Solving the generalized assignment problem: a hybrid Tabu search/branch and bound algorithm

SOLVING THE GENERALIZED ASSIGNMENT PROBLEM: A HYBRID TABU SEARCH / BRANCH AND BOUND ALGORITHM

by

Andrew John Woodcock

Doctoral thesis

submitted in partial fulfilment of the requirements

for the award of Doctor of Philosophy

of Loughborough University

25th September 2007

Table of Contents

List of Figures

List of Tables

Abstract

The research reported in this thesis considers the classical combinatorial optimization problem known as the Generalized Assignment Problem (GAP). Since the mid 1970's researchers have been developing solution approaches for this particular type of problem due to its importance both in practical and theoretical terms. Early attempts at solving GAP tended to use exact integer programming techniques such as Branch and Bound. Although these tended to be reasonably successful on small problem instances they struggle to cope with the increase in computational effort required to solve larger instances. The increase in available computing power during the 1980's and 1990's coincided with the development of some highly efficient heuristic approaches such as Tabu Search (TS), Genetic Algorithms (GA) and Simulated Annealing (SA). Heuristic approaches were subsequently developed that were able to obtain high quality solutions to larger and more complex instances of GAP. Most of these heuristic approaches were able to outperform highly sophisticated commercial mathematical programming software since the heuristics tend to be tailored to the problem and therefore exploit its structure. A new approach for solving GAP has been developed during this research that combines the exact Branch and Bound approach and the heuristic strategy of Tabu Search to produce a hybrid algorithm for solving GAP. This approach utilizes the mathematical programming software Xpress-MP as a Branch and Bound solver in order to solve sub-problems that are generated by the Tabu Search guiding heuristic. Tabu Search makes use of memory structures that record information about attributes of solutions visited during the search. This information is used to guide the search and in the case of the hybrid algorithm to generate sub problems to pass to the Branch and Bound solver. The new algorithm has been developed, implemented and tested on benchmark test problems that are extremely challenging and a comprehensive report and analysis of the experimentation is reported in this thesis.

Keywords: Integer programming, Tabu Search, Branch and Bound, Generalized Assignment Problem, heuristic.

Acknowledgements

I would firstly like to thank Professor John Wilson for providing me with the opportunity to undertake this research. John's supervision, friendship, support, knowledge, advice, encouragement and patience throughout have been invaluable. I would also like to thank Dr Alan French for his contribution as a member of the research panel, in particular for the assistance he has given me with regard to the technical aspects of the Xpress-MP software. My thanks also go to Professor Malcolm King for his contribution as Research Director.

Finally I would like to express my most heartfelt thanks and gratitude to my partner Lorraine for her support throughout. Thanks for everything Lol, I couldn't have done it without you.

1 Introduction

During the late 1930's UK scientists began to use quantitative techniques in order to study the strategic and tactical effectiveness of a number of military operations. Since military resources at this time were limited analysis of certain military operations took place in order to attempt to find the most efficient use of such resources. During World War II Operational Research (OR) groups were set up within the UK armed forces but it was not long however before similar groups were being set up in the USA and other European countries. Following the end of the war the use of OR methods began to spread into industry within the UK and USA and by the mid 1950's had been adopted by a number of other countries. The increasingly widespread usage and availability of computing power during the last 30 to 35 years has coincided with considerable growth in the development of OR techniques and applications. This is no coincidence since the majority of OR techniques are extremely computer intensive in that they require large numbers of numeric calculations to be carried out in as short a time as possible.

The science of OR attempts to take structured and semi-structured problems and develop mathematical models that can be used to represent the problems under consideration. Attempts at solving such models can then be carried out using one of a number of solution approaches. Examples of such techniques include Mathematical Programming, simulation methods, network analysis, game theory and queuing theory. Significantly OR is used to solve real-world problems encountered within business, industry and the public services, examples of these are

(i) ***Scheduling problems.*** Typical examples of scheduling problems include production scheduling where there is a requirement to schedule a number of jobs or tasks to be performed on a limited number of machines over time whilst taking into account that one or more

objectives need to be achieved as a result of such a schedule, this could be to minimise the amount of time needed to complete the last of the jobs. Another widely posed scheduling problem is that of timetabling, an example of which is the task of allocating certain resources such as staff, rooms or equipment to taught modules in a University or other educational establishment.

(ii) ***Vehicle Routing Problems.*** A typical example of a vehicle routing problem is that of how to utilise a fleet of vehicles, each with a fixed capacity, in order to meet the demands of a certain number of customers where goods have to be delivered to those customers by the fleet of vehicles from a certain number of depots or warehouses. In such circumstances there exist different distances between each customer and each depot. The problem then is one of minimising the total distance travelled by the fleet of vehicles or minimising the number of vehicles required in order to satisfy customer requirements.

(iii) ***Knapsack problems.*** Given a collection of items each having a value and a cost associated with it, this classical problem is one of selecting items from the collection to place into the knapsack having a fixed capacity with the objective of maximising the value of the items to be placed into the knapsack whilst satisfying the constraint of the capacity of the knapsack. In a more practical setting a *'knapsack'* could be thought of as representing a cargo hold in an aircraft or a container.

(iv) ***Assignment problems.*** This type of problem involves the allocation of resources to enable jobs or tasks to be performed satisfactorily. A simple example of such a problem would be where a number of jobs were required to be performed by the same number of people. Associated with each job-person combination is a cost, the problem is then to complete

all of the jobs at minimum cost. The Generalised Assignment problem falls into this category of problems and a detailed description is given in chapter 2.

The research described in this thesis focuses on a particular type of problem known as the Generalised Assignment Problem (GAP) and attempts to combine aspects of two widely used OR solution techniques for solving GAP in order to construct a hybrid solution method. The two solution techniques are the exact mathematical programming approach known as *Branch and Bound* and the meta-heuristic approach of *Tabu Search*. The remainder of this chapter describes the background to the research, outlines the research aims and objectives and finally describes the contribution and structure of the thesis.

1.1 Background to the research

Mathematical programming is an OR technique that is widely used to mathematically model and solve complex real-world decision problems. One such approach is to model the problem using a set of linear equalities and inequalities to represent the constraints of the problem together with a linear objective function and can be formally stated as

$$\text{minimize} \quad \mathbf{cx} \tag{1.1}$$

$$\text{subject to} \quad A\mathbf{x} \leq \mathbf{b} \tag{1.2}$$

where $\mathbf{c}$ is a $1 \times n$ vector of cost coefficients, $\mathbf{x}$ is an $n \times 1$ vector of decision variables where each element x_j of $\mathbf{x}$ is allowed to take on any real value, A is an $m \times n$ matrix containing the constraint coefficients of the m constraints and $\mathbf{b}$ is an $n \times 1$ vector of values that specify the right hand side of each of the m constraints.

This concept of linear programming was put forward around 1947 by George B. Dantzig who proposed an algorithm for solving such linear models

known as the simplex algorithm. This is highly significant in that it allows optimal solutions to many complex real-world problems to be obtained and is incorporated into many commercial solvers currently available. The linear constraints of the model define the feasible region of the problem and the intersections of the constraints define the extreme points of such a region, an optimal solution to the problem can be found at one or more of these extreme points. Dantzig's simplex algorithm moves from one extreme point to another evaluating the objective function at each point in order to find the optimal solution. In 1984 Karmarker (Karmarkar. 1984) presented an alternative, polynomial time algorithm for solving the linear programming problem which, in contrast to the simplex algorithm, is an interior point method that moves, by a series of iterations, through the feasible solution space to the optimal solution at the boundary of the feasible region. Karmarker's algorithm is not as well established as the simplex algorithm and the fact that a polynomial time algorithm can solve LP problems suggests that they are not as *hard* as were first thought. Existing implementations of the simplex algorithm in commercial software such as Xpress-MP and Cplex are highly efficient at solving very large complex linear programming problems to optimality. The linear programming approach however relies on the condition that the decision variables are allowed to take on continuous values and the introduction of integer constraints on the decision variables increases the complexity of the problem. In 1960 however a paper by Land and Doig (Land and A. G. Doig. 1960) proposed the Branch and Bound approach for solving such integer programming problems and is also included as the default algorithm in many commercial software solvers today. The essence of the branch and bound approach is one of solving a series of linear programming sub-problems that are constructed by dividing up the feasible region by considering those variables that take on fractional values in the optimal solution to the LP sub-problem under consideration. The bounding aspect of the algorithm uses the best integer solution found during the solution process to eliminate those LP sub-problems having an optimal solution value worse

than the value of the best integer solution.

Whilst the application of the branch and bound approach to solving many classes of integer programming problems has been quite successful, an increase in size and complexity of many instances over time has resulted in a situation where such problems are unable to be solved to optimality even using some of the most efficient commercial implementations available today. This situation applies to many classes of combinatorial optimization problems that are known to be NP-hard and so has prompted research into the development of heuristic methods in order to find good solutions to difficult problems, where good can be thought of as optimal or near optimal, within reasonably defined timescales. The development of many powerful heuristic and meta-heuristic approaches such as Genetic Algorithms (GA's), Tabu Search (TS) and Simulated Annealing (SA) has produced substantial contributions towards obtaining improvements both in respect of solution values and solution times for a variety of different classes of problems including GAP. More recently however combining heuristic methods with exact optimization methods has been the subject of research that has produced new hybrid approaches that appear to be quite successful and indeed this has been the approach adopted in this research.

A variety of real-world problems in industry, business and the public services require the allocation of resources in order to achieve certain aims and objectives. There will typically be a limit to the amount of resource available and a cost incurred by the allocation of such resource and it is therefore important to allocate such resources as efficiently as possible whilst attempting to achieve the objectives of the situation under consideration. These types of problem can typically be represented by integer programming problems and detailed descriptions of those relevant to the research are given in the following chapter. Due to their practical relevance such problems have been the subject of extensive research in recent years as problem instances have become larger and more difficult to solve resulting in the emergence of

more sophisticated and powerful algorithms for solving them.

One of the most successful heuristic approaches that has been developed and applied to many difficult problems since the late nineteen eighties is Tabu Search (TS). First introduced by Glover (Glover. 1989) and (Glover. 1990) TS has evolved as a strategy that can be adapted and tailored to construct extremely powerful algorithms that have proved to be successful for solving many very difficult problems. Despite the extensive research that has been undertaken and the advances that have resulted there remain several possibilities for further development with Tabu Search, many of which are highlighted by Glover and Laguna (Glover and Laguna 1997) including the approach of fixing problem variables in order to generate sub-problems that can be much more easily solved using exact optimization techniques. Such a strategy, it is further suggested, can be guided by a TS heuristic in order to generate complete solutions to the problem. This approach is the focus of this research using branch and bound as the exact optimization algorithm.

1.2 Aims of the research

The research focus has three main elements, these being the Generalized Assignment Problem (GAP), the meta-heuristic strategy Tabu Search (TS) and the exact optimization method Branch and Bound (B&B). The GAP has been selected as it is one of the combinatorial optimization problems that are known to be NP-hard and it has frequently been used to model a number of relevant real-world problems. Its importance has been confirmed by the extensive research that has been conducted in terms of solution methods, among which some of the most successful have been TS and B&B approaches. Many alternative approaches have been presented in the literature and a review of these is presented in chapter three of the thesis. The effectiveness and performance of each new approach to GAP tends to be assessed by testing the algorithm using a set of benchmark test instances. The result of such testing is typically compared to results of previously developed

methods in order to compare the quality of the achieved solutions. Two key indicators that are commonly used to compare the performance of different algorithms are the solution value of the objective function and the speed with which such solutions have been obtained. Whilst the aim of this research is to follow suit in terms of producing results by conducting testing on these same benchmark instances the objective is not just to produce results that outperform alternative algorithms in respect of solution value and speed although these two factors will clearly form part of the assessment as to the effectiveness of any proposed algorithm.

It seems reasonable to suggest however that, considering previous research based on the three key aspects of the thesis, combining a TS approach with a B&B approach in order to produce a hybrid approach for solving GAP could have relevance both in theoretical terms as well as practical importance and therefore provides the main aim of the research.

The main aim of the research is to investigate the possibilities for combining the TS and B&B approaches in order to produce a hybrid approach that can be implemented using existing commercial software and to test such an approach on a set of benchmark test instances in order to compare the effectiveness of the hybrid with other algorithms. The aim with respect to solution quality is to produce solutions that are of high quality when compared to solution values previously obtained and the best known solutions and, in addition, to produce such solutions within reasonable timescales when compared to the speed with which alternative algorithms are able to produce such solutions. The effective combination of the two approaches to form a hybrid algorithm should therefore highlight both practical and theoretical issues for such combinations that can also form the basis for development and further investigation into the approach.

1.3 Structure of the thesis

This thesis is comprised of seven chapters. The first of these seven chapters provides an introduction to the research by first providing an explanation of the positioning of the research within the field of Operational Research.

Chapter two gives a formal description of the *Generalized Assignment Problem* (GAP) along with a definition of its mathematical formulation and provides some examples of applications of GAP in real world settings. Additionally some problems related to GAP are also presented in this chapter that are relevant in terms of providing some insight into both the structure of GAP and its various solution approaches.

Chapter three provides a review of the relevant literature that has been published over a period spanning the last thirty plus years. This literature review describes the wide variety of solution techniques that have been developed and applied to solving GAP. The development of increasingly powerful and efficient algorithms for solving larger more complex instances of the problem is also evident in this review of the literature. In keeping with the scope of this research the review approaches this literature from two perspectives, the exact branch and bound approach and the heuristic / meta-heuristic approach. Additional literature that is not solution specific for GAP is subsequently reviewed in chapters four and five.

A general overview of both the branch and bound and Tabu Search approaches is described in chapter four. The key components of each technique are described and their relevance to solving integer programming problems is discussed. This chapter also includes the review of additional key literature relating more generally to the solution techniques as opposed to the specific applications given in chapter three.

Chapter five is concerned with the hybridisation aspect of the research and first discusses theory and techniques that have been proposed in the literature that have relevance with regard to the construction of the hybrid approach that has been developed and subsequently described later in the second part of chapter five. A description of the hybrid algorithm is given in detail along with a discussion of how the algorithm has been developed and implemented for solving GAP.

Chapter six is devoted to the presentation and discussion of the results obtained from the computational testing and experimentation performed following implementation of the proposed algorithm. Due to the extensive research conducted for solution approaches for solving GAP libraries of benchmark instances that have been used for testing the effectiveness of various algorithms have been generated and made available. Comparisons in performance of alternative approaches can be made using these GAP libraries and a detailed description of problems contained in the libraries is given in the first part of chapter six. The remainder of the chapter is concerned with the performance of the proposed new hybrid approach and its effectiveness is compared with relevant approaches that have been shown to produce results that are considered to be among the best available.

Conclusions that have been derived as a result of computational testing and experimentation are discussed in chapter seven along with the presentation of thoughts to provide some direction for future work and development.

2 The Generalized Assignment Problem

The Generalized Assignment Problem (GAP) is one of the classical combinatorial optimisation problems that are known to be NP-Hard. GAP has been the focus of much research over the last thirty years giving rise to the development of some very effective and sophisticated solution approaches that have been assessed on increasingly difficult instances. This has resulted in advances from both practical and theoretical perspectives. This chapter of the thesis gives a formal definition of the problem along with an indication of its practical relevance with a brief description of some of its practical applications. The first section of this chapter however gives an overview of some important related problems.

