) of the heuristic solution ($C(S_H)$) over the best known solution ($C(S_B)$), i.e., RPD= $100 \times (C(S_H) - C(S_B))/C(S_B)$. For each test problem, five different random starts are performed in the random start case. However, when SA schemes are compared using just one constructive initial solution, we change the random number generator seed for probabilistic move acceptances so that a total of five different runs can be obtained. All algorithms are coded in FORTRAN 77 and run on a VAX 8600 computer. We report the average computation time, ACT, in CPU seconds of the actual execution time excluding input reading and output reporting time.

3.6.2 Iterative descent algorithms.

This subsection examines the effects of the initial starting solutions and the λ–interchange neighbourhood mechanism on the solution quality of descent methods for $\lambda = 1, 2$. Computational results are listed in Table 3.1. Results are evaluated using the average relative percentage deviations ARPD. Also tabulated is the estimated probability (P) that the best known solution will be found in a single run of this algorithm. Computational results confirm our belief that, a good starting solution only reduces the computation time as can be seen in Table 3.1, without necessary improving the final solution. This can be seen since the average relative percentage deviations of all random solutions for the 1+FI and 2+FI are comparable to those with a heuristic starting solution.

These mechanisms are powerful, since they improve any random starting solution quickly. However, it might be important to use good initial solution with a weaker neighbourhood generator. The 2+FI algorithm with random solutions have improved the best and average relative percentage deviations of the 1+FI solutions by 14% and 24% respectively. However, computational time was nine time higher than that of the 1+FI algorithm.

Table 3.1. Average relative percentage deviations for descent algorithms.

	With a random starts		With a heuristic starts	
Measures	1+FI	2+FI	1+FI	2+FI
ARPD of best solutions*	1.354	1.163	5.18	3.934
ARPD of all solutions**	4.255	3.234	5.18	3.934
P	0.10	0.13	0.00	0.05
ACT	4.79	46.49	5.36	45.22

ARPD: is the relative percentage deviation from the best known solutions.
* "best" refers to the best of 5 runs for a single problem.
** "all" refers to the solutions of all 5 runs per a single problem.
ACT: is the CPU time in seconds.
P: is the probability that the best solution is found in one run.

3.6.3 Comparisons of simulated annealing algorithms.

We now compare simulated annealing algorithms SA(S, M, N, C) abbreviated as SMN.C in Table 3.2. The compared algorithms are considered under the same initial temperatures, starting solutions and neighbourhood generations. We can then assess the effectiveness of the corresponding cooling schedules.

Initial values of the control parameter T_s are chosen as suggested earlier in Section 2.2.5. A complete neighbourhood search of the initial starting solution is performed to obtain the largest (δ_{max}), the average ($\bar{\Delta}$) and the standard deviation (σ) of the cost differences between the initial costs and the generated costs. In Table 3.2, the values of 2STD and 3STD are computed using Equation (2.8) so that solutions whose costs are $2\times\sigma$ and $3\times\sigma$ worse the current initial solution are accepted with a probability of 0.8 (for example, $e^{\left(\frac{2\times\sigma}{2STD}\right)} = 0.8$, $T_s = 2STD$). The value T_s for accepting the average increase in cost, $(\overline{\Delta})$, with a probability 0.8 is denoted by AVINC. The value of AVINC is obtained by Equation (2.1). After experimentation, we use a decrement value of $\delta = 0.5$ in Equation (2.5) and a value of $\varepsilon = 0.001$ in Equation (2.7) for the stopping ratio criterion (as suggested in the cooling schedule of Aarts *at al.* [1985]). The algorithm of Lundy & Mees [1986] is run for the same number of iterations that has been used for our algorithms.

For different values of T_s, the results of Osman & Christofides [1989] algorithm, HSS.OC, out-perform the Aarts *et al.* [1985] algorithm, HSS.AV, in terms of the average relative percentage deviations, APRD, (see Section 3.4.3 for abbreviations). Moreover, the worst ARPD produced by HSS.OC is 0.448, which is twice as good as the best ARPD value of 0.866 produced by the HSS.AV scheme. The same conclusion is demonstrated when comparing the ARPD's of the Lundy & Mees [1986] algorithm, RSS.LM, to that of RSS.OC. The superiority of our cooling schedule over its counterparts is due mainly to the dynamic control of the temperature decrease and the temperature reset strategy. Unlike the other SA schemes, these strategies make the algorithm less sensitive to the choice of the initial temperature. This feature can be concluded from the results in Table 3.2.

We shall now discuss next issues concerning the neighbourhood search. Three SA algorithms, R$\overline{\text{S}}$S.OC, RSS.OC and RRR.OC, are developed with different neighbourhood search strategies. The algorithm R$\overline{\text{S}}$S.OC starts with a random solution, and searches systematically all the $p(p-1)/2$ pairs of clusters combinations but with a random permutation σ (see Section 3.3.2 and Equation 3.10). The search inside a pair of clusters is carried out systematically in order trying to shift (or exchange) every single customer from one cluster to the other cluster and vice versa. In contrast, the algorithm, RSS.OC uses systematic ordered search of clusters rather than the random ordered search in the algorithm $R\bar{S}S$.OC.

Table 3.2. Average relative percentage deviations, ARPD's, for the simulated annealing schemes.

	Values of T_s				
Schemes	δ_{max}	2STD	3STD	AVINC	$10 \times \delta_{max}$
HSS.AV	1.099	0.915	1.028	0.866	1.066
HSS.OC	0.448	0.278	0.361	0.436	0.363
RSS.LM	0.744	1.134	1.077	0.848	1.115
RSS.OC	0.405	0.571	0.405	0.494	0.294
$R\bar{S}S$.OC	0.391	0.495	0.330	0.394	0.271
RRR.OC	0.503	0.367	0.423	0.432	0.619
$R\bar{S}S$.MOC	0.256	0.221	0.356	0.310	0.372

HSS:	Heuristic starts, systematic search of cluster combinations, systematic search between clusters.
$R\bar{S}S$:	Random starts, systematic search of a random permutation of cluster combinations, systematic search between clusters.
RRR:	Random starts, random selection of clusters, random search between clusters.
AV:	Aarts *et al.* [1985] cooling scheme
LM:	Lundy & Mees [1986] cooling scheme
OC:	Our cooling scheme with a variable deferment control β_t
MOC:	Our cooling scheme with a constant deferment control β_t
T_s:	Initial starting temperature

Furthermore, the algorithm RRR.OC chooses a pair of clusters randomly and makes a random shift (or exchange) of customers. It can be seen in Table 3.2, that the $R\bar{S}S$.OC yields better results on average than that of RRR.OC. This is, perhaps because a random search may miss a neighbour having a better solution, whereas the ordered search gives a guarantee to find it within a complete cycle of search. In addition, the algorithm RRR.OC takes more running time to generate random neighbours, with no improvements in solution quality. Finally, the $R\bar{S}S$.MOC algorithm uses the same cooling schedule introduced previously (see Section 3.4.2) in which the parameter β_k is fixed to a constant value. Furthermore, the algorithm, $R\bar{S}S$.MOC, uses the same conditions for the temperature resets, Equation (3.13) and the same stopping criterion. In fact, this scheme becomes the same as Lundy & Mees [1986] but with the additional temperature reset strategy. The results of the $R\bar{S}S$.MOC algorithm are comparable to the $R\bar{S}S$.OC results. Both algorithms gave the best relative percentage deviations of less than 0.40 % at all given initial temperatures.

Experimental results show that the quality of the solution produced by HSS.OC which uses a heuristic starting solution, is as good as, or better than that of RSS.OC which uses a random initial solution, yet with less computation time. Hence, a coupling heuristic solution with SA is preferred as it reduces the computation time without loosing the solution quality. However, $R\bar{S}S$.OC results are better than the RSS.OC and HSS.OC with slightly more computation time as seen in Table 3.3. We see clearly that the random permutation of clusters and their sequential search ($R\bar{S}S$) is better than just a sequential search of cluster with a fixed order (RSS). This demonstrates the importance of the neighbourhood and the way in which it is searched.

3.6.4 Comparisons of simulated annealing and descent algorithms.

The ARPD from the best known solutions are taken over all initial temperatures. Table 3.3 also reports the total number of estimated runs with their corresponding total CPU time, T_τ, for the different algorithms. We shall establish an estimate of the number of different runs (*NR*) needed to achieve the best known solution with a given confidence probability, based on the empirical probability, P, of finding the best solution in a single run. An algorithm produces a solution within 1% of the best known solution in *NR* independent runs, when $NR \geq \frac{\ln(0.01)}{\ln(1-P)}$. The total number of estimated runs with their corresponding total time T_τ for different algorithms are given Table 3.3. We observe that all our SA versions have the highest probabilities of finding the best known value in one run with $P \in [0.482, 0.504]$, compared to $P \in [0.278, 0.365]$ of other SA schemes.

Table 3.3. Average relative percentage deviation over all initial T_s

Methods	P	*NR*	ACT	T_τ
HSS.AV	0.278	14.13	90.72	1282
HSS.OC	0.482	7	97.10	679
RSS.LM	0.365	10.3	101.79	1048
RSS.OC	0.494	6.7	101.60	680
R$\bar{S}$S.OC	0.496	6.4	117.87	754
R$\bar{S}$S.MOC	0.504	6.5	116.66	758
1+FI	0.10	43.7	4.79	210
2+FI	0.13	33.0	46.49	1534

P:	Probability of finding the best known solution in single run.
ACT :	The average CPU time of a single run.
NR :	The estimated number of runs to produce solutions within 1% of the best known solution.
T_τ :	The total CPU time for NR runs.
Others:	As defined in Table 3.2.

The total computational time T_τ for HSS.OC, and all *.OC, to achieve a 1% confidence interval is less than half of that of HSS.AV, RSS.LM and 2+FI CPU time. Our annealing algorithms found the best known results in 20 out of the 20 problems tested. The comparable figures for HSS.AV, HSS.LM and 2+FI were 15 out of 20, 16 out of 20 and 7 out of 20 respectively. Although, the 1+FI takes the least computational time of all algorithms, it may never find the best solutions. The 1+FI algorithm found the best solutions for only 7 out of 20 problems of small sizes 50×5 and failed to find the best known solution in any of the problems of size 100×10. Thus, our simulated annealing heuristic SA(R,S,S,OC) is preferred to other methods.

3.6.5 Comparisons of metastrategy SA and TS algorithms

The previous computational results deal with simulated annealing and descent algorithms. They can be found in Osman & Christofides [1989] which does not include comparisons with the recent annealing scheme of Connolly [1990] or the tabu search algorithm that we have developed here. This section investigates the performance of our SA scheme with the above mentioned approaches.

(i) Simulated annealing schemes

First, we compare the annealing schemes. We denote our SA algorithm by H$\bar{\text{S}}$S.OC and the SA algorithm of Connolly [1990] by H$\bar{\text{S}}$S.CO. H$\bar{\text{S}}$S.CO takes the initial and final temperatures T_s and T_f to be expressed in terms δ_{max}, and δ_{min}. The temperatures are then set as follows: $T_s = \delta_{max} + 0.1 \times (\delta_{max} - \delta_{min})$, and $T_f = \delta_{min}$. Our scheme starts with the initial temperature set to $T_s = \delta_{max}$. Both schemes have the same starting solutions, neighbourhood generation mechanism and the same stopping criteria. Our SA algorithms terminate, if the number of temperature resets for which there is no improvement of the best solution is greater than R, R=3. The algorithm H$\bar{\text{S}}$S.OC is run under a similar stopping criterion. The H$\bar{\text{S}}$S.CO uses the same cooling schedule of R$\bar{\text{S}}$S.MOC without the multiple resets as in Equation (3.13). H$\bar{\text{S}}$S.CO resets its temperature to T_{found} at which the best solution is found thereafter the temperature is left fixed.

Table 3.4. Computational results of our SA scheme, H$\bar{S}$S.OC and that of Connolly [1990] H$\bar{S}$S.CO.

$n \times p$	No.	H$\bar{S}$S.OC[*]	RPD	CPU	H$\bar{S}$S.CO[*]	RPD	CPU	Best
	1	820[b]	0	47.49	820[b]	0	39.18	820
	2	664[b]	0	9.13	664[b]	0	16.07	664
	3	713[b]	0	12.74	734	2.94	12.77	713
	4	829[b]	0	33.59	829[b]	0	13.69	829
50x5	5	751[b]	0	13.49	751[b]	0	22.89	751
	6	740[b]	0	10.50	740[b]	0	13.62	740
	7	715[b]	0	52.09	715[b]	0	25.21	715
	8	778[b]	0	34.03	778[b]	0	14.24	778
	9	651[b]	0	7.93	651[b]	0	18.95	651
	10	787[b]	0	11.33	805	2.28	26.76	787
ave. 50	-	-	0[a]	23.23[c]	-	0.52[a]	20.34[c]	
	11	1039	0.48	144.14	1037	0.29	269.48	1034
	12	985	0.30	131.98	982[b]	0	251.32	982
	13	1005[b]	0	484.52	1019	1.39	134.39	1005
	14	1006[b]	0	650.85	1006[b]	0	190.94	1006
100x10	15	1045	0.19	299.07	1045	0.19	156.52	1043
	16	1026[b]	0	191.42	1026[b]	0	223.51	1026
	17	1037	0.58	464.61	1032	0.09	143.08	1031
	18	1092	0.09	165.07	1091[b]	0	426.06	1091
	19	966[b]	0	631.11	966[b]	0	1536.54	966
	20	955	0.10	188.97	954[b]	0	401.13	954
ave. 100	-	-	0.17[a]	335.17[c]	-	0.19[a]	373.30[c]	
overall		-	0.08[a]	179.20[c]	-	0.36[a]	196.82[c]	

[*] : the objective function values of the corresponding SA scheme.
[a] : the value represents the, ARPD, average relative percentage deviations.
[b] : indicates that the best known solution is attained.
[c] : the value represents the, ACT, average computation time in seconds.

The constant value of β is initially computed as in Equation (2.12) with a constant, $M = n \times Nfeas$ where $Nfeas$ is the size of the feasible moves in the neighbourhood. Both algorithms use the same heuristic starting solution.

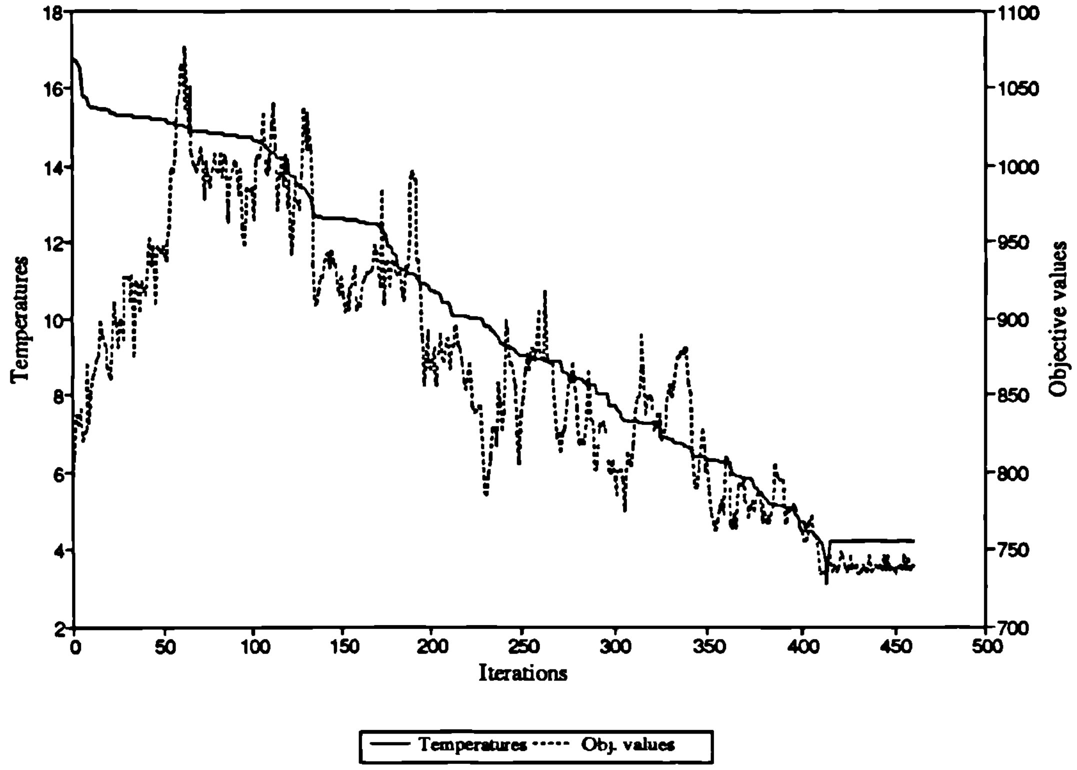


Figure 3.4 Objective value and temperature evolutions of the $H\bar{S}S$.CO scheme for Problem number 3.

Computational results, reporting the RPD and ARPD along with the total computation time in CPU seconds to achieve the best solutions, are presented in Table 3.4. This give an indication of how quickly each of the algorithms finds the best solution. We observe that the quality of solutions and computation times of $H\bar{S}S$.OC are better than that of $H\bar{S}S$.CO. The ARPD value of $H\bar{S}S$.OC is 0.08%, compared to 0.36% produced by $H\bar{S}S$.CO. This represents an improvement of 350% in the ARPD. This is accompanied by a reduction of 10% in the ACT of that of $H\bar{S}S$.CO values. More precisely, the poor performance of HSS.CO in problems 3, 10 and 13 is explained by the fact the scheme falls in a "hole" where T_{found} is either too small a value (Problems 3 and 13) or too high a value (Problem 10). In cases where T_{found} is small, it is difficult to escape from the hole of local optimality unless the temperature is reset to a higher value than T_{found}.

Figure 3.4 demonstrates the case of Problem 3, where the scheme falls into a hole at a small value of $T_f = 3.17$. However, in the case of a high T_{found}, the scheme may (Problem 19) or may not (Problem 10) escape the local optimality easily. The computation time to find the best solution for Problem 19 is 1536 CPU seconds.

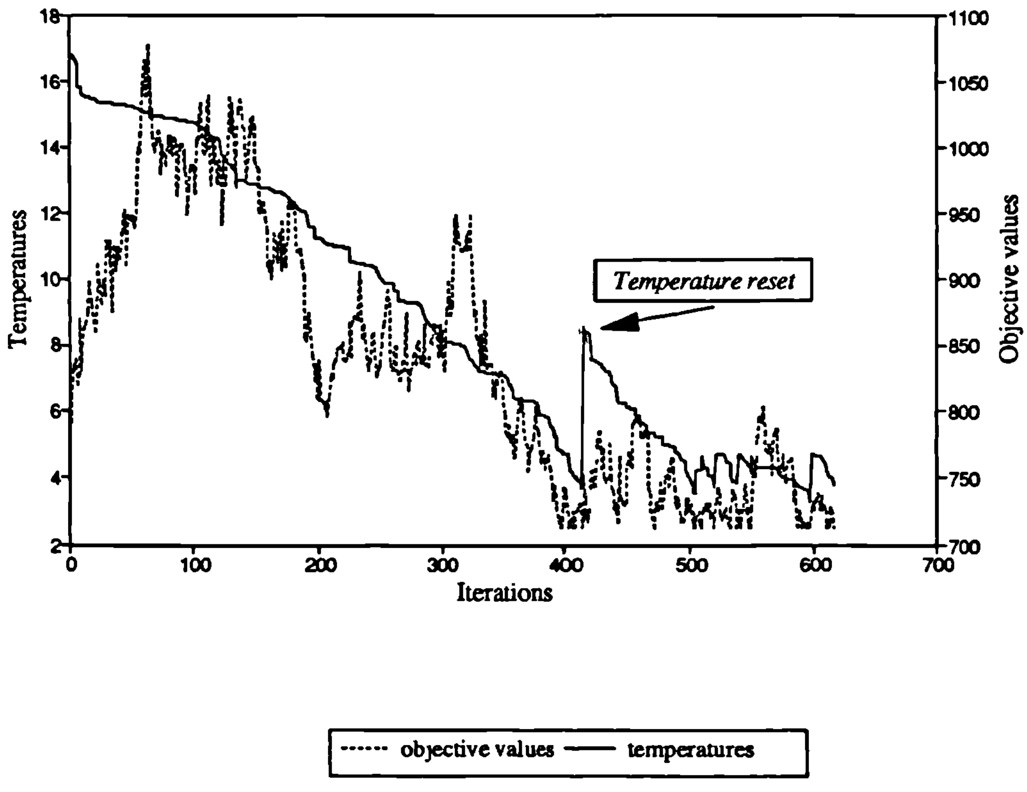


Figure 3.5 Objective values and temperatures evolutions of the HSS.OC for Problem 3.

The $H\bar{S}S$.OC scheme overcomes the problems encountered in the $H\bar{S}S$.CO scheme through the non-monotonic reduction in the temperature and the automatic update procedure. Moreover, $H\bar{S}S$.OC shows little deviation in the solutions produced and their computation requirements. However, the scheme shows a slight deterioration in the relative percentage deviations as the problem size is increased, reaching an RPD value of 0.58% for Problem 17.

Figure 3.5 shows the evolution of the temperature and the objective values for Problem 3. In the figure, we started the $H\bar{S}S$.OC from the same starting solution and initial temperature as $H\bar{S}S$.CO, but with different cooling schedules.

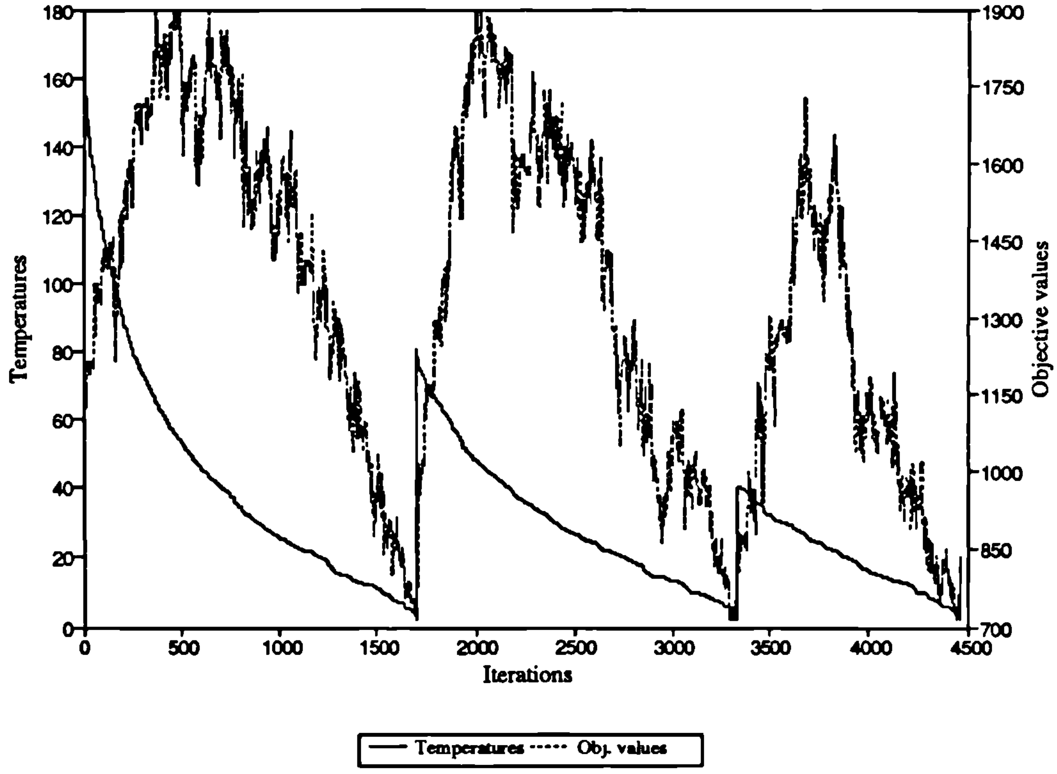


Figure 3.6 Objective values and temperature sequences evolutions of $H\bar{S}S$.OC with $T_s = \delta_{max}$ for Problem 3.

Figure 3.6 illustrates the change in the temperatures and objective value sequences as we start from a higher initial temperature value for the $\bar{\text{HSS}}$.OC scheme on Problem 3. The quality of the final solution did not change. However a longer running time was needed for the algorithm to satisfy the stopping criterion.

Table 3.5. Descent algorithm performance with FI and BI move selection strategies.

No.	ρ	H.OC*	RPD	1+FI*	RPD	1+BI*	RPD
1	0.92	945	15.24	834	1.7	837	2.07
2	0.90	804	21.08	746	12.34	677	1.95
3	0.82	786	10.23	780	9.39	818	14.72
4	0.96	1017	22.67	844	1.8	891	7.47
5	0.85	972	29.42	811	7.98	816	8.65
6	0.84	816	10.27	762	2.97	778	5.13
7	0.93	752	5.17	735	2.79	734	2.65
8	0.92	882	13.36	841	8.09	847	8.86
9	0.86	891	36.86	651	0	652	0.15
10	0.92	968	22.99	852	8.25	824	4.7
ave. 50	-		18.73		5.53		5.64
11	0.89	1665	61.02	1104	6.76	1092	5.6
12	0.88	1635	66.49	998	1.62	1054	7.33
13	0.94	1872	86.26	1036	3.08	1030	2.48
14	0.85	1761	75.04	1020	1.39	1019	1.29
15	0.89	1345	28.95	1089	4.41	1136	8.91
16	0.86	1847	80.01	1144	11.5	1053	2.63
17	0.9	1634	58.48	1105	7.17	1125	9.11
18	0.88	1517	38.91	1098	0.54	1138	4.21
19	0.85	1567	62.21	1004	3.93	974	0.82
20	0.88	1780	86.58	1063	11.42	993	4.08
ave. 100	-		64.40		5.18		4.65
overall	-		41.56		5.36		5.14

* : the objective function values of the corresponding descent algorithms.

(ii) Descent and tabu search algorithms.

Table 3.5 refers to the performance of the pure descent algorithms (1+FI and 1+BI). It displays the ratio of problem tightness, ρ, and the objective function values with their relative percentage deviations for the following: the initial constructive heuristic (H.OC of Section 3.2); the solutions obtained by the first-improve (1+FI) and the best-improve (1+BI) descent algorithms. The heuristic solutions of H.OC are very bad with RPD's ranging from 5% to 86%, at an overall average of 41%. This average value is improved significantly to ARPD values of 5.36% and 5.14% by the 1+FI and 1+BI algorithms respectively.

Let us represent the TS algorithms with FBA and BA move selection strategies by TS+FBA and TS+BA respectively. Both the TS+FBA and TS+BA algorithms are run with MAXI= $5 \times n$ and $|Ts| \in \left\{\left\lceil \frac{n}{2} \right\rceil, \left\lceil \frac{n}{3} \right\rceil, \ldots, \left\lceil \frac{n}{6} \right\rceil\right\}$ where $\left\lceil \frac{n}{x} \right\rceil$ denotes the smallest integer greater than $\frac{n}{x}$. Let TSF denotes the estimated value of $|Ts|$ obtained by Equation (3.16), and TSB represents the estimated value of $|Ts|$ obtained by Equation (3.17). In addition, both of the TS+FBA and TS+BA algorithms employ variable tabu list sizes rather than a fixed tabu size value, $|Ts|$, throughout the search. Thus, 10% values are taken around the initial estimates, TSB and TSF. Let these three values be $|Ts| - (0.1 \times |Ts|)$, $|Ts|$, and $|Ts| + (0.1 \times |Ts|)$. A random permutation of them is generated. Every $2 \times |Ts|$ iterations, $|Ts|$ is assigned to one of them in order.

Figure 3.7 shows the performance of the TS+FBA and the TS+BA algorithms with different values of $|Ts|$. We observe that the best value of $|Ts|$ is $\left\lceil \frac{n}{4} \right\rceil$ for the TS+FBA algorithm. The TS+FBA algorithm produces the best of TS results, with an ARPD value of 0.25% compared to 0.32% of that of the TS+BA algorithm with its best $|Ts|$ value at $\left\lceil \frac{n}{5} \right\rceil$. It is also clear that moving away from the best $|Ts|$ in the direction of smaller or higher $|Ts|$ values produce poor quality solutions. This is because a high $|Ts|$ value would be restrictive, while a small | *Ts* | value induces cycling. Further the TS+FBA algorithm with the tabu list estimated, TSF, produces an ARPD of 0.32% which is very close to the best ARPD of 0.25% obtained with | *Ts* |=$\lceil \frac{n}{4} \rceil$. However, the TS+BA algorithm with the estimated TSB for | *Ts* | produces an ARPD of 0.65% compared to 0.32% obtained with | *Ts* |=$\lceil \frac{n}{5} \rceil$. These results are elaborated more in Table 3.6 and Table 3.7.

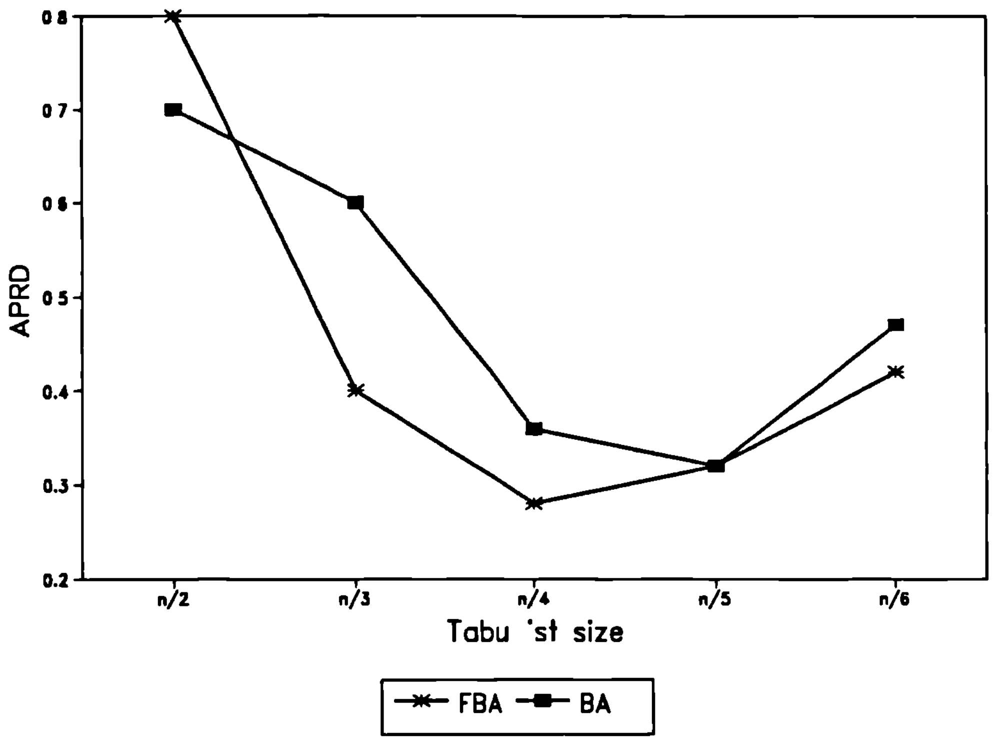


Figure 3.7. Average relative percentage deviations of the TS+FBA algorithm and of the TS+BA algorithm, with different values of tabu list size, and with MAXI=5 × *n*.

Table 3.6 presents the results which are obtained by the TS+FBA algorithms with | *Ts* |=$\lceil \frac{n}{4} \rceil$ and with the estimated | *Ts* | value, TSF. We notice that the results of the TS+FBA algorithm with TSF, are very good compared to that obtained with | *Ts* |=$\lceil \frac{n}{4} \rceil$.

This is probably due to the introduction of the ratio factor in the estimate Equation (3.15). The TS+FBA algorithm with the TSF strategy produces an ARPD value of 0.32%, which is very close to an ARPD of 0.28% that was produced with a $|Ts| = \lceil \frac{n}{4} \rceil$.

Table 3.6. Computational results of FBA tabu search scheme with the tabu size equal to $\lceil \frac{n}{4} \rceil$ and the estimated regression fit, and MAXI=5 × n

No.	TSF solutions	RPD	CPU	$\|Ts\| = \lceil \frac{n}{4} \rceil$ solutions	RPD	CPU	Best known
1	821	0.12	10.82	821	0.12	20.78	820
2	664	0	61.90	664	0	26.86	664
3	734	2.94	26.16	734	2.94	8.00	713
4	829	0	7.92	829	0	45.16	829
5	751	0	11.96	751	0	11.65	751
6	740	0	4.17	740	0	4.34	740
7	715	0	13.14	715	0	21.25	715
8	778	0	9.92	778	0	10.96	778
9	651	0	5.68	651	0	6.21	651
10	787	0	10.00	787	0	10.72	787
ave. 50		0.30	16.17	747	0.30	16.59	
11	1038	0.38	169.04	1040	0.58	1022.06	1034
12	985	0.30	550.06	985	0.30	671.96	982
13	1009	0.39	170.56	1005	0	327.93	1005
14	1020	1.39	82.10	1009	0.29	313.24	1006
15	1045	0.19	272.69	1045	0.19	142.19	1043
16	1026	0	651.70	1026	0	253.77	1026
17	1033	0.19	1259.04	1034	0.29	247.56	1031
18	1092	0	632.91	1096	0.36	50.55	1092
19	968	0.20	96.40	968	0.20	79.05	966
20	957	0.31	634.87	957	0.31	347.93	954
ave. 100		0.33	451.94		0.25	345.62	
overall		0.32	234.05		0.28	181.11	

TSF : is the statistical estimate of the tabu list size used in the TS+FBA algorithm.