2.1 Related problems

A common and effective approach that has been widely used to solve GAP is that of reducing the problem to a series of knapsack problems or a series of assignment problems by relaxing some of the problem constraints. The solution effort then focuses on solving a series of sub-problems which usually will produce infeasible solutions which need to be manipulated in order to restore feasibility. The following problems are therefore relevant both in terms of the structure of GAP and approaches to its solution.

2.1.1 The Knapsack problem

The Knapsack Problem (KP) and its variations has itself been the subject of extensive research over several years. Comprehensive coverage of the problem, its variants and solution methodologies are given in (Martello and Toth 1990) and more recent comprehensive treatment of the problem and its variants along with solution approaches, both exact and approximate, can be found in (Kellerer et al. 2004). The problem is formulated by considering which items from a set of n items should be selected to be packed into a single knapsack having a capacity c. Each of the n items has an associated

benefit if it is placed into the knapsack, which can be considered as a profit p_i, and a corresponding weight w_i if item i is included. The problem KP can therefore be formulated as an integer programming problem as defined in (Martello and Toth 1990) as

$$KP = \quad \text{maximize} \quad \sum_{i=1}^{n} p_i x_i \tag{2.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{2.2}$$

$$x_i = 0 \text{ or } 1, \quad i = 1,\ldots,n. \tag{2.3}$$

KP is the most common variant of the knapsack problem known as the 0-1 knapsack problem where each item $i \in N$ is either included in or excluded from the knapsack container. Two extensions to this version of the problem occur as a result of varying the number of each item included in the item set N.

The bounded knapsack problem

Let the item set N now represent the different types of items that can be included in the knapsack and let B represent the number of copies of each item that are available so that there are now b_i instances of type i that are available to be placed into the knapsack. Each instance of type i has an identical weight w_i and identical profit p_i. The bounded knapsack problem (BKP) can thus be represented by the model

$$BKP \quad = \quad \text{maximize} \quad \sum_{i=1}^{n} p_i x_i \tag{2.4}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq c \tag{2.5}$$

$$0 \leq x_i \leq b_i, \quad x_i \text{ integer}, \quad i = 1,\ldots,n. \tag{2.6}$$

The unbounded knapsack problem

If there exists an unlimited amount of each item type i then constraints (2.6) in BKP can be replaced to obtain the unbounded knapsack problem UKP

where

$$UKP = \text{maximize} \sum_{i=1}^{n} p_i x_i \quad (2.7)$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq c \quad (2.8)$$

$$x_i \geq 0, \; x_i \text{ integer}, \; i = 1,\ldots,n \quad (2.9)$$

Modest sized instances of KP can be solved to optimality using exact branch and bound methods, early implementations of which were proposed in (Kolesar. 1967) and later in (Martello and P. Toth. 1977). A comprehensive description and comparison of several algorithms can also be found in (Martello and Toth 1990) . Larger problem instances however require approximate approaches due to the significant increase in computational requirements associated with increases in problem sizes. One of the first approximate approaches was proposed by Sahni (Sahni. 1975).

2.1.2 The Multiple Knapsack problem

Increasing the number of knapsacks in a problem from a single knapsack to say m knapsacks each with its own capacity gives rise to a generalized form of the knapsack problem known as the multiple knapsack problem. The objective now becomes one of assigning each item to one of several knapsacks without exceeding the capacity of each knapsack whilst maximizing the overall profit from such assignments. As in the 0-1 version an available set of items $N = \{1,\ldots,n\}$ each with weight w_i and profit p_i are available to be placed into one of the m knapsacks each with capacity c_j. The problem is then formally defined in the following integer programming formulation as

$$MKP = \text{maximize} \sum_{i=1}^{n}\sum_{j=1}^{m} p_i x_{ij} \tag{2.10}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_{ij} \le c_j, \quad j = 1,\ldots,m, \tag{2.11}$$

$$\sum_{j=1}^{m} x_{ij} \le 1, \quad i = 1,\ldots,n, \tag{2.12}$$

$$x_{ij} = 0 \text{ or } 1, \quad i = 1,\ldots,n, j = 1,\ldots,m \tag{2.13}$$

where the variable x_{ij} takes on the value 1 if item i is included in knapsack j and 0 otherwise. Upper bounds for MKP can easily be computed using relaxation methods such as relaxing constraints (2.13) and replacing them with

$$0 \le x_{ij} \le 1, i = 1,\ldots,n, j = 1,\ldots,m \tag{2.14}$$

to yield a linear programming formulation which can be easily solved. Other relaxation methods such as lagrangian relaxation can also be used to calculate the upper bound. Several branch and bound algorithms have been presented as solution methods for MKP including (Martello and P. Toth. 1980) and (Pisinger. 1999).

2.1.3 The Multi-dimensional Knapsack problem

The knapsack and multiple knapsack problems consider only the weights of the items when choosing which items to include which are constrained by the total weight that can be accommodated by the knapsack i.e. the problem is constrained only in a single dimension. If one were to consider an additional constraint in relation to the knapsack, say its volume, then the resulting problem can be thought of as a two-dimensional knapsack problem. Clearly then the general case is one of a multi-dimensional knapsack problem. Each item to be packed into the knapsack now needs to be considered with respect to each dimension $j = 1,\ldots,d$ resulting in the problem formally stated as

$$MdKP = \text{maximize} \sum_{i=1}^{n} p_i x_i \quad (2.15)$$

$$\text{subject to} \sum_{i=1}^{n} w_{ij} x_i \le c_j, \quad j = 1,\ldots,d, \quad (2.16)$$

$$x_i = 0 \text{ or } 1, \quad i = 1,\ldots,n \quad (2.17)$$

Early attempts at solving the multi-dimensional knapsack problem date back to the 1960's such as the dynamic programming approaches presented in (Gilmore and R. E. Gomory. 1966) , (Weingartner and D. N. Ness. 1967) and (Nemhauser and Z. Ullmann. 1969). An exact branch and bound approach was later present by Gavish and Pirkul (Gavish and H. Pirkul. 1985).

2.1.4 The Assignment problem

The assignment problem (AP) can be considered to be a special case of GAP which has also been the subject of extensive research in its own right. The objective of the assignment problem is to minimize the cost or maximize the profit associated with allocating a set of n jobs to one of n agents that are able to carry out the jobs, where agent is a term that can be used to represent people, machines, vehicles, computer processors and a variety of other components dependent upon the problem being modelled. The constraints of the problem are that each job must be completed and each agent must be assigned a job to complete. As with the knapsack problems previously described the assignment problem can be easily stated as an integer programming problem but increases in the size of problem instances are not trivial to solve. Given the cost c_{ij} of assigning each job j to an available agent i then the integer programming problem is formally defined as

$$AP = \text{mimimize} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \quad (2.18)$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n \quad (2.19)$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1\ldots,n \quad (2.20)$$

$$x_{ij} = 0 \text{ or } 1, \; i = 1,\ldots,n, j = 1,\ldots,n \quad (2.21)$$

Although the Assignment problem is an integer programming problem solving the LP relaxation always yields an integer solution and so it is useful as a sub-problem for solving more difficult problems. GAP is constrained by aspects of both the knapsack problem and the assignment problem and so this chapter concludes with a formal definition of GAP and its significance is highlighted by some examples of its applications.

2.2 Problem formulation for GAP

The Generalized Assignment Problem (GAP) is the problem of either minimizing cost or maximizing profit by assigning n jobs to m agents such that each job is assigned to exactly one agent whilst ensuring that the resource capacities of each agent are not violated. Let $I = \{1,\ldots,m\}$ be the set of agents and $J = \{1,\ldots,n\}$ the set of jobs to be assigned to an agent $i \in I$. GAP can then be formulated as the integer programming problem:

$$GAP = \quad \text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.22}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_{ij} \le b_i \qquad \forall\, i \in I \tag{2.23}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall\, j \in J \tag{2.24}$$

$$x_{ij} = 0 \text{ or } 1 \qquad \forall\, i \in I, j \in J \tag{2.25}$$

where c_{ij} represents the cost of assigning job j to agent i, a_{ij} represents the amount of resource consumed if job j is performed by agent i, although in some variants of GAP the cost of performing job j is identical for each agent and so a_{ij} is replaced by a_j, and b_i is the amount of available resource for agent i. Constraints (2.23) prevent violation of the resource capacity for each agent $i \in I$ and can be thought of as knapsack constraints, whereas constraints (2.24) ensure that each job $j \in J$ is assigned to exactly one agent and can be thought of as assignment constraints. Constraints (2.25) represent the binary

conditions on the decision variables where x_{ij} takes on the value 1 if job j is assigned to agent i and 0 otherwise. As can be seen from the definition of *GAP* the mathematical model for GAP can be relatively easily structured as a linear and integer programming problem, however it transpires that even instances with fairly modest values of m and n can be quite difficult to solve to optimality using exact algorithms. As values of m and n grow it is soon realised that optimal solutions from solution attempts using exact methods are unlikely to be achieved. It can be seen that GAP is quite similar in its formulation to that of the multiple knapsack problem but differs in that the cost or profit values and also the weights associated with an assignment of an item to a particular knapsack will typically vary according to which knapsack it is placed into. The second difference is that in GAP all of the n items in the item set must be placed into one of the knapsacks. The similarities between GAP and AP are also apparent although in most GAP instances m will typically be small in comparison to n. All the problems described in this chapter, with the exception of the Assignment Problem, are known to be NP-hard and the reader is referred to (Martello and Toth 1990) and (Garey and Johnson 1979) for further explanation.

2.3 Applications

There are several real-world applications of problems that can be modeled using GAP and solved using an appropriate solution method. A vehicle routing problem was successfully modeled as a GAP by Fisher and Jaikumar (Fisher and R. Jaikumar. 1981) where items to be delivered were represented as jobs and the vehicles that were to deliver the items were considered to be the agents. Ross and Soland (Ross and P. Soland. 1977) presented a method for modeling facility location problems using GAP. The task of allocating jobs to computers in computer networks was also modeled using GAP by Balachandran (Balachandran. 1976). Foulds and Wilson (Foulds and J. M. Wilson. 1997) use a variation of GAP to represent the problem of allocating milk-producing farms to collection depots in the New Zealand dairy industry.

Other applications include assigning tasks to computer programmers in software development projects, storage space allocation problems, design of communication networks, sugar cane harvesting and scheduling such things as television commercials into time slots.

2.4 Summary

A formal description and definition for GAP have been provided in this chapter and its significance highlighted in terms of applications that occur in real situations. Research into solution approaches for GAP is still very much active as it provides a difficult challenge with respect to the development of new approaches for solving it. An insight into its structure has been put forward by means of the explanation of the associated problems that have also been presented. The following chapter reviews a variety of approaches for solving GAP that have been published in the literature.

3 A review of the literature

A variety of solution approaches to solving GAP have been developed and reported over the last 30 years. Most of the early methods reported in the literature for solving GAP are exact branch and bound schemes which were very effective at solving small loosely constrained instances. The development of some very efficient heuristics and meta-heuristics in the 1990's coincided with an increase in problem size and difficulty of real world instances of GAP that could not be solved to optimality using the previously reported branch and bound methods. More recently attempts have been made at combining heuristic, meta-heuristic and exact methods in order to produce some highly efficient algorithms that are capable of producing extremely high quality solutions to large, tightly constrained problems in realistic time intervals. The remainder of this chapter reviews these various approaches.

3.1 Branch and Bound approaches

In the branch and bound context the solution to a relaxed version of GAP provides both a bound on the objective function of the original problem and identifies how the relaxed problem may be further constrained by a relevant branching strategy in order to generate primal feasible solutions to GAP. This section provides a review and summary of the alternative relaxation approaches that have been exploited within a branch and bound approach for solving GAP.

3.1.1 Linear programming relaxation

The linear programming (LP) relaxation is used in a conventional branch and bound scheme by relaxing the integrality constraints and for GAP allowing $0 \le x_{ij} \le 1$ for all $i \in I$ and $j \in J$. The solution to the LP relaxation at each node provides both a bound for GAP and may contain one or more fractional variables which become candidates for selection as the branching variable. The bound provided by the LP relaxation has been rarely used in the literature in relation to branch and bound approaches for GAP as it

is fairly weak compared to some of the bounds obtained using alternative relaxations as subsequently described. The LP relaxation however does have properties which have been exploited in some heuristic approaches as discussed in 3.2 and 3.3 and is used in the branch and bound algorithm of most commercial integer programming solvers including Xpress-MP.

3.1.2 Relaxing the capacity constraints

Relaxation of the capacity constraints was implemented by Ross and Soland (Ross and P. Soland. 1975). Deleting the capacity constraints yields the following relaxed problem:

$$PR_1 = \quad \text{Min} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad (3.1)$$

$$\text{s.t} \quad \sum_{i=1}^{m} x_{ij} = 1 \qquad \forall\ j \in J \qquad (3.2)$$

$$x_{ij} = 0 \text{ or } 1 \qquad \forall\ i \in I, j \in J \qquad (3.3)$$

At each node of the branch and bound tree PR_1, subject to the relevant branching constraints, is solved by assigning each unassigned job to the least costly agent. The resulting solution to PR_1 yields a lower bound on the problem at that node. It is highly probable, except in trivial instances, that the solution to PR_1 will violate one or more of the capacity constraints for GAP and so the bound is then improved by considering the following knapsack problem for each of the agents whose capacity has been violated:

$$K_i^1 = \text{minimise} \quad \sum_{j \in j} p_j y_{ij} \qquad (3.4)$$

$$\text{subject to} \quad \sum_{j \in J} a_{ij} y_{ij} \geq d_i \qquad (3.5)$$

$$y_{ij} = 0 \text{ or } 1 \qquad (3.6)$$

where $d_i = \sum a_{ij}x_{ij}^* - b_i$, x_{ij}^* represents an assignment in the solution to PR_1

and p_j is the minimum increase in cost incurred by reassignment of job j.

Each knapsack problem K_i minimizes the penalty for reassigning jobs in order to restore feasibility for the violated agent i and the combined penalty resulting from solving each knapsack problem is added to the lower bound in order to update the bound at that node. If the solution to the relaxation at the current node is primal infeasible then branching is performed from that node.

The branching strategy used by the authors is a depth first approach that selects a variable for separation from those variables that were not selected for reassignment during the bound improvement phase. Two branches are created using the 0-1 dichotomy of the Dakin branching scheme by constraining the variable x_{ij} to take on the value 0 or 1. The variable chosen is the one that represents an assignment that is most attractive when considering the consequences in terms of the penalty incurred from reassignment and also the spare capacity of the agent to which the job is currently assigned. The algorithm first considers the problem associated with the 1-branch and this node can be fathomed if a primal feasible solution is found as a result of identifying reassignments when solving the series of knapsack problems. In such circumstances the algorithm backtracks and explores the 0-branch.

3.1.3 Relaxing the assignment constraints

Martello and Toth (Martello, Toth 1981) use a maximization version of GAP in order to demonstrate how relaxation of the assignment constraints can be applied in order to calculate an upper bound. Deletion of the assignment constraints results in the GAP relaxation defined by

$$PR_2 = \quad \text{Max} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad (3.7)$$

$$\text{s.t} \quad \sum_{j=1}^{n} a_{ij}x_{ij} \le b_i \quad \forall\, i \in I \qquad (3.8)\,.$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall\, i \in I, j \in J \qquad (3.9)$$

Relaxation PR_2 naturally decomposes into a series of $|I|$ knapsack problems

$$K_i = \text{maximise} \quad \sum_{j \in j} c_{ij} x_{ij} \tag{3.10}$$

$$\text{subject to} \quad \sum_{j \in J} a_{ij} x_{ij} \leq b_i \tag{3.11}$$

$$x_{ij} = 0 \text{ or } 1 \tag{3.12}$$

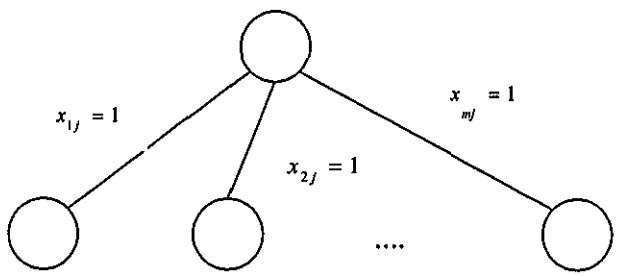
each of which can be solved to yield an objective function value z_i. The upper bound obtained is thus $\sum_{i \in I} z_i$. The authors then attempt to strengthen this bound by calculating, for each job, a lower bound, l_j, the penalty that would be incurred in order to satisfy the violated assignment constraint for job j. Each job j that has not been uniquely assigned in the solution to the relaxed problem then either belongs to $J_0 = \{j \mid \sum_{i \in I} x_{ij} = 0\}$ or $J_1 = \{j \mid \sum_{i \in I} x_{ij} > 1\}$. Each job is considered in turn and an upper bound u_{ij} is calculated for each problem K_i as a result of setting $x_{ij} = 1 \ \forall \ i \in I$ if $j \in J_0$ and $x_{ij} = 0$ if $j \in J_1$ and $x_{ij} = 1$. The penalty incurred l_j to satisfy the assignment constraint for each job $j \in J_0 \cup J_1$ is then used to calculate the revised bound $\sum_{i \in I} z_i - \max_{j \in J_0 \cup J_1} \{l_j\}$.

Consideration is given in (Martello and Toth 1990) as to whether their own bound is superior to that of Ross and Soland and conclude, based on two different examples, that the stronger of the two bounds is dependent upon the problem instance and therefore compute both their own bound and the Ross and Soland bound at each node of the branch and bound tree, except for the root node, where they also compute the bound of Fisher et al. (Fisher, et al. 1986) as subsequently described in 3.1.4.

In contrast to the Ross and Soland scheme Martello and Toth use an alternative branching strategy to the 0-1 scheme by creating multiple branches from each node based on the reassignment of jobs in the solution at that node.

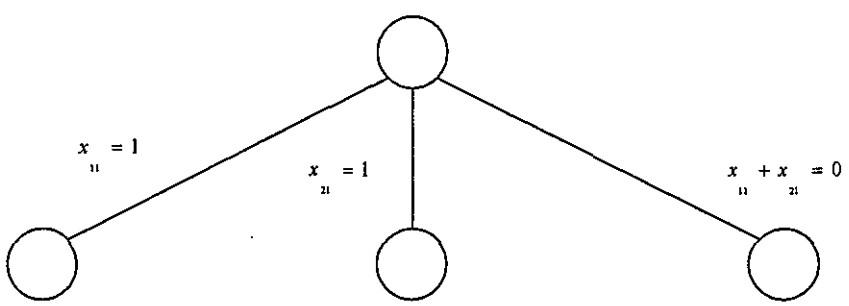
As described above the relaxed problem results in each job being allocated to 0, 1 or several agents. Each job $j \in J_0 \cup J_1$ is considered and the job having the largest l_j value is selected for branching. If job j has not been assigned in the current solution then m branches are created as in figure 3.1.1.

Figure 3.1.1 Martello and Toth branching scheme for unassigned jobs.



Alternatively if job j has been assigned to k agents where $k > 1$ then $k+1$ branches are created, one branch for each assignment and an additional branch excluding all current assignments for job j. Given a solution where job 1 is assigned to agents 1 and 2 then the branching would be as in figure 3.1.2.

Figure 3.1.2 Martello and Toth branching scheme for jobs assigned to multiple agents.



3.1.4 Lagrangian relaxation

A lagrangian relaxation of the generalized assignment problem can be achieved by dualizing either the capacity constraints or the assignment constraints. The latter is explored by Guignard and Rosenwein (Guignard and M. B. Rosenwein. 1989) in their heuristic approach as described in 3.1.5 however the former is the approach used by Fisher et al. (Fisher, et al. 1986) where the assignment constraints are dualized into the objective function to give the relaxed formulation

$$LR = \quad \text{maximize} \sum_{i \in I} \sum_{j \in J} (c_{ij} - \lambda_j) x_{ij} \tag{3.13}$$

$$\text{subject to} \sum_{j \in J} a_{ij} x_{ij} \leq b_i \quad \forall\ i \in I \tag{3.14}$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall\ i \in I,\ j \in J \tag{3.15}$$

The authors initialize the lagrange multipliers to the value of the second largest c_{ij} thus producing a bound equivalent to the Ross and Soland bound, and decompose LR into a series of knapsack problems LR_i. A solution to LR is then obtained by first assigning the value 0 or 1 to those variables having $c_{ij} - \lambda_j > 0$ according to the value of that variable in the optimal solution to LR_i, followed by an attempt to assign values to as many of the remaining free variables having $c_{ij} - \lambda_j = 0$. This second phase is achieved according to some heuristic approach the choice of which, according to the authors, is not critical as there are few feasible choices remaining and so assignments are made based on decreasing cost coefficients. The resulting solution to LR is feasible but may contain one or more jobs that have not been assigned. An attempt to assign these jobs is made by decreasing the lagrange multipliers in an attempt to include an assignment $x_{ij} = 1$ in the optimal solution to LR_i. The multiplier adjustment method proceeds until no further improvements to the solution are possible and then selects the free

variable x_{ij} having the greatest resource requirement a_{ij} for branching using the 0-1 branching strategy.

Fisher et al. (Fisher, et al. 1986) give a comparison of their own algorithm with those of Ross and Soland and Martello and Toth on a set of 160 randomly generated test problems with $m = 3$ and $5, n = 10$ and 20.

More recently Haddadi and Ouzia (Haddadi and H. Ouzia. 2004) follow the Martello and Toth convention and formulate GAP as a maximization problem but use the relaxation LR in their branch and bound approach along with subgradient optimisation to solve the relaxation. They also introduce a heuristic that is applied at each iteration of the subgradient phase in order to generate feasible solutions for GAP. At iteration k of the subgradient phase the authors fix those jobs that are uniquely assigned in the solution X^k and generate a smaller sub-problem which is that of assigning the remaining jobs that have either not been assigned or have been assigned to multiple agents. In this sub-problem the agents' capacities are reduced to take into account those assignments appearing in X^k. The heuristic of Martello and Toth (Martello and Toth 1981) is then applied to the sub-problem in order to generate an approximate solution. The resulting solution thus uniquely assigns those jobs that were not previously uniquely assigned in X^k. The authors identified that as k increases during the subgradient phase the number of uniquely assigned jobs also increases resulting in near feasible solutions to GAP and therefore smaller sub-problems to solve. The upper bound at each node of the branch and bound tree is given by the objective function value from solving the relaxed problem LR.

Having solved the relaxation and generated a relaxed solution, a job j^* is selected from those jobs not already having been assigned by the branching strategy to be the next job selected for branching based upon the contribution

to the objective function of each available assignment. The node selection strategy follows the breadth-first approach and selects the node having the largest upper bound. The number of child nodes created will be between 2 and m dependent upon how many of the agents can be assigned to job j^*.

3.1.5 Surrogate constraints

The approach of Fisher et al.(Fisher, et al. 1986) was improved upon in terms of both computational times and the size of the branch and bound trees by Guignard and Rosenwein (Guignard and M. B. Rosenwein. 1989). As a result they were able to solve larger problem instances than had previously been reported, generating problems with $m \leq 10$ and $n \leq 50$. They also use the lagrangian relaxation LR but they allow $\sum_{i \in I} x_{ij} \geq 0$ for all $j \in J$ whereas Fisher et al. use $\sum_{i \in I} x_{ij} \leq 1$. The solution process begins by solving the relaxation at the root node using subgradient optimization. At each subsequent node lagrangian dual ascent is used to solve LR and in addition the relaxation is strengthened by the addition of the surrogate constraint $\sum_{i \in I} \sum_{j \in J} x_{ij} \leq n$ or $\sum_{i \in I} \sum_{j \in J} x_{ij} \geq n$ if the number of assignments does not equal n. Multiple branching is employed from each node by considering those jobs being assigned to multiple agents and selects a job j' that minimizes the maximum resource requirements of an assignment (i, j). A separate branch for each assignment (i, j') where $x_{ij'} = 1$ is created by setting $x_{ij'} = 0$ and selecting the branch with the largest resource coefficient $a_{ij'}$ to explore first.

3.1.6 Other Branch and Bound approaches

A variety of strategies are used by Nauss (Nauss. 2003) as a means to strengthen the bounds at each node of the tree in his special purpose branch and bound algorithm for solving GAP. The algorithm begins by using the tabu search heuristic of Laguna et al. (Laguna, et al. 1995) in order to generate an initial feasible solution. The integrality constraints of P are then relaxed in

order to generate and solve the LP relaxation and the resulting objective function is used to test for optimality by comparing it with the objective function of the heuristic solution. Unless the heuristic solution is optimal the authors attempt to increase the lower bound by applying a series of linear programming cuts and then solving the lagrangian relaxation *LR* using sub-gradient optimization. They fix as many variables as possible by calculating the penalties incurred for not fixing such variables in an effort to reduce the number of variables that may be selected for branching. If the number of fixed variables at the current node of the branch and bound tree is above a certain threshold then they complete the solution using complete enumeration and update the best solution and objective function if appropriate. Alternatively if not enough variables have been fixed than they use logical feasibility tests in order to fix additional variables followed by subgradient optimisation in order to solve the relaxation. If the node cannot be fathomed then a free variable is chosen for branching using the 0-1 dichotomy .

3.2 Heuristic approaches

This section reviews some of the powerful heuristic and meta-heuristic approaches that have been implemented for solving GAP that were mostly developed during the 1990's as a means for finding high quality solutions for larger, harder problem instances.

3.2.1 Tabu Search

The origins of Tabu Search (TS) were first evident in the literature towards the end of the nineteen seventies (Glover. 1977) and early nineteen eighties (Glover. 1986) but were formally presented by Glover (Glover. 1989) and (Glover. 1990) at the end of the nineteen eighties as a strategy for solving combinatorial optimization problems. The first of these two papers describes the standard components of the strategy and the second follow up paper details more advanced aspects of TS. An overview of the strategy and its components is subsequently described in the following chapter and the

remainder of this section reviews some of the TS applications for GAP as found in the literature..

GAP can be considered to be the single-level version of the multilevel generalised assignment problem (MGAP) which was solved using a tabu search approach by Laguna et al. (Laguna, et al. 1995). An initial solution is first generated by relaxing the capacity constraints and making assignments according to minimum cost (for a minimization problem). The search process then proceeds by moving from the current solution **x** to the best neighbouring solution **x'** where the best neighbour is selected by considering the change in objective function and the violation of the capacity constraints. The novel approach presented by the authors was the construction of neighbourhoods using a series of ejection chain moves which if applied to the current solution will transform **x** into **x'**. Ejection chains are compound moves that eject either one or two assignments from the current solution and replace them with one or two new assignments in order to form a new trial solution. The authors define a dynamic tabu list for each job that records those assignments being removed from the current solution to be replaced by a new assignment and apply a tabu tenure to such an entry based on the number of available ways in which that job may be assigned to an agent. A long term memory element is also incorporated into this process using a frequency based memory which records the number of times an assignment has been part of an ejection chain move. The method also uses the concept of strategic oscillation that forces the search into the infeasible region with regard to capacity constraints when there is no solution with an objective function value better than that of the solution first encountered when the search last entered the feasible region.

The use of ejection chains for solving GAP was further developed more recently by Yagiura et al. (Yagiura, et al. 2004) where the authors construct three ejection chain neighbourhoods of the current solution that are utilised in a local search phase. The first two neighbourhoods correspond to the

commonly used shift moves which are termed *'Shift'* and *'Double-shift'* and are in fact special cases of the third type of neighbourhood termed *'long chain'* which generates neighbouring solutions by constructing a chain of reassignments. Initially an assignment (i, j) is dropped from the current solution thus increasing the spare capacity at agent i. This freeing of capacity triggers off a series of subsequent shift moves where complete trial solutions are generated by reassigning job j. These three neighbourhoods constitute the core of the algorithm which begins by randomly generating an initial solution and proceeds by attempting to improve this solution using local search with shift moves. This is followed by a second attempt at improvement using local search with double-shift and then by implementing a long-chain move. The generation of the long-chain neighbourhood ceases as soon as an improving solution is found and if no improving solution is found then all solutions in the neighbourhood are examined. The search process is allowed to visit the infeasible region by violating the capacity constraints and in such circumstances the objective function is penalized using a weighted penalty function that reflects the relative implications of an alternative assignment in terms of both cost and resource. The weights are changed dynamically throughout the search process in order to control the time spent searching both in the feasible and infeasible regions. The authors also apply subgradient optimisation to the lagrangian relaxation both in the construction of ejection chains and to calculate a lower bound on the minimization problem.

Prior to the ejection chain approach of Yagiura et al. two alternative tabu search approaches had been proposed in (Diaz and E. Fernandez. 2001) and (Higgins. 2001). The approach of Higgins was a dynamic tabu search strategy and focused on very large problems with respect to the number of jobs n. The classical shift and swap neighbourhoods for GAP were employed in order to identify moves from the current solution to the new neighbouring solution but, due to the size of the very large problems being solved, explorations of the neighbourhoods were reduced by means of sampling as complete evaluation

of the neighbourhood would prove far too costly in terms of CPU time. Two new variants of TS for GAP are proposed by the authors that employ the TS strategy of strategic oscillation where the search is allowed to alternate between the feasible and infeasible solution space. In the first instance infeasibility is defined by violation of the capacity constraints where the objective function is penalized for such a violation. Although applying penalties to the objective function had previously been employed in applications of TS the novel aspect of the authors approach was that they adapted the penalty dynamically according to the change in the objective function during phases of the search that

a) *transformed one feasible solution into a subsequent feasible solution*
b) included a move from one solution to a neighbouring solution where one or both solutions are infeasible.

Adjustment of the penalty in this manner has the effect of focusing the search toward promising areas of the feasible region by encouraging moves to feasible solutions during periods when feasible solutions have improved the objective function value and discouraging moves to feasible solutions thus encouraging the search to enter the infeasible region when the recent improvement in objective function has been poor. The second variant also incorporates a dynamically changing neighbourhood size. The sample used to select a move from the current solution to the next solution is taken from the combined shift and swap neighbourhood and the proportion of moves taken from each that are used to make up the sample is allowed to change according to which type of move has given rise to the size of the gain in objective function in the recent history of the search. It seems reasonable therefore to include a high proportion of shift moves in the neighbourhood sample if shift moves have recently caused a large improvement in objective function value. The two new variants were tested on problems with between 20 and 40 agents and between 2000 and 50000 jobs.