Table 3.7. Computational results of the TS+BA algorithm with the $|Ts|$ values equal to $\lceil \frac{n}{5} \rceil$, with the estimated value TSB, respectively and a value for MAXI=$5 \times n$

No.	TSB solutions	ARPD	CPU	$\|Ts\|=\lceil \frac{n}{5} \rceil$ solutions	RPD	CPU	Best solutions
1	820	0	4.55	820	0	6.44	820
2	664	0	43.06	664	0	23.72	664
3	734	2.94	19.80	734	2.94	17.21	713
4	829	0	11.88	829	0	11.19	829
5	751	0	17.84	751	0	21.65	751
6	740	0	4.37	740	0	9.16	740
7	715	0	10.03	715	0	20.97	715
8	778	0	10.62	778	0	38.89	778
9	651	0	7.41	651	0	4.85	651
10	787	0	80.01	787	0	83.87	787
ave. 50		0.29	20.96		0.29	23.80	
11	1058	2.32	554.18	1037	0.29	1042.40	1034
12	1011	2.95	94.97	1001	1.93	308.87	982
13	1010	0.49	80.20	1008	0.29	311.49	1005
14	1006	0	292.06	1006	0	232.36	1006
15	1076	~3.16	71.96	1045	0.19	775.59	1043
16	1026	0	241.57	1026	0	266.66	1026
17	1039	0.77	307.10	1033	0.19	947.79	1031
18	1094	0.18	1333.26	1094	0.18	685.97	1092
19	968	0.20	73.04	966	0	619.53	966
20	955	0.10	468.63	957	0.31	973.64	954
ave. 100		1.01	351.70		0.34	560.57	
overall		0.65	186.33		0.32	292.18	

TSB : is the statistical estimate of the tabu list size used in the TS+BA algorithm.

However, in Table 3.7, the results of the TS+BA algorithm with the estimated tabu size, TSB, are relatively poor with an average of 0.65% compared to 0.32% produced by the TS+BA algorithm with $|Ts| = \lceil \frac{n}{5} \rceil$.

Table 3.6 also lists the computational results of the TS+FBA algorithms on all test problems. This table provides the relative (and average relative) percentage deviations, the average CPU time in seconds to the best iterations at which the best obtained is found. Similar results are reported in Table 3.7 for the TS+BA algorithms. Both TS algorithms are run with MAXI= $5 \times n$ iterations. This stopping criterion is advantageous as it allows enough time for each problem depending on its dimension. However, the drawback of this criterion is that it requires some extra CPU time to prove the solution quality. Comparing the TS algorithms, we find that the TS+FBA algorithm produces better results in terms of solution quality and computational time than the TS+ BA algorithm. All the TS algorithms have failed to produce the best solution for Problem 3 which has an RPD of 2.94%. Since this problem has the least capacitated demand with ratio 0.82 and since it affords more feasible solutions, the failure can be contributed to its topological structure and its solution distribution.

Finally, we draw some conclusions on the performance of both SA and TS meta-strategy algorithms. The best of the TS+BA algorithm produces better results than the SA scheme $H\bar{S}S$.CO. The ARPD of the TS+FBA algorithm is 0.28% with an ACT of 181 seconds compared to 0.36% at 196 seconds of the SA algorithm $H\bar{S}S$.CO. Yet, the TS+FBA algorithm is out-performed by the SA ($H\bar{S}S$.OC) algorithm, both in solution quality and computation time. The $H\bar{S}S$.OC has an ARPD of 0.08% at an ACT of 179 seconds. It must be pointed that the ARPDs of TS+FBA could be lowered to only 0.12% if Problem 3 is removed as it has contributed a significant value of 2.94% to the ARPD value. In this case, both TS and SA would then become comparable. Moreover, the implementation of TS schemes, here, did not involve any sophisticated strategies such as a long-term memory function or any special data structures. These strategies have reduced the ACT time by half in the case of application to the VRP as will be discussed in Chapter 5.

3.7 Concluding remarks.

In this chapter, we have presented a review of the Capacitated Clustering Problem (CCP) theory and applications and investigated the performance of the SA and TS metastrategy algorithms. The CCP is an *NP-complete* problem. Hence, iterative improvement descent methods and SA and TS metastrategy algorithms are strongly recommended to the solution of the CCP problem. To our knowledge, these approaches have not been used to the CCP.

In this chapter, a constructive heuristic method is developed to generate quickly an initial solution. This is used in the iterative descent methods and the metastrategy SA and TS approaches. A λ–interchange mechanism is also developed to generate neighbouring solutions with the first-improve and best-improve search strategies. Furthermore, the SA algorithm is introduced, and SA schemes are classified according to their cooling schedule into three categories: continuous temperature reduction schemes; stepwise temperature reduction schemes and, non-monotonic temperature reduction schemes.

A non-monotonic SA algorithm is introduced for the CCP. It uses multiple temperature resets, and reduces the temperatures with a dynamic (rather than a fixed) reduction factor. This is to provide a smoother temperature reduction as the number of iteration increases. Different search and selection strategies are implemented and the best are identified. Extensive computational results on randomly generated test problems of varying sizes have demonstrated the superiority of the new coo ing schedule over the best existing cooling schedules. The SA scheme depends heavily on the neighbourhood search and generation. The SA algorithm, if combined with a heuristic start, would improve the computation time without deteriorating the solution quality. This is mainly because a good neighbourhood search scheme is implemented that improves the quality of the random starting solution.

Both descent methods and SA algorithms deserve serious consideration when selecting a heuristic. Their performance is compared in terms of solution quality and computational requirements. The 1+FI descent algorithm has the advantage of requiring smaller computation time. However, its performance decreases as the problem size and capacity tightness increases. It is only recommended when computational time is a limiting resource. A recent SA scheme due to Connolly [1990] can be seen as a special case of our non-monotonic SA scheme. Since, it resets the temperature only once to, T_{found}, the one at which the best improved solution is found and kept at that value thereafter. A comparison with this scheme is also made. Our SA scheme performs better than that of Connolly [1990], although the latter produces generally good results for problems of smooth topological structures. Furthermore, our SA gives the user control in balancing the trade-off between the amount of computation time and the quality of solution desired. This is an advantage over previous SA algorithms.

TS algorithms are also investigated, developed and experimentally tested on the same test problems. Two tabu schemes are implemented: the TS+BA is based on a first-best-admissible (FBA) selection strategy, whereas the TS+FBA scheme uses a best-admissible (BA) selection strategy. Only simple TS strategies are tested. The tabu list size is derived and expressed as a function of problem characteristics, and a method of reducing the error in its statistical estimate is implemented. Computational results showed that the new TS+FBA algorithm performs better than the classical TS+BA algorithm and takes less computational time. The TS+FBA algorithm out-performed the SA of Connolly [1990] but not our new SA scheme. This may be attributed to the poor performance of TS algorithm on one of the test problems.

Finally, both the SA and TS algorithms assume a given fixed number of clusters. This is not a restriction as it is possible to start from an infeasible number of clusters. The algorithms would always find the optimal number of clusters for which a feasible solution exists. In our schemes, only feasible moves are checked, whereas it is possible to allow infeasible neighbours to be considered with some penalties. We do not think that such variations are necessary because of the generation mechanism we employed. Both SA and TS algorithms are flexible and problem independent, in that, no apriori knowledge is required about the problem structure. They always guarantee that better solutions will be found (not worse) for a bigger number of iterations. They are the state of the art algorithms for the CCP, and must be used for the solution of other related optimization problems such as Bin-Backing Problem. Future work could consider the topological structure of the problem and its impact on the behaviour of algorithms. Other strategies such as a long-term memory function, and the data structure for TS algorithm must be improved. The CCP approach could be extended to solve the vehicle routing problem in two stages: a clustering phase followed by a routing or sequencing phase with an appropriate cost estimate. Also, problems related to the CCP such as the maximal covering location problem with capacities (Pirkul & Schilling [1991]) can be attempted by the metastrategy SA and TS algorithms.

Chapter 4

SIMULATED ANNEALING AND TABU SEARCH FOR THE GENERALISED ASSIGNMENT PROBLEM

4.0 Introduction

The generalised assignment problem (GAP) is the problem of finding a minimum cost assignment of a set of jobs to a set of agents. Each job, j, is assigned to exactly one agent, i. If agent i performs job j, a_{ij} amount of resources would be required, at a cost of c_{ij}. The total demands of all jobs assigned to any agent i can not exceed b_i, the total resources available of that agent. A pictorial representation of the GAP is given in Figure 4.1.

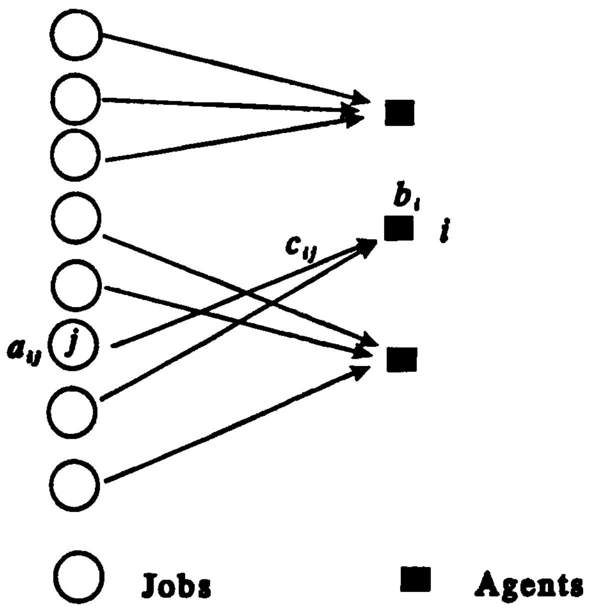


Figure 4.1 The Generalised Assignment Problem.

The GAP is a combinatorial optimization problem, which has theoretical and practical importance. Theoretically, many location problems can be modelled as a GAP, these include: the p-median problem and the plant location problem (Ross & Soland [1977]). The GAP is also a sub-problem in vehicle routing problems (Fisher & Jaikumar [1981]) and ~~in~~ of the capacitated clustering problem (See Chapter 3). Practically, the GAP has many real-life applications in, for example, fixed charge plant location problems in which customer requirements must be satisfied by a single plant, scheduling of payments on accounts where fixed-sum payments are specified; scheduling of variable-length television commercials into time slots; scheduling of project networks; assigning software development tasks to programmers; assigning jobs to computers in computer networks; design of communication networks with node capacity constraints and so on. We refer to the book by Martello & Toth [1990] for further details on recent theories and applications of the GAP.

The GAP is an *NP − Complete* optimization problem (Fisher *et al.* [1986]) and is therefore practically intractable for large-sized instances. As a consequence, our efforts are geared towards developing efficient and effective approximate algorithms for such problem instances. These approximate algorithms include: iterative improvement descent methods, simulated annealing (SA) and tabu search (TS) metastrategy algorithms.

This chapter is organised as follows: Section 4.1 introduces the integer programming formulation of the GAP and reviews some of the approximate and exact methods to solve it. Section 4.2 discusses aspects of iterative improvement methods and introduces two λ−interchange improvement descent algorithms. The first version is based on a *first-improve* (FI) strategy for the selection of neighbourhood moves. The second version is based on a *best-improve* (BI) selection strategy. Section 4.3 presents the SA algorithm and discusses its implementation. This algorithm is based on the non-monotonic cooling schedule that was introduced in Section 3.4.2. Section 4.4 describes the TS metastrategy algorithm and explains different strategies for this TS implementation. Computational experiences are given in Section 4.5, that compare the performance of the descent, SA, and the TS algorithms with the best existing approximate and exact methods in the literature. Section 4.6 contains some concluding remarks.

4.1 Mathematical programming formulation and review

We define the following input parameters for the GAP:

a_{ij} = Resource required by agent i to perform job j,

b_i = Resource capacity of agent i,

c_{ij} = Cost of assigning job j to agent i,

$I = \{1,\ldots,m\}$ Set of agents ,

$J = \{1,\ldots,n\}$ Set of jobs.

Let us define also the decision variables:

$x_{ij} = 1$, if job j is assigned to agent i

$= 0$, otherwise.

The GAP can be formulated as an integer problem as follows:

$$\text{Min} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{4.1}$$

subject to:

$$\sum_{i \in I} x_{ij} = 1, \qquad \forall j \in J, \tag{4.2}$$

$$\sum_{j \in J} a_{ij} x_{ij} \le b_i, \qquad \forall i \in I, \tag{4.3}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in I, \quad \forall j \in J, \tag{4.4}$$

The objective is to minimize the total cost of the assignment Equation (4.1) - such that each job is assigned exactly to one agent - constraint (4.2) - without exceeding the total resource capacity of each agent - constraint (4.3).

In this chapter, we give a brief description of exact and approximate algorithms for the GAP. We use some of these algorithms to compare our results with. Most exact methods are based on branch and bound techniques which use bounds derived from either a *linear programming* (LP) relaxation of the integer restrictions, or *deletion, la_grangean, surrogate, or decomposition* relaxations of the knapsack (or assignment) constraints.

Benders & Van Neunen [1983] propose an LP relaxation of constraints (4.4). They conclude that the LP bound tends to be strong when the number of jobs is large compared to the number of agents and when the resource capacities are rather loose.

Ross & Soland [1975] use a relaxation in a two-stages method, which is based on the deletion of constraints (4.3). A lower bound is obtained by assigning each job to the least costly agent. Then, every constraint in (4.3) is checked for feasibility. In a second step, minimum penalties are computed for reassigning jobs from one agent to another in order to satisfy the capacity restrictions. These penalties are then added to the lower bound. Their branching strategy is based on the penalty for not using the least cost assignment of a job, and also on the remaining resources available to an agent.

Martello & Toth [1981] use a similar procedure in which the GAP is re-formulated as a maximization problem. They remove the assignment constraints to compute an upper bound. Furthermore, Martello & Toth [1981] describe a heuristic for the maximization problem. We will denote this heuristic as MT. The MT heuristic is used to generate a lower bound for their branch and bound procedure. Computational results are given for problems of size up to 5-agents and 20-jobs. They compare their results with that of Ross & Soland [1975]. The conclusion was that, for easy problems, where the capacity constraints are loose, the algorithm of Ross & Soland [1975] performs better, and that the average running times are less. For hard problems, where capacity constraints are tight or when the objective function coefficients and capacity requirements are correlated, the exact algorithm of Martello & Toth [1981] out-performs the algorithm of Ross & Soland [1975] in terms of both running time as well as the number of tree nodes. Furthermore, for the large and easy problems of size varying from 5-agents and 20-jobs to 20-agents and 200-jobs, the MT heuristic produces an average relative error less than 0.1%. More recently, Martello & Toth [1990] present exact solution methods with different combinations of bounding procedures, problem up to 30-jobs and 10-agents are solved. They show that their results are better than that of Fisher *et al.* [1986]. Martello & Toth [1990] also report results with the MT heuristic to problems with size up to 500-jobs and 50-agents, with average relative percentage deviations that vary from 0.14% to 13%.

There are two types of Lagrangean relaxations for the GAP. The first is obtained by dualizing the assignment constraints. This reduces the GAP to $|I|$ 0-1 knapsack problems. The second relaxation is obtained by dualizing the capacity constraints. This would give $|J|$ generalised upper bound (GUB) problems which can be solved by simple inspection. Geoffrion [1974], shows that bounds from a Langrangean relaxation can do no better than LP-relaxation bounds when the Lagrangean problem satisfies the integrality property. It follows that relaxations of the second type, in which the subproblems are GUB problems, produce solutions that are at best, equivalent to the LP-relaxation. However, the first relaxation with 0-1 knapsack sub-problems, whose solutions are not naturally integer, would produce superior bounds to those produced by an LP-relaxation.

A langrangean relaxation of the first type is reported in Fisher *et al.* [1986]. Their procedure consists of two phases. In the first phase, they relax the assignment constraints (4.2) and set the lagrangean multipliers (μ_j, $\forall j \in J$), in a maximization problem, to the second largest c_{ij}. A job j is assigned to an agent i if the cost of this assignment is such that $c_{ij} - \mu_j > 0$. An assignment problem is solved for those jobs with $c_{ij} - \mu_j = 0$. This yields the same initial bound as that generated by Ross & Soland [1975]. In the second phase, a heuristic procedure is used to adjust the multipliers in order to assign more jobs and strengthen the bound. This procedure is repeated until no further improvements are possible. At the end, the branch and bound algorithm selects the un-assigned variable x_{ij} with the largest a_{ij} to branch on. Results of this algorithm are compared with those from Martello & Toth [1981] and Ross & Soland [1975]. The algorithm of Fisher *et al.* [1986] is faster and requires less number of tree nodes on small-sized problems of size 5-agents and 20-jobs. Guinard & Rosenwein [1989] enhance the dual ascent procedure of Fisher *et al.* [1986]. They exploit violations of assignment constraints that are not considered in the previous approach, and add a surrogate constraint to the lagrangean relaxed model. The new algorithm uses a branching scheme that is based on a combination of a depth-first, and breadth-first strategy. They also perform a sub-gradient optimization procedure at the root node to increase the bound. As a result, they solve problems of size up to 10-agents and 50-jobs using less CPU time and fewer number of tree nodes.

A langrangean relaxation heuristic of the second type is reported by Klastorin [1979]. In the first phase, a modified sub-gradient procedure is used to find the optimal dual solution. This procedure continues until a *primal* feasible solution is found. In a second phase, a tree search is performed in the neighbourhood of the feasible solution obtained in order to improve it. Computational tests show that the heuristic procedure yields solutions with an average deviation of 1% from optimality. Moreover, they also conclude that the second phase is computationally expensive to provide any significant improvements.

Jörnsten & Näsberg [1986] propose a surrogate Lagrangean relaxation of the knapsack constraints (4.3). This relaxation produces multiple choice knapsack problems. The bounds obtained are at least better than the Lagrangean relaxation of constraints (4.3) or the LP-relaxation. However, they are slightly inferior to those obtained by the Lagrangean relaxation of the assignment constraints (4.2). Jörnsten & Näsberg [1986] also report another approach based on a Lagrangean decomposition. In this approach, additional integer variables, y_{ij}, are added to the GAP formulation in the form of equality constraints $x_{ij} = y_{ij} \; \forall \, i \in I, \forall j \in J$. In addition to the other constraints, these new constraints are also relaxed. They multiply the objective function by (A + B = 1), where A and B are constant, such that A + B = 1. The resulting relaxation is decomposed into sub-problems: 0-1 knapsack problems in the x-variables and GUB problems in the y-variables. The bounds obtained through this methods are compared to those from classical Lagrangean relaxation procedures. Based on a small set of 10 test problems of sizes 4-agents and 25-jobs, they conclude that this bound is stronger than the one obtained by relaxing either constraints (4.2) or constraints (4.3). Theoretical bounds, which are based on a combination of Lagrangean decomposition and bound improving sequences, are provided by Barcia *et al.* [1990].

Set partitioning (SP) and simulated annealing (SA) approximate approaches are proposed by Cattrysse [1990] for the GAP. These approaches are used either as stand-alone heuristics or in conjunction with a pre-processing variable reduction (fixing) procedure. We denote the combined procedures as FSP (for fixing-set-partitioning) and FSA for fixing-simulated annealing). The fixing procedure uses the knapsack constraints to add valid inequalities (or facets) to strengthen the LP-bound. For every violated knapsack constraint, a valid inequality of the facet of the knapsack polytope is added to the LP-formulation. The extended formulation is then solved to generate more facets. This procedure is repeated until no further valid inequalities can be generated. At the end of the procedure, all integer variables, which have $x_{ij} = 1$, are used to assign jobs to corresponding agents. The capacity of each agent is adjusted, the fixed variables are removed and a reduced problem of smaller size is created. We refer to Cattrysse [1990] for further information on how to generate valid inequalities for the GAP and to Hoffman & Padberg [1986] for a general overview of LP-based solution methods for combinatorial optimization problems. In the second phase, Cattrysse [1990] solves the reduced problem either by the SA or by the SP approaches. All the algorithms (SA, SP, FSA, FSP) are tested on 60 test problems of size varying for 3-agents and 15-jobs to 10-agents and 60-jobs. Computational results show that the SA algorithm performs badly in comparison to the SP algorithm. Furthermore, the quality of solutions and the computation time of both SA and SP are improved by the combined FSA and FSP procedures. Although, the performance of the FSA is better than that of the SA, it is still poor when compared to the FSP. On the basis of these findings, Cattrysse [1990] advocates the use of the FSP algorithm for the solution of the GAP.

Although, as indicated above, there are a few heuristics for the GAP, to our knowledge, there is no comparison made between them. We believe that, due to a weak implementation by Cattrysse [1990], the SA and FSA algorithms do not perform well. A better SA implementation is discussed in Section 2.2.7.7. There seems to be no TS approach applied to the GAP. In this chapter, we will show that our implementations of the SA and the TS metastrategy algorithms out-perform the FSA, and FSP heuristics in both solution quality as well as computation time on the same test problems.

4.2 Iterative improvement methods

An iterative improvement method has been defined in Section 2.1.6. We will summarise the method as follows: The method starts with an initial feasible solution which can be generated either by a random procedure or by the constructive heuristic of Martello & Toth [1981]. It then attempts to improve the initial solution by a series of local changes which are produced by a suitably defined neighbourhood generation mechanism. The procedure is repeated until no further improvements in the objective function can be made. Before describing our implemention of the iterative descent method for the GAP, we must define the following generic components: (i) Problem definition, a solution representation and an objective function; (ii) a generation mechanism for perturbing solutions; (iii) a selection strategy for choosing and accepting alternate solutions and (iv) Evaluation of the cost of a move; (v) a *long term* strategy to generate different starting solutions using a single constructive heuristic; (vi) a stopping criterion.

4.2.1 Problem definition and a solution representation

The generalised assignment problem is one of finding the minimum cost assignment of a set of jobs to a set of agents. Each job is assigned to exactly one agent. The total demand of each agent must not be exceeded.

A feasible solution for the GAP is represented by $S = (S_1, S_2, \ldots S_m)$ where S_i, $i = 1, \ldots, m$ are disjoint sets. Each S_i uniquely defines the set of jobs, $j \in J = \{1, \ldots, n\}$ assigned to agent $i \in I = \{1, \ldots, m\}$. The resource capacity, b_i, of each agent i must not be exceed by the total demand required from the jobs in S_i. More formally, the GAP can be formulated as follows:

$$J = \bigcup_{i=1}^{m} S_i; \qquad S_{i_1} \cap S_{i_2} = \emptyset, \quad \forall\, i_1, i_2 \in I; \qquad \sum_{j \in S_i} a_{ij} \le b_i, \quad \forall\, i \in I.$$

The objective function value of this assignment becomes:

$$C(S) = \sum_{i \in I} \sum_{j \in S_i} c_{ij} \tag{4.5}$$

4.2.2 The Martello and Toth (MT) heuristic for the GAP.

Martello & Toth [1981] propose a heuristic for the GAP which we denote as, MT. The algorithm can be described for a maximization (minimization) problem as follows: Let f_{ij} be a measure of the *desirability* of assigning job j to agent i. In an iterative manner, the MT heuristic considers all the un-assigned jobs and determines the *job*, j^*, which has the maximum difference between the largest (smallest) and the second largest (smallest) f_{ij}. The job j^* is then assigned to the agent for which f_{ij^*}, $\forall i \in I$, is a maximum (minimum). In the second part of the heuristic, the current solution is improved by a simple local exchange procedure, which checks whether an improved solution can be found by a *shift procedure*. The shift procedure considers each job and tries to reassign it to the agent with the maximum (minimum) f_{ij}.

From different f_{ij} functions are used and the best solution produced by the different f_{ij} values is considered as the MT heuristic solution. The different expression for f_{ij} are:

(a) $f_{ij} = c_{ij}$ - with this choice the second improvement phase can be skipped,

(b) $f_{ij} = \frac{c_{ij}}{a_{ij}}$,

(c) $f_{ij} = -a_{ij}$,

(d) $f_{ij} = -\frac{a_{ij}}{b_i}$.

4.2.3 λ-interchange generation mechanism

A generation mechanism defines how a given solution S is amended to generate another solution S' in its neighbourhood i.e., $S' \in N_\lambda(S)$. In this section, we will show how the λ-interchange mechanism, that is defined in Section 3.3.1 for the CCP, can be adopted and modified for the solution of the GAP.

Consider a solution $S = (S_1,\dots,S_{m_1},\dots,S_{m_2},\dots,S_m)$ for the GAP, in which S consists of sets of jobs S_i. Each S_i is assigned to agent i. A λ-interchange mechanism between a pair of job sets S_{m_1} and S_{m_2} is a replacement of a subset $\overline{S_{m_1}} \subseteq S_{m_1}$ of size $|\overline{S_{m_1}}| \leq \lambda$ with another subset $\overline{S_{m_2}} \subseteq S_{m_2}$ of size $|\overline{S_{m_2}}| \leq \lambda$, and vice versa, to get two new sets of jobs. These job sets become represented by $S_{m_1} \leftarrow (S_{m_1} - \overline{S_{m_1}}) \cup \overline{S_{m_2}}$, and $S_{m_2} \leftarrow \left(S_{m_2} - \overline{S_{m_2}}\right) \cup \overline{S_{m_1}}$. Hence, a new neighbour, $S' \equiv (S_1,\dots,S_{m_1},\dots,S_{m_2},\dots,S_m)$. is obtained. The λ-interchange neighbourhood $N_\lambda(S)$ of solution S is the family of all possible solutions S' that can be reached from S in one λ-interchange, over all combinations of pairs of job sets in S.

The effectiveness of the generation mechanism does not depend only on the generation of neighbours. It also depends on the order in which these neighbours are selected. Thus, for any iterative improvement algorithm, it is necessary to specify the order of selection of the *pairs of sets* and the order in which the λ-interchange mechanism searches the *selected pairs*.

Let the permutation, $\sigma \equiv (1,\dots,m_1,\dots,m_2,\dots m)$ be the order of agents in a given GAP solution, $S = (S_1, S_2, \dots S_m)$, where $\sigma(i) = i, \forall i = 1,2,\dots,m$. An ordered search would typically examine all possible combinations of pairs of agents (S_{m_1}, S_{m_2}) according to the order specified by the permutation σ without repetition. A total of $m(m\text{-}1)/2$

combinations of (S_{m_1}, S_{m_2}) are examined in the following order:

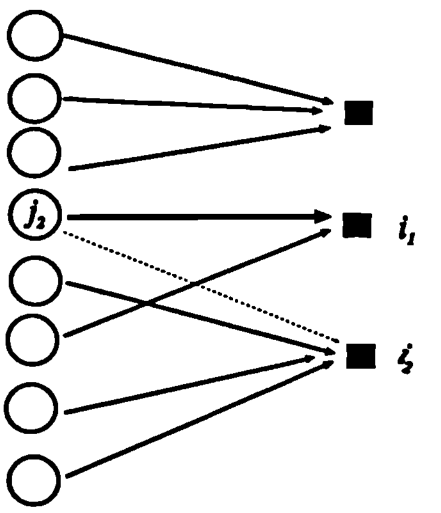
$$(S_{\sigma(1)}, S_{\sigma(2)}), \ldots, (S_{\sigma(1)}, S_{\sigma(m)}), (S_{\sigma(2)}, S_{\sigma(3)}), \ldots, (S_{\sigma(m-1)}, S_{\sigma(m)}) \qquad (4.6)$$

We define a *cycle* to be a complete search of all the neighbourhood $N_\lambda(S)$. These neighbours are generated by the λ-interchange mechanism in the order of a permutation σover all pairs of agents (S_{m_1}, S_{m_2}) without repetition. Note that, for the descent and TS algorithms, the permutation σ is of a fixed order, $\sigma(i) = i, \forall i = 1, 2, \ldots, m$ for all cycles. However, for the case of the SA algorithm, a new permutation σ $(\sigma(i) \neq i,$ for some $i = 1, \ldots, m)$ is generated at the end of each cycle. Since the SA decision on whether to accept a change or not is made after each move, in which case the sets of agents S_{m_1} are updated and the Equation 4.6 is affected.

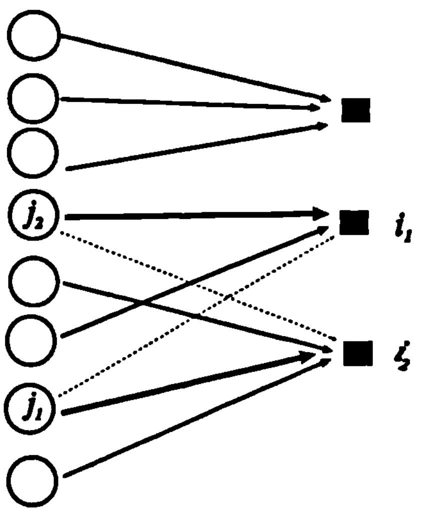
Furthermore, for a given pair (S_{m_1}, S_{m_2}), we must define the order of *search of jobs*, to be exchanged by the λ-interchange mechanism. We consider the λ-interchange with $\lambda = 1$ for the descent, SA, and TS and use $\lambda = 2$ only for the descent algorithm. There are two types of processes which are found to be successful for searching the neighbourhood of the given pair of jobs $(S_{m_1}$, and $S_{m_2})$:

(*a*) A *shift process*: the shift process is represented by the (0,1), (1,0), (0,2) and (2,0) operators. The (0,1) and (1,0) operators denote the shift of one job from one set (say S_{m_1}) to another set (say S_{m_2}). The (0,2) and (2,0) operators similarly denote the shift of two jobs from one set to another set of jobs. Figure 4.2a illustrates the shift process of a job j_2: Initially, the job j_2 is assigned to agent i_2 (this assignment is represented by a doted line). Subsequently, the shift process changes the assignment of j_2 to i_1 (the new assignment is represented by a solid line). In a given set of jobs, the shift process attempts to shift sequentially every single job in a sequential manner and checks for improved feasible solutions.

(*b*) An *interchange process*: the interchange process is represented by (1,1) operator for $\lambda = 1$, and by (1,2), (2,1) and (2,2) operators for $\lambda = 2$. We define $\overline{S_{m_1}}$ to be a subset of the set of jobs S_{m_1} ($\overline{S_{m_1}} \subseteq S_{m_1}$) of size $|\overline{S_{m_1}}| \in \{1,2\}$. Let $\overline{S_{m_2}}$ to be a subset of the set of jobs S_{m_2} ($\overline{S_{m_1}} \subseteq S_{m_1}$) of size $|\overline{S_{m_2}}| \in \{1,2\}$. The interchange process exchanges a subset of jobs $\overline{S_{m_1}}$ with another subset of jobs $\overline{S_{m_2}}$ to generate feasible solutions. Figure 4.2b shows an illustration of a (1,1) exchange process. There are two jobs j_1 and j_2 which are assigned to two agents i_1 and i_2 respectively. This assignment is represented by two dotted lines (to denote the assignment before the interchange process is carried out). The interchange process involves removing the two doted lines and adding two heavy lines (to depict the new assignment, in which job j_1 is assigned to agent i_2 and job j_2 is assigned to agent i_1). An attempt is made to interchange every job in the set S_{m_1} with another job in the set S_{m_2} in a sequential manner in a bid to obtain improved feasible solutions.



(a)



(b)

Figure 4.2 The shift and interchange processes.

The search process which we use for a given pair of sets in the case of $\lambda = 1$ is a sequence of the shift and interchange processes selected in the following operation order: (0,1), (1,0) and (1,1). After all the processes are performed, a new pair of job sets is selected until the cycle of search is completed.

4.2.4 Selection strategy of alternate solution

Given a solution S for the GAP and its neighbourhood $N_\lambda(S)$, we propose two selection strategies for choosing an alternate solution S' from $N_\lambda(S)$:

(i) The *best-improve* (BI) strategy which examines all possible feasible solutions in the neighbourhood and selects that $S' \in N_\lambda(S)$ which yields the largest improvement in the objective function.

(ii) The *first-improve* (FI) strategy which selects the first solution $S' \in N_\lambda(S)$ that satisfies the acceptance criterion.

The BI strategy uses an aggressive orientation towards a near optimal solution, possibly at a high computation time. The FI strategy follows a less aggressive orientation.

4.2.5 Long term strategy for initial solutions

A formal definition of the LT, long term strategy is given in Section 2.3.5. The main objective of the long term strategy is to generate different starting solutions using a single constructive heuristic. This strategy uses a long term memory function which collects information about the assignment of jobs to agents during the search. This information is used to penalise jobs by adding the LT memory function values to the actual costs. The adjusted costs are then used by the heuristic method to generate different starting solutions.