Diaz and Fernandez (Diaz and E. Fernandez. 2001) adopt a similar dynamic oscillation approach to the search in that they too use the shift and swap neighbourhoods along with an evaluation function where the objective function is penalized by a weighted penalty based on whether the search has most recently been conducted in the feasible or infeasible region and adjust this weight dynamically as the search progresses. Whereas Higgins uses a simple tabu list that keeps track of the t most recent solutions visited during the search where experimentation indicated values of $t > 1000$ were necessary in order to improve final solution values, Diaz and Fernandez use additional TS concepts in order to improve the efficiency of their algorithm. Firstly they incorporate a short-term recency memory that renders certain assignments tabu for the next t iterations where the value of t changes dynamically within a fixed range $[t_{\min}, t_{\max}]$. This is implemented by means of an $m \times n$ matrix T that records those assignments that are forbidden from being included in subsequent solutions and is updated whenever an assignment (i, j) has been dropped from a solution i.e. when a job j is reassigned from an agent i to a new agent i' at iteration k then T_{ij} is given the value $k + t$ where $t_{\min} \leq t \leq t_{\max}$. In addition to this short-term memory aspect the authors also incorporate a longer-term frequency based memory that is utilised for intensification and diversification strategies. This frequency memory also takes the form of an $m \times n$ matrix and records the number of solutions containing the assignment (i, j). As a result an intensification strategy is implemented by first recovering the best solution found during the search and fixing those assignments that have occurred in at least 85% of solutions and a short term phase is subsequently implemented in order to search for a solution to the resulting reduced problem. Diversification is achieved by penalising the objective function coefficient for those high frequency assignments thus encouraging the selection of less frequently occurring assignments resulting in a series of high influence moves that transform the structure of the solution and drive the search into regions that have remained previously unexplored.

Finally a standard aspiration criterion is used in order to override tabu status when a feasible solution is encountered that improves the objective function of the current best solution. The method was tested on a set of standard benchmark problems and performed favourably when compared to other high quality solution methods.

3.2.2 Genetic Algorithms

In contrast to the deterministic approach of Tabu Search, Genetic Algorithms (GA's) adopt a randomised approach in an attempt to mathematically simulate the biological process of evolution within a population and were first proposed and developed by Holland (Holland 1975). A population of solutions is generated and allowed to evolve by reproductive means in a survival of the fittest approach. One GA approach was formulated and applied to GAP by Chu and Beasley (Beasley and P. C. Chu. 1997) which begins by constructing an initial population where each solution in the population is simply generated by randomly assigning each job j to one of the m agents. The structure of each solution takes the form of vector $\mathbf{s}_k$ of size n where each element s_{kj} is given the value i where job j is assigned to agent i. As a result of these random assignments it is likely that this initial population contains some highly infeasible solutions when considering the capacity constraints. The fitness of each solution in the population is assessed by computing two values, firstly the objective function value f_k is used as a measure of *fitness* and secondly a measure of infeasibility, with respect to violation of the capacity constraints, $u_k(x)$ is calculated to represent the *unfitness* of a solution where

$$f_k = \sum_{j \in J} c_{s_{kj} j}$$

and

$$u_k = \sum_{i \in I} \max\left[0, \left(\sum_{j \in J, s_{kj}=i} a_{ij}\right) - b_i\right].$$

Reproduction is achieved by selecting two parent solutions from the population to which crossover is applied followed by mutation in order to generate a child solution. The selection of the two parents is achieved by means of *binary tournament selection* where two randomly chosen solutions are selected from the population and compete to become a parent by comparison of the fitness values only. The two winners of the binary tournaments subsequently go on to produce a child solution consisting of the first p elements of one parent and the remaining $n-p$ elements from the other parent where the value of p is selected randomly from J. Two elements of the child solution are randomly selected from J and their values swapped in order to achieve mutation. The resulting child solution is then subjected to an improvement heuristic which attempts to improve both the fitness and unfitness values by reassigning jobs prior to replacing a solution in the current population. The solution chosen from the current population to be replaced by the child is the one that is most unfit i.e. the most infeasible solution and if the entire population consists of feasible solutions then the one with the poorest fitness is selected for replacement. Children are not allowed to enter the population if an identical solution already exists within the current population. The GA was tested on a set of benchmark instances ranging in size from 5 to 20 agents and 15 to 200 jobs and compared favourably with other heuristics available at that time in terms of solution quality although this was at the expense of longer computational times when compared with some methods.

An alternative GA approach was proposed at around the same time in (Wilson. 1997) where the emphasis focused on producing a population of solutions that have potentially optimal objective function values but are infeasible with regard to constraint satisfaction and subsequently concentrates on improving feasibility whilst maintaining solution quality in terms of objective function value. This is in contrast with the approach of Chu and Beasley who focused on improving feasibility and optimality simultaneously. The author first generates an initial population based on the optimal solution

to a relaxed problem obtained by deletion of the capacity constraints. Each solution in the population is generated by selecting one job at random and randomly assigning it to one agent, with the remaining jobs in the trial solution being assigned according to the optimal solution to the relaxed problem. This initial population is then subjected to the standard GA procedures of *reproduction, crossover* and *mutation* in an effort to improve the fitness of the population where fitness is calculated according to the degree of infeasibility $\sum_{i=1}^{m} r_i$ where, given a solution x, $r_i = \max\left[0, \left(\sum_{j=1}^{n} a_{ij}x_{ij} - b_i\right)\right] (i \in m)$. Parents are selected from the population using the binary tournament selection method as in Chu and Beasley (Beasley and P. C. Chu. 1997) although crossover is performed by transferring those assignments common to both parents to the resulting child and allocating the remaining assignments according to some probability based on the fitness of each parent, effectively encouraging the inheritance of attributes from the fittest parent. In addition to the crossover procedure a variable rate of mutation is implemented which is increased as the average fitness value of the population approaches zero which randomly mutates the assignments of one or more jobs according to the current mutation rate. The resulting child solution then replaces a solution from the current population and the solution to be replaced must be one with a lower fitness value than the child in order to maintain a population of the fittest solutions. As in Chu and Beasley (Beasley and P. C. Chu. 1997) a child solution is only added to the population if it is different to all solutions in the current population. The GA phase of the algorithm terminates when a solution of fitness value zero is obtained or the generation of children reaches a specified limit and the best solution is subjected to improvement using local search that in a first phase allows the best solution found to take on values from the optimal solution to the initial relaxed problem if this would produce a feasible solution, and in the second phase attempts to swap assignments of jobs in order to improve objective

function value. The GA was tested on problem instances with values of m between 10 and 50 and n between 20 and 500 and performed well with respect to generating near optimal solutions quite quickly. On the smaller problems most improvement seemed to be gained during the GA phase whereas on the larger problems there seemed to be greater gain as a result of the local search phase.

More recently the GA approach has been further researched and an improved GA for solving GAP was proposed by Raidl and Feltl (Raidl, and Feltl 2004) which is largely based on the method of Chu and Beasley (Beasley and P. C. Chu. 1997). The authors present two variants of their GA that differ from each other in the way that the initial population of solutions are generated and effort is concentrated on generating a high proportion of feasible solutions. In the first variant this is achieved by considering the *desirability* of assigning jobs to agents where desirability is measured by means of a desirability index $\gamma_{ij} = \frac{c_{ij} a_{ij}}{b_i}$. Assignments are made by considering each job in random order and identifying those agents having spare capacity for performing the job, of these agents the one chosen to perform the job is determined by comparison of the relevant desirability indices and the most desirable agent is selected, and if no agent can feasibly perform the job then an assignment to one of the agents is made at random. In the second instance an initial integer solution is obtained from the linear programming relaxation for GAP which typically contains a large number of assignments, which are subsequently adopted by the initial solution. For those jobs that are split between two or more agents in the LP relaxation the agent who contributes most to performing job j is selected to perform the entire job. The resulting integer solution will typically be highly infeasible with regard to the capacity constraints, particularly for the more difficult problem instances, is thus subjected to the heuristic improvement strategy described earlier and developed in (Beasley and P. C. Chu. 1997) in order to produce a

number of more suitable solutions for the GA. The evaluation function used to assess the fitness of each solution is the objective function penalised for infeasibility where a measure of infeasibility is calculated according to the *average relative capacity excess.* Binary tournament selection is also used here in order to select solutions from the population for reproduction and its offspring replaces the worst solution in the population. Crossover and production of a child is achieved as in the Chu and Beasley algorithm, however a heuristic mutation operator is then applied by selecting a specified number of jobs to be reassigned using the heuristic approach of Martello and Toth (Martello, Toth 1981) as subsequently reviewed in 3.2.6. The new GA algorithms are subsequently tested and the results compared with those of Chu and Beasley and also with the commercial solver CPLEX on an existing set of benchmark test problems along with a set of new, larger and more difficult test problems generated by the authors. Results of such testing seemed to indicate that the new GA variants were able to significantly outperform their predecessor and with the more difficult problems for which CPLEX was unable to provide optimal solutions could outperform the commercial solver as well. Of the two variants proposed the version that uses the LP relaxation as its starting point seemed to be the stronger.

3.2.3 Simulated Annealing

Whilst Genetic Algorithms attempt to mimic the evolutionary process of reproduction, Simulated Annealing (SA) algorithms attempt to simulate the annealing process of a solid and is a strategy that is used to guide a local search method in the direction of high quality local optima. Starting from an initial point in the solution space the local search begins its descent towards a local optimum by a series of moves to solutions in the neighbourhood of the current solution. The move selected from the current neighbourhood may not necessarily be an improving move as SA allows non-improving as well as improving moves. In the SA setting a neighbouring solution is selected at random and a move to that solution is executed providing that it produces an

improvement in terms of evaluation of the quality of that solution, if however the randomly selected solution is non-improving then it is accepted according to some probability determined by the current *temperature* as updated by the *cooling schedule.*

An SA approach to solving GAP was proposed by Osman(Osman. 1995) using a cooling schedule that periodically increases the temperature in an oscillating manner in contrast to the common approach of gradually decreasing the temperature during the execution of the algorithm. The neighbouring solutions are generated by interchanging the assignment of one or more jobs from an agent i with one or more jobs from an agent i', where the number of jobs to be shifted is determined by the parameter λ where typically $\lambda = 1$ or $\lambda = 2$ due to the increasing amount of work required to consider neighbourhoods of size $\lambda > 2$. The algorithm combines the oscillating SA approach with the metaheuristic approach of Tabu search and proceeds by moving from a current solution to a neighbouring solution as would occur during a standard TS approach by accepting the best non- tabu move to a neighbouring solution, however whereas a non-improving move would normally be accepted with probability 1, in this hybrid approach a non-improving move is only accepted subject to the probability determined by the current temperature as defined by the oscillating cooling schedule as stated. The author proposes and tests six variants of the algorithm based upon

- Selection strategy
- Strength of tabu restriction
- Aspiration criteria.

The selection strategy refers to exploration of the neighbourhood and considers either terminating exploration of the neighbourhood as soon as the first improving move is found, or exploring the entire neighbourhood and accepting the best solution found. The strength of the tabu restriction is determined by the number of attributes involved in a move that are recorded as tabu and the aspiration criteria determines the circumstances under which

tabu status can be overridden. Results are given on a set of 60 test problems in order to make comparisons of the different strategies and also to highlight comparisons with the solutions obtained using the Martello and Toth heuristic in (Martello and Toth 1990). The results obtained at the time seemed to indicate that combining aspects of simulated annealing and tabu search formed an effective approach for solving GAP and suggested exploration of alternative forms of cooling schedule, such as probabilistic tabu search approach, may produce further improvements.

3.2.4 Path Relinking

In the path relinking approach the focus is on generating new solutions by combining attributes from two different solutions and in particular by examining solutions on the path from the first solution known as the *starting solution* to the second solution known as the *guiding solution.* This is essentially achieved by transforming the starting solution into the guiding solution by changing one or more attributes of the starting solution resulting in new solutions that sit on the path between the two solutions. A more detailed description of the method is given in (Glover 1997).

This approach was applied to GAP in (Yagiura, et al. 2006) where the authors further develop their ejection chain approach by incorporating a path-relinking aspect. The ejection chain approach as previously described in (Yagiura, et al. 2004)is applied in turn to a reference set of 20 high quality solutions that are initially generated randomly. This reference set is updated by searching for feasible solutions using the ejection chain process described previously. The resulting locally optimal solution is then compared with the worst solution in the reference set and replaces that solution if it has a better penalized objective function. Each path relinking phase selects two solutions from the reference set and generates a path of solutions between the two which starts at the source solution and by executing shift moves leads to the destination solution. Each solution along this path is subjected to the ejection

chain search phase and the reference set updated with those solutions encountered that improve the worst solution in the reference set. The process then iterates by selecting two new solutions from the reference set to generate a new path. The two solutions are either selected randomly from the reference set or deterministically by considering pairs of solutions that have not yet been combined. An element of diversification is applied to the reference set by only allowing solutions to be admitted if they are at least a minimum distance from the other member of the set.

3.2.5 Variable Depth Search

A variable depth search procedure for GAP was proposed in (Amini and M. Racer. 1994) that consists of a two phase effort to improve solutions . An initial solution is first generated randomly by assigning jobs to one of $m+1$ agents where the agent $i = m+1$ represents a dummy agent with an infinite cost for its use and having infinite capacity. Along with its associated objective function a lower bound on the problem is obtained by means of solving the linear relaxation to GAP. In the subsequent search phase improved solutions are sought by generating sequences of feasible task re-assignments that result in the reduction of the objective function value, where reassignments are achieved by means of standard shift and swap moves commonly used for GAP, until no improving sequence can be found in which case the algorithm terminates. Following presentation of the algorithm the remainder of the paper concentrates on the construction of a comprehensive testing system that is used to compare the relative efficiencies of the VDSH approach with that of three of the exact methods presented in 3.1 due to (Martello, Toth 1981), (Ross and P. Soland. 1975) and (Fisher, et al. 1986). The testing system has three components

a) A random problem generator
b) A set of user supplied codes to solve the randomly generated problems
c) An analysis module to perform statistical analysis.

Subsequently three separate experiments are described and the results reported

that compare firstly the performance of the four methods on a set of small problems, secondly the experimentation focuses purely on the comparison of the VDSH with the Martello and Toth algorithm in respect of the small test problems, and finally the third experiment compares the VDSH with the heuristic of Martello and Toth (Martello, Toth 1981) on a set of larger randomly generated test problems. The results indicate that VDSH performs well in terms of being able to find solutions of comparable quality to the three exact methods on the small test problems and in much shorter solution times, although performance seems inferior to that of the Martello and Toth heuristic on the large test problems and is only able to perform favourably if the VDSH is given the correct parameter settings thus highlighting the trade-off between quality of solution verses computation time.

Subsequent research undertaken and reported in (Amini and M. Racer. 1995) follows up on these findings by constructing a hybrid heuristic that generates solutions using the Martello and Toth heuristic and then subjects these high quality solutions to a refinement process using VDSH after first checking for optimality. Testing of the hybrid algorithm still seemed to indicate that the Martello and Toth algorithm was able to significantly outperform VDSH and the hybrid heuristic in terms of computational times however given longer running times the hybrid algorithm was able to find improved solutions.

A branching variable depth search approach to solving GAP was given in (Yagiura, et al. 1998) where branching trees are constructed by generating child nodes that represent solutions derived from the parent by means of searching the commonly used shift and swap neighbourhoods for GAP. The process begins initially by generation of a random solution which represents node 0 in the branching tree and is then subjected to a procedure termed "*SSS-Probe*", which is a local search procedure that uses modifications to the standard *shift* and *swap* neighbourhoods that are commonly used when

searching the solution space for GAP. The modification to the shift neighbourhood essentially involves the introduction of a memory structure that prevents a reversal of previous shift moves from being executed as would be utilised in a standard tabu search procedure. The modification to the swap neighbourhood however results in a much restricted neighbourhood in respect of size as the restriction imposed is to only consider swap moves that involve at least one of the agents included in the previous shift move. Each iteration of the algorithm consists of construction and evaluation of tree structures where the nodes of the tree relate to solutions which have been generated using "*SSS-Probe*" which begins by first shifting a job from its current agent to a new agent subject to the restrictions placed on the neighbourhood by the tabu list. The shift move chosen to be executed is the one that minimises an objective function penalised for violation of capacity constraints and the local search continues to a local optimum by subsequent moves to improving solutions within the modified swap neighbourhood and the locally optimal solution is used to create a child node which is then added to the tree. The number of child nodes generated by each parent is determined by a parameter of the method whose value produces tree structures of different types since the number of nodes generated at a given depth is a function of the improvement in objective function between depth 0 and depth d. If a new best solution is found whilst searching the current tree then this forms node zero of a new tree to be subsequently searched. The algorithm was tested on a set of benchmark problems of size $m = 5,10,20$ and $n = 100,200$ and performed better than most existing algorithms at the time both in terms of solution quality and the computational times required to obtain those solutions.

3.2.6 Other heuristics

3.2.6.1 Improvement heuristic of Martello and Toth

In addition to the exact branch and bound approach in (Martello, Toth 1981) also subsequently described in (Martello and Toth 1990) the authors present a heuristic approach to solving GAP using a two-phase approach that

first generates a feasible solution by allocating the assignment of jobs according to the difference between the best and second best assignments for each job. In the first instance the algorithm iteratively considers all unassigned jobs and allocates each a value based on the desirability of allocating the job j to the agent i and i' where the desirability of such an assignment (i, j) is determined by one of the following measures

a) c_{ij}

b) c_{ij}/a_{ij}

c) $-a_{ij}$

d) $-a_{ij}/b_i$.

Comparisons are then made to identify the job j having the largest difference between the assignments (i, j) and (i', j) and the assignment (i, j) is subsequently the next to be made. If as a result of this first phase, a feasible solution has been found then this is subjected to an improvement phase shifting the assignment of jobs to alternative agents whilst maintaining feasibility with regard to the capacity constraints. In addition to the heuristic algorithm for generating and improving feasible solutions the authors further propose a heuristic approach for fixing variables in order to determine whether such a solution is optimal. In the first instance variables are fixed to 0 if fixing them to one would exceed the best bound as calculated in the previously described branch and bound algorithm and determining whether a situation occurs whereby for a job j, all assignments $(i, j), i \in m$ are fixed to zero then the previously found feasible solution must be optimal. The second reduction phase considers bounds on each of the I knapsacks with regard to setting a variable equal to 0 or 1 as also described earlier and fixing such a variable dependent upon their effect on the best bound.

3.2.6.2 Variable fixing approach

A similar approach of fixing variables in order to reduce the problem was

subsequently implemented by Trick (Trick. 1992) who uses the solution to the linear programming relaxation of GAP in order to render certain variables *'useless'* thereby allowing them to be fixed to 0. As discussed in chapter 2.2 such a relaxation produces at least $n-m$ assignments and Trick provides proof that at least one variable relating to an assignment (i, j) for one of the unassigned jobs will require an amount of resource that is greater than the available capacity for such an assignment, if the naturally occurring assignments to the LP relaxation are fixed, and as such the corresponding variable can be fixed to 0. This result subsequently yields a quite straightforward heuristic approach comprising:

- solving an LP relaxation
- fixing those variables with value 1
- removing the *'useless'* variables and solving the LP relaxation of the resulting reduced problem.

This procedure iterates until no *'useless'* variables occur, in which case a feasible but not necessarily optimal solution has been found. The solution found using the LP heuristic is subsequently subjected to an improvement heuristic that attempts to find a better solution. The improvement phase also takes advantage of the results previously described by randomly fixing some portion of the previously obtained solution and subsequently solving for the remaining portion of the problem using the original relaxation heuristic. The approach was implemented and tested on standard type problems with up to 500 jobs and 100 agents and was able to produce some reasonably good solutions but still left room for improvements to be gained. A later paper by Cattrysse et al. (Cattrysse, et al. 1998) points out some deficiencies in Trick's method in that it does not always yield a feasible solution to GAP

3.2.6.3 Adaptive search heuristics

A general framework for adaptive search heuristics for GAP is proposed in (Lourenco and D. Serra. 2002)which considers two alternative strategies for generating initial solutions followed by local search procedures that begin

from these initial solutions in order to obtain better solutions. The approaches used for generation of initial solutions are based on the MAX-MIN ant system (MMAS) heuristic and the greedy randomized adaptive search procedure (GRASP). These two procedures are then combined with the two local search procedures, local descent and tabu search, to produce the following four alternative heuristics for solving GAP

a) GRASP with Local Descent
b) MMAS with Local Descent
c) GRASP with Tabu Search
d) MMAS with Tabu search.

In heuristics a) and c) an initial solution is generated by means of a greedy heuristic that considers each job $j \in J$ and allocates the job to an agent $i \in I$ according to some probability p_{ij} determined by the available resource at agent i and the resource required to perform the assignment a_{ij} . The chosen assignment is then made according to such probabilities except where the chosen agent has insufficient capacity to accommodate job j in which case the job is assigned to the first agent having such spare capacity, assuming one exists, otherwise the job is assigned to a random agent. This process is repeated until all jobs have been assigned which may result in an infeasible solution with regard to the capacity constraints in which case the objective function is penalised by addition of a penalty function $\alpha \sum_{i=1}^{m} \max\left[0, \sum_{j=1}^{n} a_{ij} x_{ij} - b_i\right]$ where $\alpha > 0$ represents the cost of using one extra unit of capacity. In heuristics b) and d) jobs are assigned to agents in the same greedy manner apart from the probability that is used being based upon the desirability of making such an assignment which the authors define to be based on the cost incurred by making such an assignment. The association with the ant colony approach (Dorigo, Di Caro 1999)is that as new improved solutions are found by means of the local search algorithm these initial probabilities change based upon the difference in the desirabilities between

the assignments in the starting solution and those in the improved solution. The second aspect of the general framework is that of local search for which the authors define two types of neighbourhoods, the first being a simple shift neighbourhood and the second an ejection chain neighbourhood. In the descent local search algorithm the neighbourhoods of the current solution are searched until an improving solution is found, known as the first improvement strategy, and a move to the new solution is executed. The search continues in this manner until no improving solution can be found in the neighbourhoods in which case the local search phase stops, necessary parameters are updated and a new iteration is started by generation of a new starting solution. The tabu search strategy uses the same neighbourhoods and moves but does not necessarily stop at a local optimum since non-improving moves are allowed during this approach and a return to previous solutions is prohibited by means of a tabu list that prevents reassignment of a job to an agent that has previously been shifted except where the aspiration criteria allows acceptance of a tabu assignment if the resulting solution is better than the best solution found during the search. Computational testing revealed that the addition of local search and tabu search to the MMAS and GRASP approach to generating initial solutions outperforms algorithms where MMAS and GRASP alone are used by experimenting with different combinations of the stated approaches and also indicated that a more sophisticated tabu search implementation might provide further improvements.

3.3 Assessment of the reviewed methods

The methods reviewed in this chapter have been compared against alternative methods on a variety of test problems. The methods of Martello and Toth (Martello and Toth 1990), Ross and Soland (Ross and P. Soland. 1975) and Fisher et al. (Fisher, et al. 1986) have all been compared on a set of randomly generated test problems with $m = 2,3,5$ and $n = 10,20$. There are 4 levels of difficulty and the reader is referred to (Martello and Toth 1990) or (Fisher, et al. 1986) for a description of how these problems were generated.

The results are reported in both (Fisher, et al. 1986)and (Martello and Toth 1990) but with different interpretations as to which performs best. Martello and Toth suggest that the Ross and Soland algorithm performs better on the easier problems whilst the Martello and Toth branch and bound algorithm performs better on the harder problems and the algorithm of Fisher et al. performs worse. This is in contrast to the interpretation of results in (Fisher, et al. 1986)where it is suggested that the algorithm of Fisher et al. performs better than both Ross and Soland and Martello and Toth, the latter suggesting that this may be due to the fact that results were obtained on different machines in the earlier research reported in (Fisher, et al. 1986).

Along with the presentation of their GA algorithm Chu and Beasley generated two sets of benchmark test problems for GAP. The first consists of 60 *small* problem instances with $5 \leq m \leq 10$ and $15 \leq n \leq 60$. Chu and Beasley report results of several methods on this small set which are summarized in table 3.3.2 along with the results of Haddadi and Ouzia (Haddadi and H. Ouzia. 2004), Diaz and Fernandez (Diaz and E. Fernandez. 2001) and the ejection chain approach in(Yagiura, et al. 2004)for this problem set.

The ejection chain tabu search was able to find all 60 optimal solutions for these small test problems in less than 1 second indicating that this set of problems was no longer suitable for testing more recent algorithms. As a result more recently developed and more powerful algorithms have been tested on larger problem instances. A description of these larger test problems along with a detailed comparison of a variety of algorithms including TSBB is presented in chapter 6.

Table 3.3.1 Number of optimal solutions found for small problems.

Method	Number of optimal solutions found
Martello and Toth heuristic	0
Fisher et al. branch and bound	26
Martello and Toth branch and bound	24
Hybrid simulated annealing/tabu search	39
Chu and Beasley Genetic algorithm	60
Haddadi and Ouzia branch and bound	57
Diaz and Fernandez Tabu Search	60
Ejection Chain Tabu Search	60

3.4 Summary

A wide variety of solution approaches for GAP have been researched, implemented and tested during a period spanning the last thirty years. During this time significant improvements have been made both in terms of practical and theoretical issues. Technological developments with regard to computational power have been significant as has the introduction of some powerful heuristic and metaheuristic procedures. The trade off between solution quality and computational times however is still a relevant issue and there are some methods that seem to be able to produce good quality solutions in very short computational times whilst other approaches tend to focus on producing extremely high quality solutions at the expense of longer running times. The suitability of a method therefore may be context dependent in terms of a real application. In conjunction with such developments researchers have attempted to solve harder and more complex test instances that have resulted in a challenging set of test libraries. These are now freely available and important in terms of measuring further development as they provide a benchmark against which alternative algorithms can be compared. More

recently attempts have been made to hybridise certain strategies and, as can be seen from much of the research carried out, there are a number of possibilities for constructing new algorithms that use aspects from, and combinations of, alternative approaches in the construction of such hybrid methods and this research has been conducted with this hybrid approach in mind. It is thought however that there still exists scope for further research into solution methodologies for GAP. This is motivated by evidence that the problem is still actively being researched by leading researchers in the field of combinatorial optimization, in addition to the requirement of solutions to ever more complex problem instances. Two of the approaches, Tabu Search and Branch and Bound, reviewed in this chapter have been the subject of significant research and have provided some very efficient algorithms for solving GAP and so it seems reasonable to suggest that a combination of the two might prove interesting. The following chapter provides an outline and overview of these two methodologies and at the same time identifies and summarizes some of the key literature that is relevant to the development of the two methodologies.

4 An overview of Branch and Bound and Tabu Search

This chapter presents an overview of both the *Branch and Bound* (B&B) and *Tabu Search* (TS) methods. In the following section 4.1, an outline of how the B&B algorithm works is presented which includes relaxation of the integer problem, selecting variables for branching to create new sub-problems, consideration of how such variables should be separated and the types of constraints that can be used to achieve such separation and how to select the next sub-problem to be solved from those outstanding sub-problems yet to be explored. Section 4.2 then presents a description of the basic TS approach and how this incorporates the use of local search techniques to search the solution space. Explanation is also provided as to how the use of memory, both short and long term, is incorporated into the search process in order to enhance its ability to find good integer solutions, intensifying the search around previously found good solutions and also by escaping local optima and diversifying the search in order to explore previously unexplored areas of the solution space. More sophisticated implementations of TS require the use of more strategic techniques and so some more strategic aspects of the method are outlined in this section too. Finally some examples of how TS has been applied to solve a variety of real-world combinatorial optimization problems will be discussed.

4.1 Branch and Bound

The Branch and Bound algorithm is an exact optimization algorithm used for solving integer programming problems and was first introduced by Land and Doig (Land and A. G. Doig. 1960) and is today incorporated into several commercial solvers, including Xpress-MP, as the default algorithm used to solve integer and mixed integer programming problems. The following subsections describe the basic components that form the basis of the algorithm although there is considerable scope with regard to implementation of the

algorithm for a variety of problems. The B&B strategy is essentially one of divide and conquer and begins by first solving a relaxed version of the integer problem. An optimal solution to the relaxed problem can typically be found quite quickly and is subsequently used to define two or more sub-regions that exclude the relaxed solution previously found but still include the optimal integer solution to the original problem, assuming that one exists. This separation process is defined by a tree structure where the sub-problems are represented by nodes of the tree and the branches represent the rules for separating the nodes of the tree.

4.1.1 Relaxation

The remainder of this chapter considers the integer programming problem formally stated as

$$\begin{aligned} &\text{minimize} && \mathbf{cx} && (4.1)\\ &\text{subject to} && A\mathbf{x} \le \mathbf{b} && (4.2)\\ & && \mathbf{l} \le \mathbf{x} \le \mathbf{u} && (4.3)\\ & && x_j \text{ integer } \forall j \in J, J = \{1,2,\ldots,n\} && (4.4) \end{aligned}$$

where n is the size of the column vectors $\mathbf{x}, \mathbf{l}$ and $\mathbf{u}$ with objective function 4.1 and constraints 4.2 as previously described in chapter 1.1 with the stipulation that all elements of $\mathbf{c}$ are integer values. Constraints 4.3 specify restrictions on the upper and lower values allowed to be taken on by each decision variable x_j in the column vector $\mathbf{x}$. Constraints 4.4 specify that each decision variable x_j must have an integer value in the solution.

Generally the B&B algorithm will first solve the linear programming (LP) relaxation obtained by removing the integrality constraints 4.4 and allowing the decision variables to take on fractional values within the specified upper and lower limits. This LP relaxation can typically be solved quite quickly to yield one of three possible outcomes:

i. The LP relaxation is infeasible.
ii. An optimal solution to the LP relaxation has been found and one or more of the n decision variables has a fractional value.
iii. An optimal solution to the LP relaxation has been found and all n decision variables are integer valued.

Outcome **i** indicates that the integer problem must be infeasible, outcome **iii** indicates that the optimal solution to the integer problem has been found and only if outcome **ii** occurs is additional work required in order to solve the integer problem. In case **ii** it is necessary to select a decision variable having a fractional value in the solution to the relaxation and to then generate two or more sub-problems each having a feasible region that excludes the optimal solution to the parent problem, whilst further ensuring that one of the newly generated problems has a feasible region that contains the optimal solution to the integer problem. This is achieved by the addition of branches to represent constraints that restrict the values that the selected decision variable is allowed to take in a solution to the original problem.