The *long term memory* function is represented by an $n \times m$ matrix, denoted as LTM, that records the number of times each job remains assigned to an agent during the run of an algorithm. According to these numbers, jobs are then penalised by adding the LTM values to the original cost matrix $[c_{ij}]$. The COST, new cost matrix (COST= LTM+ $[c_{ij}]$), is used to generate a new different starting solution using the MT heuristic of Section 4.2.2. The objective function value for the new solution is calculated from the original cost matrix. Each time the current solution is improved, the LTM function is updated in the following way:

Let a solution be represented by $S \equiv (S_1,S_2,\ldots,S_m)$ and let e_j be a job in S_i. Then the long term memory function is updated using Equation (4.7).

$$\mathrm{LTM}(e_j,S_i) = \mathrm{LTM}(e_j,S_i) + 1, \quad \forall\, e_j \in S_i, \quad \forall\, i \in I, \tag{4.7}$$

4.2.6 Evaluation of the cost of a move

A move is defined as a transition from a solution, S, to another solution, $S' \in N_\lambda(S)$, generated by the λ-interchange mechanism. A move is identified by attributes. Such attributes could be, for example, the index of jobs involved in a move that produced that solution or the set S_{m1} containing the above jobs. The cost of a move is the difference in the objective values of both solutions: $\Delta = C(S') - C(S)$.

Let S_i be the set of jobs assigned to agent i, and let j be the job to be added (or deleted) by the 1-interchange mechanism either produces $S'_i = S_i \cup \{j\}$ (shift add operation) or $S'_i = S_i - \{j\}$ (shift delete operation). The cost of the new assignment $C(S')$ is computed as follows:

Shift delete operation: $$C(S'_i) = C(S_i) - c_{ij} \qquad (4.8)$$

Shift add operation: $$C(S'_i) = C(S_i) + c_{ij} \qquad (4.9)$$

The cost of a move, $\Delta = C(S') - C(S)$, can be done by a simple calculation using Equations (4.8) and (4.9). Similar principles are used for the evaluation of moves generated by the 2-interchange mechanism.

4.2.7 The λ-interchange descent algorithm

The λ-interchange descent algorithm is an iterative improvement method. It starts with a solution chosen by the application of the constructive heuristic, MT. The iterative descent method selects a neighbouring solution $S' \in N_\lambda(S)$ using the 1-interchange or the 2-interchange mechanism according to the FI or BI selection strategies and with or without LT, the long term strategy. Then, it evaluates the objective function value, $C(S)$ and $C(S')$, and computes $\Delta = C(S') - C(S)$. If $\Delta < 0$, S' is accepted as the current solution. Alternatively, if $\Delta \geq 0$, S is retained. The search usually continues until a cycle of search is completed without any improvements. At this point, the algorithm stops, and declares S as the λ-optimal solution. We also use the LT strategy in the descent algorithms with the 1-interchange mechanism and the FI and BI selection strategies. The resulting algorithms are denoted by LT1+FI and LT1+BI respectively. The descent algorithms which do not uses the LT strategy are represented by 1+FI, 2+FI and 1+BI. The 2+FI descent algorithm produces its solution by starting from the final solution of the 1+FI algorithm. This is similar to what is proposed for the CCP in Section

3.3.4. The LT1+FI and LT1+BI algorithms terminate after at least three different initial solutions, generated by the LT strategy, do not improve the current best solution found during the different runs.

Let COST denote the cost matrix adjusted by the LT memory function, let S_{best} represent the best found solution and let R_{best} be the number of runs required to restart the LT strategy without improving upon the S_{best} solution.

The descent algorithm steps can be presented as follows:

```
Step 0. Initialise the LT memory function.
        Let COST= $[c_{ij}]$.
        Let $\mathrm{LTM}(j,i)=0, \quad \forall j \in J, \quad \forall i \in I$.
        Set the counter $r_{best} \leftarrow 0$, and $R_{best} \leftarrow 3$.
Step 1. Generate the MT solution, $S$, using the cost matrix, COST. Evaluate the
        initial objective function $C(S)$ using the original cost matrix, $[c_{ij}]$.
Step 2. Choose a solution $S' \in N_\lambda(S)$ according to either of the FI or BI strategy.
Step 3. If $S'$ is better than $S$ $(\Delta < 0)$,
        Then replace $S$ by $S'$.
        Else go to Step 2.
Step 4. If the neighbourhood $N_\lambda(S)$ of $S$ has been completely searched with no
        improved solutions,
        Then declare $S$ as the $\lambda-$optimal solution and go to Step 5,
        Else go to Step 1.
Step 5. If $C(S_{best}) < C(S)$ Then; Set: $S_{best} \leftarrow S$, $r_{best} \leftarrow 0$, COST= COST + LTM and
        go to Step 2;
        Else Set $r_{best} \leftarrow r_{best} + 1$.
        If $r_{best} > R_{best}$ Then go to Step 6,
        Else; Set COST= COST + LTM and go to Step 1.

Step 6. Stop the algorithm and output all the solutions.
```

4.3 Simulated annealing

4.3.1 Simulated annealing implementation

The SA metastrategy algorithm is an iterative improvement approximate method. SA attempts to overcome the disadvantage of poor local optima inherent in a local descent method, by employing random acceptance strategies. The random acceptance strategy allows occasional uphill moves to be accepted in a controlled fashion. Downhill moves are always accepted. In this sense, SA resembles the descent method. For any SA implementation, it is necessary to specify: (i) the method by which a starting solution is selected, (ii) a neighbourhood generation mechanism, (iii) how the neighbourhood is searched, (iv) a cooling schedule which, specifies an *initial* and a *final* value of the control parameter, defines a decrement function for decreasing it, indicates the number of iterations to be performed at each temperature, and defines a stopping criterion that terminates the algorithm.

In this section, we will describe a SA implementation for the GAP. In Chapter 3, we concluded that for the CCP a combination of SA with a heuristic starting solution would reduce the computation time without scarificing the quality of the solution. Thus, in this implementation, we start the SA algorithm with the MT heuristic of Section 4.2.2 and use the 1-interchange neighbourhood generation mechanism. Pairs of jobs to be searched by the 1-interchange mechanism are selected systematically according to

Equation (4.6). A new random permutation σ is generated after each cycle of search is completed. This is then used in Equation (4.6). For the GAP, we adopt the same non-monotonic cooling schedule which is introduced for the CCP in Section 3.4.2. This cooling schedule is a general approach in that the generic cooling schedule parameters, (Section 2.2.3) do not need a manual change. They are obtained automatically from the problem specific choices (Section 2.2.2) and their suggested values are explained in Section 3.4.2.

In this application, we introduce two possible avenues for exploration. The first possibility is the introduction of the LT, long term strategy to generate different initial solutions (Section 3.4.5). The LT strategy is an alternative approach to changing the initial acceptance *seed* for the random generation function, thereby enabling us to use the MT heuristic for several runs. The simulated annealing algorithm with the LT strategy is denoted by SA^{lt}. The SA algorithm which uses different seed values is represented by SA^{r}. The second possibility is a slight modification of the SA algorithm of Section 3.4.2. In this modification, the moment the temperature is reset, we also reset the current solution with the best solution found so far. In the previous scheme, only the temperature is reset and the current solution is remained as it is. The modified scheme is represented by SA^{br}

A stopping criterion can be arbitrarily chosen to reflect the amount of time to be spent on solving the problem. The stopping criterion, we used, is based on R, the number of temperature resets since the best solution is found.

4.3.2 Simulated annealing algorithm

The same definition for COST and LTM matrices is used as in case of the descent algorithm (Section 4.2.7). We also define S_{best} to be the best solution in a given run and S_{lt} to be the best solution found when the LT strategy is applied.

The steps of the SA algorithm:

(i) Initialisation:

Step 0. Initialise the long term memory function.

Let $\mathrm{LTM}(j,i)=0, \quad \forall j \in J, \quad \forall i \in I,$

Set $S_{best} \leftarrow S, S_{lt} \leftarrow S, \; r_{lt} \leftarrow 0$ and $R_{lt} \leftarrow 3$.

Step 1. Generate S an initial heuristic solution using the MT heuristic (of Section 4.2.2) using the adjusted cost matrix, COST.

Evaluate C(S), the objective function value using the original cost matrix, $[c_{ij}]$.

Step 2. Initialisation of the *SA cooling schedule parameters*:

Perform a cycle of search, $\mathrm{N}_1(S)$ without accepting neighbours to obtain $\delta_{\max}$, and $\delta_{\min}$, the largest and smallest change in objective values respectively, and also the total number of feasible solutions *Nfeas* in the neighbourhood, $\mathrm{N}_1(S)$, of S.

Set the initial, final, reset and best found temperatures respectively using $T_s \leftarrow \delta_{\max}$, $T_f \leftarrow \delta_{\min}$, $T_{reset} \leftarrow \; T_s$ and $T_{found} \leftarrow \; T_s$.

Set the other cooling schedule parameters $\alpha \leftarrow n \times Nfeas, \quad \gamma \leftarrow n, \quad R_{best} \leftarrow 3$, and the counters $k \leftarrow 1$, and $r_{best} \leftarrow 0$

(ii) Selection and acceptance of generated neighbours.

Step 3. Select a solution $S' \in N_1(S)$ in sequential and systematic search (see Section 4.2.3, and Section 4.2.4).

Compute the move value $\Delta = C(S') - C(S)$, (Section 4.2.6).

Step 4 **If** $\Delta \leq 0$, or $e^{(-\Delta/T_k)} > \theta$, where θ is a uniform random number $0 < \theta < 1$, **Then** accept S' and set $S \leftarrow S'$.

If $C(S') < C(S_{best})$ **Then** Set: $S_{best} \leftarrow S'$, $r_{best} \leftarrow 0$, and update T_{found}.

(iii) Temperature updates.

Step 5. Evaluate the temperature decrement ratio β_k as in Equation (3.11), and update temperatures according to the rule (3.12) or the conditions for occasional increase Equation (3.13) of Section 3.4.2.

If the rule (3.13) is used, **then** $r_{best} \leftarrow r_{best} + 1$.

Set $k \leftarrow k + 1$.

Step 6. **If** $r_{best} > R_{best}$ **Then**, go to Step 7,

Else go to Step 3.

Step 7. **If** $C(S_{best}) < C(S_{lt})$ **Then** Set: $S_{lt} \leftarrow S_{best}$, $r_{lt} \leftarrow 0$, COST= COST + LTM and go to Step 1;

Else; Set $r_{lt} \leftarrow r_{lt} + 1$.

If $r_{lt} > R_{lt}$ **Then** go to Step 8,

Else Set COST= COST + LTM and go to Step 1.

Step 8. Stop the SA algorithm and output all the generated solutions with the total computation time.

4.4 Tabu search

Tabu search (TS) is a metastrategy procedure. TS methodology and applications were introduced in Section 2.3. In this section, we will explain a TS implementation to solve the GAP. Tabu search shares with SA, the ability to guide local search methods in such a way as to prevent the occurrence of bad local optima. As a result, near optimal solutions may be produced. The TS algorithm seeks to transcend local optima by employing the following strategies:

(i) *Forbidding and freeing* strategies that forbid *tabu* moves from selection and free them when their tabu tenures have expired. In other words, they manage what comes in and what goes out of the list of tabu moves;

(ii) A *short term* strategy which manages the interplay between different TS strategies (see Section 2.3.7);

(iii) A *learning* strategy based on *a long term memory* function that gathers information during the algorithm run to improve the choice of an initial heuristic solution (see Section 4.2.5).

The TS algorithm follows the same basic steps of the 1-interchange descent method. One major difference is in the decision making of selecting alternate solutions. The TS algorithm examines all moves in the neighbourhood $N_1(S)$ and chooses the *best admissible* move (whether improving or not) at each iteration. This choice is guided by above strategies in order to continue the search when local optima are encountered. The TS algorithm is also different to the SA algorithm which selects and accepts moves with random strategies. The success of TS algorithm depends on the components of the above strategies which we will identify in greater detail for the GAP. Details of other possible strategies and successful applications can be found in Glover [1989a, 1989b].

4.4.1 The forbidding and freeing strategies.

The forbidding strategy constrains the search by classifying certain moves as *tabu* (or forbidden). The goal of classifying certain moves as tabu is to prevent cyclıng and to induce the exploration of new regions.

For the GAP, we define a set of attributes which characterize a move using the two pairs (e_1, S_{m_1}) and (e_2, S_{m_2}). These pairs are sufficient to identify that a job e_1 from the set S_{m_1} is interchanged with a job e_2 from the set S_{m_2}. These attributes (e_1, S_{m_1}) and (e_2, S_{m_2}) are stored in a *tabu list* and are used to check the tabu status of future moves. Thereby, providing a crude but effective way to prevent cycling. We will consider two tabu conditions based on the above attribute to identify a tabu move:

(1) If e_1 is returned to S_{m_1} ***and*** e_2 is returned to S_{m_2}. This is denoted by criterion B.

(2) If ***either*** e_1 is returned to S_{m_1} ***or*** e_2 is returned to S_{m_2}. This is denoted by criterion E.

A move is considered *tabu* if it satisfies one of the above tabu conditions depending on which condition is used in the TS algorithm. Tabu condition (2) is more restrictive than the tabu condition (1).

The set of forbidden move is recorded in a data structure for a tabu list denoted by *TABL*. The *TABL* is a matrix of size $(n+1)\times m$ (n rows one per job e_1, one extra for the *null* job involved in the (0,1) and (1,0) operators and m columns for subsets S_{m_1}). The entry, TABL(e_1, S_{m_1}) records the iteration number at which job e_1 is removed from the set S_{m_1} of jobs assigned to agent m_1. A move remains on the tabu list for a tenure of $|\,Ts\,|$ iterations, starting from the moment that the move was selected by the TS algorithm. $|\,Ts\,|$ is called the tabu list size. After $|\,Ts\,|$ iterations are lapsed, the tabu status is updated by the freeing strategy and the move is allowed to be re-selected again. By preventing the selection of a previously performed move during a sequence of $| Ts |$ iterations, the likelihood of a return to a solution that was previously obtained diminishes. The advantage of this data structure is that we can easily determine the tabu status of a given move by a simple check. This can be of great importance when the problem and the tabu list size are very large. More clearly, if we are at iteration k, a move is classified "tabu" according to condition (1) when Equation 4.10 is satisfied, i.e.,

$$k - TABL(e_1, S_{m_1}) \leq | Ts |$$

and
$$k - TABL(e_2, S_{m_2}) \leq | Ts | \qquad (4.10)$$

By storing in the tabu list, and the iteration number when a move is made, the tabu status of such a move will be updated automatically as the number of iteration increases. Hence, the move is freed from its tabu status when its tabu tenure expires. Therefore, there is no need to use any freeing functions with this data structure. This is different to the classical circular update data structure, in which the pointer returns to the beginning of the tabu list to update the status of the tabu moves each time the end is reached (FIFO procedure Section 2.3.3) to update the status of the tabu moves. The freeing strategy is needed when the circular update is used and this adds another advantage of the tabu list data structure.

4.4.2 The short term strategy.

The short term strategy is the core of any TS algorithm. It is based on a *short term memory* function. This strategy is an overall strategy which manages the interplay among all TS strategies (see Section 2.3.7). It records more information about the past moves in a flexible memory structure designed to permit the evaluation of the best *admissible* move in the neighbourhood. A move is considered admissible if it is not a tabu move or if its tabu status is overridden by *aspiration criteria*. Aspiration criteria are measures mainly designed to override tabu status of a move if a move is *good enough* and *sufficient enough* to prevent cycling. Such criteria are necessitated by our use of approximations, based on move attributes, to prevent cycling.

Tabu restrictions and aspiration criteria play a dual role in constraining and guiding the search process (Glover [1989a]). Tabu restriction allows moves to be regarded as admissible only if they do not hold, while aspiration criteria allow moves to be regarded as admissible regardless of their tabu status. This strategy also decides the selection strategy for a candidate move and a stopping criterion.

(i) Aspiration functions

Two aspiration functions are used in our implementation. The underlying motive of these functions is to demonstrate that different aspiration criteria can have different influences on the quality of solutions. Aspiration functions are based on criteria that allows a tabu move to be accepted if Equation (4.11) or Equation (4.12) are satisfied. Let S, S', S_{best} be the current, the potential and the best obtained solutions so far, respectively. Let $\Delta = C(S') - C(S)$ be the change in the objective function value of the current move. The two different criterion can be formulated as follows:

The first aspiration function which is denoted by criterion B, is:

$$C(S)+\Delta < C(S_{best}) \tag{4.11}$$

The second aspiration function which is denoted by criterion M, is:

$$C(S)+\Delta \quad < \quad \text{Min}\,\{\text{ASP}(e_1), \text{ASP}(e_2)\} \tag{4.12}$$

where, ASP(e_1), and ASP(e_2) are the objective function values of corresponding jobs involved at the time their moves have been made tabu. The aspiration level of each job, ASP, is initially set to a large value and subsequently updated. This criterion is more flexible and allow more moves to be accepted than the first criterion in Equation (4.11).

(ii) Move selection strategies

There are two strategies for move selections. At each iteration, the first strategy is the best admissible strategy which chooses the best admissible (BA) move which has the minimum (greatest improvement or least disimprovement) value in $N_1(S)$, the whole neighbourhood of the current solution, S. The corresponding TS algorithm which uses this strategy is denoted by TS+BA. The BA strategy is a generalisation of the best improve (BI) selection strategy in the descent improvement method of Section 4.2.3. However, as the neighbourhood increases with the problem size, the (BA) strategy becomes more expensive to execute and a sampling strategy that shrinks the set of admissible moves is recommended. This approach also requires more effort to store and compare the moves. The second strategy which we suggest is called the FBA strategy. This strategy accepts the first admissible move which gives a reduction in the objective value of the current solution. Then, as in the descent method, the search starts in the neighbourhood of the new solution. If we search the whole neighbourhood of the new solution without any improvements, then the best (least disimprovement) admissible move is accepted. This strategy combines the first improve strategy with the BA strategy in a more dynamic selection strategy for sampling the neighbourhood. In this strategy, we update the tabu list more frequently in regions of good solutions and less in others. This strategy is denoted by FBA and the TS algorithm which uses this strategy is denoted by TS+FBA. Further details on these strategies can be found in Section 2.3.7.

(iii) The stopping criterion, we use in the TS algorithm is based on a maximum number of iterations (*MAXI*) elapsed since the iteration at which the best solution was found. However, other alternative criteria can be used such as a total pre-specified number of iterations before terminating the algorithm.

TS algorithm steps

Step 0 Initialisation of the long term memory function LTM. This is similar to that of the descent and SA algorithms.

Step 1. Generate an initial heuristic solution S by the MT heuristic. In the case of a long term memory function where it is not possible to find different starting solutions other than those previously generated and where no random starting solutions are possible, go to Step 5. Otherwise, initialise the *short term memory function TABL, Tabu list size,* and *MAXI* iterations number.

Step 2. Choose an admissible move according to the BA or the FBA strategy to generate a new solution $S' \in N_1(S)$. Update the current and best solutions; the tabu list; the short and long term memory functions. This is pictorially illustrated in Figure 2.10 - Figure 2.11 of Section 2.3.7.

Step 3. **If** a maximum number of iterations (*MAXI*) has elapsed since the best found solution, **then** go to Step 4,

Otherwise, go to Step 2.

Step 4. **If** the TS algorithm is restarted for at least three times from different starting solutions (generated with the LT memory function) without improving upon the best tabu solution, **then** go to Step 5.

Otherwise, invoke the long term memory function. Replace the cost matrix with the penalised cost matrix as in SA algorithm and go to Step 1.

Note: For test problems from the literature, since the optimal solution is known, an additional stopping criterion terminates the algorithm as the optimum is obtained.

Step 5. Display the final TS and initial MT solutions of every start together with their computation time.

Stop.

4.5 Computational Experience

4.5.1 Test Problems

The algorithms we developed are tested on twelve sets of test problems with $m \in \{5, 8, 10\}$ and $\frac{n}{m} \in \{3, 4, 5, 6\}$ generated by Cattrysse [1990]. Every set contains five randomly generated problems, thus yielding a total of 60 test problems. For each problem, a_{ij}, c_{ij} are integers generated from the uniform distribution U[5, 15], while $b_i = \frac{0.8 \sum_{j \in J} a_{ij}}{m}$, according to Fisher *et al.* [1986].

Our algorithms are designed to solve minimization problems. However, since the data provided corresponds to maximization problems, it is necessary to effect a transformations so that comparisons are possible. The problems are known to be the more highly capacitated problems of type C. The generic notation we give for a problem is *Tm-n-i*, where *T* is the problem type, *m* is the number of agents in the problem, *n* is the number of jobs in the problem, and *i* is the index of the problem instance, (i=1,...,5). For example, a set of five problems with m=10, n=60 from Type-C is denoted as C10-60, and the *i*th problem instance from this set is represented as C10-60-*i*.

The data from Cattrysse [1990] are for maximization problems, in which the costs in a minimization problem are replaced by profits (revenues). The problems are transformed into equivalent minimization problems as follows: if p_{ij} is the profit from assigning job j to an agent i in the maximization problem of size n jobs, the costs in a corresponding minimization problem are calculated using: $c_{ij} = 35 - p_{ij}$, where 35 is an upper bound on $p_{ij}\ \forall i, j$. Then, $Z_{\max}$ the objective value of the maximization problem is given by: $Z_{\max} = 35 \times n - Z_{\min}$, where $Z_{\min}$ is the objective function value of the transformed minimization problem.

Computational results are evaluated using the ARPD (average relative percentage deviation) of solution Z_h from the optimal solution Z_{opt}, where Z_h is a heuristic solution, i.e. $ARPD = 100\frac{(Z_h - Z_{opt})}{Z_{opt}}$. The ACT (average CPU time) in seconds of the actual execution is reported excluding input and output time. The algorithms are programmed in FORTRAN 77 and run on a VAX 8600 computer.

4.5.2 Descent Methods

This section discusses the λ-interchange descent methods with $\lambda \in \{1,2\}$, and studies the effects on solution quality and computing time of employing different strategies. Computational results are listed in Table 4.1, reporting the ARPD of the best solutions from optimal ones and the ACT. We make a number of observations: Firstly, the 1+FI descent algorithm significantly improves the initial starting solutions of the MT heuristic in that the ARPD is reduced from 2.56 % to 1.42 % with only a doubling of the ACT. The 2+FI algorithm which starts from the solution of 1+FI algorithm requires eleven times more CPU seconds than that of the MT and five times more than the 1+FI algorithm with only a slightly better ARPD of 1.33 %.

From Table 4.1, we notice that the 2+FI algorithm produces improved ARPD compared to that of the 1+FI only in the case of the first half of problem sets; where the problems are of small sizes. However, both of them have exactly the same ARPD in the case of large-sized problem sets. Consequently, the 1-interchange neighbourhood mechanism is preferred as it is powerful enough to generate good solutions without the need of exploring larger neighbouring solutions thereby increasing the computational time.

Table 4.1: Average relative percentage deviation for the λ-interchange descent methods over 60 test problems.

Prob.'sets	MT	1+FI	2+FI	1+BI	LT1+FI	LT1+BI
C5-15	5.43	2.69	2.32	1.91	1.74	1.61
C5-20	5.02	1.64	1.54	1.13	0.89	0.75
C5-25	2.14	1.58	1.51	1.51	1.26	1.51
C5-30	2.35	0.72	0.66	1.00	0.72	1.01
C8-24	2.63	1.85	1.74	1.63	1.42	1.53
C8-32	1.67	1.05	0.79	1.05	0.82	0.85
C8-40	2.02	1.26	1.11	1.20	1.22	1.18
C8-48	2.45	1.13	1.13	1.11	1.13	1.11
C10-30	2.18	1.73	1.73	1.73	1.48	1.48
C10-40	1.75	1.27	1.25	1.21	1.19	1.13
C10-50	1.78	1.25	1.25	1.15	1.17	0.84
C10-60	1.37	0.88	0.88	0.88	0.81	0.83
ARPD[b]	2.56	1.42	1.33	1.29	1.15	1.15
ARPD[a]	2.56	1.42	1.33	1.29	1.38	1.34
ACT[c]	0.04	0.09	0.45	0.22	0.10	0.16
NOPT[o]	0	2	2	2	3	3

a : Average relative percentage deviation of all solutions.
b : Average relative percentage deviation of best solutions.
c : Average CPU time in seconds of one start per problem.
o : Number of optimal solutions obtained.

Secondly, the 1+BI descent algorithm has an ARPD of 1.29%. Thus, it out-performs both the 1+FI and the 2+FI algorithms in solution quality. However, it has an ACT which is double that of the former and a half that of the latter. To investigate this issue further, we embedded the two algorithms, the 1+FI and 1+BI into a long term LT strategy discussed in Section 4.4.2 so that we could generate different starting solutions using the MT heuristic. We find that the LT1+FI and LT1+BI algorithms produce the same number of optimal solutions, NOPT, the same ARPD of the best solutions, and are comparable in the ARPD of all solutions generated. However, the LT1+FI requires a total of 18.5 CPU seconds while the LT1+BI needs a total of 28 CPU seconds. Hence, the LT1+FI, which follows a less aggressive orientation in the search for an improved solution than the LT1+BI is recommended for large-sized problems, as it requires less ACT.

4.5.3 Simulated Annealing

Although, our SA algorithm is generally flexible and robust, the parameters α and γ need a little tuning and a careful choice when applied to new problems. The values of α and γ are set after an analysis with respect to quality of solution and CPU time. The SA algorithm is executed by starting with an initial solution from the MT heuristic. The random generator is varied with different seed values at the beginning of each run, so that we obtain different final solutions. The SA is restarted at least three times and the algorithm is stopped when there is no improvement on the best solution obtained during the restart process.

Computational results for the same test problems are reported in Table 4.2. The SA algorithm gives the user control over balancing the trade-off between running time and solution quality through the number of temperature resets parameter R. This is illustrated by the ARPD improvements and the number of optimal solutions found, as R increases. In particular, by changing the value of R from 1 to 3, the ARPD^b drops from 0.11% to 0.04% (an improvement of 63%) at an increase of 45% in the total computation time (since we know that the ARPD of MT starting solution is 2.56%). Consequently, NOPT, the number of optimal solutions rises from 30 to 39 out of 60.

We tested two more schemes. The first, SA^{br}, restarts from the best solution obtained each time the temperature is reset. Results are not as good as SA^r for $R = 3$. This can be explained by the fact that we are returning to local minimum rather than carrying on from where we were in the previous scheme. The second scheme is SA^{lt}, which invokes the LT strategy. In this respect, this scheme is similar to tabu search. The results from this scheme are better than those of the SA^{br} scheme. However, they are not better than that of SA^{r} scheme in term of the best ARPD and in the number of optimal solutions found. This is because the SA^{lt} algorithm stops earlier than the SA^{r} algorithm as it is not able to generate different starting solutions.

Table 4.2. Average relative percentage deviations of SA algorithms.

	SA^{r}			SA^{br}	SA^{lt}
R	1	2	3	3	3
$ARPD^{w}$	0.56	0.46	0.40	0.48	0.36
$ARPD^{b}$	0.11	0.07	0.04	0.11	0.10
$ARPD^{a}$	0.32	0.26	0.21	0.29	0.22
$NOPT^{o}$	30	35	39	29	31
$RUNS^{t}$	219	193	177	211	148
ACT^{c}	7.91	12.14	14.26	16.11	14.56

br : SA algorithm with a scheme that starts from best solution found when temperature is reset.
lt : SA algorithm with long term strategy.
r : SA algorithm with different seed values for restarts with a temperature reset only.
t : Total number of runs executed by SA to solve the 60 problems.
w : Average relative percentage deviation of worst solutions from optimum.
: Other legends are the same as in Table 4.1.

Cattrysse [1990] suggested SA algorithms for the GAP (The best of them called FSA) in which the temperature is held fixed for a number of potential randomly chosen moves before being dropped. Thus, the cooling takes place in a series of stages (stepwise temperature reduction scheme). In addition, the algorithm crosses infeasible regions by accepting infeasible solutions during the search process. The FSA algorithm which utilises a reduction heuristic in the initial stage produces a best ARPD of 0.72 % at a much higher computation time - twenty times more of ACT when compared with our quickest SA scheme (R=1) in which the ARPD of worst solution is 0.56%. The FSA algorithm also utilises ten times more ACT than that of the SA scheme with R=3 where the ARPD of the worst and best solutions are 0.40 % and 0.04 % respectively. The FSA algorithm performs poorly due to the following reasons: (a) the cooling schedule which is classified as stepwise reduction type - we have shown in Osman & Christofides [1989] we show the superiority of this cooling schedule over such types of schedules; (b) the random selection might miss good solutions; (c) the infeasibility permission requires more time for checking. Full results of these comparisons are presented in Section 4.6 where we compare the different algorithms.

4.5.4 Tabu Search

This subsection addresses three aspects concerning the choice of TS parameters. First, the need for tabu restrictions and the effects of tabu list size are illustrated on the problem C5-20-4. The initial solution from the MT heuristic for the problem is 400. Note that, we are dealing with data of a maximization problem. The TS+FBA algorithm is applied to the MT solution with a tabu list size $|\,Ts\,| = 0$ and a $MAXI = 80$. The results are plotted in Figure 4.2 and show cycling in which the solution value oscillates in the range of 409 to 414. However, if the $|\,Ts\,|$ is chosen reasonably well, the optimal solution would be obtained. Figure 4.3 shows the progress of the TS+FBA algorithm for this problem with $|\,Ts\,| = \left\lceil \frac{20}{4} \right\rceil = 6$ and $MAXI = 4 \times n = 80$. The optimal solution of 419 is obtained first at iteration 104 and is repeated at later iterations as indicated. We observe that, a small tabu list size will result in cycling, while a larger tabu list size could drive the search process away from the global optimum. This would, thereby, require more iterations in addition to the need of other sophisticated corrective techniques to achieve good solutions. The effect of a large tabu list size is illustrated in Figure 4.4, in which $|\,Ts\,| = n = 20$ and $MAXI = 80$. The TS+FBA algorithm was embedded into a long term memory function so that the optimal solution is attained only in the third run.

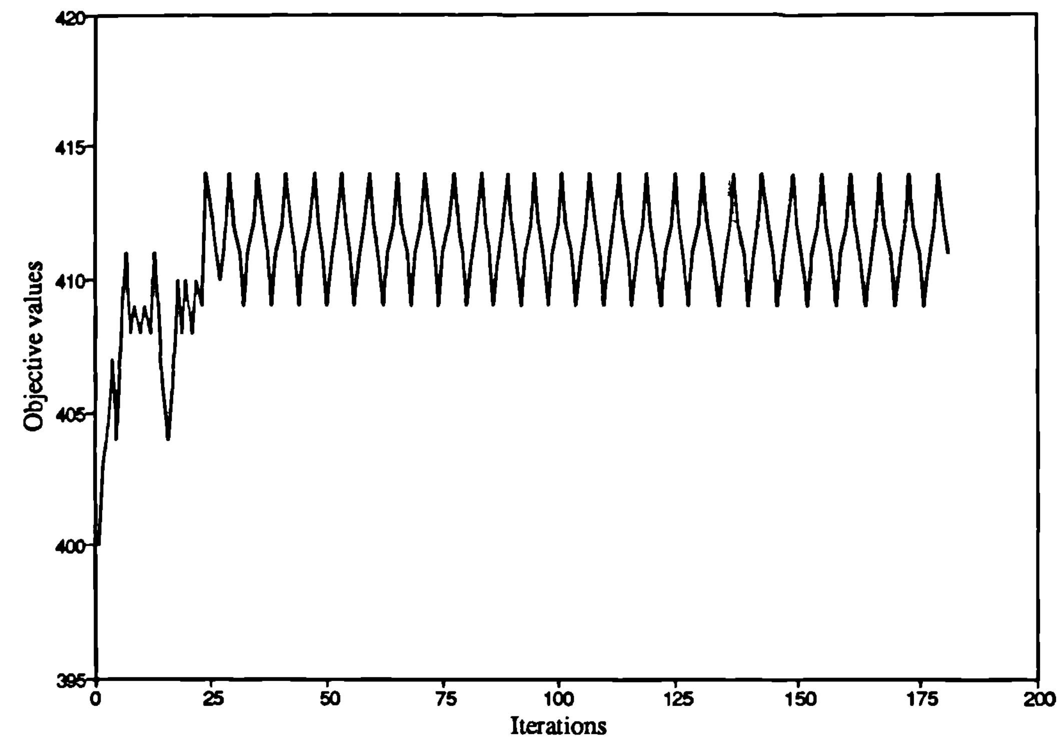


Figure 4.2. The TS+FBA algorithm with a tabu list size | *Ts* |= 0.

Figure 4.4 also illustrates the effect of the LT memory function which enables us to generate different initial solutions by the same constructive heuristic. The TS+FBA algorithm with a LT strategy starts with an objective value of 400 in the first run and terminates with a near optimal best value of 416. In the second run, another initial value of 406 was generated by the MT heuristic. Again, the algorithm fails to find the optimal solution due to the large tabu list size. In the third run of the algorithm, the optimal solution of 419 is obtained with an initial starting solution of 404. A compromise has to be reached and experiments are conducted to find out a reasonable range of the tabu list sizes for different tabu conditions. This is discused at a later stage in this section.

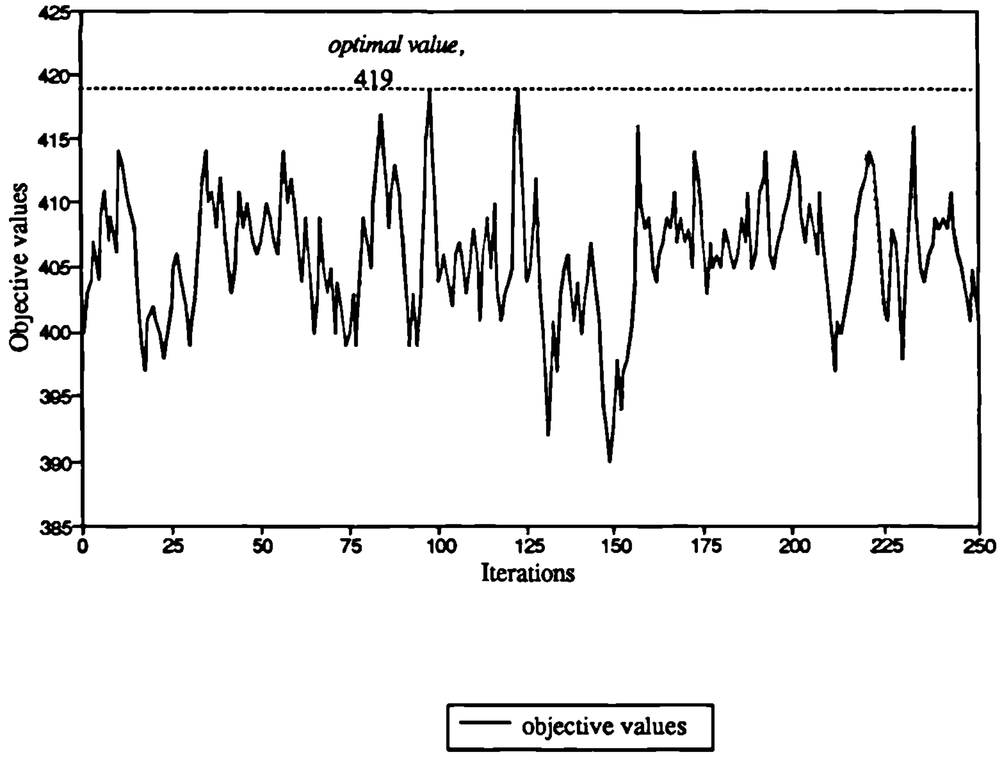


Figure 4.3. The TS+FBA algorithm with a tabu list size, | *Ts* |= 6.

We experimented to identify the best selection strategy to be used. An FBA selection strategy that provides a dynamic detection of regions for search intensification and diversifications is compared to the classical BA selection strategy (discussed in Section 4.4.2.). Both strategies are embedded into two long term TS algorithms, which use the same tabu restrictions, aspiration level and stopping criterion of $MAXI$=4×n. A move is considered tabu if both attributes are tabu. Its tabu status is overridden if the objective value of the new solution is less than the best objective value (Equation 4.11). Computational results are listed in Table 4.3 for different values of| *Ts* |. We observe that for all values of | *Ts* | the TS+FBA algorithm yields better quality solutions in less ACT than the TS+BA algorithm. The best ARPD[b] of the TS+FBA with | *Ts* | $=\lceil \frac{n}{4} \rceil$ (integer part of $\frac{n}{4}$ plus 1) is 50 % smaller at 17 % less ACT than that of the best of the

TS+BA algorithm with $|Ts| = \lceil \frac{n}{5} \rceil$. Moreover, the quality of the TS+FBA solutions seems to be relatively stable and show only slight deteriorations when used in conjunction with smaller or larger values of $|Ts|$.

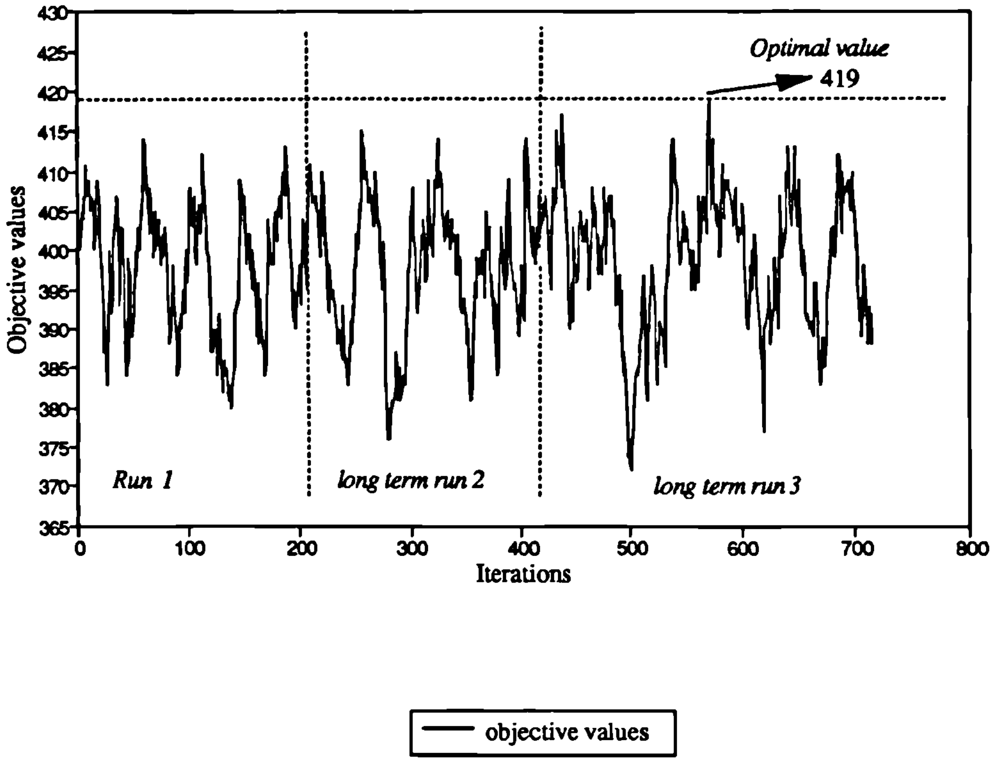


Figure 4.4. The TS+FBA algorithm with an embedded long term strategy, and a tabu list size, $|Ts| = 20$.

Second, we also investigate the dual role of tabu restriction and aspiration level in constraining and guiding the search. Computational results in Table 4.4 demonstrate that tabu restrictions must be coupled with the right aspiration level to serve their purposes. We notice that the algorithm with BB combinations (both of move attributes are tabu using criterion B and aspiration criterion of using criterion B of Equation 4.11) provides the best results when compared to those produced by other algorithms, which range from the very bad results of the BM scheme (Both of move attributes are tabu and the most flexible aspiration criterion E) to the most restrictive EB scheme that imposes a stronger tabu restriction using criterion E. The BB scheme also better than the BN scheme (tabu condition using criterion B with no aspiration level is used).

Table 4.3. Computational results comparing long term TS+FBA algorithms against the TS+BA algorithm using different tabu sizes $\lceil . \rceil$ and MAXI= 4 × *n*.

$\lvert Ts \rvert$	$\lceil \frac{n}{6} \rceil$		$\lceil \frac{n}{5} \rceil$		$\lceil \frac{n}{4} \rceil$		$\lceil \frac{n}{3} \rceil$		$\lceil \frac{n}{2} \rceil$	
TS	FBA	BA	FBA	BA	FBA	BA	FBA	BA	FBA	BA
$ARPD_w$	0.12	0.21	0.11	0.13	0.09	0.16	0.13	0.21	0.18	0.23
$ARPD^b$	0.05	0.06	0.07	0.06	0.03	0.07	0.07	0.09	0.07	0.08
$ARPD^a$	0.09	0.13	0.09	0.10	0.07	0.12	0.10	0.15	0.12	0.15
$NOPT^o$	40	38	40	40	45	40	36	34	33	32
$RUNS^t$	118	114	105	116	122	117	115	119	139	137
ACT^c	14.19	16.30	12.17	15.66	12.94	16.46	12.47	14.01	11.97	14.09

FBA: Tabu search scheme with the combined first and best admissible move strategies
BA: Tabu search with the best admissible move strategy only.
: All legends are the same as in previous tables.

From Table 4.4 we observe that, by using the long term strategy in the LT+BB scheme, the solution quality has improved upon the scheme MT+BB, which is a single run that does not invoke the long term memory function. The number of optimal solutions found has risen from 32 to 45 and the ARPD of best solutions has dropped from 0.08% to 0.03% with a percentage improvement of 62% obtained at an extra increase of 80% in the total CPU time and 11% less of the ACT per problem. We therefore observed that the TS algorithm with LT, FBA and BB strategies is the best. Figure 4.5 is a pictorial representation of the changes in the ARPD and the ACT as the *MAXI* increases in the stopping criterion. From this figure, we draw two conclusions: The first is that the ARPD of best solution decreases in a sequence, of 27%, 37%, 40%, 13%, with increases in the ACT of 92%, 55%, 29%, 19% as the values of *MAXI* varies from 1×*n* to 5×*n*. The percentage improvements of ARPD[b] reaches a peak of 40% with only 29% increases in the corresponding computer time and then declines to 13% with 19% increases in the ACT. This gives us an indication of a good estimate value of MAXI= 4x*n*. This is because, beyond *MAXI*=4x*n*, there are little improvements to justify the extra increase in the ACT.

Table 4.4. Average relative percentage deviations of the TS+FBA algorithm using different tabu restrictions, aspiration levels, with MAXI= 4×*n* and | *Ts* |= $\lceil \frac{n}{4} \rceil$.

	LT+TS				MT+TS
	BB	BM	BN	EB	BB
ARPD[w]	0.09	1.56	0.12	0.36	0.08
ARPD[b]	0.03	1.09	0.05	0.16	0.08
ARPD[a]	0.07	1.30	0.08	0.26	0.08
NOPT[o]	45	5	39	27	32
RUNS[t]	122	155	103	151	60
ACT[c]	12.94	7.04	13.49	11.17	14.66

BB : Tabu conditions using criterion B; and aspiration criterion B (in Equation 4.11)
BM : Tabu conditions using criterion B; and aspiration criterion M (in Equation 4.12)
BN : Tabu conditions using criterion B; and No aspiration criterion is used.
EB : Tabu conditions using criterion E; and aspiration criterion B (in Equation 4.11)
: Other legends are the same as in previous tables.

The second conclusion is that the error bars that represent the differences between the ARPD[w] and ARPD[b] decrease as the value of MAXI of increases and level off at a value of MAXI= 4×*n*, 5×*n*. This reflects the stability of TS if it is let to run with a reasonable estimate of its parameters.

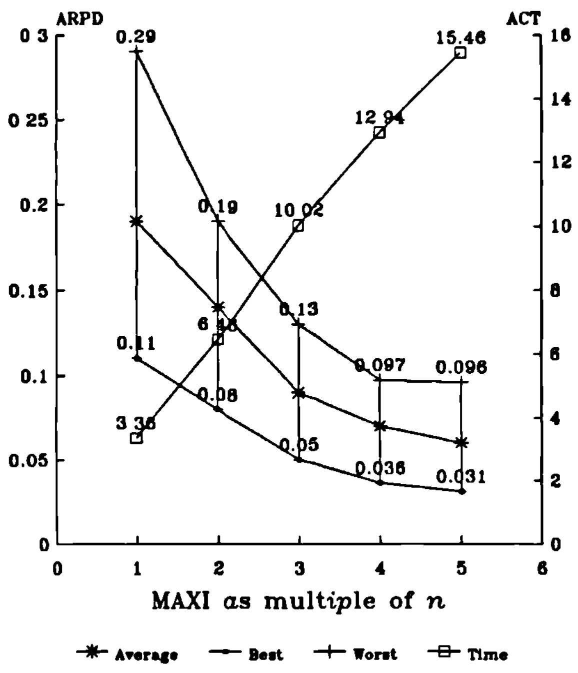


Figure 4.5. Effects of stopping values, *MAXI*, on the quality solution and computing time using the LT+FBA+BB tabu search algorithm with $| Ts | = \lceil \frac{n}{4} \rceil$

4.5.5 Comparison of algorithms

Having discussed and found the best parameters for the descent, SA and TS algorithms in previous sections, we now compare their performances against one another and with FSP, the set partitioning heuristic and the FSA, simulated annealing of Cattrysse [1990] and with MAM, the tree search with multiplier adjustment method of Fisher *et al.* [1986]. Table 4.5 shows the ARPDs and ACTs for the relevant algorithms. We observe that, LT1+FI generates better quality solutions than MT at a reasonably extra time. However, the LT1+FI algorithm has the advantage of requiring much smaller computation times when compared to other more sophisticated algorithms such as the SA or TS algorithms. The LT1+FI descent method could only be used when computation time is a limiting resource.

Table 4.5. Average relative percentage deviation ARPD and average computing time ACT for most of discussed algorithms.

Prob.'sets	MT	LT1+FI	2+FI	FSA	SA	FSP	BA	FBA	MAM
C5-15	5.43	1.74	2.32	0.00	0.00	0.00	0.00	0.00	0.00^{+0}
C5-20	5.02	0.89	1.54	0.19	0.00	0.19	0.24	0.10	0.00^{+1}
C5-25	2.14	1.26	1.51	0.00	0.00	0.03	0.03	0.00	0.00^{+1}
C5-30	2.35	0.72	0.66	0.06	0.00	0.21	0.03	0.03	0.83^{+3}
C8-24	2.63	1.42	1.74	0.11	0.00	0.07	0.04	0.00	0.07^{+1}
C8-32	1.67	0.82	0.79	0.85	0.05	0.05	0.00	0.03	0.58^{+5}
C8-40	2.02	1.22	1.11	0.99	0.02	0.00	0.02	0.00	1.58^{+5}
C8-48	2.45	1.13	1.13	0.41	0.10	0.19	0.14	0.09	2.48^{+5}
C10-30	2.18	1.48	1.73	1.46	0.08	0.14	0.06	0.06	0.61^{+4}
C10-40	1.75	1.19	1.25	1.72	0.14	0.19	0.15	0.08	1.29^{+5}
C10-50	1.78	1.17	1.25	1.10	0.05	0.00	0.02	0.02	1.32^{+4}
C10-60	1.37	0.81	0.88	1.68	0.11	0.03	0.07	0.04	1.37^{+5}
ARPD[b]	2.56	1.15	1.33	0.72	0.04	0.09	0.06	0.03	0.84
ARPD[a]	2.56	1.15	1.33	0.72	0.21	0.09	0.10	0.07	0.84
ARPD[w]	2.56	1.38	1.33	0.72	0.40	0.09	0.13	0.09	0.84
ACT[c]	0.04	0.10	0.45	520*	14.26	68*	15.66	12.94	86
Adj. ACT	0.04	0.10	0.45	157.57	14.26	20.60	15.66	12.94	86

* : ACT in seconds on an IBM Model 80 PS/2 with Math. coprocessor 386, based on benchmark runs, the VAX 8600 is about 3.3 times faster than the IBM. To obtain comparable times entries under FSA and FSP are divided by 3.3 to get the adjusted ACT.

+p : Number of problems unsolved within the time limit of 250, 750, 1000 seconds for $m = 5$; 8; 10 respectively

BA : Tabu search BA algorithm with tabu size= $\left\lceil \frac{n}{5} \right\rceil$ and $MAXI = 4 \times n$

FBA : Tabu search FBA algorithm with tabu size= $\left\lceil \frac{n}{4} \right\rceil$ and $MAXI = 4 \times n$

FSA : Fixing-simulated-annealing algorithm of Cattrysse [1990]

FSP : Fixing-set-partitioning algorithm of Cattrysse [1990]

MAM : Tree search with multiplier adjustment method of fisher *et al.* [1986]

MT : Martello & Toth [1981] constructive heuristic

LT1+FI : Long term strategy embeded with the 1-interchange descent heuristic

2+FI : The 2-interchange descent heuristic

SA : Our simulated annealing denoted SA^r with number of resets R=3

Our SA algorithm produces better ARPDs than that of the FSA and the MAM algorithms in a total of 2524 seconds compared to 9454 seconds for the FSA and 5160 for the MAM algorithm. Note that the average and total times of MAM are based only on the average computation time of those problems which are solved optimally. We exclude the time limit terminations which transform the tree search into a heuristic. The

SA algorithm also has an ARPD^b of 0.04% which is better than the corresponding value of 0.09% produced by FSP, the set partitioning heuristic. Further, the FSP has a total computation time of 1236 seconds, which is 44% more than that of the SA algorithm.

Our final tests compare tabu search algorithms with the SA and FSP algorithms. The ARPD's of the TS+FBA algorithm are better than the ARPD's of the SA algorithm. In addition, the TS+FBA is 10% faster than the SA algorithm with respect to the ACT. The TS algorithm produces more stable results due to the small gap of 0.06 between the worst and the ARPDs compared to the SA algorithm, which has a relatively huge gap of 0.36. Simulated annealing solution quality deteriorates as the problem size increases while the Ts exhibits a steady behaviour in this respect. Tabu search is also better than the FSP with respect to quality of solutions and computation time. The best and worst ARPD of the TS+FSA algorithm are 0.03% and 0.09% respectively compared to an ARPD of 0.09% of the FSP. Moreover, the average computation time ACT of the TS+FSA is equal to 60% of the ACT of the FSP algorithm.

4.6 Conclusion

In this study, we have developed λ-interchange descent methods, metastrategy simulated annealing and tabu search algorithms for the generalised assignment problem. We compared their performance with other existing methods with respect to solution quality and computation time. The major conclusions of this study are as follows:

1. The constructive method of Martello & Toth [1981], MT, generates solutions which deviate significantly (2.56%) from the optimum. Further, these solutions were improved to 1.42% by the introduction of λ-interchange neighbourhood generation mechanism into descent methods. Due to the power of the neigh-

bourhood mechanism, we find a value of $\lambda = 1$ is sufficient to produce good solutions. Increasing the value of λ would require more computation time with little improvement. Finally, the first-improve selection strategy produces the same quality solution of 1.15% as the best-improve strategy at half the computation time when both are incorporated into algorithms that use long term memory functions.

2. Our simulated annealing SA gives the user control over the trade-off between running time and solution quality through the number of temperature resets. The SA algorithm always finds better or equal solutions as the time increases holding constant other cooling schedule parameters. However, this obviously most important is not provided by many other SA schemes. Our SA that uses a random permutation of the set of agents with a systematic search, out-performs the FSA algorithm of Cattrysse [1990] in quality of solution and computation time. It also out-performs MAM, the tree search heuristic with a modified multiplier adjustment method both in solution quality and computation time. The SA algorithm produces better solutions than FSP, the set partitioning heuristic but a slightly higher computation time.
3. The tabu search algorithms, TS+FBA and the TS+FBA with the LT memory function (LT+TS+FBA) fairly consistently out-performs the SA algorithm and the FSP algorithm with respect to execution time and solution quality. In addition, its solution quality is much more stable than the SA algorithm where a large variation in the quality of solutions is found by different runs. The dynamic selection in the LT+TS+FBA algorithm provides an automatic sampling of the variable-sized neighbourhood so that many neighbourhoods can be examined in good regions. This is not the case in the classical LT+TS+BA algorithm. Tabu search conditions and aspiration levels play a dual role and must be chosen adequately. Tabu size is found to be a function of problem sizes but does not have

a major effect if its value is within a certain range. Tabu search algorithms share with simulated annealing the trade off between quality of solution and computation time thereby giving the user an opportunity to optimize his/her implementation according to the computing budget.

In both the TS algorithms all the entire neighbourhood of a current solution, generated by $m(m-1)/2$ combinations of sets of agents, is re-evaluated after any acceptance, while only $2 \times m$ sets are affected. Hence for $m \geq 4$, we can always save computational effort, by storing the unchanged sets in an additional data structure so that these move will be used if necessary in the subsequent iteration, without the need of move re-evaluation. This extra data structure is implemented in the next Chapter (Section 5.3.2). It has produced a further reduction in the CPU time especially for large problems without sacrificing solution quality. It is worth investigating the strategic oscillation and different candidate list strategies suggested by Glover [1989c]. In these infeasible solutions are accepted with penalties added to the objective function value. This would help in oscillating between feasible and infeasible regions, with the hypothesis that the optimal solution would be found quickly. Further, with recent rapid developments in computer technology, we feel that parallel versions of SA and TS further enhance the performances of our SA and TS implementations. In conclusion, the findings in this study are supported by our results obtained from other applications; in particular, the vehicle routing problem and the capacitated clustering problem and the maximum planar graph.

Chapter 5

SIMULATED ANNEALING AND TABU SEARCH FOR THE VEHICLE ROUTING PROBLEM

5.0 Introduction

The vehicle routing problem (VRP) involves the design of minimum cost delivery routes for a fleet of vehicles, originating and terminating at a central depot, which services a set of customers. Each customer is supplied by exactly one route. The total demand of the customers on each trip must not exceed the vehicle capacity. A pictorial representation of the problem is given in Figure 5.1.

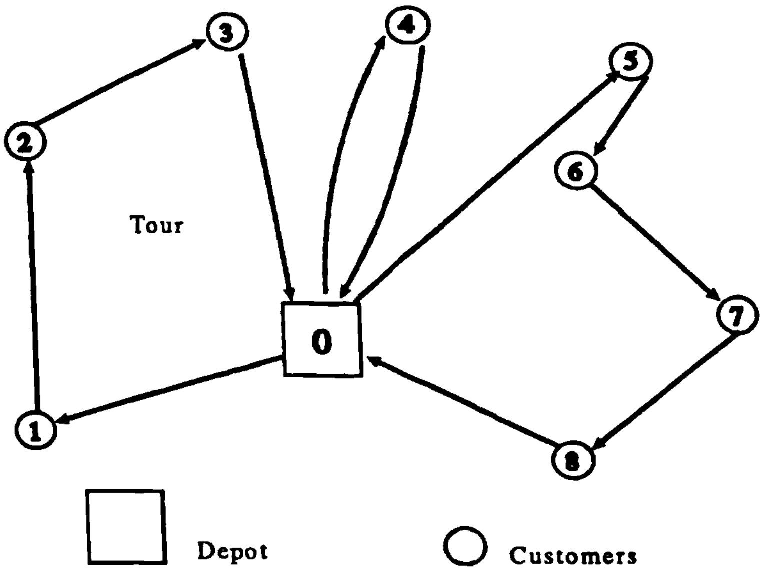


Figure 5.1. The vehicle routing problem.

The VRP is in an extremely active research area that has seen an exciting interplay between theory and practice. In the last decade, the VRP stands out as *one of the great success stories of operations research.* If vehicle routing does constitute a major success story, a share in this must be contributed to effective modelling and efficient methods for handling physical distribution problems as well as to the intriguing nature of the underlying combinatorial optimization models.

Practical applications of the VRP are numerous, and many companies and organizations have achieved major economic benefits by implementing vehicle routing algorithms to control daily fleet operations. Brown *et al.* [1981] reduced the transportation costs of petroleum products for Chevron by 13%; Fisher *et al.* [1982] developed a system for Du Pont, that resulted in a 15% reduction in delivery costs; Bell *et al.* [1983] implemented a distribution of industrial gases system for Air Products and Chemicals Inc. that helped to produce savings of 6-10% in operating costs; Evans *et al.* [1985] reported savings of 10.7% in food service delivery for Kraft, Inc.; Golden *et.al.* [1987] implemented a system in the soft drink industry. An extensive list of other applications has been identified by practitioners such as school-bus routing and scheduling, design of dial-a-ride systems, collection of mail (coins) from mail (telephone) boxes, etc. are only a few of the many applications arising in practice. The interested reader may consult a number of useful surveys of the fields including Christofides [1985], Golden *et al.* [1988].

Operational Researchers's interest in the VRP's is due in part to their practical importance, but also to their intrinsic difficulties. The VRP is a generalisation of the travelling salesman problem (TSP) - easy to describe but difficult to solve. It belongs to the class of *NP-hard* problem Lenstra *et. al.* [1981]. Computational complexity theory has provided strong evidence that any optimization algorithm for the VRP's solution is likely to require a number of computations that grows exponentially with problem size. Hence, there have been few attempts to solve the VRP optimally, notably by branch and bound procedures adopted by Christofides *et al.* [1981a, 1981b]. The bounds for the tree search were obtained from the shortest path, spanning tree or state space relaxations of the VRP. Laporte *et al.* [1985] reported exact algorithms which are based on a TSP formulation involving sub-tour breaking constraints and on cutting plane techniques; the largest problems that were solved contained 50 and 60 customers in the Euclidean and Non-Euclidean case respectively. However, the problems were of a special simple structure. Agarwal *et al.* [1989] proposed an exact algorithm based on a set partitioning formulation. Although, these approaches can address small VRP's adequately, there does not appear to be any way of incorporating other real-world constraints within these approaches. The largest general VRP that could be solved optimally is of size up to 50 customers with 8 vehicles, Christofides [1985]. Laporte *et al.* [1987] provide an excellent review of exact methods.

Although the VRP is difficult to solve in an optimizing sense, it is solved in an operational sense. The world's economies could not operate if it were not for the fact that VRP's and their extended variations have readily available practical solutions. Researchers have recognized early the importance of approximate methods (or heuristics) which can provide near optimal solutions for large sized problems. Research efforts have resulted in the development of a large number of heuristic algorithms which are able to solve the VRP of several hundred customers. Some of the well-known heuristic approaches are classified as follows:

Insertion or savings heuristics, Clarke & Wright [1964], Gaskell [1967], Mole *et al.* [1976], Paessens [1988];

Two-step methods, Gillett *et al.* [1974], Christofides *et al.* [1979], Fisher *et al.* [1981], Beasley [1983];

Improvement methods, Christofides *et al.* [1969], Russel [1977];

Exact but incomplete tree search methods, Christofides *et al.* [1979].

Useful surveys and other classifications of heuristics are given in Paessens [1988] and Bodin *et al.* [1983], Golden *et al.* [1986].

The aim of this chapter is to develop, and investigate the algorithmic performance of the metastrategy SA and TS in a new area, namely the vehicle routing problem under capacity and distance constraints. This Chapter is organised as follows: Section 5.1 introduces the definition of the VRP problem with these constraints. Section 5.2 discusses aspects of iterative improvement methods and introduces descent λ-interchange algorithms based on *first-improve and best-improve* selection criteria of neighbours, as well as introducing different move cost evaluations. Section 5.3 presents SA algorithm methodology and discusses its implementation based on the cooling schedule in Osman and Christofides [1989], (see Section 3.4.2). Section 5.4 describes details of TS metastrategy aspects and different implementations based on a new dynamic tabu list, and a new data structure that has resulted in a substantial speed up of the algorithms. Computational experiences are given in Section 5.5, comparing TS, SA and Descent algorithms against the best published results for all bench-mark problems reported in the literature, in addition to some new randomly generated problems. Section 5.6 contains some concluding remarks.

5.1 Problem definition and solution representation

The vehicle routing problem under capacity and distance restrictions is defined as follows: Let the customers be indexed $i=1,...,n$ and $i=0$ refers to the depot. The vehicles are indexed $k=1, ..., v$. A customer i has a demand of q_i and a service time of δ_i. We assume that $q_0=0$, $\delta_0=0$. The travel time (distance) between customers i and j is

c_{ij}. The total demand of vehicle route k may not exceed the vehicle capacity Q_k with $Q = Q_1 = Q_2 = \ldots = Q_v$. The length (duration) of vehicle route k which is made up of inter-customer travel times and service times (δ_i) at each customer i included on the route must not exceed a pre-specified bound L.

Solution representation

A feasible solution for the VRP is represented by $S = (S_1, S_2, \ldots S_v)$, where each S_k defines uniquely the subset of customers of $N = \{1, \ldots, n\}$ routed by a vehicle k from the set of vehicles $V = \{1, \ldots, v\}$ such that,

$$\bigcup_{k=1}^{v} S_k = N, \qquad S_{k_1} \cap S_{k_2} = \emptyset, \quad \forall k_1, k_2 \in V,$$

$$C(S_k) = \sum_{j \in S_k \cup \{0\}} (c_{j\pi(j)} + \delta_j) \le L, \qquad \sum_{i \in S_k} q_i \le Q, \quad \forall k \in V,$$

$$C(S) = \sum_{k \in V} C(S_k) \tag{5.1}$$

$C(S_k)$ is the length of an optimal (or approximate) TSP tour over the set of customers in $S_k \cup \{0\}$ delivered by a route k. This tour length must be less than a given bound L. It consists of the travel distances of the route given by the permutation π and the service times of customers in S_k. $C(S)$ is the total length of all routes.

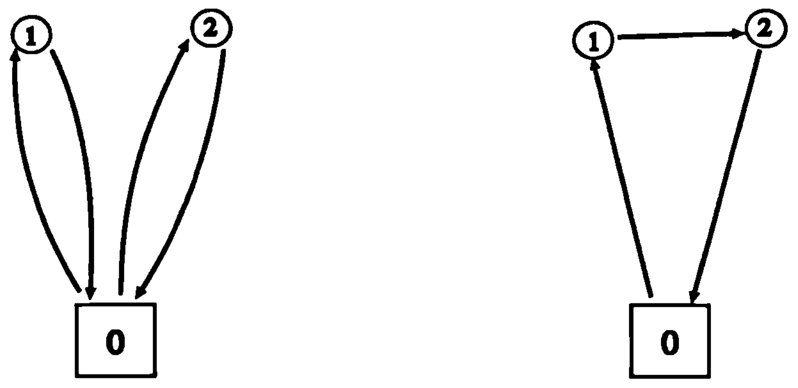
5.2 Iterative improvement methods

An iterative improvement method has a strong relationship with the metastrategy SA and TS. Hence, a number of common aspects must be elaborated in more detail, as they play a major role in the success of the above metastrategy algorithms. Most iterative improvement methods invoke the successive application of two modules: a construction method that produces a feasible initial solution S with a cost value $C(S)$, and an improvement technique that maintains feasibility whilst reducing the cost iteratively. The latter consists of: a generation mechanism to alter this initial solution; selection strategies of alternate solutions; and a stopping criterion (see Section 2.1.6).

5.2.1 An initial solution (saving method)

An initial solution can be obtained by the savings procedure of Clarke & Wright [1964]. It is the most widely known heuristic for the VRP. A survey of its variations can be found in Paessens [1988]. At each step, the procedure exchanges one set of routes with a better set. Initially, we suppose that every customer is supplied by a separate route (Figure 5.2 a).

$$S_{ij} = (2c_{0i} + 2c_{0j}) - (c_{0i} + c_{ij} + c_{0j}) = c_{0i} + c_{0j} - c_{ij} \tag{5.2}$$



Initial routes 1 and 2. A single saving route of routes 1 and 2.

(a) (b)

Figure 5.2. Saving routes

Instead of using two vehicles to service customers *i* and *j*, if we use one only, then we obtain a savings of S_{ij} in the travel distance. The savings are given by Equation (5.2) assuming symmetric distances. An illustration is shown for two customers in Figure 5.2b. We order these savings from the largest to smallest and starting from the top of the list. We link nodes *i* and *j* with the maximum savings unless the problem constraints are violated.

The saving heuristic steps can be summarised in the followings:

Step 1. Compute the savings $S_{ij} = c_{0i} + c_{0j} - c_{ij}$ for all pairs of customers i and j.

Step 2. Order the savings in descending order.

Step 3. Starting at the top of the list and moving downwards and do the following:

(i) Find the first feasible link in the list, which can be used to extend one of the two ends of the currently constructed route.

(ii) If the route cannot be expanded due to the constraints, or no route exists, choose the first feasible link in the list to start a new route.

(iii) Repeat (i) and (ii) until no more links can be chosen. At the end of the procedure, the connected links are picked to form a feasible solution for the VRP.

5.2.2 The λ-interchange generation mechanism

The generation mechanism, we use for the VRP, is already defined in Section 3.3.1. It is one essential component of the generic choices of any iterative improvement algorithms. In this section, we only give an illustration on how the generation mechanism would work using the VRP specific choices. A natural neighbourhood generation mechanism for the problem is as follows:

Given a solution $S \equiv (S_1, S_2, \ldots, S_v)$ for the VRP, that consists of the sets of routes S_i. A λ-interchange between a pair of route sets S_{k_1} and S_{k_2} is a replacement of a subset $\overline{S_{k_1}} \subseteq S_{k_1}$ of size $|\overline{S_{k_1}}| \leq \lambda$ with another subset $\overline{S_{k_2}} \subseteq S_{k_2}$ of size $|\overline{S_{k_2}}| \leq \lambda$, and vice versa, to get two new route sets. These sets become represented as $S_{k_1} \leftarrow \left(S_{k_1} - \overline{S_{k_1}}\right) \cup \overline{S_{k_2}}$, $S_{k_2} \leftarrow \left(S_{k_2} - \overline{S_{k_2}}\right) \cup \overline{S_{k_1}}$. Hence, a new neighbouring solution, S', becomes $S' \equiv (S_1, .., S_{k_1}, .., S_{k_2}, .., S_v)$. Furthermore, the neighbourhood, $N_\lambda(S)$, of a given solution S is the set of all neighbours S' generated by the λ-interchange mechanism.