Whilst most commercial solvers use an LP relaxation as the basis for the B&B solver alternative relaxations can also be used in problem specific implementations of B&B such as those reviewed in the previous chapter by (Ross and P. Soland. 1975) and (Martello and Toth 1981). Both of these approaches maintain the integrality constraints and the former of these two relaxes the knapsack constraints whilst the latter relaxes the assignment constraints which result in two very different branching schemes. Both of these relaxations exploit the structure of the problem although the Ross and Soland (Ross and P. Soland. 1975) scheme still proceeds by selecting a single decision variable for branching, whilst the Martello and Toth (Martello, Toth 1981) scheme constructs a branching scheme based on a multiple choice approach. Having solved the relaxed problem it is then necessary to identify those variables that are infeasible with regard to the problem constraints. In

the case of the LP relaxation to the integer programming problem those decision variables having fractional values violate the integer constraints.

4.1.2 Variable selection

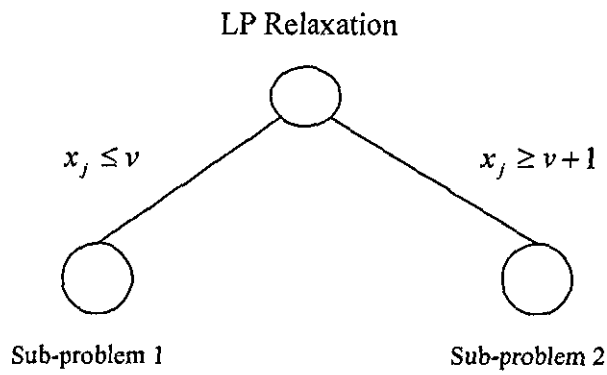
Having identified those variables that violate the problem constraints with regard to the relaxed problem it is then necessary to select a variable or set of variables which will define the division of the current sub-problem into two or more child problems. Consider a solution $\mathbf{x}$ to an LP relaxation of an integer problem which contains one or more decision variables whose values are fractional. One relatively simple way to select a decision variable on which to separate might be to select the variable whose value is most infeasible in relation to integrality. In order to assess feasibility of a decision variable x_j where $v < x_j < u$ and v and u are integer values and $u - v = 1$ then a measure of infeasibility for x_j can be defined as $\min\{x_j - v, u - x_j\}$. Other frequently used approaches for deciding which variable to select for separation are those of specifying priorities, penalties and degradations. Knowledge of a particular problem context could be used in order to set priorities that determine the order in which the decision variables are to be chosen for separation. For example a variable that represents a major decision within the problem formulation may be given a higher priority than one whose variation in the solution has less of an impact. Applying penalties to a relaxed solution and estimating the degradation in the change in objective function that would occur by forcing a fractional decision variable to an integer value are also commonly used approaches for selecting the variable on which to separate next. A detailed description of applying penalties and estimating degradation can be found in (Nemhauser and Wolsey 1998).

4.1.3 Separation

The structure of the branch and bound tree is determined by the way in which separation is applied to the current problem under consideration to produce subsequent child problems. In the general case where each integer

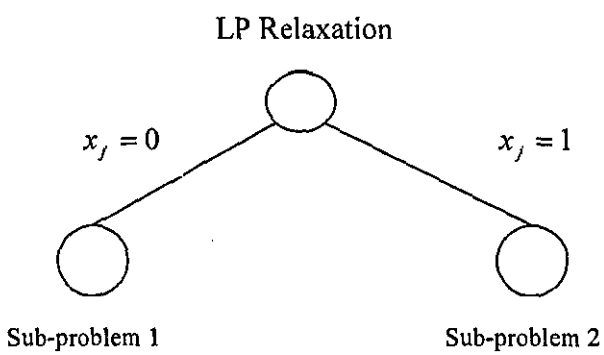
decision variable x_j is restricted by an upper bound, u_j, and lower bound, l_j, on its value in the integer problem, and its value in the solution to the current LP relaxation is fractional where $v < x_j < v+1$ and v is the greatest integer less than x_j, then separation can be achieved by imposing two branches that will ensure that x_j cannot take on values between v and $v+1$ in the solution to any subsequent relaxation as shown in figure 4.1.1.

Figure 4.1.1 Branching on integer variables.



In the zero-one case where some or all of the decision variables are constrained to be binary, branching on a binary decision variable x_j is achieved by means of applying two equality constraints as apposed to the inequalities used for branching in the general case as in figure 4.1.2. This results in a binary decision tree that fixes the value of the variables x_j to either 0 or 1 as in figure 4.1.2

Figure 4.1.2 Branching on binary variables



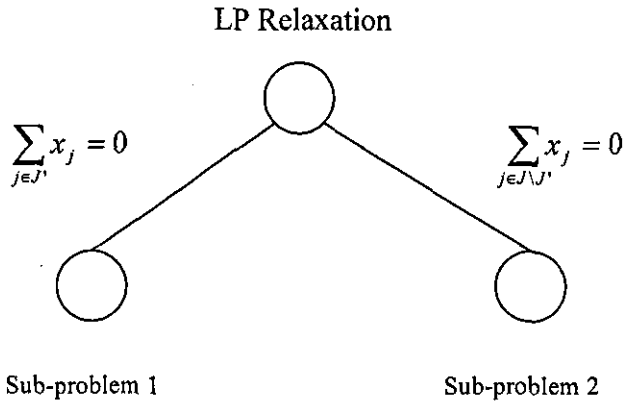
Consistent with the latter of the two approaches already described, a multiple branching scheme can also be applied where appropriate by imposing additional branches using equality constraints where the right hand side of each constraint is an integer between u_j and l_j.

The two branching schemes previously described branch on a single decision variable however there are many integer programming problems (including GAP) that are modelled using constraints of the form

$$\sum_{j \in J} x_j = 1 \qquad J = \{1,2,\ldots,n\}$$

where there is a requirement to select one variable from a subset of the problem's decision variables to take on the value 1. In such a case it may be beneficial to branch from the current node using a subset of variables as in the example of figure 4.1.3.

Figure 4.1.3 Branching on a subset of variables.



where J' can be constructed using information available from the solution at the parent node.

The separation approaches outlined in this section are commonly used to solve a variety of integer programming problems and are utilised in commercial solvers including Xpress-MP.

As the branch and bound tree grows during the solution process there will be a number of nodes representing sub-problems that are yet to be solved, these are commonly called the leaf nodes and the process of selecting which node to consider next in the solution process is now described in the following section.

4.1.4 Node selection

There are several ways that the next node to be examined from the set of *leaf* or *active* nodes can be chosen including

a) *Best First:* As the name suggests this approach considers all of the outstanding nodes and selects the node that is best where best is assessed according to some pre-determined evaluation criteria. Such evaluation might be achieved by considering the value of the objective function of the solution to the relaxed problem at the node, or by using some type of estimating approach to approximate the value of a possible integer solution from that node. However the evaluation is achieved the node with the best evaluation is the one chosen next for development. Clearly this approach can become time consuming as the size and complexity of the problem increases and so may not be suitable in some cases.

b) *Depth First:* This strategy always considers the child nodes that are generated as a result of branching being applied to the current node. The child nodes are one level deeper in the tree than the parent hence the approach is known as the depth first approach. The child node to be selected for further development can be chosen by means of evaluation as described in a) or by means of consideration according to some pre-defined order. If the current node does not require any further development then the algorithm is said to *backtrack* up the tree to the deepest parent node that has child nodes that have not yet been examined.

c) *Local First:* The local first approach considers the children and sibling nodes of the current node as candidates for selection as the next node to be processed. If all child and sibling nodes require no further development then the algorithm must backtrack and select a node according to some alternative criteria, possibly the deepest or best node available in the tree.

Whilst these are some of the most commonly used strategies for node selection it is possible to combine these kinds of approaches to produce other alternatives for specifying the order in which nodes are selected.

4.1.5 Bounds, fathoming and pruning

Construction of the branch and bound tree using the strategies described in the previous subsections would result in complete enumeration as a means of finding an optimal integer solution. The effort required to solve an integer programming problem to optimality can be significantly reduced by the use of *bounds* and *pruning.*

4.1.5.1 Bounds

Solving the LP relaxation of the original problem will yield an objective function value that can be used to generate a bound on the best integer solution that can be found, assuming that the relaxation has a feasible solution. In a minimization problem the bound can be generated by rounding up the objective function value to the nearest integer since all of the objective function coefficients are integer. If, during the solution process, an integer solution is found such that the difference between its objective function value and the generated bound is zero then clearly this integer solution must be optimal and therefore there is no need to continue the solution run. Various strategies can also be used to improve the bound for example by generating additional constraints, fixing the values of variables, specifying restricted ranges of values that variables are allowed to take and numerous heuristic strategies. Improving the bound however obviously requires additional effort

during the solution process and so consideration should be given to the amount of effort expended in improving the bound verses the actual gains to be had from such improvements.

4.1.5.2 Fathoming

During the branch and bound search a bound on the objective function value of the best integer solution obtainable will be available, as described in the preceding section. An integer solution to the problem however may or may not be available. Solving the LP relaxation at any node of the branch and bound tree will result in one of the following situations:

a) The relaxed sub-problem has no feasible solution.
b) The relaxed sub-problem has a feasible solution but its objective function value is worse than the best known integer solution.
c) The relaxed sub-problem has a feasible solution, its objective function value is better than the best known integer solution but one or more of the variables required to take integer values are fractional.
d) The relaxed sub-problem has a feasible solution, its objective function value is better than the best known integer solution and all of the integer constraints are satisfied.

In situations a, b and d the node is said to have been fathomed since in situation a) there is no feasible region that can be further divided, and in situations b) and d) the feasible region of the sub-problem cannot contain an integer solution that is better than the best integer solution already known. It is therefore only necessary to branch on the current node in situation c).

4.1.5.3 Pruning

When a new best integer solution to the problem is found during the branch and bound search this new information can be used to *prune* the tree by essentially fathoming those leaf nodes that have yet to be developed whose relaxed objective function is worse than the objective function value of the new best solution. If all outstanding nodes have been fathomed then clearly the branch and bound process ceases and the best integer solution found, if

indeed one has been found must be the optimal integer solution to the problem.

4.2 Tabu Search

In contrast to the exact approach described in the preceding section Tabu Search (TS) is a heuristic strategy that has been successfully utilized in order to solve a variety of hard real world optimization problems. TS is one of a group of heuristic approaches that are known as *meta-heuristics* which are essentially heuristic strategies that are used to guide other heuristics to explore the solution space of a problem. TS was first introduced by Glover in 1989 (Glover. 1989) where he presents the fundamental principles that form the basis of the approach. This was followed by a second paper, (Glover. 1990), that presents refinements and some more sophisticated aspects of the method. In the first paper a basic TS algorithm is proposed termed *Simple Tabu Search* which can be used to guide a local search heuristic beyond locally optimal solutions where the local search procedure would normally cease, in order to search for alternative local optima. This is achieved by allowing the search to execute moves to non-improving solutions whilst maintaining a record of changes in solution attributes that have occurred during the previous t moves. Forbidding the reversal of the changes in solution attributes recorded during the previous t moves renders such solutions *tabu.* In its simplest form such a record is just a list T, of size t, where each entry in the list identifies the changes in attributes that have occurred during the previous t moves. Clearly a list that is allowed to grow up to size $t = +\infty$ will ensure that each new solution visited will never be revisited however this can prove very costly in terms of computational time and effort particularly as problems become large and complex. Empirical evidence has shown that small values of t can be quite effective in controlling the search by preventing a return to those solutions visited during the previous t moves with a constant amount of effort required at all stages of the search process with regard to checking whether a solution is in the list T. The flowchart in figure 4.2.1 depicts the simple tabu

search algorithm that can be used as a basis for development of a more sophisticated TS algorithm.

The description of *Simple Tabu Search* highlights the need for certain basic ingredients to incorporate into most TS formulations

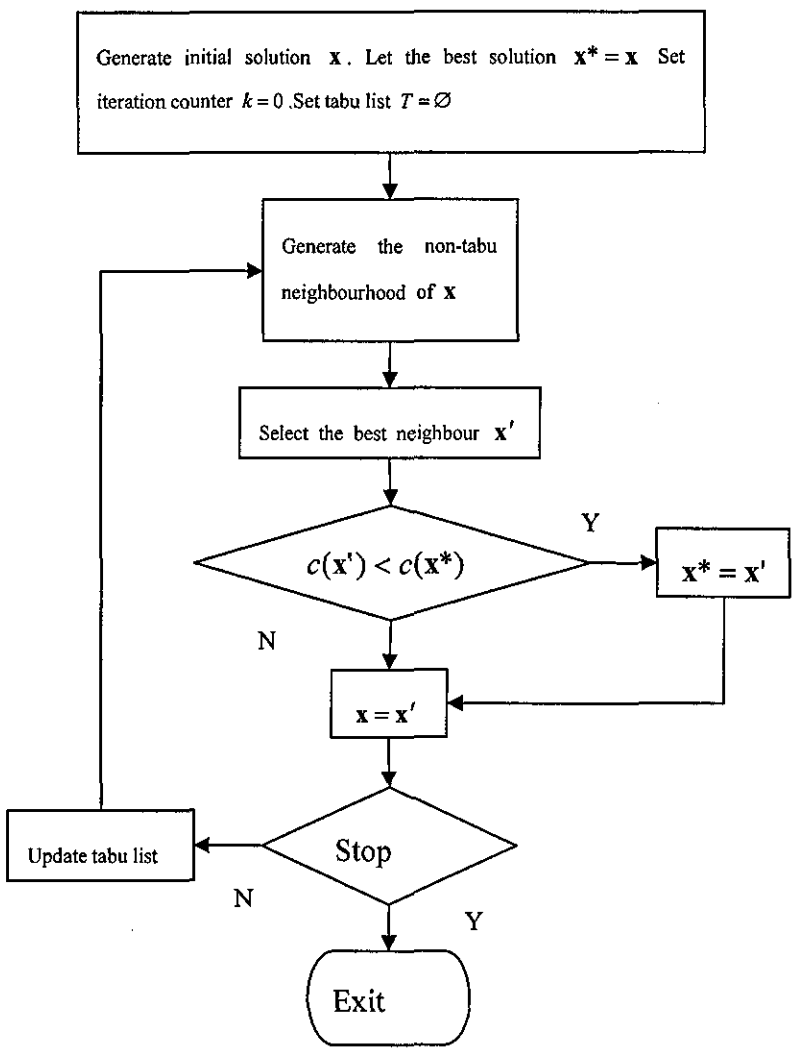
a) A suitable initial starting solution.
b) A relevant neighbourhood structure.
c) A suitable Tabu list.
d) A suitable evaluation function.
e) Some form of stopping criteria.

All of the above are largely problem dependent although Glover does give some guidelines relating to the form of the evaluation function *'OPTIMUM'* indicating that this may take on an *'aggressive orientation'* in contrast to other methods that use a less aggressive approach based on the premise that this gives a better chance of reaching a local optimum that is also global. A discussion of the size and structure of the tabu list is also given along with an example of how this might be achieved.

In the remainder of Part I Glover goes on to outline some more advanced aspects of the strategy that can be incorporated into the algorithm in order to produce more powerful and sophisticated applications.

The first of these aspects is the use of some form of aspiration criteria that define the circumstances under which tabu status may be overridden, such as allowing a move to be admissible if it leads to a solution that improves the best solution found so far, or in circumstances where all neighbouring solutions to the current solution are rendered tabu then the oldest tabu restriction may be lifted in order that the search may continue. Glover goes into some detail about how aspiration criteria may be defined but the key idea here appears to be that tabu restriction and aspiration criteria have a dual role to play in an effective implementation of TS.

Figure 4.2.1 Simple Tabu Search algorithm



Two other key strategies that may be included in a TS approach are those of strategic oscillation and medium-term and long-term memory structures. Strategic oscillation can be useful in certain circumstances where it may be advantageous to allow the search to cross the feasible boundary and allow the search to access a feasible solution via a path that would not otherwise be available. Some control is required however to ensure that the search does not travel too far into the infeasible space and that it does not travel too far in the opposite direction either, hence the term oscillation. This is a particularly

relevant consideration for this research since some harder GAPs are very tightly constrained and thus may have very small feasible regions and in fact some problem instances may have no feasible region at all. This technique has been incorporated into a TS strategy for solving GAP by Diaz and Fernandez (Diaz and Fernandez 2001) with some success.

The use of medium and long-term memory structures are also introduced as a means of intensifying or diversifying the search at certain stages of the process. Tabu search utilizes these types of memory as a means of identifying regions of the solution space that deserve more thorough consideration and also to move the search to previously unexamined regions. In contrast to heuristic methods that use randomization in order to restart the search at different points, TS seeks to use information gathered during the search in order to make more informed decisions as to which regions of the solution space should be searched.

The final concept presented in this first part is the use of a probabilistic approach within the TS framework, see (Lokketangen and Glover 1996). This initially appears to be in contrast with the deterministic approach of TS since inclusion of a probabilistic strategy introduces an element of randomization to the search, whereas the principles of TS are those of an intelligent search procedure basing decisions regarding the selection of moves on information collected by the search process as opposed to making decisions for executing a move according to some probability. Glover points out however that there are gains to be made in terms of efficiency by a reduction in record keeping and evaluation operations, whilst also losing some of its efficiency due to possible repetition and duplication that would not be present in the more systematic approach.

The incorporation of some or all of these more advanced aspects in various forms provide considerable scope for the development of different TS strategies that can be tailored to suit the required approach for different

problem types. Many new applications of TS have been successfully constructed and tested during the last 17 years as a result of the concepts and ideas presented in Part I and thus this work is considered extremely important since it provides a broad foundation for other researchers to develop more efficient solution methods for many different types of problems.

Of prime importance to the efficiency of a TS approach is the way in which the tabu lists are managed. In Part I Glover describes the structure of a static tabu list that is fixed in size and contains the reversal of those moves executed in the previous t iterations, where t is the size of the tabu list. In Part II the concept of dynamic tabu lists are introduced and a detailed explanation of two types of dynamic tabu list are given. One such approach is the Reverse Elimination Method (REM) also described in (Glover and Laguna 1997) and (Dammeyer, F. and Voss, S. 1993). Both of these strategies involve the construction of a new tabu list at each iteration by identifying sequences of moves that would result in a return to a previously visited solution. Whilst this approach can be advantageous in terms of preventing a return to a previously visited solution it has to be realised that there will be an increase in the amount of computational effort required to manage this type of list as the search progresses. As such there is some scope for researchers in discovering ways to reduce this computational effort whilst maintaining the advantage given to the search by the use of these types of dynamic tabu lists.

Glover goes on to discuss the application of dynamic tabu strategies to search processes that can be viewed in terms of stages or levels. The suggested strategy is that of marrying tabu lists to different stages and in the case where the stages, or levels, can be viewed as a hierarchy such as a branch and bound tree, then it is appropriate to allow the list to grow as the search progresses to deeper levels of the tree, and to purge the list associated with deeper levels as the search backtracks towards the root node of the tree. Glover concludes that this may be a highly desirable procedure, whether this

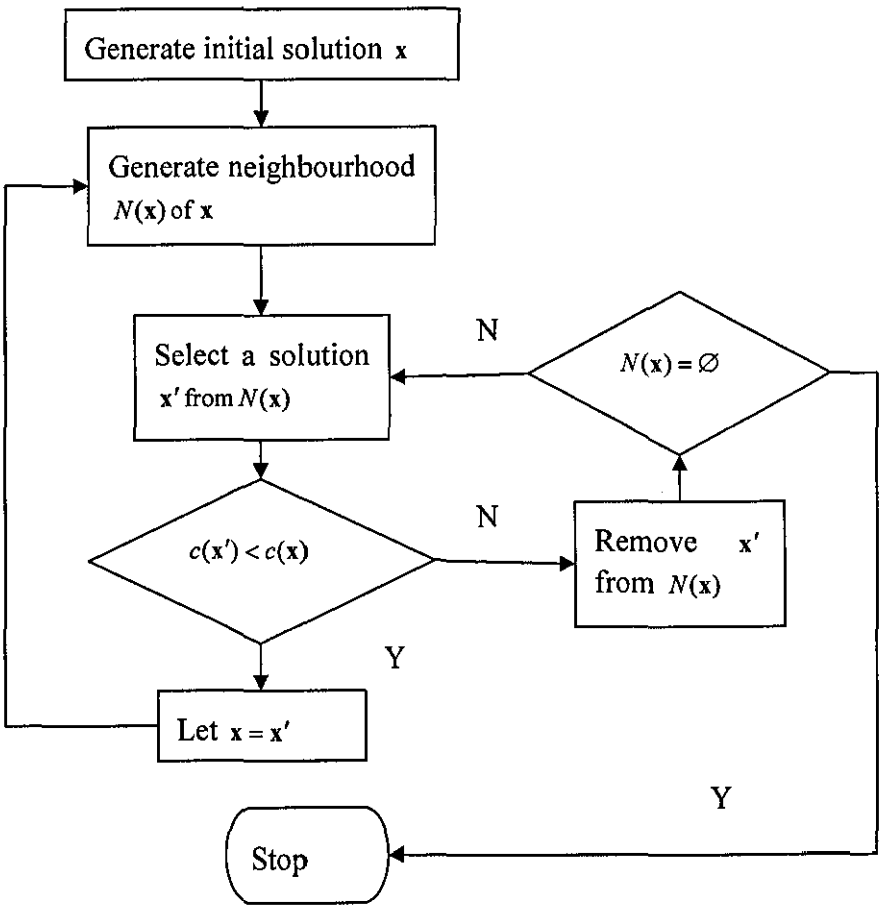
is actually achievable in terms of implementation, and if so whether the benefits of doing so confirm what Glover is suggesting remain to be proven. The remainder of this chapter now describes the local search procedure and how the use of memory can be used to strategically guide a local search heuristic through the problem solution space in order to discover good locally optimal solutions.

4.2.1 Local Search

Local search heuristics can be used to search the solution space of a problem by executing a sequence of moves from one solution to the next where the next solution is in the neighbourhood of the current solution. In order for the search to begin an initial solution must first be generated and methods for doing this are numerous but some commonly used approaches include random generation where values are allocated randomly to decision variables, and heuristically where values are allocated to decision variables according to a set of rules that may take into account the structure of the problem. The neighbourhood of this initial solution must then be defined along with a set of rules specifying how the defined neighbourhood will be searched. During the search of the neighbourhood an evaluation must be made with regard to the benefit of moving from the current solution to the neighbouring solution such as the change in objective function obtained by implementing such a move. The goal in searching the neighbourhood is to find a solution that is better than the current solution. A straightforward evaluation can be achieved by comparing the objective function value of the current solution $\mathbf{x}$, $c(\mathbf{x})$, with the objective function value of the neighbouring solution $\mathbf{x}'$, $c(\mathbf{x}')$. If the aim is to minimize the objective function then clearly if $c(\mathbf{x}') < c(\mathbf{x})$ then a move from $\mathbf{x}$ to $\mathbf{x}'$ will be an improving move. Figure 4.2.2 represents a simple local search procedure where the objective is one of minimization. The solution output from this local search is clearly locally optimal with regard to its neighbourhood, however it may not be globally optimal in terms of the original problem. A

key consideration is the approach taken to searching the neighbourhood to find improving solutions. Clearly one could evaluate every solution in the neighbourhood and execute a move to the solution that yields the greatest improvement, assuming that the current solution is not a local optimum with regard to the current neighbourhood. Complete evaluation of the neighbourhood may not always be practical in terms of the amount of computational effort required for such an evaluation at each iteration of the search. This may occur where the size of the neighbourhood is extremely large or where generation and evaluation of each neighbour requires considerable effort. Tabu search incorporates several strategies for more efficient ways of searching the neighbourhoods and these are described in 4.2.3.5.

Figure 4.2.2 Local search algorithm.



4.2.2 The use of memory

Introducing the use of memory into a local search procedure, as in a tabu search setting, brings several benefits in terms of exploring the solution space more fully. On reaching a local optimum, the simple local search described in the previous section must cease as there are no improving moves available. Allowing the search to accept non-improving moves will overcome this situation and allow the search to continue. In certain circumstances this may result in the search simply reversing the sequence of improving moves that facilitated its arrival at the local optimum. On its return to the solution from which the sequence of improving moves began, depending on how improving moves are selected from the neighbourhood, the search may simply descend once again to the same local optimum from which it began its ascent. Such behaviour is known as *cycling.* Cycling can be overcome by the introduction of a short-term or *recency* based memory that forbids a return to the t most recently visited solutions. There are several alternative approaches for implementing a short-term memory aspect and these are further discussed in sub-section 4.2.2.1.

Whilst short-term memory seeks to prevent the search from becoming trapped at a local optimum and allows the search to continue past this point, two important strategic aspects of TS are those of *Intensification* and *Diversification.* Both of these longer-term strategies can often be achieved with the aid of frequency based memory in order to identify solution attributes or collections of solution attributes that may indicate regions of the solution space that seem promising with regard to providing good solutions and also to guide the solution towards previously unexplored areas of the solution space .

4.2.2.1 Recency based memory

A standard approach to implementing a recency based short term memory would usually be to maintain one or more Tabu lists that record information about the solutions visited during the previous t iterations of the search. Each

entry on such a list might be a vector that represents the solution explicitly or, for reasons of practicality and to reduce computational effort, might simply identify those attributes that have changed in order to transform one solution into the next solution. Maintaining a simple static list of size t is guaranteed to prevent the occurrence of cycles up to size t. Clearly if cycles of size greater than t are detected then action is required in order to break such a cycle and guide the search towards some alternative path. The list size t is a parameter of the algorithm and will usually require experimentation in order to identify values, or ranges of values that are effective. Empirical evidence has shown however that quite small values of t can be quite effective for a variety of problems. The tabu list is updated at each iteration of the search by simply replacing the oldest member of the list with the newest, or attribute information of the newest solution.

An alternative, and computationally efficient method of maintaining a short-term recency based memory, can be achieved using a tabu structure whose elements represent solution attributes of the problem. The value of each element then defines whether each attribute is tabu at a given iteration of the search. Those attributes that are involved in a move from one solution to the next at iteration k will have their tabu status recorded by setting the corresponding elements in the tabu structure to a value of $k+t$. At iteration $k+1$ any attribute having a tabu status of greater than or equal to $k+1$ is forbidden from being included in a move at the current iteration. Once again the value of t can be fixed or alternatively can be allowed to vary dynamically within some pre-defined range. The dynamic approach may allow t to be selected randomly within the given range at each iteration or may be deliberately increased or decreased during the search according to the recent history of the search.

4.2.2.2 Frequency based memory

Longer term strategies in TS tend to incorporate some form of frequency

based approach. Counting the frequency with which solution attributes are included in a solution is commonly known as *residence* frequency whilst counting the number of times that a solution attribute changes during the search is usually known as *transition* frequency. Residence frequencies can be useful in identifying attributes or collections of attributes that consistently appear in a number of good quality solutions or indeed poor quality solutions. Such information can be useful as a means of focusing the search temporarily on regions of the solution space that contain solutions with attributes that have high residence frequencies when considering good quality solutions and low residence frequencies when considering poor quality solutions. Diversification strategies can also take advantage of residence frequencies for example by forcing the inclusion into the solution of a small number of attributes that have rarely or never been included in solutions that have previously been visited. In the GAP context this could mean the inclusion or exclusion of assignments of jobs to agents.

It can sometimes be beneficial to count how many times certain solution attributes change as this can help to identify parts of solutions which are less important in relation to how they can affect the quality of the solution. In other words it might be more productive to focus on assigning values to variables based on high residence values and allowing those attributes with high transition values to be dealt with subsequently. How the frequency memory is implemented and what information is collected in terms of frequencies however depends on the strategic approach to searching the solution space and the following section 4.2.3 now considers some of these aspects and how they can be incorporated into a TS algorithm

4.2.3 Strategic aspects

Strategically TS incorporates a number of principles that can be combined in order to guide a local search heuristic to find high quality solutions to a

variety of difficult problems. This section outlines some of the more widely used aspects and refers the reader to (Glover and Laguna 1997) for a more detailed description of these and other TS strategies.

4.2.3.1 Searching the neighbourhood

It is important to define a set of rules that determine how the neighbourhood of the current solution will be examined at each iteration of the search as this can impact both on the quality of the best solution found and also on the amount of computational effort required to execute each iteration. Two commonly used approaches for examining the neighbourhood are to (a) accept the best admissible solution or (b) accept the first admissible solution. In order to apply strategy (b) then it is first necessary to order the solution attributes in some way that they become candidates to be changed as part of a move to the next solution. Having ordered the attributes they should then be examined in order by evaluating the moves associated with them until an admissible move is found. For example a move may be admissible if it exceeds an amount of improvement in objective function or some other threshold that is relevant. When an admissible solution has been found however the search of the neighbourhood ceases and the solution is accepted as the next solution. Applying strategy (a) however does not require any ordering of the attributes of the current solution since in order to find the best admissible solution then it is usually necessary to search the entire neighbourhood.

4.2.3.2 Intensification

The purpose of an intensification strategy in a TS approach is to identify regions of the solution space that appear attractive by some criteria and thus to implement a more thorough search of search areas. Such strategies are usually based on some form of historical data that is collected and stored during the search process. One way of doing this would be to maintain a frequency based memory as discussed in section 4.2.2.2 and then to fix or constrain a large

subset of the problem variables based on information which indicates that certain variables take on certain values or ranges of values with a relatively high frequency and allow the search to continue under these conditions for a short period of time.

An alternative method to that of focusing the search on sub-regions that contain a large number of attributes found in good solutions could be to maintain a list of *elite* solutions and to systematically or periodically initiate an intensification phase of the search beginning from one of the elite solutions and at the same time clearing the short-term memory. An elite solution list would typically contain the e best solutions found during the search process but will require updating as solutions are used for each intensification phase. Consideration must also be given to criteria other than the evaluation function when assessing the suitability of a solution to be added to the elite list, for example if a solution is close by some measure to a solution already in the list then its inclusion may simply result in a duplication of effort at a subsequent intensification phase. One way to achieve this may be to establish a threshold in respect of distance from each solution already in the list and to only add a solution if it exceeds such a threshold, alternatively it may be acceptable to penalize the evaluation of a potential elite solution according to how close it is to those solutions already in the list and then add it to the list if the penalized evaluation is acceptable.

A lesser used strategy is that of recording information about the unvisited neighbours of previously visited solutions and selecting such high quality solutions as a means of initiating a return to regions of the solution space that have been visited previously but starting from a different solution in this region.