In addition to defining the neighbourhood, we must also specify the order in which neighbours are searched.

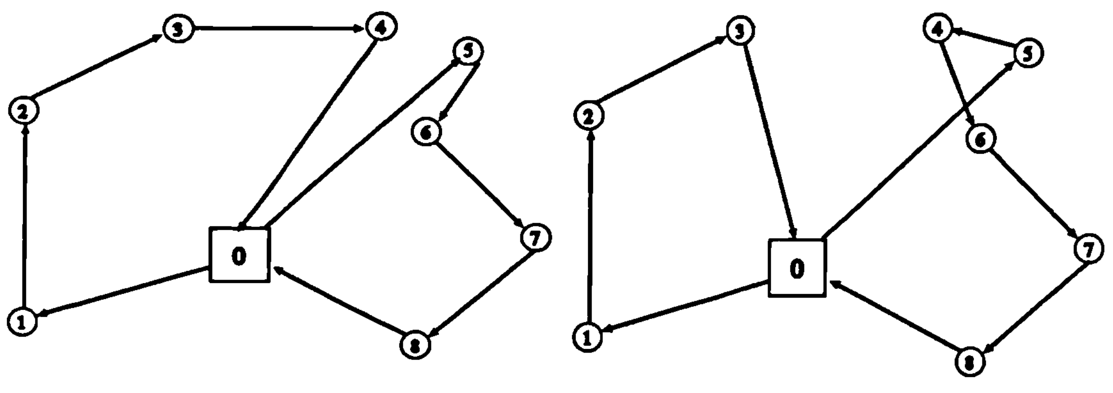
First, an ordered search is usually adopted in which all possible combinations of a pair of route sets S_{k_1} and S_{k_2} in S are considered systematically without repetition (see Section 3.3.2). An ordered search of the interchange neighbourhood would typically examine a total number of $v(v-1)/2$ combinations of pair of routes (S_{k_1}, S_{k_2}) according to the order specified by a permutation σ. Each time a cycle of search is completed a new permutation σ is generated. The search continues with the new combination order, defined as follows.

$$(S_{\sigma(1)}, S_{\sigma(2)}), \ldots, (S_{\sigma(1)}, S_{\sigma(v)}), (S_{\sigma(2)}, S_{\sigma(3)}), \ldots, (S_{\sigma(v-1)}, S_{\sigma(v)}) \qquad (5.3)$$

It has to be pointed out that the permutation σ does not change in the descent and the tabu search algorithms. It takes a fixed order throughout the search, say $\{\sigma(i) = i \ \forall\, i = 1, \ldots, v\}$, whereas this order would change in simulated annealing algorithm. At the end of each completed cycle of search, a new permutation σ is generated and the search continues selecting pairs of routes in the order specified in Equation 5.3 until the SA algorithm terminates.

Second, for the given pair of routes chosen above, we consider the λ-interchanges with $\lambda = 1$. Thus, there are two processes to search a given pair of routes S_{k_1} and S_{k_2}: a shift process is represented by (0,1), (1,0) denoting the shift of one customer from one route to another, and an interchange process represented by (1,1) denoting the exchange of one customer in the first route with another customer in the second to generate all the possible feasible solutions. The neighbours are generated systematically searching for improved solutions in the shift and the interchange processes.

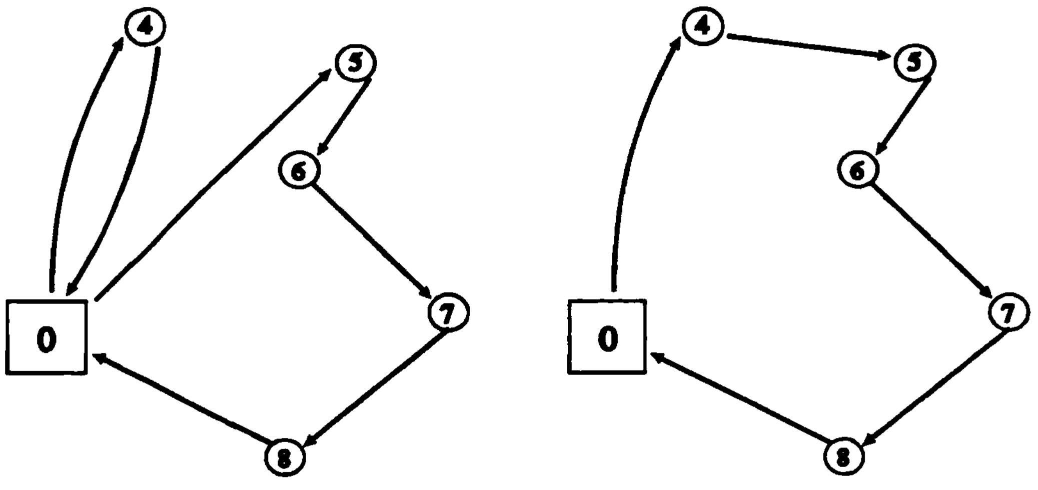
For example, let $S_1 = \{1,2,3,4\}$ and $S_2 = \{5,6,7,8\}$ be two set of vehicle routes. In the case of a (0,1) shift process, we attempt to shift every customer: 5, 6, 7, 8 one at a time from S_2 and add it to the set of customers in S_1. At each attempted move, we check the feasibility and take the appropriate decision according to the adopted acceptance criterion. Figure 5.3 (a) shows an example of a (1,0) shift process which removes a subset of one customer (index 4) from S_1 and inserts it in S_2. The new routes after the shift are shown in Figure 5.3 (b), and become $S_1 = \{1,2,3\}$ and $S_2 = \{5,4,6,7,8\}$.



(a) Before the shift (b) After the shift

Figure 5.3 A (1,0) shift process

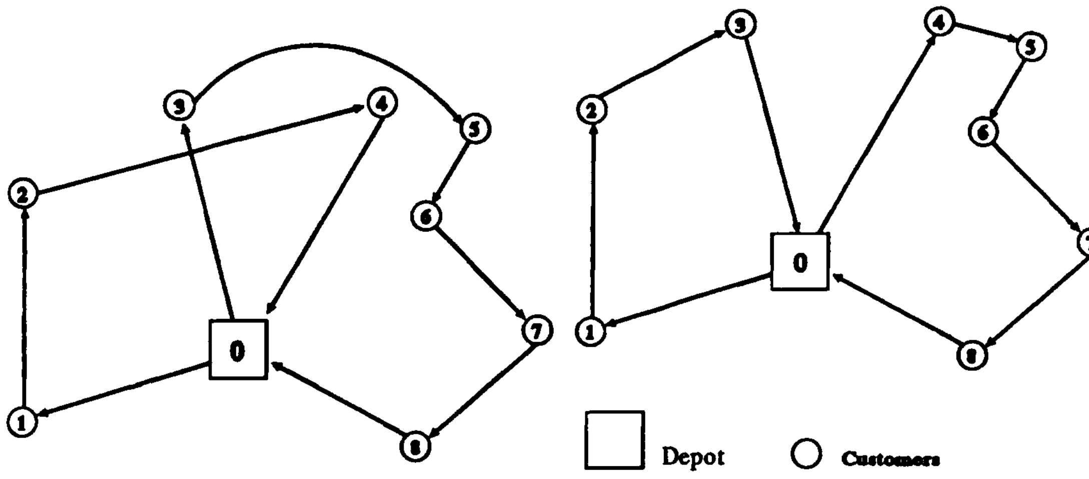
Note that, if set S_1 (say) contained only one customer then the (1,0) shift process would produce the solution $(S_1, S_2, \ldots, S_v)$ with $S_1 = \emptyset$. This is illustrated in Figure 5.4 (a) and Figure 5.4 (b). As a result, one vehicle route is reduced, i.e., the route corresponding to $S_1 = \emptyset$ and the total number of routes is reduced by 1. This is of great practical importance and is an important property of the generation mechanism.



(a) Before the shift (b) After the shift

Figure 5.4 A shift process with one vehicle reduction

In the case of a (1,1) interchange process, we attempt to exchange each customer in $S_1 = \{1,2,4\}$ with every customer in $S_2 = \{3,5,6,7,8\}$. We refer to Figure 5.5.



(a) Before the move (b) After the exchange

Figure 5.5. A (1,1) interchange process

For example, we try to exchange systematically customer number $1 \in S_1$ with every possible customer $3, 5, 6, 7, 8 \in S_2$ and similarly for 2 and 4 to generate feasible moves according to the acceptance criterion. Figure 5.5 (a) and Figure 5.5 (b) show an example, where a customer number, 4, from route S_1 is exchanged with customer number, 3, from S_2 to generate a new pair of routes.

5.2.3 Selection strategy of alternate solutions

We are proposing two selection strategies for selecting alternate solutions in the neighbourhood $N_\lambda(S)$ of a given solution S.

(i) *Best-improve (BI) strategy,* which examines all solutions $S' \in N_\lambda(S)$ in the neighbourhood of S and accepts the one which yields the *best admissible* solution according to a given *acceptance criterion.*

(ii) *First-improve (FI) strategy,* which immediately accepts the first solution in the neighbourhood that satisfies the acceptance criterion upon its first discovery.

The BI and FI strategies would be used in the descent algorithms as well as in the SA algorithms. However, these are generalised and extended for the case of tabu search algorithms. In tabu search, a move is considered admissible, if it is a non-tabu move, or a tabu move which passed the aspiration level criterion. The best-improve selection strategy selects that *admissible* move which yields the *highest evaluations* (i.e, the greatest improvement or the least disimprovement in the objective function), subject to the tabu restriction and aspiration criterion being satisfied, see Section 2.3.6b. This is then denoted by the BA strategy. However, the first-improve strategy is modified for tabu search as follows: The first and best-improve strategies are combined in a new strategy denoted by the FBA strategy. The FBA strategy selects the first admissible move in the neighbourhoods that provides an improvement over the current objective value. If there is no such an improving move, the move with the highest evaluated which is recorded in the searched neighbourhood is then selected. In this case, the FBA resembles the BA strategy. For further details on this strategy, we refer to Section 2.3.6c.

5.2.4 Evaluation of the cost of a move

A *move* is a transition from a solution S to a solution $S' \in N_\lambda(S)$. The value of the move is the difference in the objective function values, $\Delta = C(S') - C(S)$. This requires the evaluations of $f(S_{k_1})$,$f(S_{k_2})$ which are the costs of the TSP tours over customers in routes S_{k_1}, S_{k_2} respectively generated by the λ–interchange mechanism of section 5.2.2. These functions are very complex, and cannot be written down explicitly at each move. Consequently, iterative improvement methods become computationally expensive to implement. However, if we use approximate methods to obtain $f(S_k)$'s, a substantial saving of computer time would result and solving of the VRP becomes more effective. We propose two approximate evaluation methods and show their evaluations for the case of $\lambda = 1$. First of all, we need the following:

Let S_k be the set of customers in route k, and i be the customer to be added (or deleted) from S_k by the 1-interchange to obtain the new $S'_k = S_k \cup \{i\}$ (added) or $S'_k = S_k - \{i\}$ (deleted) case.

(a) ***Insertion (deletion) cost evaluation procedure***

Recall that the length of the tour over the set of customers in S_k is denoted by $f(S_k)$.

Let i be the customer to be inserted (or deleted) between two consecutive customers r and s of S_k, and define:

$l_i(r,s) = \quad c_{ri} + c_{is} - c_{rs}$ be the cost of inserting i between r and s,

$$l_t(S_k) = \underset{r,s \in S_k}{Minimum}\{l_t(r,s)\} \qquad (5.4)$$

then the cost of the new generated route over S'_k becomes:

Insertion case: $f(S'_k) = f(S_k) + l_t(S_k)$ (5.5)

Deletion case: $f(S'_k) = f(S_k) - l_t(r,s)$ (5.6)

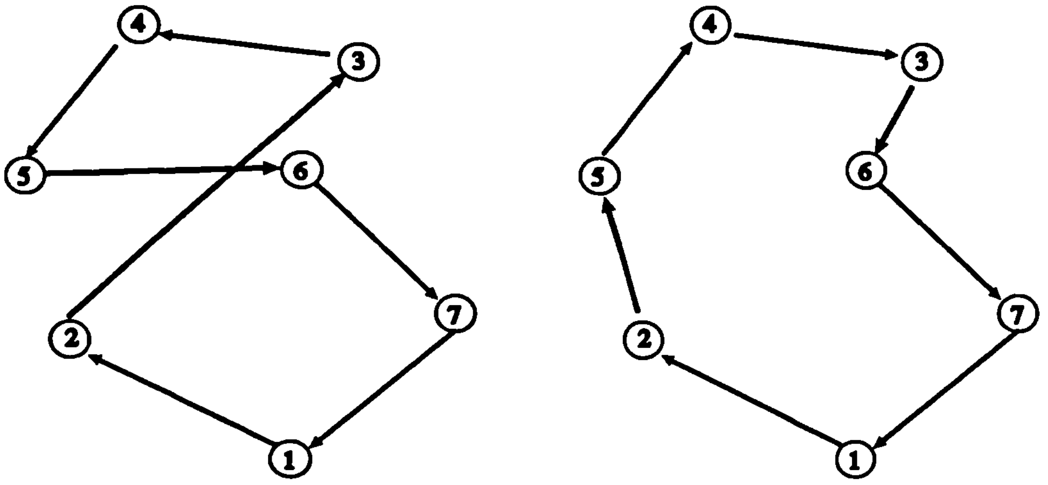
(b) ***2-opt cost evaluation procedure***

Perhaps the best known heuristic for the TSP is the arc exchange heuristic of Lin *et al.* [1974]. The procedure works as follows:

(i) Find an initial random tour over the set of customers in S_k.

(ii) Improve the tour using an arc exchange procedure.

(iii) Continue (ii) until no additional improvement can be made.



(a) (b)

Figure 5.6 A 2-change of a tour

The arc exchange procedure used in this work, involves exchanging just 2 arcs. A 2-change of a tour consists of the deletion of 2 arcs in a tour, e.g., arcs (2,3) and (5,6) in Figure 5.6 (a), and their replacement by 2 other arcs, say, (2,5) and (3,6) to form a new reduced cost tour in Figure 5.6 (b). Note that, the arcs between customers 3 and 5 have reversed their order in the new tour. A tour is called 2-optimal (2-opt), if it is not possible to improve the tour via any 2-change. Thus, we have used a 2-opt tour S_k to evaluate approximately the function $f(S_k)$.

(c) Combination of procedures (a) and (b)

This cost combination is used only in the case where a decision is made to accept a move. Each move is evaluated by insertion/ deletion; if it is accepted then a 2-opt procedure is invoked. The reason for this evaluation is depicted in Figure 5.3 (b) where criterion (a) may produce crossing arcs in the new tour in which case criterion (b) is necessary to produce the 2-opt tour of Figure 5.4 (b), (See also Figure 5.6). Thus criterion (c) provides a fast way to approximate exchanges which are then evaluated more thoroughly (and slowly) if they seem worthwhile. Evaluation method (a) works alone if only the condition of the following theorem 1 is held.

Theorem 1. (Agarwal *et al.* [1989])

Given a set of customer points S, $T^*(S)$ is the optimal TSP tour, and a customer i not in S; ***if*** the insertion cost of i in the best position in the tour $T^*(S)$ is equal to the best insertion cost between any two customers in the set S, ***then*** the tour obtained by inserting i in $T^*(S)$ will be the optimal TSP tour for the set $S' = S \cup \{i\}$

Proof:

Recall that the length of tour $T^*(S)$ is denoted by $f(S)$. After inserting i in its best position, let the new tour be $T_1(S')$ with length f_1. Then according to Equation (5.5)

$$f_1 = f(S) + l_i(T^*)$$

or

$$f(S) = f_1 - l_t(T^*) \tag{5.7}$$

$$\text{where } l_t(T^*) = \underset{(r,s)\in T^*}{\text{Minimum}}\{l_t(r,s)\}$$

Let $T^*(S')$ be the optimal tour over the set S' with length $f(S')$. Suppose stop i appears between stops r and s in this tour. Then, we can create a tour $T_2(S)$ by deleting i from $T^*(S')$ and joining r and s directly. Let f_2 be the length of this tour. Then

$$f_2 = f(S') - l_t(r,s)$$

Since $l_t(r,s) \geq l_t(S) = l_t(T^*)$ by definition and assumption respectively, we have

$$f_2 \leq f(S') - l_t(T^*) \tag{5.8}$$

Since $f(S)$ is the length of the optimal tour over S, we must have $f_2 \geq f(S)$. The last inequality with Expressions (5.7) and (5.8) yields

$$f(S') - l_t(T^*) \geq f(S) = f_1 - l_t(T^*)$$

or

$$f(S') \geq f_1$$

Since f_1 is the distance of the feasible tour T_1 and $f(S')$ is the length of the optimal tour, the above relation must hold with equality. Hence, T_1 is optimal for S'.

5.2.5 The λ-interchange descent algorithm

The λ-interchange descent algorithm is an iterative improvement method. It starts either with a solution chosen at random or by the application of a constructive heuristic such as the savings heuristic. The latter approach is sometimes adopted either to reduce the computing time or when a random feasible solution is difficult to generate. The descent method selects its starting solution S and chooses a neighbouring solution $S' \in N_\lambda(S)$ according to FI or BI strategies, evaluates the cost function $C(S)$ and $C(S')$ and computes $\Delta = C(S') - C(S)$. If $\Delta < 0$ then S' is accepted as the current solution. Alternatively, when $\Delta \geq 0$, S is retained as the current solution. The search usually continues until a (local minimum) λ-optimal solution is found (see Section 2.1.6).

Definition 5.3

A solution S is called *locally optimal* with respect to N_λ *(or* λ-opt for short) if and only if:

$$C(S) \leq C(S') \quad \forall S' \in N_\lambda(S)$$

The λ-interchange descent algorithm steps for the VRP are summarised below

Step 1. Generate an initial heuristic solution S by the savings method (Section 5.2.1).

Step 2. Choose a solution $S' \in N_\lambda(S)$ according to Section 5.2.2 and Section 5.2.3.

Step 3. If S' is better than S $(\Delta < 0)$, replace S by S', and go step 2.

Step 4. If a complete cycle of search - the neighbourhood $N_\lambda(S)$ of S - has been searched without any improvements **then**, stop with a 2-opt solution, **else** go to step 2. Since at least one route set has changed in the latter case, the search is restarted with the new solution for a new *cycle*.

The above descent algorithm is simple to implement and flexible to handle any additional aspects of a problem. Several descent algorithms can be derived according to the neighbourhood generation mechanism (Section 5.2.2), and the selection strategy (Section 5.2.3). As a result, we develop the following descent algorithms: 1+FI, 2+FI and 1+BI algorithms, which use the 1-interchange mechanism with the first-improve selection strategy and the 1-interchange mechanism with the best improve selection strategy, respectively. All these algorithms use the move cost evaluation of Section 5.2.4c. Despite the excellent performance of these heuristic algorithms, there is a major limitation that the local optimum achieved may be far from the global optimum. Also, the quality of the final solution depends critically on the initial starting solution. Many techniques have been devised to deal with the above limitations and allow the local search algorithms to continue its exploration beyond the local optimum region to find a near optimum solution. In the next section, we will focus on and explain in more detail SA approximate algorithms that overcome local optimality by embedding a randomised search and acceptance strategy into the local search methods.

5.3 Simulated annealing implementation

In the following, we shall adopt the same SA cooling schedule that we have introduced and implemented in Section 3.4 for the CCP, and in Section 4.3 for the GAP. This section explains the SA implementation to the VRP. We have discussed earlier the relationship between the SA algorithm and the local search method. The SA algorithm uses procedures that are identical those discussed in Section 5.2 such as: the initial starting solution, the 1-interchange mechanism, and the move cost evaluation. However, the differences are in the search of the neighbourhood, the acceptance of alternate solutions and the algorithm stopping criteria.

The neighbourhoods are searched according to random permutations, σ, of the route order $\{1,...,v\}$. These permutations are generated each time a complete cycle of search is completed. Each of the $v(v-1)/2$ possible pairs of route combinations is considered in turn according to the order of the permutation σ. More precisely, the neighbourhood is searched by selecting systematically pairs of sets (routes), (S_{k_1}, S_{k_2}) in the order indicated by the Expression (5.3) according to the permutation

$\{\sigma(1),\sigma(2),..,\sigma(v)\}$. This is in contrast to the local search descent methods where σ is fixed to an order of $\{1,2,..,v\}$ throughout the descent algorithm run. Furthermore, the search for a given pair (S_{k_1}, S_{k_2}) is systematic for all potential customer moves as in the descent methods (Section 5.2.2). This implementation is in contrast to classical SA schemes that recourse to random neighbourhood search, which can lead to pockets that remain unexplored for undesirable lengths of time. Hence, those methods may miss the few good moves at *lower* temperatures. This implementation of the SA algorithm keeps the best solution found during the search rather then the one at which it stops. The SA algorithm performs a single iteration (one attempted feasible move) at each temperature. In any SA implementation, we must additionally define a cooling schedule. We adopt for the VRP the non-monotonic SA cooling schedule which was introduced in Section 3.4.2. This cooling schedule require the following:

(i) The starting temperature T_s, the final temperature T_f.

(ii) The decrement rule for updating the temperature after each iteration k.

(iii) The condition for the temperature reset variables, T_{reset}, the value to which T is reset after the system freezes - a complete search of the neighbourhood (a cycle) is completed with no change. The temperature reset variable itself needs an update rule.

(iv) The total number of temperature resets R to be performed since the best solution was found. This parameter R is, therefore, the stopping criterion.

We will sketch briefly our cooling schedule parameters as follows:

(i) The initial and final temperature values.

The initial temperature, T_s, is given the largest change in the objective function value, δ_{max}. The final T_f is assigned the smallest change in the objective function value, δ_{min}.

The T_f parameter is used only to show the relationship with Lundy & Mees [1986] and Connolly [1990]. It is no longer used to detect the algorithm stopping as our cooling schedule uses another stopping criterion which does not depend on T_f.

(ii) The decrement temperature rule.

After each iteration k, the temperature is decreased according to a parameter β_k which depends on the number of iterations. Let us define

$$\beta_k = \frac{T_s - T_f}{(\alpha + \gamma\sqrt{k})T_sT_f} \tag{5.9}$$

where α and γ are constants. Then, the temperatures are updated according to the rule

$$T_{k+1} = \frac{T_k}{(1 + \beta_k T_k)} \tag{5.10}$$

Note that, if in Equation (5.9), $\gamma = 0$ then β_k becomes a constant, and this cooling schedule without the temperature resets is similar to the cooling schedules of Lundy & Mees [1986], and that of Connolly [1990]. There is a trade-off in the choice of the α and γ values. This trade-off can be determined experimentally as explained in Section 3.4.2. The aim of these settings is to provide fast reductions in the value of the temperatures at the beginning, and ideal slower reductions near the end of the search. The reasons are to spread the SA search and to concentrate on good regions.

(iii) The condition for occasional temperature increase

If a cycle of search is made without accepting any λ-interchange move then it is likely that with the current temperatures dropping even further there will be no accepted moves (if such a cycle has occurred, we say that the reset condition is satisfied). Therefore, the current temperature T_k needs to be reset to a higher value where we do not deviate much from the current solution but allow for some further moves to be made. The reason is to escape from the current local optimum. The temperature at iteration $k+1$ is then reset to a temperature value of T_{reset} i.e. ($T_{k+1} = T_{reset}$). We define how T_{reset} is updated. Initially, the T_{reset} is set to the initial value of T_s. The value of T_{reset} is updated first before its value is passed to T_{k+1} in the following way.

$$T_{reset} = \frac{T_{reset}}{\delta} \qquad \text{provided that } T_{reset} > T_k$$

$$T_{reset} = T_{found} \qquad \text{otherwise} \qquad (5.11)$$

where T_{found} records the temperature value at which the current best solution is found, and δ is a constant equals to 2. After this reset, the temperatures are updated as normal using the above rules Equation (5.10) and (5.11) until the algorithm stops.

(iv) The Stopping Criterion.

A stopping criterion can be arbitrarily chosen to reflect the amount of time one wants to spend on solving the problem. The stopping criterion we used is based on the number of temperature resets, R, since the best solution was found.

5.3.1 Simulated annealing algorithm

The SA algorithm steps are as follows:

(i) Initialisation.

Step 1. Generate an initial *heuristic* solution S by the savings method of Section 5.2.1.

Step 2. Initialisation of the *cooling schedule parameters*:

perform a *test* cycle of search over the neighbourhood, $N_1(S)$, of the initial solution (but without performing the exchanges themselves) in order to obtain the largest and smallest δ_{max}, δ_{min} change in objective function values, and an estimate of the total number of feasible exchanges *Nfeas*.

Set $T_s \leftarrow \delta_{max}$, $T_f \leftarrow \delta_{min}$, and $T_{reset} \leftarrow T_s$, $\alpha \leftarrow n \times Nfeas$, $\gamma \leftarrow n$, $R \leftarrow 3$.

Set the counters $k \leftarrow 1$,

(ii) Selection and acceptance of generated neighbours.

Step 3. Select a solution $S' \in N_1(S)$ systematically as explained in Section 5.2.2.

Compute $\Delta = C(S') - C(S)$ (Section 5.2.4).

Step 4 **If** $\{\Delta \leq 0$ or $e^{(-\Delta/T_k)} > \theta\}$ where θ is a uniform random parameter $0 < \theta < 1$,

then accept the new solution S' and set $S \leftarrow S'$.

If S' is better than the best so far **then**

keep S' and update T_{found}.

(iii) Temperature updates.

Step 5. Evaluate the variable decrement ratio β_k as in (5.9), and update temperatures according to rule (5.10) and the conditions for occasional increase in Equation (5.11)

Set $k \leftarrow k + 1$.

Step 6. **If** the stopping criterion (R resets were performed since the best was found) is met then stop and report the best found solution and the computation time, **otherwise**, go to Step 3.

Due to the temperature resets, we are able to continue the search beyond the **freezing** point. Thus, increasing the available computation of the above SA algorithm could lead to a better solution. This obviously most important property is not provided by many other cooling schedules in the literature.

5.4 Tabu search implementation

In this section, we implement tabu search (TS) algorithms for the VRP. It follows a similar approach which we used for the GAP (see Section 4.4). However, we present, here, some new ideas and modifications. Tabu search shares with simulated annealing, the ability to guide the search of iterative descent methods. In this context, TS provides a guiding framework for exploring the solution space beyond points where an embedded heuristic would become trapped in a local minimum. The process by which TS seek to transcend local optimality is based on an evaluation function. This function selects the highest evaluation move which produces the most improvement or the least disimprovement in the objective function at each iteration. The *best admissible* (not in the *tabu-list*) move is the highest evaluation move in $N_1(S)$, the 1-interchange neighbourhood of the current solution S, in terms of the objective function value and *tabu restrictions*. The reason for introducing a tabu-list is to store in that list characteristic of accepted moves so that these characteristic can be used to classify certain moves as tabu in future iterations. By accepting disimproving moves, it becomes possible to return to solutions already visited. Hence cycling may occur and the purpose of the tabu list is to prevent such occurrence. Thus, it is necessary for any implementation to define the following:

(i) A *forbidding* strategy which manages what goes into the tabu list. A *freeing* strategy which manages what goes out off the tabu list.

(ii) A *short term* strategy which is an overall strategy that manages the interplay between the above strategies including: (a) an *aspiration* strategy which ignores tabu conditions of certain moves; (b) selection strategies which choose trial solutions from $N_1(S)$ based on the best-admissible, BA, or the first-best-admissible FBA strategies of Section 5.2.3.

(iii) A stopping criterion

A principal contribution of this implementation, is the construction of special data structures for the tabu lists and for the candidate list of moves. A candidate list is a sublist of the possible admissible moves in the neighbourhood. The second contribution is the identification of an initial value for the tabu list size, $| Ts |$ and a way to update subsequently its value. This initial value of $| Ts |$ is determined statistically from the problem characteristics.

5.4.1 The forbidding and the freeing strategies

(a) The forbidding strategy

This strategy constrains the search by classifying certain moves *forbidden* (or tabu) based on *tabu conditions* which are identified by *attributes* of a move. A move is a transition from one solution to another generated by the 1-interchange mechanism which identifies the attributes of the move. In order to avoid cycling which involves preventing the move from a solution to other solutions $S' \in N_1(S)$. It is clearly sufficient to check that previously visited states are not revisited. Ideally, the tabu-list should store all such states and the list would be checked prior to any new move. However, the process of checking the tabu status of a move based on the above generally requires a great deal of memory and computational effort. This is so because we have to store attributes of a solution that are needed to check the tabu status of future move. A crude but reasonably effective way is instead to store instead a partial range of these attributes. Hence a special data structure for the tabu-list must be identified.

A tabu-list data structure is represented by a matrix which stores a simpler set of move attributes. This set consists of the two pairs (e_i,S_i) and (e_j,S_j). These attributes are sufficient to identify that a customer e_i from the set S_i of customers on route i has interchanged with a customer e_j from the set S_j of customers on route j, and vice versa. These attributes (e_i,S_i) and (e_j,S_j) are used to specify the *tabu restrictions* that forbid a move from being performed. A move is deemed tabu, if e_i is returned to S_i ***and*** e_j is returned to S_j. The same attributes could be used instead to forbid moves if ***either*** e_i is returned to S_i ***or*** e_j is returned to S_j. The advantage of this approximation is that more states can be represented and check those states faster. However, in this approximation some of non-tabu solutions are also prevented as they share some of these attributes which are stored in the tabu list. The set of forbidden moves is recorded in a *tabu list* for a period of $| Ts |$ iterations.

Route sets

Customers e_i \ S_i	S_1	S_2	S_3
0	80	75	100
1	1	•	5
2	-999	-999	-999
3	•	5	•
4	•	100	•
5	•	•	-999
6	•	80	75
7	-999	•	•
8	•	•	1

TABL matrix

Figure 5.7. A tabu list data structure, which is represented by an $(n+1)\times v$ TABL matrix.

Implicitly, we define the data structure, TABL, to take the form of an $(n+1)\times m$ matrix (n rows one per customer e_i and one extra for the *null* customer involved in the shift process (0,1) or (1,0) and m subsets S_i. The TABL(e_i,S_i) records the iteration number at which a customer e_i is removed from the set S_i, similarly for TABL(e_j,S_j). The TABL matrix is drawn in Figure 5.7 for an example of 8 customers and 3 routes. In this Figure, a high negative number value means that the corresponding customer has not been used in any previous moves (either a customer is in its best route or it is not feasible to be exchanged). Precisely, this negative value is to avoid false identification of customers as tabu during the initial iterations. Entries which have the same values mean that these customers are interchanged by a (1,1) interchange operation e.g., customer 1 from S_1 is interchanged with customer 8 from S_3 at iteration number 1, whereas the (1,0) shift operation of customer 4 from S_2 and adding it to S_3 at iteration 100 is represented by putting 100 in entries TABL(4,S_2) and TABL(0,S_3) respectively, and a similar representation is made for the (0,1) operations.

***(b) The tabu list size functions*:**

We believe that the tabu list size, $| Ts |$, should be a function depending on the problem characteristics (dimensions, ratio of required demands to available capacity, ρ, given in Table 5.1. It also depends on the selection strategies FBA or BA that is employed. Through observation of the tabu list sizes for various test problems, we have used a multiple regression analysis using to find an estimate for the $| Ts |$ value. A statistical package, MINITAB, is used to find an equation fit for $| Ts |$ using the data in Table 5.4 and Table 5.5. Based on the analytical results, we propose that a tabu size should take the following values. These Equations (5.12) and (5.13) have produced high R-squared statistical values.

For the case the TS+FBA algorithm:

Let us define ρ to be the capacity ratio of the required demands to the available vehicle capacity. The tabu list size, $| Ts |$, which we use in the TS+FBA algorithm is estimated from the data as follows:

$$| Ts | = 8 + (0.078 - 0.067 \times \rho) \times n \times v \qquad (5.12)$$

For the case of the TS+BA+DS algorithm:

The tabu list size which we use in the TS+BA+DS algorithm is estimated similarly as follows:

$$| Ts | = \mathrm{Max}\,\{7, \quad -40 + 9.6 \times \ln(n \times v)\,\} \qquad (5.13)$$

Since, the $| Ts |$ value is statistically estimated, an error might occur. To avoid such an error, an interval with its end-points are the 10% around the estimated value (say ts) i.e., these values are $0.9 \times ts$, and $1.1 \times ts$. The value of $| Ts |$ is varied in a systematic order to take each of the three values $0.9 \times ts$,, ts, and $1.1 \times ts$. $| Ts |$ retains each of the assigned value for $2 \times | Ts |$ iterations before its assigned another value. If all the three values are chosen a random permutation is obtained and the assignment continues in the same manner. Similarly, experiments of Taillard [1990] show that varying $| Ts |$ randomly to take a value inside a given interval has an advantage.

(c) The freeing strategy

This strategy is concerned with the management of what comes out of the tabu list. The attributes of a tabu move remain on the tabu list for a duration of $| Ts |$ iterations. If this tenure has elapsed, then the tabu restrictions imposed are removed so that those move can be reconsidered as admissible in any future search. The proposed data structure (Figure 5.7) which stores the iteration number of those attributes updates automatically the tabu status. Also, with this data structure we can easily determine the tabu status of a potential move. A simple and fast check for tabu status is of great importance, especially when the problem and the tabu size increase. More clearly, if we are at iteration k then a move is classified as tabu if

$$k - \mathrm{TABL}(e_i, S_i) \leqslant |Ts|$$

and

$$k - \mathrm{TABL}(e_j, S_j) \leqslant |Ts| \tag{5.14}$$

In other words, neither e_i should return to S_i nor e_j should return to S_j during the following $| Ts |$ iterations. By storing in the tabu list the time (iteration number) when a move is made tabu, tabu status can be checked in two simple operations in Equations

(5.14). This data structure provides a better and fast way to check the tabu status of a move than that in the classical circular approach explained in Section 2.3.3 which needs more input control for the freeing strategy.

5.4.2 The short term strategy

The short term strategy is an overall strategy and the core of the TS algorithm. It manages the interplay between the forbidding, the freeing, the aspiration, the move selection strategies, and the stopping criterion. It is also designed to permit the evaluation of the best admissible move in the neighbourhood based on *tabu restrictions* and *aspiration criteria*. Aspiration criteria are measures mainly designed to override the tabu status of a move if this move is *good enough* and sufficient to prevent cycling. The tabu restrictions and aspiration criteria play a dual role in constraining and guiding the search process, Glover [1989a]. Tabu restrictions allow a move to be regarded admissible, if these restriction are not violated. A tabu move is also admissible if aspiration criterion apply regardless of the tabu status (see Section 2.3.4).

(a) The aspiration function:

This function can have different influences on the quality of solutions (Section 2.3.4). Based on experience, we have learned in the early applications, only the following aspiration function will be used. Let S_{best} be the current best solution found so far, during the search. Let S' be a tabu solution in $N_1(S)$ and $\Delta = C\ S') - C(S)$ be the value of the move. The solution S' is admissible by the aspiration criterion if Equation (5.15) is satisfied.

$$C(S) + \Delta \quad < \quad C(S_{best}) \qquad (5.15)$$

In other words, a tabu move considered admissible if it only improves the best solution found rather than the current one. This move provide a new direction of search and guarantees that cycling would not occur.

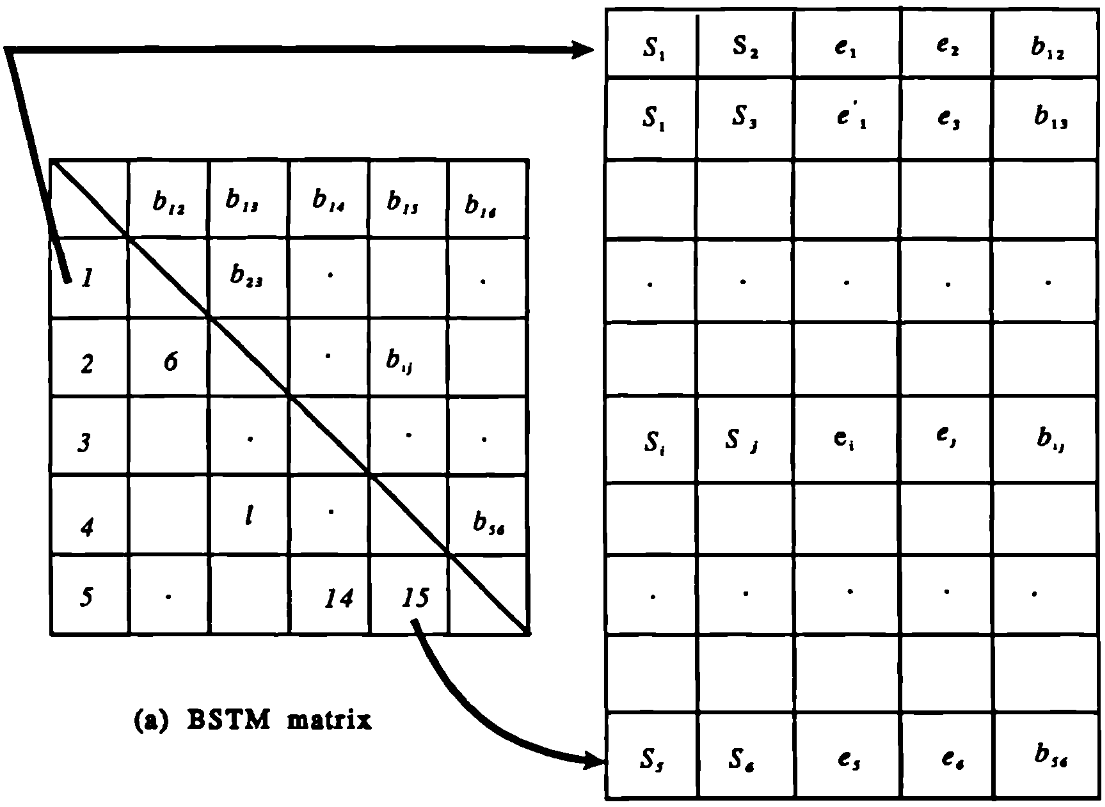
(b) ***The move selection strategies.***

We consider, here, two selection strategies to select admissible move from the *candidate list* of moves. The first strategy is the best-admissible selection strategy, BA. The BA strategy selects the admissible move from the current solution which yield the greatest improvement or the least disimprovement in the objective function, subject to the tabu restriction and the aspiration criterion (see Section 2.3.7b). The corresponding TS algorithm which employs this strategy is denoted by TS+BA. The second strategy is the first-best-admissible strategy, FBA, which is based on the combination the first and best improve strategies of Section 5.2.3. This strategy uses a greedy approach which selects the admissible move that provides an improvement over the current solution in the objective value. If all moves in the candidate list are tried without any improvement, the FBA strategy selects the best disimproving move (see Section 2.3.7c). Similarly, the TS algorithm which uses this strategy is represented by TS+FBA.

The candidate list for the TS+FBA algorithm is the whole neighbourhood $N_1(S)$ generated by the 1-interchange mechanism. The size of the candidate list is dynamic and determined automatically by the search itself. The dynamic sampling is a desirable way to search a large neighbourhood. The candidate list for the TS+BA algorithm is the set of the best admissible moves in the neighbourhood. This list is very expense to compute especially for large-sized problems. The reason is that the whole neighbourhood must be re-evaluated to select the best move after each iteration.

(c) A special data structure for the BA selection strategy:

Candidate lists occupy an important practical role in tabu search by balancing the computational efficiency against the goal of obtaining a highest evaluation move, subject to satisfying the associated tabu restrictions. The computational effort to find the highest evaluation move is notably reduced by the use of the proposed data structures. The data structure we propose permit only a small number of evaluations to be recalculated (in order to identify a *new best* move) from one iteration to another. The need for a data structure is that the neighbourhood increases with the problem size, the BA and BI strategies become more expensive to execute, when implemented in any descent or Any TS algorithms. It also requires more computation effort to store and compare moves. We have designed a special data structure, that provide the reduction in the computation time *without shrinking or sacrificing* the quality of solution.



b_{12}
b_{13}
b_{14}
b_{15}
b_{16}
1
2
3
4
5
b_{23}
6
b_{ij}
l
b_{56}
14
15
(a) BSTM matrix
S_1 S_2 e_1 e_2 b_{12}
S_1 S_3 e'_1 e_3 b_{13}
S_i S_j e_i e_j b_{ij}
S_5 S_6 e_5 e_6 b_{56}

(b) RECM matrix

Figure 5.8 A data structure representation for the BA selection strategy.

This data structure needs two matrices: Let BSTM be matrix of size $v \times v$, and RECM be the other matrix of size $\{v(v-1)/2\} \times 5$. The top triangular part of the matrix BSTM(i,j) stores the objective value of the best move obtained by the 1-interhange mechanism between a given pairs S_i and S_j if such a move exists, otherwise a high value is stored instead. The lower triangular part BSTM(j,i) stores the index l of this combination S_i and S_j in the set $\{1,\ldots,v(v-1)/2\}$ (see Equation 5.3); this index is then used to indicate the position in which we store the attributes of the highest evaluation move obtained from the two route sets i.e. RECM(l,1)= S_i, RECM(l, 2)= S_j, RECM(l, 3)= e_i , RECM(l,4)= e_j, RECM(l,5)= b_{ij} with $b_{ij} = \Delta$ as the move value. This data structure is illustrated in Figure 5.8 with an example of v= 6 vehicles and undefined number of customers, as the data structure is independent of the customers number. There are 6(5-1)/2= 15 possible pair combinations, which represent the number of rows in Figure 5.8b as well as the number of entries in the lower and upper matrices in Figure 5.8a. The arrows show the chronological mapping order between the two matrices BSTM and RECM. The columns in RECM represent the two pair route sets, the two interchanged customers and the move value b_{ij}. During the search, the b_{ij} and the corresponding index l of the route sets are first identified from the BSTM then the l value is used to get the attributes of the move from the data matrix RECM.

This data structure evaluates all moves in the neighbourhood $N_1(S)$ once at the first iteration. At each iteration, the upper matrix of BSTM is scanned and the best stored move is identified and accepted. Then after an accepted move, only the two corresponding route sets are affected, and the others remain intact. Thus, only the moves in $2 \times v$ pair combinations of route sets $((S_i,S_k)\ \forall k \neq i$ and $(S_j,S_k)\ \forall k \neq j)$ need to be evaluated rather than all moves in the whole neighbourhood considering the $v(v-1)/2$ pair combinations. As a result, the time requirement for this data structure to evaluate the neighbourhood is $O(v)$ rather than $O(v^2)$ of $O\left(\left(\frac{n}{v}\right)^2\right)$. The $O\left(\left(\frac{n}{v}\right)^2\right)$ order is the number moves examination checked inside a given pairs of route sets. This data structure will save a lot of computation for any value of $v \geq 5$ as illustrated in Figure 5.9 without sacrificing the quality of the solution.

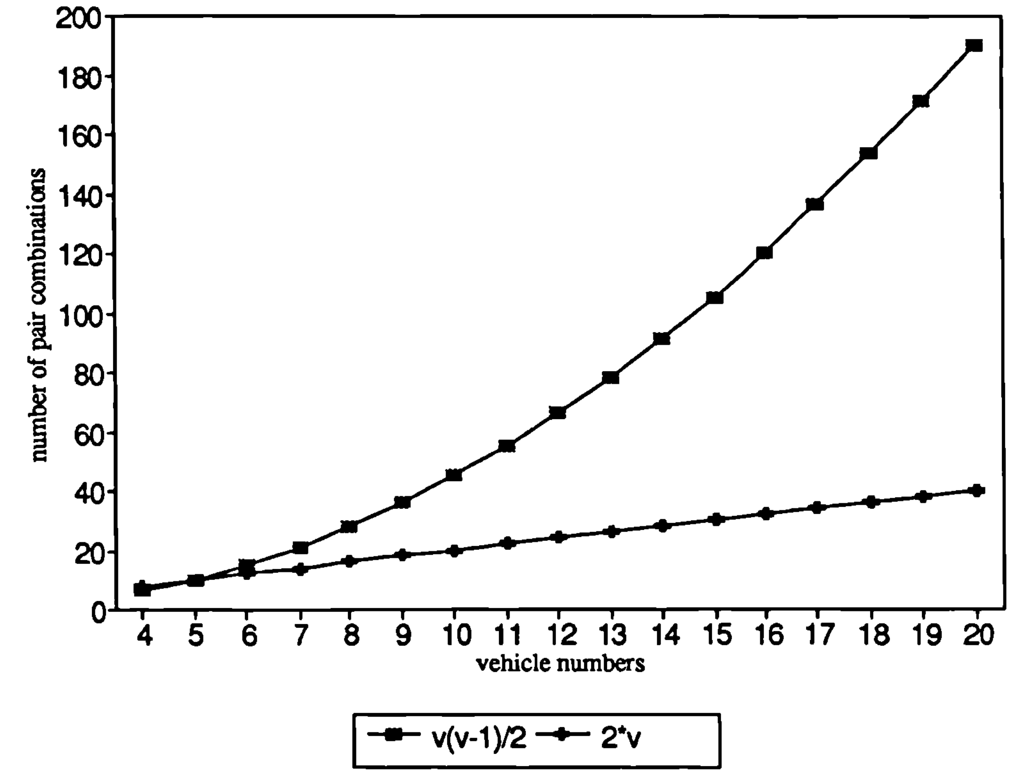


Figure 5.9. Computational requirements with and without the data structure per one iteration of the BA strategy.

The TS+BA algorithm which uses this data structure, DS, for move selection is denoted by TS+BA+DS algorithm. Also, the descent algorithm which uses the best improve selection strategy, BI, and this data structure is denoted by BI+DS algorithm.

Note that, the TS+FBA algorithm can not use any special data structure for its move selection strategy. The TS+FBA algorithm accepts the first admissible move, which gives a reduction in the objective value of the current solution say *S*. The search is then starts from $N_1(S)$ as in the search by the descent method. As a result, the neighbourhood size is not generally fixed. It is, however, of a variable size, which is determined automatically by the search. Furthermore, the FBA algorithm records and updates the best admissible move during the search, for the following reason. If we search the whole neighbourhood, $N_1(S)$, without finding any improved solution over the current one, then the best admissible move is accepted. At this moment, the TS+FBA is similar to the TS+BA algorithm. Moreover, the TS+FBA algorithm accepts more moves in good regions, hence that tabu list is updated more frequently and as a consequence a larger part of the solution space is finally searched.

(d) The stopping criterion.

The stopping criterion, we used in the TS algorithms is based on a maximum iterations number (*MAXI*) elapsed since the iteration at which the solution is found. This has the advantage of relating the stopping criterion to the problem solution changes. However, the *MAXI* iterations at the end of the search are a loss of computational effort, and it is good to estimate a total number of iterations to be performed so that such a loss can be saved.

We have used a statistical method based a multiple regression analysis to analyse the iteration numbers during which a significant improvement in the objective value occurred. A fitted equation to get an approximate value for the total number of iterations, M, based on problem characteristic is similarly estimated like the $|Ts|$ value. This is merely a guidance so that extra time can be saved and good solutions can be obtained with a reasonable computation time. This estimate, for the TS+BA+DS is obtained with an R-squared value of 81.8%, and its value is computed by:

$$M = 340 + 0.000353 \times \rho \times (n \times v)^2 \qquad (5.16)$$

where ρ is a measure of a problem tightness called *capacity ratio* in Table 5.1.

5.4.3 Tabu search algorithm.

In this section the general tabu search steps will be presented.

(a) Initialisation.

Step 1. Get an *initial heuristic* solution S, e.g by applying the savings algorithm of section 5.2.1.

Initialise, the tabu list size $|T_s|$;

Set *the initial tabu list* TABL$(i,j) \leftarrow -\infty \; \forall i,j$;

Perform a cycle of search to initialise the data structure matrices BSTM and RECM if the BA strategy is used;

Set a value for *MAXI* (a maximum iteration number to be elapsed since the best solution is found) or a total number of iterations M, according to Equation (5.16);

Set counters $k \leftarrow 1$, and $k_{best} \leftarrow 0$.

(b) Selection and acceptance of generated neighbours.

Step 2. *Choose* an admissible move $S' \in N_1(S)$ (feasible and not tabu or tabu status is overridden by aspiration criterion) according to the **BA** or **FBA** selection strategy.

Update as follows:

- Storing the attributes of the newly accepted move in the tabu list matrix TABL.
- Update the current solution S.

Set $k \leftarrow k+1$.

If current solution is better than the best solution found so far **then** update the latter and set $k_{best} \leftarrow k$.

- **If** we are using the BA strategy, **then** update the BSTM and RECM matrices. (Data structure for the selection the best move).

(c) **The stopping test.**

Step 3. **If** ($k - k_{best}$ > MAXI) **then** go to Step 4, otherwise go to Step 2.

Step 4. Display the TS final and initial solutions together with the computation times. Stop

5.5 Computational experience

5.5.1 Test problems

The set of test problems we used in this work for comparing the solutions of the SA and TS methods with the best solutions reported in the literature, are to be discussed in this section. In the literature some data with customer locations defined by coordinates are published and the calculation of Euclidean distances is assumed between the customers. This could be done as a real floating point operation or as an integer-operation with truncation. Hence, different results were reported in the literature for the same problems. Our distance data is calculated in a real floating point operation point. Descent, SA and TS algorithms are compared to solution methods that gave the best published routes. These methods are summarised in Paessens [1989].

The test problems are of two types according to the existence (or absence) of maximum route time limits. They are the 17 classical problems in the literature ranging from tight or loose capacity, with or without time constraints to some geographically

Table 5.1. Specification and characteristic of test problems.

Problem number	Problem origin	Problem size	Vehicle capacity	Maximum route length	Service time	Capacity ratio
G1	Gas[1967]	29	4500	240	10	0.70
CL	Cla[1964]	30	140	-	-	0.92
G2	Gas[1967]	32	8000	240	10	0.91
C1	Chr[1969]	50	160	-	-	0.97
C2	Chr[1969]	75	140	-	-	0.97
C3	Chr[1969]	100	200	-	-	0.91
C4	Chr[1979]	100	200	-	-	0.90
C5	Chr[1979]	120	200	-	-	0.98
C6	Chr[1979]	150	200	-	-	0.93
C7	Chr[1979]	199	200	-	-	0.98
C8	Chr[1969]	50	160	200	10	0.80
C9	Chr[1969]	75	140	160	10	0.88
C10	Chr[1969]	100	200	230	10	0.81
C11	Chr[1979]	100	200	1040	90	0.82
C12	Chr[1979]	120	200	720	50	0.62
C13	Chr[1979]	150	200	200	10	0.80
C14	Chr[1979]	199	200	200	10	0.88

Table 5.1. (*continued ...*)

Table 5.1. Specification and characteristic of test problems.

Problem number	Problem origin	Problem size	Vehicle capacity	Maximum route length	Service time	Capacity ratio
N1	New	50	275	-	-	0.90
N2	New	50	195	-	-	0.90
N3	New	50	165	-	-	0.90
N4	New	75	350	-	-	0.91
N5	New	75	265	-	-	0.90
N6	New	75	223	-	-	0.91
N7	New	100	410	-	-	0.90
N8	New	100	330	-	-	0.91
N9	New	100	266	-	-	0.91

Table 5.1. (*concluded*)

Chr[1969]	:	Christofides and Eilon [1969]
Chr[1979]	:	Christofides *et al.* [1979]
Cla[1964]	:	Clark & Wright [1964]
Gas[1967]	:	Gaskell [1967]
New	:	New randomly generated data

clustered problems. Their sizes range from 29 to 199 customers. In addition, we generated randomly 9 new problems, 3 each of sizes 50, 75 and 100. Their coordinates points are taken from the uniform distribution between U[1,100], while the depot coordinates are chosen from the U[45,55]. The customer demands are generated in the interval U[20,40], while vehicle capacity are fixed so that ratios of the required demands to the available supplies are in [0.90,0.92]. More details on problem characteristics are summarised in Table 5.1.

Computational results are evaluated using the RPD relative percentage of a solution, Z_k, generated by a heuristic *h* from the *new best* solution Z_{best} i.e. $RPD = \frac{100(Z_k - Z_{best})}{Z_{best}}$. The average computation time, ACT, in CPU seconds of the actual execution is reported excluding input and output time. The algorithms are programmed in FORTRAN 77 and run on a VAX 8600 computer.

5.5.2 Descent algorithms

In order to design good approximate algorithms, we need to equip them with the right choice of: a neighbourhood generation mechanism; a selection strategy of alternate solutions and a cost evaluation function. We varied these parameters in order to examine not only their effect on running time, but also their effect on the solution quality. More specifically, we tested the effect of the neighbourhood size such as produced by the 1-interchange and the 2-interchange mechanisms. The descent method using these neighbourhoods are implemented using a first-improve selection strategy of section 5.2.3(i) in the 1+FI, 2+FI with 2-opt TSP procedure of Section 5.2.4(b). The best-improve selection strategy BI of Section 5.2.3(ii) is only implemented using 1-interchange in the 1+BI descent procedure with the same 2-opt TSP for route calculation. Furthermore, the best-improve strategy is implemented using the data structure of Section 5.4.2(c) with cost evaluation criterion of Section 5.2.4(c) in the BI+DS descent algorithm. The

BI+DS also uses the 1-interchange neighbourhood mechanism. The objective of these variations is to establish the best combination of strategies from the experiment, and use them in the metastrategy SA and TS algorithms for further research analysis.

Computational results are listed in Table 5.2, reporting the obtained solutions and their computation time in CPU seconds. In evaluating the results, we seek to answer several questions. *First*, how big the neighbourhood should be? We have used the λ-interchange descent algorithms with $\lambda \in \{1,2\}$ embedding the first-improve selection strategy of Section 5.2.3(i), with the 2-opt route cost evaluation criterion of Section 5.2.4(c) at each attempted move. We observe that the resulted algorithms namely, the 1+FI and the 2+FI, improve significantly the initial starting solutions of (C&W), Clark & Wright [1964]. The average relative percentage deviation, (ARPD), is reduced from 26.65% (C&W) to 10.07% and 4.87% for the 1+FI and 2+FI algorithms, respectively. Their corresponding average computation time ACT increases from 1.51 to 141.22 and 2941.22 seconds. The 2+FI algorithm improves over the solution of 1+FI algorithm by 106.77% at an extra ACT (a factor of about 20). However, the latter 1+FI improves the C&W by 164.64% at only 92.5% increases in ACT. The rate of improvement of the 1+FI algorithm is better than the 2+FI algorithm with respect to computation time and solution quality. Although, the 2+FI found the best known solution for problems G1 and G2, it shows a great variability in computing requirements, it takes 0.89 seconds for problem (CL) of size 30, and 14,886 seconds for (C5) of size 120 but 4,274 seconds for (C7) of size 199, although the last two problems have the same capacity ratio of 0.98. The reason for this variability is partially due to the large neighbourhood search with $\lambda = 2$ if at least one improvement has occurred over the 1+FI solutions.

The 1+FI algorithm has started from bad C&W solutions with RPD's of 24% and 28% for problem C5 and C7 respectively. This means a large number of steps were

Table 5.2. Computational results for the λ-Interchange descent methods.

Problem number	C&W L (V)	Time	1+FI	Time	2+FI	Time	1+BI	Time	BI+DS	Time	New best known
G1	1017 (5)	0.08	953	2.22	875*	60.64	953	5.68	953	0.55	875
CL	1258 (7)	0.12	1255	0.20	1255	0.89	1255	0.20	1255	0.07	1205
G2	888 (5)	0.11	833	3.11	810*	36.13	833	5.22	833	0.49	810
C1	625 (5)	0.26	588	12.97	588	52.16	588	18.29	592	1.14	524
C2	1005 (10)	0.70	864	5.05	859	71.97	871	17.58	871	2.40	838
C3	982 (8)	1.18	942	46.30	910	15.00	962	91.26	956	5.61	830
C4	939 (10)	1.06	834	159.15	834	736.00	877	150.04	887	3.40	819
C5	1291 (7)	2.17	1104	555.45	1053	14886.35	1086	1824.90	1167	10.20	1042
C6	1299 (12)	3.32	1186	137.27	1127	4028.54	1158	410.00	1165	22.70	1044
C7	1707 (16)	7.04	1515	205.48	1421	4274.92	1474	705.40	1519	20.22	1334
C8	670 (6)	0.27	604	15.22	568	474.63	570	65.94	584	4.85	555
C9	989 (12)	0.70	964	24.18	953	190.33	953	51.42	947	9.40	909
C10	1055 (10)	1.20	980	146.00	923	7628.00	966	795.33	978	27.27	866
C11	952 (11)	1.08	884	91.04	884	558.65	885	203.20	885	12.30	866
C12	1646 (11)	2.19	1620	292.00	1564	9205.00	1613	1062.00	1620	43.26	1545
C13	1383 (15)	3.08	1298	423.00	1282	7094.00	1293	2475.00	1294	149.14	1169
C14	1671 (20)	7.37	1550	786.00	1504	13100.00	1535	5472.00	1542	240.54	1418

Table 5.2. *(continued ...)*

Table 5.2. Computational results for the λ–Interchange descent methods.

Problem number	C&W L (V)	Time	1+FI	Time	2+FI	Time	1+BI	Time	BI+DS	Time	New best known
N1	948 (6)	0.21	754	13.67	729	148.50	765	21.70	811	1.48	709
N2	1134 (8)	0.22	897	9.67	876	38.43	966	10.92	932	2.69	814
N3	1282 (10)	0.25	1061	3.78	1010	18.00	1100	8.99	1122	2.34	994
N4	1476 (7)	0.66	1072	87.32	991	2983.00	1075	74.01	1287	3.96	925
N5	1534 (9)	0.65	1277	24.82	1143	538.50	1287	50.23	1352	4.54	1045
N6	1532 (11)	0.70	1169	18.69	1052	207.15	1126	69.97	1306	5.55	1011
N7	1579 (8)	1.55	1147	483.33	1133	4277.16	1174	1056.00	1341	14.47	1035
N8	1704 (10)	1.58	1412	88.01	1196	5010.80	1360	290.00	1512	12.00	1185
N9	1902 (12)	1.63	1480	38.02	1295	842.30	1435	203.78	1658	7.07	1234
ARPD	26.65	-	10.07	-	4.87	-	9.91	-	14.36	-	-
ACT	-	1.51	-	141.22	-	2941.42	-	582.27	-	23.37	-

Table 5.2. (*concluded*)

C&W:	Clark and wright [1964] saving heuristic. L is the total route length excluding service times, with a, V, number of vehicles.
1+FI:	Frist-improve acceptance strategy with 2-opt TSP route evaluations and 1-interchange descent methods.
2+FI	First-Improve acceptance strategy with 2-opt TSP route evaluations and 2-interchange descent methods.
1+BI:	Best-Improve acceptance strategy with 2-opt TSP route evaluations and 1-interchange descent methods.
BI+DS:	Best insertion evaluations during the neighbourhood search for moves stored in a special data structure, 2-opt route TSP evaluations for the best-improve selected moves from the stored ones.
ARPD:	Average relative percentage deviations from the best known solutions over all test problems.
ACT:	Average computation time in seconds.

needed to find good solutions. The C&W also produces very bad initial solutions for the random data with the relative percentage deviations, RPD, varying form 28% to 59% and from 0.87% to 28% for the published data. This demonstrates that descent algorithms depend heavily on good initial starting solutions to save computation time, for acceptable quality solutions, in addition to, a good neighbourhood search mechanism.

Second, which selection strategy - a best-improve or a first-improve - is best ? Table 5.2 shows that the 1+FI has an ARPD of 10.07%, which is almost the same as, 9.91%, of the best improve descent algorithm 1+BI. The latter requires an ACT of 582.42 seconds which is 312% more than the ACT of 1+FI algorithm.

Last, what effects does the approximate route cost evaluation criterion (c) and the use of a special data structure have on solution quality and computation time? Not surprisingly, an examination of Table 5.2 reveals that the best-improve with a data structure algorithm, BI+DS, improves the ACT of 1+BI by 2390% and that of the 1+FI by 504%. However, its average relative percentage deviation is 14.36% which is worse than the 1+BI value by 44%. Consequently, we are left with two competing algorithms: 1+FI which gives better results than the BI+DS; however the latter requires less computation time.

5.5.3 Metastrategy algorithms

In this section, we evaluate the performance of the SA and TS algorithms. Computational results are reported on the same test problems. *First,* we discuss the implementation of SA and its cooling schedule parameters. The C&W solution is used as a starting solution. The cooling schedule is used as explained in Section 5.3.1. It is superimposed on the 1+FI descent algorithm with a cost evaluation criterion of Section 5.2.4(c). From our experience with SA on the CCP and GAP, we found that the cooling schedule parameter values are related to the problem size and estimated from one run of SA on a cycle of the initial solution without move replacements. We suggest that α is set to be the product of the number of customers n and the size of the neighbourhood - the number of feasible solution; the number of customers n to γ; the largest, and smallest increases in solution costs are given to T_s, and T_f respectively. A value of three is given for the stopping criterion parameter (R) of Section 5.3.1(iv).

Computational results of SA are listed in Table 5.3. Results show that the simulated annealing finds 10 new best (or previously known) solutions as marked with [*] for the equal best and [b] for new best in Table 5.3. SA fails to reach the previously best known solutions for tight capacity problems C1 and C5. However, SA solutions quality is not robust and varies with the problem making the 2+FI solutions better in some cases e.g., SA solution for C5 is about 13% away from the best solution compared to 1.05% of 2+FI, and SA were worse than the 2+FI's for 6 out of the 9 random problems. The algorithm that performs the best, uses the longest computation time. However, simulated annealing solutions can be improved by further tuning its parameters for problems where running time is short. The ACT to the best solutions together with the ACT to the end of runs are reported in Table 5.3. This is to indicate how early a solution can be found, as extra time is spent to prove that we can not improve over the best solution obtained so far by the algorithm. The overall ARPD of SA is 3.27% which is better than 4.87% of 2+FI, with an ACT of 3275 seconds for SA compared to 2941 seconds for the 2+FI algorithm. This presents a percentage improvement in solutions for SA of 48.92% over 2+FI at only an 11.35% percent increase in ACT. Furthermore, the SA algorithm generates a reduction in the total number of vehicles used and finds new reduced numbers for 4 problems marked with [a] in Table 5.3. However, this was not the case for the 2+FI where no such reductions occurred.

Table 5.3. Computational results for the (SA) simulated annealing algorithm

Problem number	SA solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Best known
G1	875*	0.00	4	1817	12.11	16045	107.00	875
CL	1213	0.66	7	9918	44.75	12870	58.07	1205
G2	810*	0.00	4	310	2.88	654	6.08	810
C1	528	0.76	5	566	8.77	10802	167.49	524
C2	838*	0.00	10	26010	3564.31	46953	6434.35	838^b
C3	830*	0.00	8	59244	6171.20	89607	9334.00	830^b
C4	826	0.85	10	1516	48.79	19634	632.00	819
C5	1176	12.85	7	2252	266.20	2672	315.85	1042
C6	1058	1.34	12	69609	4293.00	81272	5012.30	1044
C7	1378	3.29	16^a	34431	1373.82	58097	2318.11	1334
C8	555*	0.00	6	10427	697.87	50952	3410.20	555^b
C9	909*	0.00	11^a	17478	311.30	35177	626.55	909^b
C10	866*	0.00	9	9810	364.26	25780	957.26	866^b
C11	890	2.27	11	6457	300.49	6560	305.29	866
C12	1545*	0.00	11	39162	4569.26	65331	7622.56	1545^b
C13	1169*	0.00	14^a	739981	59017.16	1057000	84301.0	1169^b
C14	1418*	0.00	18^a	25648	2417.37	60561	5708.00	1418^b

Table 5 3. (*continued* .)

Table 5.3. Computational results for (SA) simulated annealing algorithm

Problem number	SA solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Best known
N1	757	6.77	6	3020	43.84	3437	49.90	709
N2	864	6.14	8	8900	71.33	12464	99.90	814
N3	1032	3.82	10	399	21.51	539	29.06	994
N4	1014	9.62	7	4327	122.76	4411	125.15	925
N5	1098	5.07	9	3074	145.21	3196	150.98	1045
N6	1091	7.91	11	10664	102.56	17260	166.00	1011
N7	1135	9.66	8	9910	513.90	10588	549.06	1035
N8	1264	6.66	10	3403	319.17	3880	363.91	1185
N9	1350	9.40	12	15052	350.75	15118	352.29	1234
Averages	-	3.37	-	-	3275.18	-	4969.32	-

Table 5.3. *(Concluded)*

*: Indicates that the best known route length is attained

■: Indicates that the best known vehicle number is found by SA algorithm

b: Ind cs that the best known route length is found by SA algorithm

RPD : Relative percentage deviation over the best solutions.

Second, we analyse the behaviour of two tabu search algorithms namely: the TS+FBA algorithm and the TS+BA+DS algorithm. Both schemes follow the implementation discussed in Section 5.4 with a cost evaluation criterion of Section 5.2.4(c), and a value of $5 \times n$ is given for the stopping parameter (MAXI). They were run with different tabu list sizes taking integer values from $\lceil \frac{n}{2} \rceil, \lceil \frac{n}{3} \rceil, \ldots$ to $\lceil \frac{n}{6} \rceil$ and the best of computational results are reported in Table 5.4, and Table 5.5 for the TS+FBA and the TS+BA algorithms respectively, and the use of Equations (5.12) and (5.13) for the large-sized problems. Computational results in Table 5.4 show that The TS+FBA algorithm provides the new (or previously) best known solutions - marked by ^{b} the new best or * for the equal best - for 15 problems, and gives solution costs that do not exceed 1.96% with respect to the best solutions found by the others. On the other hand, looking at the results of TS+BA+DS listed in Table 5.5, we see that the algorithm finds the new best (or best) known solutions for 11 problems. The solutions for the other do not go beyond 2.67%, except for the case of problem, N8, where the solution is obtained with an RPD of 2.02% compared to an RPD of 0.94% for the 2+FI solution. However, the 2+FI algorithm takes 5010 CPU seconds and only 888 seconds for the former.

Both tabu search schemes find the 4 new best reduced vehicle numbers marked with ^{a} in each Table. Average performance analysis demonstrates the superiority of the TS+FBA over the TS+BA+DS algorithm with respect to solution quality. The ARPD is 0.43% compared to 0.66% with an ACT of 966 seconds for the former associated with an ACT of 499 seconds for the latter. This reduction in ACT is mainly due to the special data structure we have used.

Finally, an excellent regression fit is observed for Equation (5.13) with R-squared equals to 82.5%. The coefficient values of the equation are significant with a 99% confidence level. The error in regression is due to some case where either a small (or

Table 5.4. Computational results of the TS+FBA algorithm with a first-best-admissible selection strategy

Problem number	TS+FBA solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Tabu size	Best known
G1	875*	0.00	4	107	10.89	220	22.4	10	875
CL	1205*	0.00	7	1365	36.15	1919	50.83	22	1205^b
G2	810*	0.00	4	96	5.39	404	22.69	9	810
C1	524*	0.00	5	529	61.47	981	114	13	524
C2	844	0.71	10	247	50.34	877	178.75	26	838
C3	838	0.96	8	1260	894.69	2173	1543	26	830
C4	819*	0.00	10	339	339.85	890	892.24	26	819^b
C5	1043	0.09	7	745	780.33	1380	1445.46	41	1042
C6	1044*	0.00	12	1373	1761.39	2775	3560	36	1044^b
C7	1334*	0.00	16^a	895	1703.91	1705	3246	34	1334^b
C8	555*	0.00	6	233	62.98	640	173	17	555^b
C9	911	0.21	11^a	1654	744.69	2347	1056.71	16	909
C10	878	1.36	9	1641	1964.74	2504	2998	21	866
C11	866*	0.00	11	543	581.55	1098	1175.97	34	866^b
C12	1547	0.12	11	821	1576.36	1476	2834	61	1545
C13	1184	1.26	14^a	895	2474.70	1720	4755 85	51	1169
C14	1441	1.59	18^a	968	4024.65	1097	4561	100	1418

Table 5.4. (*continued...*)

Table 5.4. Computational results of the TS+FBA algorithm with the first-best-admissible selection strategy.

Problem number	TS+FBA solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Tabu size	Best known
N1	716	0.98	6	153	38.47	543	136.56	11	709
N2	830	1.96	8	747	146.54	1189	233.25	9	814
N3	994*	0.00	10	501	87.67	919	160.83	26	994^{b}
N4	925*	0.00	7	827	540.74	1428	933.71	13	925^{b}
N5	1066	2.00	9	1234	726.39	1870	1100.77	19	1045
N6	1011*	0.00	11	718	339.36	1351	638.56	26	1011^{b}
N7	1035*	0.00	8	1762	2444.80	2621	3636.68	17	1035^{b}
N8	1185*	0.00	10	1742	1935.03	2590	2877	26	1185^{b}
N9	1234*	0.00	12	1882	1786.86	2730	2592	26	1234^{b}
Averages	-	0.43	-	-	966.15	-	1574.58	-	-

Table 5.4. (*Concluded*)

*: Indicates that the new best know route length is attained

^{a}: Indicates that the new best vehicle number is found by the tabu search with FA criterion.

^{b}: Indicates that the new best know route length is found by the tabu search with FA criterion.

Table 5.5. Computational results for the TS+BA+DS algorithm with the best admissible and the special data structure for move selection.

Problem number	TS+BA+DS solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Tabu size	Best known
G1	875^*	0.00	4	75	5.77	220	16.94	10	875
CL	1210	0.41	7	410	10.67	710	18.49	18	1205
G2	810^*	0.00	4	24	2.094	154	13.44	9	810
C1	524^*	0.00	5	278	35.38	528	67.2	11	524
C2	844	0.71	10	190	23.81	565	70.82	26	838
C3	835	0.60	8	730	400.60	1230	675	34	830
C4	819^*	0.00	10	249	127.01	799	407.57	21	819^b
C5	1042^*	0.00	7	249	127.01	799	407.57	31	1042^b
C6	1052	0.76	12	3434	2488.11	4244	3075	38	1044
C7	1354	1.49	16^a	3851	1542.22	4926	1972.74	40	1334
C8	555^*	0.00	6	381	84.65	631	140.2	13	555^b
C9	913	0.44	11^a	593	124.35	968	203	19	909
C10	866^*	0.00	9	1075	819.04	1575	1200	21	866^b
C11	866^*	0.00	11	24	413.25	324	5579	29	866^b
C12	1547	0.12	11	551	613.59	1206	1343	31	1545
C13	1188	1.62	14^a	1196	1446.09	2021	2443.61	38	1169
C14	1422	0.28	18^a	1194	1726.67	2289	3310.19	34	1418

Table 5.5. *(Continued)*

Table 5.5. Computational results for the TS+BA+DS algorithm with the best admissible and the special data structure for move selection.

Problem number	TS+BA+DS solution	RPD	Vehicle numbers	Best iteration	Time to best	Total iteration	Total time	Tabu size	Best known
N1	709[*]	0.00	6	56	11.43	306	62.5	9	709[b]
N2	814[*]	0.00	8	752	101.31	1002	135	17	814[b]
N3	1005	1.10	10	201	18.53	451	41.58	17	994
N4	946	2.27	7	224	107.16	599	286.58	76	925
N5	1045[*]	0.00	9	620	200.02	995	321	19	1045[b]
N6	1017	0.59	11	903	203.03	1278	287.35	38	1011
N7	1056	2.02	8	308	271.78	808	713	21	1035
N8	1209	2.02	10	1472	888.13	1972	1189.81	34	1185
N9	1267	2.67	12	1143	509.11	1643	731.82	17	1234
Averages		0.66			473.11		950.47		

Table 5.5. *(Concluded)*

[*] : Indicates that the new best know route length is attained

[a] : Indicates that the new best vehicle number is found by the tabu search with BA criterion.

[b] : Indicates that the new best know route length is found by the tabu search with BA criterion.

large) number of iterations are needed to obtain a good solution. Similarly, a good fit for Equation (5.12) with an R-squared value equals to 67% which is not as good as the previous one. However, the error in the estimated tabu list sizes can be reduced as suggested in Section 5.4.2b. The alternate tabu list size values are used for the large sized problems as the idea has emerged after analysis on relatively small problems is made.

5.6 Comparative analysis and conclusions

In this study, we have developed λ-interchange descent methods, and superimposed metastrategy simulated annealing and tabu search algorithms on the best of descent methods for the vehicle routing problem. The objective is to compare their performance with respect to solution quality and computation time. We tested these approaches on classical routing problems with capacity and maximum distance constraints, and on randomly generated data with capacity constraints only. The results of our investigation is listed in Table 5.6, and can be summarised as follows:

1. The constructive method of Clark *et al.* [1964] produces solutions with an ARPD of 26.65% and a total number of 170 vehicles which is about 4.3% away from the optimal number. λ-interchange descent methods with ($\lambda = 1$, and 2) improve substantially the C&W results in that the 1+FI, the 2+FI and the 1+BI algorithms have an ARPD of 10.07, 4.87 and 9.91% respectively. The best improve with approximate cost and special data structure BI+DS has reduced immensely the average computation time (ACT) of the 1+BI algorithm by 2390% with a little sacrifice in solution quality of an ARPD of 14.36%. Unfortunately, descent methods fails to reduce the vehicle numbers and uses the same published vehicles number of 167.

Table 5.6. Average relative percentage deviations, ARPD, of the starting and old best results over the new best found by metastrategy methods.

Problem number	Clark & wright		published best			New obtained best	
	solution	vehicle #	solution	RPD	vehicle #	solution	vehicle #
G1	1017	5	875	0.00	4	875	4
CL	1258	7	1214	0.74	7	1205*	7
G2	888	5	810	0.00	4	810	4
C1	625	5	524	0.00	5	524	5
C2	1005	10	844	0.71	10	838*	10
C3	982	8	832	0.24	8	830*	8
C4	939	10	829	1.22	10	819*	10
C5	1291	7	1051	0.86	7	1042*	7
C6	1299	12	1082	3.63	12	1044*	12
C7	1707	16	1387	3.97	17	1334*	16*
C8	670	6	560	0.90	6	555*	6
C9	989	12	916	0.77	12	909*	11*
C10	1055	10	885	2.19	9	866*	9
C11	952	11	876	1.15	11	866*	11
C12	1646	11	1567	1.42	11	1545*	11
C13	1383	15	1211	3.59	15	1169*	14*
C14	1671	20	1494	5.35	19	1418*	18*
ARPD	26.65	4.29 (170)	-	1.58	2.45 (167)	0.00	0.00 (163)

* : Indicates a new best route length or a new vehicle number is found.
(x) : Shows the total vehicle number used by the algorithm.

2. Simulated annealing leads to 163 vehicles, and some of the new best know solutions. However, simulated annealing shows big variabilities with regard to solution quality and computation time. The ARPD is 3.27% with an ACT to the best of 3275 seconds.

3. Both tabu search schemes with a first admissible strategy TS+FBA and a best admissible strategy TS+BA+DS, out-performs the SA algorithm in solution quality and computation time. Tabu search results are also more robust than SA. TS+FBA produces an average relative percentage deviation (ARPD) of 0.44% which is better than 0.66% of that of TS+BA+DS, but at a double of the ACT. This time reduction is due to the sophisticated data structure which can not be applied at the moment for TS+FBA. Since the difference in the ARPD's of TS schemes is not huge, we recommend the TS+BA+DS to be used rather than the dynamic tabu version of TS+FBA when the computer time is a scarce resource.

4. Good estimates for the tabu list size and the total number of iterations for tabu search schemes is established and found to depend on the characteristic of the problem data (customers, vehicles, ratio). A new dynamic approach for tabu size is introduced to reduce the error of the estimated tabu size value.

5. The published vehicle numbers of 167 (in total) is substantially larger than the new vehicles number of 163 with an RPD of 2.45% . Also, the ARPD of solutions is worse by 1.58% on average. Best solutions are found for 14 out of the 17 classical problems, and are equal in the other three as they seem to be optimal due the VRP's small sizes. A summary of results is found in Table 5.6. The largest improvements are found for problems of medium and large sizes with (and without) time constraints, e.g. the RPD of problem C14 (199 customers) is 5.35% and the new vehicle number is 18 rather than 19 for the best published and 20 for the C&W, and in problem C9 (75 customers) there is a reduction of one vehicle in addition to cost reduction. This reflects the fact that the old methods are not able to handle efficiently large sizes and extra real-life constraints.

It should be pointed out that in all developed algorithms there was extra running time involved to handle the vehicle distance constraints. The vehicle service times are evaluated in O(n) operations at each attempted move assuming that the service time would change according to the vehicle sequence visits. This is not the case here as the service time is constant throughout, and the vehicle service time can be updated in one addition and one subtraction after each move. Hence, a further reduction in the CPU can be made. Further more, it is not necessary to confine ourselves to a feasible starting solution if they are difficult to obtain. The metastrategy SA and TS would find feasible solution as shown for problems where Clark and wright initial solutions were infeasible in terms of the optimal vehicle numbers.

Chapter 6

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

6.1 Conclusion

This thesis sets out to design, develop and analyse the performance of metastrategy simulated annealing (SA) and tabu search (TS) algorithms on three, theoretically and practically, important combinatorial optimization problems: namely the capacitated clustering (CCP), the generalised assignment (GAP) and the vehicle routing (VRP) problems. These problems appear in many business applications of strategic, tactical and operational nature. They arise in situations involving location, resource allocation, logistics and distribution management systems. They have attracted extensive studies in the last two decades in an effort to provide practically applicable and theoretically sound solutions. The result is a variety of exact optimization and heuristic methods. Heuristic methods provide fast solutions for large-sized problems, with solutions that can sometimes be quiet bad. Exact optimization methods can solve only small-sized problems, and are often unable to handle any additional constraints. For the CCP, the largest problems solved with heuristic methods are of sizes up to one hundred customers. For the GAP, a few exact algorithms are available. Problems with 50 jobs can be solved exactly. Heuristic methods exist for solving problem with instances of sizes up to 500 jobs. For the VRP, (which is the most practically useful of the three) the situation is very different. VRP's involving thirty to fifty customers can be solved exactly. Heuristics exist solving VRPs with up to several hundred customers.

This thesis constitutes an effort to bridge the gap between the two extreme solution methods and presents metastrategy SA and TS algorithms, able to deliver near optimal solutions for realistic size problems within reasonable amount of computational requirements. SA and TS approaches have attracted the interest of many researchers. Many successful applications to combinatorial optimization problems were reported in the literature. However, a few comparisons regarding their relative performances were conducted objectively. In cases where such a comparison exists, the superiority of the TS algorithm over SA is claimed, but this is partly because of the use of poor annealing schemes in their trials. We think that both algorithms deserve further attention. We propose a comparison of the SA, TS and descent algorithms with the available existing methods, on the CCP, the GAP and the VRP. To our knowledge, these methods have not been applied to the above problems apart from a poor application of SA to the GAP.

The main contributions of this thesis are as follows. *First,* the neighbourhood structure and its best size are investigated. Different selection and search strategies, such as a first-improve, a best-improve, and a long term memory are implemented for the descent (local search) algorithms. Their best combinations are integrated in the metastrategy algorithms. *Second,* a number of interesting modifications are made to the basic SA technique. Hence, a new SA annealing scheme is developed which performed well on a wide range of problems. It easily out-performs all the other annealing algorithms known in the literature. The scheme has improved solutions to some of the largest problems available in the literature. The cooling schedule parameter values are derived from the problem characteristics. The best neighbourhood size and search strategies are defined. *Third,* tabu search algorithms are considered. A new TS algorithm, TS+FBA, is developed. The TS+FBA algorithm is based on a combined first and the best admissible search strategy (BA) rather than the best admissible strategy. The TS+BA algorithm which is more faithful to the original TS philosophy uses the BA strategy. For both TS algorithms, the tabu list size, | *Ts* |, is statistically defined with a good confidence interval, in relation to the problem characteristics. A way to vary | *Ts* | during the search is indicated. In addition, an estimate for the minimum number of iterations required to obtain good results is also estimated. The TS+FBA algorithm out-performed the TS+BA algorithm in both solution quality and computer time for all the three types of problem. The introduction of a good data structure for the tabu list and the search processing into the TS+BA algorithm has reduced its computational requirements by half. This data processing makes the TS+BA more competitive as it requires less CPU time with no substantial difference in solution quality.

Both metastrategy SA and TS algorithms are flexible in handling changes (constraints, objective function) in the models. They are generally applicable to any new area without problem specific knowledge requirements. They out-perform other existing methods for the above problems. The 1-intercharge 1+FI descent method improves the initial starting solutions significantly at little extra CPU time. They are only recommended if computer time is a scarce resource. The 2-interchange descent methods 2+FI require more CPU time than that of the 1+FI algorithm with a marginal improvement over the 1+FI solutions. This demonstrates the power of the proposed generation mechanism. Long term memory function has improved the starting heuristic solutions and enabled us to use the same constructive methods to generate different initial solutions.

How do TS and SA compare with each other?

For the CCP, our simulated annealing scheme and tabu search were compared on 20 test problems of sizes up to 100 customers. The SA algorithm, $H\bar{S}S$.OC, produces solutions with an average relative percentage deviation, ARPD, of 0.08% compared with 0.28% that of the TS+FBA algorithm. Each of the SA and TS algorithms requires a similar amount of the average computation time, ACT, of 179 and 181 CPU seconds, respectively.

For the GAP, the situation is reversed. Sixty test problems were solved of size up to 60-jobs and 10-agents. The ARPD of the best solutions of SA is 0.04% with an ACT of 14.26 seconds, compared to an ARPD of 0.03% with an ACT of 12.94 seconds for the TS+FBA algorithm. The TS+FBA algorithm uses a long-term strategy. However, the TS+FSA algorithm produces ARPDs of all trials (and of the worst) solutions of 0.07% (and of 0.09%) while the corresponding SA figures are 0.21% (and 0.40%). This demonstrates the robustness of the TS and the variability of SA algorithm. Both SA and TS algorithms have outperformed the results obtained by the set partitioning and SA algorithms of Cattrysse [1990] in solution quality and computation time.

For the VRP, results are reported on 26 test problems, 17 from the literature and 9 new randomly generated. The ARPD of SA is 3.37% with an ACT of 3275 seconds, compared with 0.43% and 966 seconds obtained by the TS+FBA algorithm, and with an ARPD of 0.66% and an ACT of 473 seconds for the TS+BA algorithm with the special data structure. We found 14 new best known solutions out of the 17 test problems taken from the literature. Both SA and TS algorithms have reduced the number of vehicle and the total mileages. For example, the number of vehicles used in our algorithms are 163 vehicles compared to 167 (the best published in the literature), and 170 the initial vehicle number by Clarke & Wright [1964]. The ARPD of the best published solutions is 1.58% away from the new best solutions. Most improvements have happened in large-sized problems with capacities and distance restrictions (e.g., one instance of size 199 customers with capacity and maximum distance limit has been improved by 5.35%, in addition to a reduction in the number of vehicles). The best known solutions for the other three of sized problems seem to be optimal and can not be improved.

The TS algorithms seem to be more robust. The ARPD's of the TS+FBA are always less than 0.5%, but this is not the case for the SA algorithm. Although the SA algorithm produces unstable results at a higher computation time, for some problem instances, it finds solutions better than those produced by the TS algorithm. The SA and TS algorithms are also able to deliver feasible solutions if one exits even if the starting solution is infeasible. This is demonstrated in the reduction of vehicle numbers when the initial solutions are infeasible. Both algorithms should not be looked at from a competitive angle, and our feeling is that both need to be tried when encountered with a new problem domain.

6.2 Future research

During the research reported in the thesis, new problems were encountered, and improvements were envisaged. These surely deserve further investigation:

- The special data structure, DS, for the TS+BA algorithm is implemented only to the VRP. Since the results produced by this algorithm, TS+BA+DS, are similar to that of the TS+FBA algorithm, the computation time of the TS+BA algorithm for the CCP and the GAP can be reduced using the same data structure. As the main objective is to solve large-sized problems, this saving becomes practically important.

- The computation time of the metastrategy SA and TS algorithms can be speeded up using parallel computing machines, and hence, better quality solutions could be obtained (See Aarts & Korts [1988], Malek *et al.* [1989] for further details). Research along this line must be conducted.

- Development of hybrid algorithms which can combine the best features of the simulated annealing and tabu search algorithms should be investigated. Also a combination of the metastrategy algorithms and exact procedures provides an avenue for further research.

- The SA and TS algorithms could be applied to extensions of the above problems. In particular, the many extensions of the VRP (e.g., VRP with time windows, etc.). Solving the VRP's in two stages: a cluster-first and a route-second approach could be investigated. The clustering (assignment) stage can be solved with the help of an appropriate approximation of the cost by using the CCP or the GAP models. The second stage would involve solving a TSP for each cluster. Also, problems related to the CCP such as the maximal covering location problem with capacities (Pirkul & Schilling [1991]) can be attempted by these metastrategy approaches.

References

Aarts, E. And Korts, J., (1988), *Simulated annealing and boltzmann machines* Wiley & Sons

Aarts, E.H.L., and Van Laarhoven, P.J.M., (1985), Statistical cooling: A general approach to combinatorial optimization problems, *Philips Journal of Research,* 40, 193-226.

Agarwal, Y., Mathur, K. And Salkin, H., (1989), A set partitioning based exact algorithm for the vehicle routing problem, *Networks,* 19, 731-749

Ahn S., Cooper C., Cornuejols G., and Frieze A., (1988), Probabilistic Analysis for the k-median problem, *Mathematics of Operations Research 13*, 1-31

Aho, A., Hopcroft, J., and Ullman, J. (1974), *The design and analysis of computer algorithms*, Addision-Wesley, London.

Ball, M., and Magazine, M., (1981), The design and analysis of heuristics, *Networks,* 11, 215-219.

Barcelo, J., and Casanovas J., (1984), A heuristic lagrangean algorithm for the Capacitated location problem, *European Journal Of Operational Research,* 15, 212-226

Barcia, P., and K., Jornsten, (1990), Improved Lagrangean decomposition: An application to the generalised assignment problem, *European Journal of Operational Research,* 46, 84-92

Beasley, J., (1983), Route first-cluster second methods for vehicle routing, *Omega,* 11, 403-408

Beasley, J., (1990), OR-library: distributing test problems by electronic mail, *Journal of Operational Research Society*, 41, 1069-1072.

Beasley, J.E., (1984), A note on solving large P-medians, *European Journal of Operational Research*, 21, 270-273.

Bell, W., Dalberto, L., Fisher, M., Greenfield, A., Jaikumar, R., Mack, R. And Prutzman, P., (1983), Improving distribution fo industrial gases with an on-line computerized routing and scheduling systems, *Interfaces*, 13, 4-23

Benders, J.F., and Van Nunen, J.A., (1983), A property of assignment type mixed linear programming problems, *Operations Research Letters*, 2, 47-52.

Bodin, L., Golden, B., Assad, A. And Ball, M. (1983), Routing and scheduling of vehicles and crews, The state of the art, *Computers & Operations Research*, 10, 69-211

Bohachevsky, I., Mark, J., and Stein, M., (1986), Generalized simulated annealing for function optimization, *Technometrics*, 28, 209-217.

Bonomi, E., and Lutton, J., (1984), The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm, *SIAM Review*, 26, 551-568

Bookbinder, J.H., and Reece, K., (1988), Vehicle routing considerations in distribution system design, *European Journal of Operational Research*, 37, 204-213

Brandeau, M.L., and Chiu, S.S., (1989), An overview of representative problem in location research, *Management Science*, 35, 645-674

Brown, G. And Graves, G., (1981), Real-time dispatch of petroleum tank trucks, *Management Science*, 27, 19-32

Bryant J., (1978) On the clustering of multidimentional pictorial data, *proceeding of LACIE symposium, Johnson-space centre*, Houston, Texas, 647-659

Burbank, F., (1972), A sequential space time analysis of cancer mortality in united states: etiologic implementations, *American Journal of Epdemology*, 95

Burkard, R., and F., Rendel, (1984), A thermodynamically motivated simulated procedure for combinatorial optimization problems, *European Journal of Operational Research,* 17, 169-174

Campell, H., Dudek, R., and Smith, (1970), A heuristic algorithm for the *n* job, *m* machine sequencing problem, *Management Science*, 16 B, 630-637.

Cattrysse, D., (1990), *Set partitioning approaches to combinatorial optimization problems*, Ph.D dissertation, Katholieke Universiteit Leuven, Department Werktuigkunde Afdeling Industrieel Beleid, Belgium.

Cerny, V., (1985), A thermodynamical approach to the travelling salesman problem: an efficient simulated annealing algorithm. *Journal of Optimization Theory and Applications* 45, 41-51

Chaffray, J., and Lilien, G., (1973), A new approach to industrial market segmentation, *Sloan Management Review*, 18, 41-54

Chams, M., Hertz, A., And De Werra, D., (1987), Some experiments with simulated annealing for colouring graph, *European Journal of Operational Research,* 32, 260-266.

Christofides, N., And Eilon, S., (1969), An algorithm for the vehicle dispatching problem, *Operational Research Quarterly,* 20, 309-318

Christofides, N. (1979), *The travelling salesman problem.* Christofides, N., Mingozzi, A., Sandi, C. (eds). Combinatorial Optimization, Wiley, Chichester, 131-149.

Christofides, N., Mingozzi, A., And Toth, P., (1979), The vehicle routing problem, in: N. Christofides *et al.* (eds), *Combinatorial Optimization*, Wiley, 315-338.

Christofides, N., Mingozzi, A., And Toth, P., (1981a), Exact algorithms for the vehicle routing problem, based on spanning tree shortest path relaxation, *Mathematical Programming*, 20, 255-282

Christofides, N., Mingozzi, A., And Toth, P., (1981b), Space state relaxation procedures for the computation of bounds to routing problem, *Networks,* 11, 145-164.

Christofides, N., and Beasley, J.E., (1982), A tree search for the p-median problem, European Journal of Operational Research, 10, 196-204

Christofides, N., (1985), Vehicle routing, In: Lawler, E., Lenstra, J., Rinnoy Kan, A. And shmoys, D. (eds.), *The travelling salesman problem: A guided tour of combinatorial optimization,* Wiley

Clarke, G., And Wright, J.W., (1964), Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research,* 12, 568-581

Collins, N.E., Eglese, R.W., and Golden, B.L., (1988), Simulated annealing - An annotated bibliography, *American Journal of Mathematical and Management Sciences,* 9, 209-307.

Connolly, D., (1990), An improved annealing scheme for the QAP, *European Journal of Operational Research,* 46, 93-100.

Connolly, D., (1991), *Private communication.*

Cornuejols, G., Fisher, M.L., and Nemhauser, G.L., (1977), Location of bank accounts to optimize float: an analytical study of exact and approximate algorithms, *Management Science,* 23, 789-810

Cornuejols, G., Sridharan, and Thizy, J.M., (1991) A comparison of heuristic and relaxations for the Capacitated Plant Location Problem, *European Journal of Operational Research,* 50, 280-297.

Cullen, F., Jarvis, J., and Ratiff, D., (1981), Set partitioning heuristics for interactive routing, *Networks,* 11, 125-143.

Darby-Dowman, K., and Lewis, H.S., (1988), Lagrangean relaxation and the single-source capacitated facility-location problem, *Journal of Operational Research Society,* 39, 1035-1040

De Amorim, S.G., Barthelemy, J.P., And Ribeiro, C. (1989), Clustering and clique partitioning: Simulated annealing and tabu search, *Departamento de engenharia eletrica,* Catholic Universi'ty of Rio de Janeiro.

De Bont, F., Aarts, E., Meehan, P., And O'brien, C., [1988], Placement of shapeable blocks, *Philips Journal of Research*, 43, 1-22.

Drexl, A., (1988) A simulated annealing approach to the Multi-constraint zero-one knapsack problem, *Computing*, 40, 1-8.

Elgese, R., (1986), Heuristics in Operational research, *Recent developments in operational research*, (eds Belton and O'keefe), Pergamon Press.

Eglese, R.W., and Rand, G.K., (1987), Conference seminar timetabling, *Journal of Operational Research Society*, 38, 591-598

Eglese R., (1990) Simulated Annealing: A tool for Operational research, *European Journal of Operational Research*, 46, 271-281

Evans, S. And Norback, J., (1985), The impact of a decision-support system for vehicle routing in a food service supply situation, *Journal of Operational Research Society*, 36, 467-472

Fiechter, C.N., (1990), A parallel tabu search algorithm for large travelling salesman problems, *ORWP 90/1, Ecole Polytechniques Federale de Lausanne*, Departement de Mathematiques, CH-1015 Lausanne.

Fisher M., and Jaikumar, R., (1978) *A decomposition algorithm for the large scale vehicle routing, Report* 78-11-05, Department of decision Sciences, The Wharton School, University of pennsylvania, Philadelphia.

Fisher, M., (1980), Worst-case analysis of heuristic algorithms, *Management Science*, 26, 1-15.

Fisher, M.L., And Jaikumar, R., (1981), A generalised assignment heuristic for vehicle routing, *Networks*, 11, 109-124.

Fisher M., Greenfield, R., Jaikumar, R. And Lester, J., (1982), A computerized vehicle routing application, *Interfaces*, 12, 45-52

Fisher M., Jaikumar, R., and Van Wassenhove, L., (1986) A multiplier adjustment method for the generalised assignment problem. *Management Science 32*, 1095-1103

Fisher M. (1987) Lagrangean optimization algorithms for vehicle routing problems. In *Operational Research'87, IFORS, 1988 ,(G.K. Rand Editor)* Elsiver Science Publishers, North Holland,

Fisher, M.L, and Rinnoy Kan, A.H.G., (1989), The Design, Analysis and implementation of heuristics, *Management Science*, 34, 263-265.

Garey, M.R., and Johnson, D.S., (1979), *Computers and intractability: A guide to the theory of NP – completeness*, W.H. Freeman and company, San Fransisco.

Gaskell, T., (1967), Bases for vehicle fleet scheduling, *Operational Research Quarterly*, 18, 367-384

Gavish, B., Pirkul, H., (1985), Efficient algorithms for solving multiconstraint Zero-one knapsack problems to optimality, *Mathematical programming*, 31, 78-105.

Geoffrion, A.M., (1974), Lagrangean relaxation for integer programming, *Mathematical Programming Study 2: An approach to integer programming*, editor M. Balinski, 82-114.

Geoffrion, A.M., and McBbride, R.C., (1978) Lagrangean relaxation applied to capacitated facility location problems, *AIIE Transactions*, 10, 40-71

Gidas, B., (1988) Nonstationary Markov chains and convergence of the annealing algorithm. *Journal of Statistical Physics* 39, 73-131

Gillett, B., And Miller, L., (1976), A heuristic algorithm for the vehicle dispatches, *Operations Research*, 340-349

Glover, F., (1986), Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research.*

Glover F. (1989a) Tabu search Part I, *ORSA Journal on Computing* 1:3, 190-206

Glover, F., and Greenberg, H.J., (1989b), New approaches for heuristic search: A bilateral linkage with artificial intelligence, *European Journal of Operational Research*, 39, 119-130.

Glover F. (1989c) Candidate list strategies and tabu search, *CAAI research report,* University of Colorado, Boulder

Glover, F., (1990), Tabu search Part II, *ORSA Journal on Computing* 2:1, 4-32.

Glover, F., (1990a), Tabu search: A tutorial, *Report, Centre for Applied Artificial Intelligence*, University of Colorado, Boulder.

Glover, F., And De Werra, D., (1991), The tabu search metaheuristic: How we used it, to appear in *Annals of Mathematics and Artificial Intelligence.*

Goldberg, D.E., (1989), *Genetic algorithms in search,Optimization, and Machine Learning,* Addison-Wesley.

Golden, B., And Stewart, W., (1985), Empirical analysis of Heuristics in: Lawler *et al.* (eds), *The Travelling Salesman Problem,* Wiley, 207-235.

Golden, B. And Assad, A. (1986), Perspective on vehicle routing: exciting new developments, *Operations Research,* 34, 803-810

Golden, B., And Skiscim, C., (1986), Using simulated annealing to solve routing and location problems, *Naval Research Logistics Quarterly* 33, 261-279.

Golden, B. And Watts, E., (1987), Computerized vehicle routing in the soft drink industry, *Operations Research,* 35, 6-17

Golden, B. And Assad, A. (1988), *Vehicle routing: Methods and studies* , Elsevier Science Publishers, North Holland

Guinard, M., and M., Rosenwein, (1989) An improved dual based algorithm for the generalised assignment problem. *Operations Research* 17:4, 658-663

Haines, L., (1987), The application of the annealing algorithm to the construction of exact optimal designs for linear-regression models, *Technometrics*, 29, 439-447.

Hajek, B., (1988), Cooling schedule for optimal annealing. *Mathematics of operations research,* 13, 311-329

Hansen, P., (1986), The steepest ascent mildest descent heuristic for combinatorial programming, *Presented at the congress on numerical methods in combinatorial optimization,* Capri, Italy.

Held, M., Wolfe, P., and Crowder, H.P., (1974), Validation of sub-gradient optimization, *Mathematical Programming, 6,* 62-88

Hertz, A., And De Werra, D., (1987), Using tabu search techniques for graph colouring. *Computing*, 39, 345-351.

Hoffman, K., Padberg, M., (1986) LP-based combinatorial optimization problem solving, *Annals of Operations Research,* 4, 145-194.

Holland, J.H., (1975), *Adaptation in Natural and Artificial Systems,* The University of Michigan Press, Ann Arbor.

Huang, M., Remeo, F., And Sangiovani-Vincentelli, A., (1986), An efficient general cooling schedule for simulated annealing, IEEE, *Int. Conf. on Computer aided design, Santa clara,* 381-384.

Ibraki, T., (1987), *Combinatorial optimization problems and their complexity: In enumerative approaches to combinatorial optimization,* part I, *Annals of Operations Research,* 11, 3-49.

Johnson, D.S., and Papidimitriou, C.H., (1985), *Computational complexity,* in Lawler *et al.* [1985], The travelling salesman problem: A guided tour of combinatorial optimization, Wiley, Chichester.

Johnson, M., (1988) *Simulated annealing and optimization: Modern algorithms with VLSI, Optimal design. and Missile defense applications,* a special issue of *American Journal of Mathematical and Management Sciences,* 8, Nos. 3&4.

Johnson, S., Aragon, C., Mccgeoch, L., and Schevon, C., (1989), Optimization by simulated annealing: An experimental evaluation, Part I, Graph Partitoning. *Operations Research* 37, 865-892

Joneker, R., And Volgenant, T., (1982), Identifications of non-optimal arcs for the travelling salesman problem, *Operations Research Letters*, 1, 85-88.

Jörnsten, K.O, and Näsberg, M., (1986), A new lagrangean relaxation approach to the generalised assignment problem, *European Journal of Operational research,* 27, 313-323.

Karp, R.M., (1977), Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane, *Mathematics of Operations Research,* 2, 209-224

Kelly, J., Golden, B., And Assad, A., [1990], Using simulated annealing to solve controlled rounding problems, *ORSA* Journal on computing, 2, 174-185.

Kernighan, B., And Lin, S., (1970), An efficient heuristic procedure for partitioning graph, *Bell System Technical Journal*, 49, 291-307.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, P.M., (1983), Optimization by simulated annealing, *Science, 220, 671-680.*

Kirkpatrick S. (1984), Optimization by simulated annealing; Qualitative studies, *Journal of Statistical Physics,* 34, 974-986

Klastorin, T.D., (1979), An effective sub-gradient algorithm for the generalised assignment problem, *Computers and Operations Research*, 6, 155-164.

Klincewicz, J.G., and Luss, H., (1986), A lagrangean relaxation heuristic for the capacitated facility location with single constraints, *Journal of Operational Research Society,* 37, 495-500

Klincewicz J.G., H. Luss, and Yu, C.S., (1988), A large multi-location capacity planning model, *European Journal of Operational Research*, 34, 178-190

Knox, J., And Glover, F., (1989), Comparative testing of travelling salesman heuristics derived from tabu search, Genetic algorithms and Simulated annealing., *Centre for • Applied Artificial Intelligence,* University of Colorado (September).

Koontz Warren L.G., Narendra Patrenahall, M. and Keindsuke Fukunaga (1975), A branch and bound clustering algorithm, *IIE transactions on computers,* 24, 908-915

Krolak, P.D., Flets, W., And Marble, G., (1971), A man-machime approach towards solving the travelling salesman problem, *Communications ACM,* 14, 327-334.

Krolak, P.D., Flets, W., And Nelson, J.H., (1972), A man-machime approach towards solving the generalised truck dispatching problem, *Transportation Science,* 6, 149-170.

Kuik, R., And Solomon, M., (1990), Multi-level lot-sizing problem: Evaluation of a simulated-annealing heuristic, *European Journal of Operational Research,* 45, 25-37.

Lagauna, M., Branes, J.W., And Glover, F., (1989), Scheduling jobs with linear delay penalties and sequence dependent set-up costs using tabu search, *Research report, Department of Mechanical Engineering,* The university of Texas-Austin.

Laporte, G., Nobert, Y. And Desrochers, M., (1985), Optimal routing under capacity and distance restrictions, *Operations Research,* 33, 1050-1073

Laporte, G. And Nobert, Y., (1987), Exact algorithms for the vehicle routing problem, *Annals of Discrete Mathematics,* 31, 147-184

Lawler, E.L. (1976), *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York.

Lenstra, J., And Rinnoy Kan, A., (1981), Complexity of vehicle routing and scheduling problems, *Networks,* 11, 221-228

Lin, S. (1965), Computer solutions of the travelling salesman problem, *Bell System Technical Journal,* 44, 2245-2269.

Lin, S., And Kernighan, B., (1973), An efficient heuristic for the travelling salesman problem, *Operations Research,* 21, 498-516.

Lin, S., (1975), Heuristic programming as an aid to network design, *Networks,* 5, 33-43.

Lundy, S., and Mees, A., (1986), Convergence of an annealing algorithm. *Mathematical Programming* 34, 111-124.

Malek M., Gunrunswamy, M., Owens, H., and Pandya, M., (1989), Serial and parallel search techniques for the travelling salesman problem, *Annals of OR: Linkages with Artificial Intelligence,* 21, 59-84.

Martello, S. and Toth, P., (1981), An algorithm for the generalised assignment problem *In proceedings of the 9th IFORS conference,* Hamburg, Germany

Martello, S., and Toth, P., (1990), *Knapsack problems: Algorithms and computer implementations,* Weily

Masson, E., Wang, Y., (1990), Introduction to computation and learning in artificial neural networks, *European Journal of Operational Research,* 47,1-28.

Metropolis, W., Roenbluth, A., Rosenbluth, M., teller, A., and Teller, E., (1953), Equation of the state calculations by fast computing machines, *Journal of Chemical Physics,* 21, 1087-1092

Michael, G., (1972), A review of heuristic programming, *Decision Science,* 3, 74-100.

Mirzaian A., (1985), Lagrangean relaxation for the start-star concentrator location problem: approximation algorithm and bounds, *Networks,* 15, 1-20

Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A., (1986), Convergence and Finite time Behaviour of Simulated Annealing, Advance Applied Probability, 18, 747-771.

Muller-Merbach, H., (1981), Heuristic and their design: a survey, *European Journal of Operational Research,* 8, 1-23.

Muller-Merbach, H., (1984), A five facets frame for the design of heuristics, *European Journal of Operational Research,* 17, 313-316.

Mulvey, J.M., and Crowder, H.P. (1979), Cluster analysis: An application of lagrangean relaxation, *Management Science,* 24, 329-340.

Mulvey, J.M., and Beck, M.P., (1984), Solving capacitated clustering problems, *European Journal of Operational Research,* 18, 339-348

Nawaz, M., Enscore, E., And Ham, I., (1983), A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega,* 11, 91-95.

Neebe, A.W., and Rao, M.R. (1983), An algorithm for the fixed charge assigning users to sources problem, *Journal Of Operational Research Society,* 34, 1107-1113

Nemhauser, G.L. and Wolsey, L.A. (1988), *Integer and Combinatorial Optimization.* John Wiley, Chichester.

Nicholson, T., (1971), *Optimization in industry, Vol. 1, Optimization techniques,* Chapter 10, Longman press, London

Orloff, C., (1976), Route constrained fleet scheduling, *Transportation Science,* 10, 149-168.

Osman, I.H., (1987), *Application of simulated annealing to flow-shop scheduling problems,* Msc dissertation in O.R., Faculty of Mathematical Studies, Southampton, England.

Osman, I.H., and Potts, C.N., (1989), Simulated annealing for permutation flow-shop scheduling, *Omega* 17, 551-557

Osman, I.H., Christofides, N., (1989), *Simulated annealing and descent algorithms for capacitated clustering problems,* presented at EURO X, Beograd, Yugoslavia.

Osman, I.H., Christofides, N., (1990), *Simulated annealing and tabu search techniques for the generalised assignment problem*, presented at the 12th triennial conference on operations research of the international federation of operations research societies, IFORS'90, Athens, Greece.

Padberg, M., And Rinaldi, G., [1987], Optimization of a 512-city symmetric travelling salesman problem by branch and cut, *Operations Research Letters*, 6, 1-7.

Paessens, H., (1988), Saving algorithms for the vehicle routing problem, *European Journal of Operational Research*, 34, 336-344

Palmer, S., (1965), Sequencing jobs through a multistage process in the minimum total time- a quick method of obtaining a near optimum, *Operational Research Quarterly*, 16, 101-107.

Papadimitriou, C.H., And Steiglitz, K., (1982), *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, New York.

Pirkul, H., and Schilling, D.A., (1991) The maximal covering location problem with capacities on total workload, *Management Science*, 37, 233-248.

Polya, G., (1948), *How to solve it*, Princeton university press, princeton.

Rao, M.R., (1971), Clustering analysis and mathematical programming, *Journal of the American Statistical Association*, 66, 622-626

Ross, G.T., Soland, R.M., (1975), A branch and bound algorithm for the generalised assignment problem, *Mathematical Programming*, 8, 91-103.

Ross, G.T., Soland, R.M., (1977), Modelling facility location problems as generalised assignment problems, *Management Science*, 24, 345-357.

Russel, R.A., (1977), An effective heuristic for the M-tour travelling salesman Problem with some side conditions, *Operations Research, 25, 5217-524*

Silver, A., Vidal, R., And De Werra, D., (1980), A tutorial on heuristic methods, *European Journal of Operational Research*, 5, 153-162.

Skorin-Kapov, J. (1990), Tabu search to the quadratic assignment problem, *ORSA Journal on Computing* 2:1, 33-44.

Stanfel, L.E., (1986), A lagrangean treatment of certain nonlinear clustering problem, *European Journal of Operational Research,* 27, 332-342

Taillard, E., (1990), Some efficient heuristic methods for the flow shop sequencing problem, *European Journal of Operational Research,* 47, 65-74.

Tarajan, R., [1983], *Data structures and network algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, 44, Siam, Philadelphia.

Teitz, M., and Bart, P., (1968), Heuristic methods for estimating the generalised vertex median of a weighted graph, *Operations Research,* 16, 955-961

Vakharia, A., And Chang, Y., (1990), A simulated annealing approach to scheduling a manufacturing cell, *Naval Research Logistics,* 37, 559-577.

Van Laarhoven, P., And Aarts, E., (1987), *Simulated annealing: Theory and applications,* D. Reidel, Dordrecht

Whitaker, R.A., (1983), A fast algorithm for the greedy interchange for large scale clustering and median location problems, *Canadian Journal Of Operational Research and Information Processing,* 21, 95-108

White, S.R., (1984), Concept of scale in simulated annealing, *IEEE international conference on computer design,* Portchester, (1984) 646-651

Widmer, M., and Hertz, A., (1989) A new heuristic for the flow shop sequencing problem, *European Journal of Operational Research,* 41, 186-193.

Widmer, M., (1991), Job shop scheduling with tooling constraints: a tabu search approach, *Journal of Operational Research Society,* 42, 75-82.

Wilhem, M.R., And Ward, T.L., (1987), Solving quadratic assignment problems by simulated annealing, *IIE Transactions* 107-119.

Wright, M.B, (1989), Applying stochastic algorithms to a locomotive scheduling problem, *Journal of Operational Researchn Society,* 40, 187-1192.

Zanakis, H., and Evans, J.R., and Vazacopoulos, A., (1989), Heuristic methods and applications: A categorized survey, *European Journal of Operational Research*, 43, 88-110.

Appendix A.1

This 1.2 MB diskette contains 3 subdirectories in which there are the input data for the test problems, we use in the thesis. Also, there are the output files for the best obtained solutions of these problems. These are produced by the algorithms which are developed in different chapters.

Each subdirectory also contains input data files (called XXX.IN) and output results files (XXX.RES) for each of the test problems.

1. Subdirectory \CCP.

" The Capacitated Clustering Problem ". There are 57 files in this subdirectory of different types:

Input data files :	CCPDXX.IN (for distance data).
Input data files :	CCPXXX.IN (for coordinate points).

Output result file:	CCPXXX.RES

2. Subdirectory \GAP.

" The Generalised Assignment Problem ". There are 60 files in this subdirectory.

Input data files : CXXXXXX.IN

3. Subdirectory \VRP.

" The Vehicle Routing Problem ". There are 57 files in this directory.

Input data files :	CXX.IN (for the test problem in the literature).
Input data files :	NX.IN (for the new random problems).
Output result files:	CXX.RES or NX.RES.

Appendix A.2

This appendix explains the data sets in the subdirectory \CCP for the CCP. They are the 20-test randomly generated problems which we use in Chapter 3. In the thesis, these test problems are numbered from 1 to 20. On the diskette we add this number as a prefix and suffix. For each problems say number *i*, there are three files: the first two are for input data files: CCPDi.IN is a file for the inter-customer distances, CCPXi.IN is a file for the (X,Y) coordinates, and CCPi.RES is an output result file which represents the best solution obtained fo problem *i*.

We will explain the data storage of the CCPD1.IN file. The data in this file is formatted as follows. We define a *flag* to take the value of 0 if the file stores the inter-distances, or 1 if the file contains the (X,Y) coordinates. We define *j* to represent a job number *j*.

0 50 5 120	: [*flag*=0], [n= 50], [p=5], [$Q_1 = 120$]
1 4	: [j= 1], [$d_1 = 4$]
2 16 34	: [j= 2], [$c_{21} = 16$], [$d_2 = 34$]
3 6 56 43	: [j= 3], [$c_{32} = 6$], [$c_{31} = 56$], [$d_3 = 43$]
4 16 43 89 86	
5 12 39 63 82 94	
6 19 35 30 27 59 61	
7 6 42 56 17 45 98 88	
8 9 63 67 102 72 41 85 55	
9 4 84 103 65 90 95 60 9 30	
10 12 17 77 87 48 73 78 46 10 31	

11 1 26 42 62 61 23 55 52 24 36 39

12 9 49 54 57 29 78 64 100 81 38 59 27

13 4 91 48 68 85 92 49 27 9 31 54 77 87

14 11 42 110 61 66 80 121 90 55 37 73 80 71 95

15 12 75 44 50 28 53 68 48 34 20 53 31 12 64

55

16 14 21 86 62 29 26 45 56 35 52 35 71 52 9

54 36

17 2 19 29 102 73 30 45 64 74 19 49 48 83 55

22 72 49

18 7 24 6 18 80 57 34 20 41 54 42 51 30 65

49 6 51 37

19 3 52 74 58 46 28 23 84 35 49 65 93 65 26

25 49 51 57 73

20 5 15 67 88 73 60 16 26 99 50 61 77 107 74

40 22 57 66 68 87

21 1 68 58 56 78 60 69 70 80 65 41 15 11 91

103 62 84 93 62 6 39

22 15 83 22 37 89 111 96 83 13 44 121 72 79 93

130 93 63 37 75 89 84 107

23 11 33 57 12 7 57 80 63 53 22 28 88 39 49

65 98 72 33 29 55 57 56 75

24 18 7 29 64 6 8 60 82 67 54 21 23 93 44

56 72 101 70 34 22 53 60 63 81

25 12 57 59 80 94 61 52 47 51 50 29 77 36 78

53 79 95 68 13 31 44 5 41 89 84

26 15 48 84 82 113 84 90 76 29 6 24 29 104 73

36 49 69 80 19 45 50 82 52 25 78 55

27 5 97 76 20 17 27 56 19 24 71 94 77 69 14

44 99 51 52 66 113 89 50 41 72 72 57 83

28 20 61 80 93 67 60 87 6 72 61 53 74 56 67

74 81 60 40 14 5 86 101 62 86 92 59 4 33

29 10 74 46 62 30 27 29 52 73 31 22 46 62 51

33 47 11 80 38 60 78 81 43 16 20 26 42 70 77

30 11 45 78 18 104 74 20 23 9 74 14 28 80 102

87 74 5 39 112 62 69 83 121 87 54 33 70 80 74

97

31 19 30 48 49 12 92 77 26 20 39 44 27 25 64

88 70 65 26 48 90 43 41 54 106 89 48 48 74 66

45 72

32 9 62 75 38 58 67 30 39 55 52 84 61 61 47

9 27 13 9 75 49 42 21 45 60 46 44 22 58 40

5 55 45

33 19 44 52 45 7 81 48 67 29 28 32 50 80 32

26 52 67 57 38 48 6 87 45 67 85 86 42 23 15

25 49 77 84

34 14 41 14 48 64 34 48 54 43 45 44 41 73 49

51 36 16 40 22 20 63 44 48 8 34 50 58 53 18

52 45 17 44 43

35 13 32 29 44 24 31 24 56 24 73 53 13 8 41

54 19 5 48 71 55 44 31 27 80 31 44 61 90 65

25 30 50 48 52 68

36 15 9 40 26 51 26 24 24 64 22 80 54 4 6

33 62 10 4 56 78 62 50 25 22 89 40 53 69 97

67 30 23 50 55 60 77

37 9 68 60 28 67 24 71 91 60 49 80 31 62 72

68 101 54 79 64 15 25 12 32 90 71 20 29 40 48

37 64 44 80 64 20 48 24

38 10 20 89 80 48 86 42 91 112 80 61 100 34 77

93 89 122 67 99 84 34 28 28 49 111 91 2 49 56

59 27 76 64 100 80 37 61 29

39 13 28 13 65 55 28 69 30 63 86 62 36 73 44

69 68 63 96 41 75 60 22 38 22 39 84 72 27 25

27 35 50 74 46 79 70 28 35 15

40 14 63 87 67 1 8 39 26 49 26 25 23 64 23

78 53 5 6 34 61 12 3 55 77 61 49 26 22 87

38 52 68 95 66 29 23 49 54 60 76

41 20 68 55 53 44 69 65 41 52 28 87 93 48 87

90 21 28 73 73 101 89 79 66 33 25 32 22 94 58

55 49 74 87 40 24 40 67 33 27 84 68

42 10 87 28 61 89 70 28 25 47 53 61 3 33 49

46 15 91 78 28 21 42 41 30 27 63 87 69 65 29

50 88 42 38 51 105 90 48 50 75 65 42 70

43 16 23 67 31 38 66 46 32 23 26 48 40 25 49

42 33 35 69 65 35 28 59 31 40 28 40 65 46 45

47 48 65 19 21 37 81 75 33 52 63 43 29 48

44 13 50 35 89 23 85 111 90 21 30 62 40 72 32

5 41 80 21 101 70 17 22 11 76 11 26 78 100 84

71 10 34 110 61 70 85 119 83 51 28 65 77 76 97

45 11 18 54 47 77 23 84 106 86 22 31 58 25 65

44 24 29 87 36 91 54 19 27 27 84 17 26 72 91

```
78 62 28 18 106 59 75 91 110 67 43 10 49 70 83
98
46 14 51 46 5 18 71 28 43 71 51 29 21 30 48
44 20 45 42 35 30 73 67 32 25 55 32 37 25 44
69 50 49 42 47 70 24 24 40 86 78 35 51 65 47
31 52
47 13 48 28 42 48 54 49 27 67 84 64 27 29 39
2 41 53 47 5 80 50 64 27 30 33 53 79 34 27
50 65 55 36 50 9 84 43 66 84 84 40 21 18 23
46 76 82
48 7 79 33 85 78 32 43 88 62 38 64 51 63 55
48 80 59 46 76 73 2 58 82 93 65 58 85 3 70
59 54 76 58 67 72 80 62 40 14 8 88 101 62 85
92 60 3 36
49 18 26 57 9 59 52 10 19 78 37 43 71 53 37
29 36 57 50 22 50 51 28 33 78 75 39 32 60 24
44 34 49 73 54 56 47 56 70 29 19 33 89 86 43
60 74 53 24 50
50 5 79 84 60 73 87 98 69 90 14 76 48 41 36
77 71 43 62 29 91 101 58 83 95 7 42 81 80 109
87 87 73 30 12 26 26 102 69 42 50 71 83 25 38
47 78 46 26 81 60
```

Note that, if more than fifty entries exist per customer, the data is continued on other lines.

The file CCPX1.IN has the same first row as in CCPD1.IN. However, the coordinate values start from the second row up to row number 51. Each row of data contains four entries: first, the customer number i; second, the demand of the customer; third, the X-coordinate and the fourth is the Y-coordinate.

The file CCP1.RES is the solution representation of problem number 1. The content of this file is shown below.

```
wpt=  552, wct=    600,  RATIO  0.92,  PR=         1
```

$[\sum_{j=1}^{50} d_j]$, $[\sum_{i=1}^{5} Q_i]$, $[\rho = \frac{\sum_{j=1}^{50} d_j}{\sum_{i=1}^{5} Q_i}]$, $[i=1]$

S 820 (*S is the best obtained solution with an objective value of 820 units*)

cen,	cost,	num,	lwc,	elements of clusters
ζ_i,	$C(S_i)$,	$\lvert S_i \rvert$,	$Q_i - \sum_{j \in S_i} d_j$	List of jobs $j \in S_i$ assigned to cluster i.
16	249	13	10	3 8 37 16 17 32 26 18 12 38 50 34 11
30	124	8	25	14 22 30 44 31 42 45 27
2	154	10	9	2 9 10 21 28 48 49 43 39 1
25	162	8	3	7 4 25 47 33 41 6 15
40	131	11	1	46 5 23 19 36 40 13 35 24 29 20

Appendix A.3

This appendix is concerned with the set of data for the GAP in the subdirectory \GAP. There are 60 test problems on the floppy diskette under the GAP directory. These problems were solved in Chapter 4. The problem notations are explained as follows:

c515-1 : denotes a problem of type *c* with 5-agents and 15-jobs, problem number 1 in a group of 5 problems of the same sizes.

c1060-1 : indicates 10-agents, and 60-jobs, problem number 1 in 5 problems of the same type.

The data of a problem is stored in the following order:

3 7	number of agents ***m***, number of jobs ***n***
21 25 23 10 29 34 21	c_{1j} (costs or revenues)
23 13 24 15 23 15 16	c_{2j}
34 12 34 25 12 23 23	c_{3j}
10 23 12 21 21 20 12	a_{1j} (capacity requirements)
34 12 33 44 12 12 13	a_{2j}
13 13 14 12 15 13 12	a_{3j}
120 120 120	b_1 b_2 b_3 (capacity of agents)

To transform a maximization problem into a minimization problem, we need to modify the input data sets as explained in Section 4.5.1. This transformation is briefly summarised as follows:

Let $C_{ij} = 35 - c_{ij}$ (same distribution of coefficients [10,25]) where 35 is an upper bound on the values of c_{ij}'s. The created minimization problem is then solved by the minimization algorithms that were developed in Chapter 4. The maximization problem is then has a maximum objective value, Z_{max}, which can be obtained by simply letting $Z_{max} = 35 \times n - Z_{min}$ where Z_{min} is the objective value of the minimization problem.

The maximum solution values (Z_{max}'s) of the 60 problems are given below for all the 60 test problems. This is explained as follows: problem c515-1 with an optimal value of 336 is written as c515-1 336:

c515-1	336	c520-1	434	c525-1	580	c530-1	656
c515-2	327	c520-2	436	c525-2	564	c530-2	644
c515-3	339	c520-3	420	c525-3	573	c530-3	673
c515-4	341	c520-4	419	c525-4	570	c530-4	647
c515-5	326	c520-5	428	c525-5	564	c530-5	664
c824-1	563	c832-1	761	c840-1	942	c848-1	1133
c824-2	558	c832-2	759	c840-2	949	c848-2	1134
c824-3	564	c832-3	758	c840-3	968	c848-3	1141
c824-4	568	c832-4	752	c840-4	945	c848-4	1117
c824-5	559	c832-5	747	c840-5	951	c848-5	1127
c1030-1	709	c1040-1	958	c1050-1	1139	c1060-1	1451
c1030-2	717	c1040-2	963	c1050-2	1178	c1060-2	1449
c1030-3	712	c1040-3	960	c1050-3	1195	c1060-3	1433
c1030-4	723	c1040-4	947	c1050-4	1171	c1060-4	1447
c1030-5	706	c1040-5	947	c1050-5	1171	c1060-5	1446

Appendix A.4

This appendix explains the files in the subdirectory \VRP. The input data files are used in Chapter 5. There are the 17-test problems in the literature. In addition to, there is also 9 problems of random data we have generated. The name of the problems in the diskette, are exactly the same as in the thesis.

There are two types of input data files as in the case of Appendix A.2. We will explain first one of the coordinate data file of the problem G1.IN. The data in the file are the following. Let i denote customer number i. We define a *flag* to take the value of 0 if the file stores the inter-distances, or 1 if the file contains the (X,Y) coordinates. We define j to represent a customer number j. L denote the maximum distance limit and δ_i be the drop time per customer number i. Here , we have all δ_i have the same value.

1 29 240 4500 10	: [*flag*=1], [n= 29], [L = 240], [Q = 4500, [δ_i = 10].
162 354	: [X_{30} = 162], [Y_{30} = 354], (X,Y) coordinates of the depot "30".
1 218 382 300	: [i =1], [X_1 = 218], [Y_1 = 382], [d_1 = 300]
2 218 358 3100	
3 201 370 125	
4 214 371 100	
5 224 370 200	
6 210 382 150	
7 104 354 150	
8 126 338 450	
9 119 340 300	

```
10 129 349  100
11 126 347  950
12 125 346  125
13 116 355  150
14 126 335  150
15 125 355  550
16 119 357  150
17 115 341  100
18 153 351  150
19 175 363  400
20 180 360  300
21 159 331 1500
22 188 357  100
23 152 349  300
24 215 389  500
25 212 394  800
26 188 393  300
27 207 406  100
28 184 410  150
29 207 392 1000
```

The output data file of G1.RES is show below:

PROBLEM G1.RES (*name of result file)*

No. OF CUST.= 29, VEH. No.= 5, VEH CAP =4500, VEHDIST 240,

BEST COST 875.0 (*the best obtained objective value*)

V.No	BRCST	LEFTW	NLC	RLIST
1	233.95	1650	7	30 26 28 27 25 24 29
2	000.00	4500	1	30 *One vehicle reduction*
3	236.59	125	9	30 22 2 5 4 1 6 3 20
4	177.24	1700	6	30 23 8 14 21 19
5	227.21	1775	11	30 15 16 13 7 17 9 12 11 10 18

Note that, the empty route in row 2 is included. This is to show that a vehicle reduction has occurred. The savings heuristic procedure produced a solution with 5 vehicles rather than the optimal number of 4.

The column headings are:

V.No	is the vehicle index
BRCST	is the length of the corresponding tour
LEFTW	is the unused capacity of the vehicle
NLC	is the number of customer in the route including depot
RLIST	is the list of customer assigned to the vehicle.

The other data files are for problems with the inter-customer distances are given. Let consider the data in the input file N1.IN for problem N1.

0 50 9999 275 0	: [$flag$= 0], [n= 50], [L = 9999], [Q = 275, [δ_t = 0].
1 21 41	:i =1, [$c_{1,51}$ = 21], [d_1 = 41]
2 33 49 65	: i= 2, [$c_{2,1}$ = 33], [$c_{2,51}$ = 49], [d_2 = 65]
3 20 49 87 26	: i= 3, [$c_{3,2}$ = 20], [$c_{3,1}$ = 49], [$c_{3,51}$ = 87], [d_2 = 26]

4 37 43 78 81 81

5 24 50 78 1 88 27

6 33 37 37 44 37 87 52

7 37 44 32 69 16 68 89 77

8 29 39 37 5 32 49 32 89 49

9 37 16 53 46 49 66 38 65 43 56

10 23 46 48 35 4 30 67 20 67 92 77

11 39 48 62 64 27 64 32 6 73 6 88 31

12 38 60 92 105 57 97 103 96 91 96 90 17 65

13 35 27 33 66 72 26 65 70 64 66 64 65 23 45

14 28 31 6 31 72 74 26 70 71 68 72 65 71 18

52

15 24 42 53 55 82 85 42 30 65 38 60 88 23 88

65 84

16 39 50 92 62 56 75 27 81 65 50 82 54 22 87

21 75 10

17 28 42 72 48 72 69 102 52 10 48 25 12 20 57

27 57 90 69

18 30 37 74 76 45 14 20 37 83 74 26 77 71 74

84 61 83 20 66

19 29 17 24 49 65 30 25 25 56 65 50 1 54 47

50 67 39 66 42 56

20 33 47 61 85 26 55 70 78 73 105 30 36 60 8

39 11 36 53 36 97 55

21 24 26 57 18 39 38 76 16 43 43 74 69 35 17

51 32 46 72 21 72 59 68

22 36 39 13 66 29 45 44 89 4 52 54 82 81 38

28 61 34 56 85 19 84 66 81

23 27 37 73 60 33 54 71 50 22 77 60 54 80 12

59 54 28 61 32 12 68 11 76 21

24 39 36 73 12 15 72 21 33 52 87 11 42 44 70

82 49 21 66 45 61 85 31 84 54 78

25 33 55 23 92 33 38 95 38 32 75 103 29 46 51

67 102 71 38 88 67 84 105 52 104 50 93

26 21 50 105 85 15 83 71 30 68 86 53 28 87 75

69 95 3 63 67 28 65 33 6 75 7 91 33

27 25 50 97 37 44 82 56 52 97 37 13 87 84 55

22 28 30 94 87 39 89 84 87 94 73 93 14 74

28 28 59 89 36 110 95 32 99 85 64 74 82 82 10

102 68 62 77 35 91 74 60 92 64 30 97 29 79 17

29 35 41 31 86 11 96 77 4 77 64 33 58 75 52

21 80 64 58 84 8 61 58 29 63 33 8 71 7 80

23

30 31 67 89 81 35 101 72 76 86 88 80 111 62 42

109 79 88 37 39 6 98 111 64 103 109 103 96 102 95

22 70

31 39 36 31 62 62 27 74 56 52 59 62 52 81 33

23 78 58 63 10 8 24 71 81 35 73 79 72 70 73

69 18 47

32 30 26 11 42 51 54 36 62 59 50 47 58 47 69

29 28 68 48 59 14 8 35 59 72 30 61 70 61 59

67 58 29 37

33 24 45 20 9 22 69 65 28 80 61 60 65 70 61

89 43 29 87 62 71 18 18 15 77 90 44 81 88 81

76 82 75 15 52

34 35 49 54 51 49 66 90 103 31 99 7 20 86 32

34 91 31 25 73 96 29 39 44 61 97 69 32 84 65

80 99 51 98 44 86

35 35 58 68 103 84 94 125 61 92 103 59 90 67 60

57 53 28 66 90 16 82 61 88 85 118 58 19 64 32

23 28 64 39 64 106 80

36 32 39 97 54 6 13 5 28 62 60 30 74 61 57

59 67 57 83 38 28 81 56 68 15 13 22 71 85 39

75 83 74 70 78 69 19 46

37 30 36 74 32 56 79 62 70 100 61 88 74 65 58

35 58 26 22 42 37 61 18 78 30 62 61 93 63 13

36 39 10 33 68 10 67 79 72

38 23 37 45 68 42 84 75 55 67 97 19 50 86 20

89 68 18 65 53 16 53 74 34 40 68 65 60 91 18

43 52 10 45 14 23 55 23 84 39

39 22 36 9 51 64 51 85 70 51 63 92 10 41 84

14 91 70 9 69 57 24 53 73 42 31 73 62 57 86

12 51 52 19 53 23 16 62 15 80 30

40 26 50 83 85 77 25 105 37 22 33 23 29 84 85

6 95 43 48 80 61 55 97 39 15 89 81 60 20 26

24 92 89 41 89 86 87 92 77 91 8 70

41 37 40 78 56 49 4 77 33 55 82 65 73 102 65

92 75 69 57 35 63 24 23 46 39 63 20 82 28 64

64 96 67 14 38 43 10 37 72 6 72 81 77

42 27 48 25 95 45 36 23 87 10 78 93 74 84 115

55 86 93 55 81 58 54 49 44 26 56 80 6 76 53

```
78 75 109 54 11 54 27 15 22 60 31 59 97 73
43 26 44 6 19 90 45 37 17 83 15 72 89 70 80
110 55 85 88 56 75 52 53 43 38 29 50 75 3 75
47 73 71 104 55 7 49 28 10 23 61 25 60 91 72
44 20 55 75 74 84 89 29 38 80 65 80 102 71 56
66 88 19 13 92 23 109 92 21 93 80 52 71 84 72
9 97 70 64 84 22 81 71 48 82 52 16 89 16 84
19
45 35 36 90 43 49 25 57 66 63 26 63 58 30 66
54 58 84 74 95 53 81 32 10 70 4 9 65 25 42
44 86 6 48 50 78 79 39 24 60 35 55 82 21 81
62 78
46 26 38 62 65 82 87 76 25 63 68 73 1 96 53
7 12 4 29 62 60 30 73 60 56 58 66 56 82 37
27 80 56 67 14 12 23 71 84 38 74 82 73 69 77
68 19 46
47 22 10 46 27 63 37 42 30 53 41 39 27 46 52
44 52 34 43 73 48 68 51 56 49 29 44 30 17 47
14 38 36 59 33 36 34 67 53 38 13 40 36 36 56
33 55 54 51
48 35 45 43 40 38 95 76 82 62 23 81 82 61 41
92 14 41 41 36 52 85 94 17 95 20 28 81 41 39
90 29 12 76 88 39 27 33 47 92 74 30 83 71 80
93 59 92 30 77
49 23 25 35 32 13 50 61 69 73 62 29 54 57 60
14 83 46 20 5 10 41 55 59 31 66 53 45 51 54
43 71 25 23 67 53 55 9 3 35 63 70 26 63 68
```

```
61  63  64  62  26  43
50  22  51  26  34  56  18  65  83  94  99  84  12  80  84
82  18  108  48  12  29  17  18  80  77  18  91  54  57  76
69  62  98  44  24  92  74  69  20  24  13  88  94  46  89
92  88  87  84  86   4  64
```

The output file is N1.RES and its abbreviation are the same as in G1.RES file above.

```
PROBLEM No1.RES
No. OF CUST.=  50 VEH. No.=  6VEH CAP = 275
BEST COST  709.0
V.No BRCST  LEFTW  NLC  RLIST
 1   111.00   32  10   12 20 23 26 43 33 21 51 47 28
 2   146.00    8  10   22 40 38 32 51 37 27 25 42  7
 3    88.00   13  10    6 11  8  2  5 36  3 10 19 51
 4   113.00    6   9   50 48 30 45 51  9 34 41  4
 5   128.00   46   8   24 29 18 35 49 17 31 51
 6   123.00   60   9   39 44 16 15 46 13 51 14  1
```