4.2.3.3 Diversification

Diversification strategies are important in the TS approach as they use

historical information as a means of forcing the search into sub regions of the solution space that have previously not been explored. This may be as a means of breaking a cycle that has trapped the search process and cannot be overcome by means of the short term memory in operation, or to overcome barriers or peaks in the solution landscape that cannot otherwise be traversed during the normal search process. Diversification can often be achieved by modification of the rules for selecting solution attributes to be changed that would otherwise not be selected under normal circumstances. A common approach is to employ some form of trigger that instigates a diversification move or series of moves such as the number of iterations performed without improving the best known solution. In such a situation a frequency memory can then be consulted to identify attributes that have never or seldom been included in the solution. A move or series of moves can subsequently be implemented that encourage, or in some cases force one or more of the rarely seen attributes to be included and possibly remain in the solution for some period of time. Conversely, a similar approach can be employed in order to exclude attributes that have high residence frequencies thus encouraging the search to incorporate solutions that have possibly not been seen before. The normal search procedure resumes from the solution identified by the diversification phase, which may be of poor quality in terms of evaluation but may provide access to a path that can lead to a local optimum that improves the best know solution. It may be the case that such a solution may never have been accessible without such modification.

4.2.3.4 Strategic Oscillation

The strategy of controlling the search so that its movement can be viewed as a form of oscillatory pattern about some form of boundary can be useful as a means of approaching such a boundary from a number of different directions. If, for example, the boundary is deemed to be the boundary of the feasible region where, under normal circumstances the search would stop or turn and retreat away from the boundary, then there may be an advantage to

be had by allowing the search to cross the boundary and move into the infeasible region for a period of time as an approach back towards the boundary from a different direction may allow access to solutions on or near the boundary that could not otherwise have been reached if the search were restricted only to the feasible region. A key component of the strategy is that of controlling the oscillation so that it moves away from the boundary sufficiently to allow the approach back towards it to be different from previous approaches, whilst at the same time restricting the movement away from the boundary so as to prevent subsequent approaches from bypassing good solutions that are close to the boundary. This is commonly achieved by modifying the evaluation function dependent upon which side of the boundary the search is, and which direction it is moving relative to the boundary. An example of such a strategy is used by Diaz and Fernandez (Diaz and E. Fernandez. 2001) who use an evaluation function that includes a weighted penalty for infeasibility which is systematically increased and decreased according to which side of the boundary the search has occupied in the previous iterations.

4.2.3.5 Candidate Lists

Candidate list strategies attempt to reduce the amount of effort involved in identifying good solutions in the neighbourhood of the current solution, particularly where such neighbourhoods tend to be very large, very complex in terms of evaluation, or both. The *Aspiration Plus* candidate list approach as used by Budenbender (Budenbender, et al. 2000) defines a threshold for the quality of a move prior to beginning the neighbourhood scan. The neighbourhood is then examined until the first move that exceeds the threshold is found, at which point a counter is initialized and the number of moves examined subsequently is restricted to *plus* and the best move found is chosen to be applied in order to continue the search. Additionally the scan can be forced to examine at least a minimum number of moves and at most a maximum number of moves. In this situation if the threshold is reached by a

move prior to the minimum number of moves having been examined then the counter does not initialize until the minimum value is reached. Also if the maximum value is reached prior to *plus* moves having been examined then the maximum value overrides the value of *plus*.

An alternative candidate list strategy is known as the *Elite Candidate List* strategy. In this approach a large number of neighbouring moves, or possibly even the entire neighbourhood, are examined and the best k moves are recorded. In the subsequent iterations the best move from the list of *elite* candidates is chosen to be executed as opposed to scanning the neighbourhood at each iteration. Since the quality of a move included in the candidate list may change at each iteration a threshold is established so that when the best move on the list falls below this threshold the list is reconstructed using the neighbourhood of the current solution.

A more detailed discussion of these and other candidate list strategies such as the *Successive Filter Strategy* and *Sequential Fan Candidate Lists* can be found in (Glover and Laguna 1997).

4.3 Summary

This chapter has attempted to provide the reader with an overview of the two contrasting approaches of Branch and Bound and Tabu Search. The vast and wide ranging literature for both of these approaches includes numerous variations and implementations of both methods being applied to a variety of difficult real-world problems. Many such methods have been tailored to be problem specific and as such some highly sophisticated and novel approaches have evolved. The basic components of each approach as briefly described in this outline of the two methods form the basis for many of the more sophisticated algorithms that have been constructed and the following chapter describes the hybrid approach developed as part of this research.

5 Hybrid Tabu Search / Branch and Bound

Recently researchers have attempted to combine aspects of both exact and heuristic approaches in order to construct more powerful solution methods for solving hard combinatorial optimisation problems. A survey and classification of such approaches is given in (Puchinger and Raidl 2005) where the authors classify such approaches into *Integrative* and *Collaborative* methods. The hybrid approach described subsequently falls into the integrative category and uses a Tabu Search metaheuristic as a master strategy and the branch and bound solver Xpress-MP as a sub-strategy for solving restricted instances of the original problem. The approach of restricting a large number of problem variables and using an exact method to solve the resulting sub-problem is suggested by Glover and Laguna in (Glover and Laguna 1997) and is termed *Referent Domain Optimisation* . An application of such an approach is given in (Budenbender, et al. 2000). Fischetti and Lodi (Fischetti and A. Lodi. 2003) and (Danna, et al. 2005) propose solution methods for general Mixed Integer Programming (MIP) problems also based on this approach and these two methods are further discussed and outlined in sections 5.2 and 5.3 respectively due to their relevance to this research, in terms of their strategic approach to generating neighbourhoods defined by smaller sub-problems of the original.

5.1 Referent Domain Optimization

Referent domain optimisation is referred to in Glover and Laguna (Glover and Laguna 1997) as being a process of using optimisation methods together with heuristic processes to generate trial solutions which can then be optimised to generate a new solution and the process repeated. Several examples of how this can be achieved are given.

The strategy being developed as part of this research is in keeping with an observation made by Glover and Laguna in (Glover and Laguna 1997) which

is based on the fact that a problem with a small number of zero-one variables can typically be solved quite quickly using a branch and bound solver such as Xpress-MP. The idea therefore is to fix a large number of the problem variables to produce a restricted integer problem and then to call the branch and bound solver to assign values to the relatively small number of remaining free variables. In relation to GAP advantage can be gained from the fact that solving the LP relaxation of an instance of GAP will yield a solution containing a large number (at least $n-m$) of assignments of jobs to agents. It is not unreasonable to suggest that such assignments could be considered attractive in terms of generating a complete integer solution and so the approach of fixing the values of the relevant variables in order to generate a restricted integer problem seems sensible. The remaining unassigned jobs can then be considered by the branch and bound solver in an attempt to produce a complete solution. As a result of optimising the restricted problem a feasible integer solution may or may not have been found and it is then necessary to continue the search by generating a new collection of assignments that are different to the previous collection in order to continue the process. The use of Tabu memory structures is required in order to guide the process with regard to the generation of sets of assignments that can be fixed in order to provide restricted regions that can quickly be searched by the branch and bound solver, along with longer term strategies to provide information for intensification and diversification strategies.

A key question here is that of deciding which of the problem variables should be fixed and which should be left to be dealt with by the exact solution method. The Local Branching method subsequently described in section 5.2 uses what is termed a *soft* fixing approach.

5.2 Local Branching

The approach of defining solution neighbourhoods to be the feasible region of a sub-problem of the original problem is described by Fischetti and

Lodi in (Fischetti and A. Lodi. 2003). Having first generated an initial solution to the problem (the incumbent solution) their approach constructs a branching tree where each node of the tree represents a sub-problem that is constrained to be within a certain distance of the current incumbent solution. Given an incumbent solution $\mathbf{x}$ whose elements are constrained to take on binary values then the distance $\Delta(\mathbf{x},\mathbf{x}')$ where $\mathbf{x}'$ is an alternative feasible solution to the problem, can be measured as

$$\sum_{j\in S_1}(1-x'_j)+\sum_{j\in S_0}x'_j \qquad (5.1)$$

where $S_0=\{j:x_j=0\}$ and $S_1=\{j:x_j=1\}$. Furthermore if $|S_0|$ and $|S_1|$ for all feasible solutions to a problem are constant then the distance between two solutions can be considered to be $\Delta'(\mathbf{x},\mathbf{x}')$ where

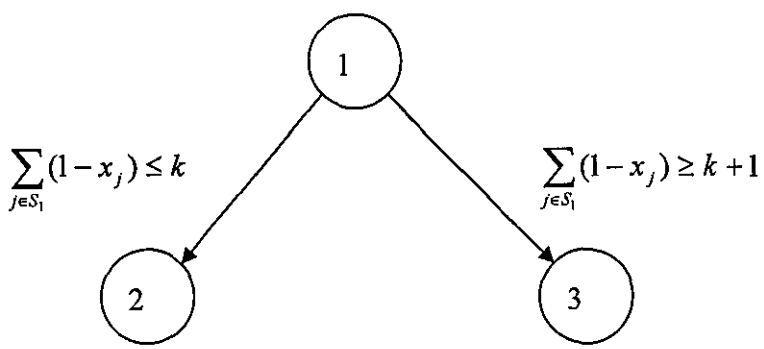
$$\Delta'(\mathbf{x},\mathbf{x}')=\sum_{S_1}(1-x_j) \qquad (5.2)$$

or alternatively

$$\Delta'(\mathbf{x},\mathbf{x}')=\sum_{S_0}x_j \qquad (5.3) \quad .$$

A neighbourhood of the current incumbent solution $\mathbf{x}$ can therefore be considered to be those solutions that are within a distance k of $\mathbf{x}$ and therefore branching from the current node of the tree can be achieved by applying inequality constraints using either equations 5.2. or 5.3 as shown in figure 5.2.1.

Figure 5.2.1Local branching node separation.



In this situation the neighbourhood of the incumbent solution defined by node 2 is explored first using a branch and bound solver. A time limit is applied for searching the neighbourhood and the search ceases either when a new incumbent solution has been found or the time limit has been reached without improvement of the incumbent solution. If the time limit is reached without discovering a new incumbent solution then the search backtracks to node 1 and reduces the size of the neighbourhood k to $k/2$ and nodes 2 and 3 are regenerated for the new value of $k/2$. If a new incumbent solution is found within the time limit then the search backtracks to node 1 and the new incumbent solution is used to identify a new neighbourhood to be explored and nodes 2 and 3 are regenerated. Local Branching also incorporates a diversification aspect that is triggered whenever no improved solution can be found at the current node. In such a situation the search is widened by increasing the size of the neighbourhood by say $k/2$, if no improved solution can be found in the enlarged neighbourhood then a second diversification step is invoked which accepts the first solution found in the neighbourhood after removing the bound restriction to the problem.

The approach being used in this local branching method is consistent with referent domain optimization in that the objective is to fix a large number of variables in order to define a much smaller sub-problem that can then more easily be solved using an exact method (in this case Branch and Bound) . Local Branching however defines the proportion of problem variables in the incumbent solution that are to be fixed when moving from one solution to the next, and so is less restrictive since any of the variables may change value. This is in contrast to the approaches of sections 5.3 and 5.4 which specify the variables that must be fixed to the same values in both the current and the new solution which is more in keeping with the referent domain optimization approach. An important consideration common to all of these approaches is the size of the neighbourhood k as explicitly fixing say 90% of the problem variables to take specific values results in a sub-problem requiring less

computational time and effort to solve than one generated by the local branching approach which only specifies that 90% of the problem variables should be fixed but does not specify which ones.

5.3 Relaxation Induced Neighbourhood Search (RINS)

As has already been suggested in section 5.1 the difficulty in constructing sub-problems obtained from fixing a large number of problem variables is deciding which problem variables should be fixed, and which should be left to the exact solver to take care of. In a heuristic approach the attractiveness of fixing certain variables to certain values can be assessed by considering the objective function coefficients or constraint coefficients or sometimes a combination of both. In addition Tabu Search can also make use of longer term frequency memories in order to identify values, or indeed ranges of values, taken by certain variables in high quality solutions. Exact methods such as branch and bound also use evaluation techniques such as assessing reduced costs in order to decide which sub-regions should be subsequently explored as defined by the branches that restrict the values of certain problem variables.

The Relaxation Induced Neighbourhood Search (RINS) approach, as proposed by Danna et al in (Danna, et al. 2005), takes advantage of information supplied by two high quality solutions available at various nodes of the branch and bound tree. Both solutions are available at any feasible node of the tree. The first of these is the current incumbent solution (the best feasible solution found during the search up to the current point) and the second is the relaxed solution obtained when solving a particular node. Since both solutions can be considered to be attractive in some respect, in terms of objective function value with regard to the relaxed solution, and in terms of feasibility with regard to the incumbent solution, it seems fairly logical to isolate solution attributes that are common to both solutions and to treat them as being highly attractive when attempting to construct a new improved best

solution. The RINS algorithm was designed to be a general mixed integer problem (MIP) solver and as such does not take into account the underlying structure of the problem being solved. It simply attempts to construct new improved solutions to the problem by formulating and searching neighbourhoods, by means of generating sub-problems, where the neighbourhoods derived are common to both solutions available at the node of the tree under consideration.

The RINS method proceeds by searching the branch and bound tree of the problem using a commercial solver (in this case CPLEX) but pauses at certain intervals where the size, f, of each interval is quite large and is measured by the number of nodes processed. This interval is set to be quite large due to the fact that neighbourhoods generated by nodes that are close together tend to be quite similar resulting in considerable amounts of duplicated effort. Assuming that the node at which the search pauses is a feasible node, and that an incumbent feasible solution is available, a sub-problem is defined by fixing those variables having the same values in both the incumbent and relaxed solutions, and then allocating values to the remaining variables according to the solution of the sub-problem. Generating and solving these sub-problems is purely an attempt to improve on the incumbent solution and in practice such sub-problems have the potential to be quite large and thus require extensive computational effort. The amount of time spent exploring the sub-problem is therefore tactically truncated by the imposition of a node limit. If the solution attempt yields a solution with better objective function value than the current incumbent then the incumbent solution is updated, as are the bounds for the original problem and the main branch and bound search continues. Despite the use of large intervals between generations of sub-problems there still exists the potential to generate neighbourhoods that can be quite similar and searching them may require a certain amount of duplicated effort. This is accepted without any attempts to eliminate such duplicated effort. By contrast in the TSBB approach subsequently described in the following section 5.4, the

use of memory does assist attempts to reduce some of the computational effort that may be involved when searching neighbourhoods that may be similar to others that have already been searched.

Section 5.4 now describes the new algorithm, TSBB, which has been developed as part of this research. As with the approaches already described in this chapter, TSBB incorporates a variable fixing approach in order to generate sub-problems that represent neighbourhoods of solutions that can be searched using the exact branch and bound method incorporated within the Xpress-MP solver. A Tabu Search strategy is employed, incorporating both short-term and longer-term memory structures that guide the search in terms of generating neighbourhoods that are different to neighbourhoods that have been previously generated during the search process. A detailed description is now given.

5.4 The Hybrid TSBB Algorithm

The motivation behind the development of the hybrid approach for solving GAP has been provided by the successful implementations of both Branch and Bound and Tabu Search (see Chapter 3) for solving GAP. The review of the literature of solution approaches for solving GAP has provided two key insights into the development of the hybrid algorithm. The first of these is that Branch and Bound is quite effective at solving small to medium sized instances of the problem, however this effectiveness deteriorates quite quickly in relation to larger, and more difficult instances. The second is that TS approaches have outperformed most alternative heuristic approaches in terms of solution quality when compared with the results of these alternative approaches on a set of benchmark test problems. This is particularly true in the case of the larger, more difficult problems. Also evident from the literature is that the approach of relaxing problem constraints for GAP can be extremely advantageous when constructing heuristic algorithms although most approaches tend to focus on relaxation of the capacity constraints or the

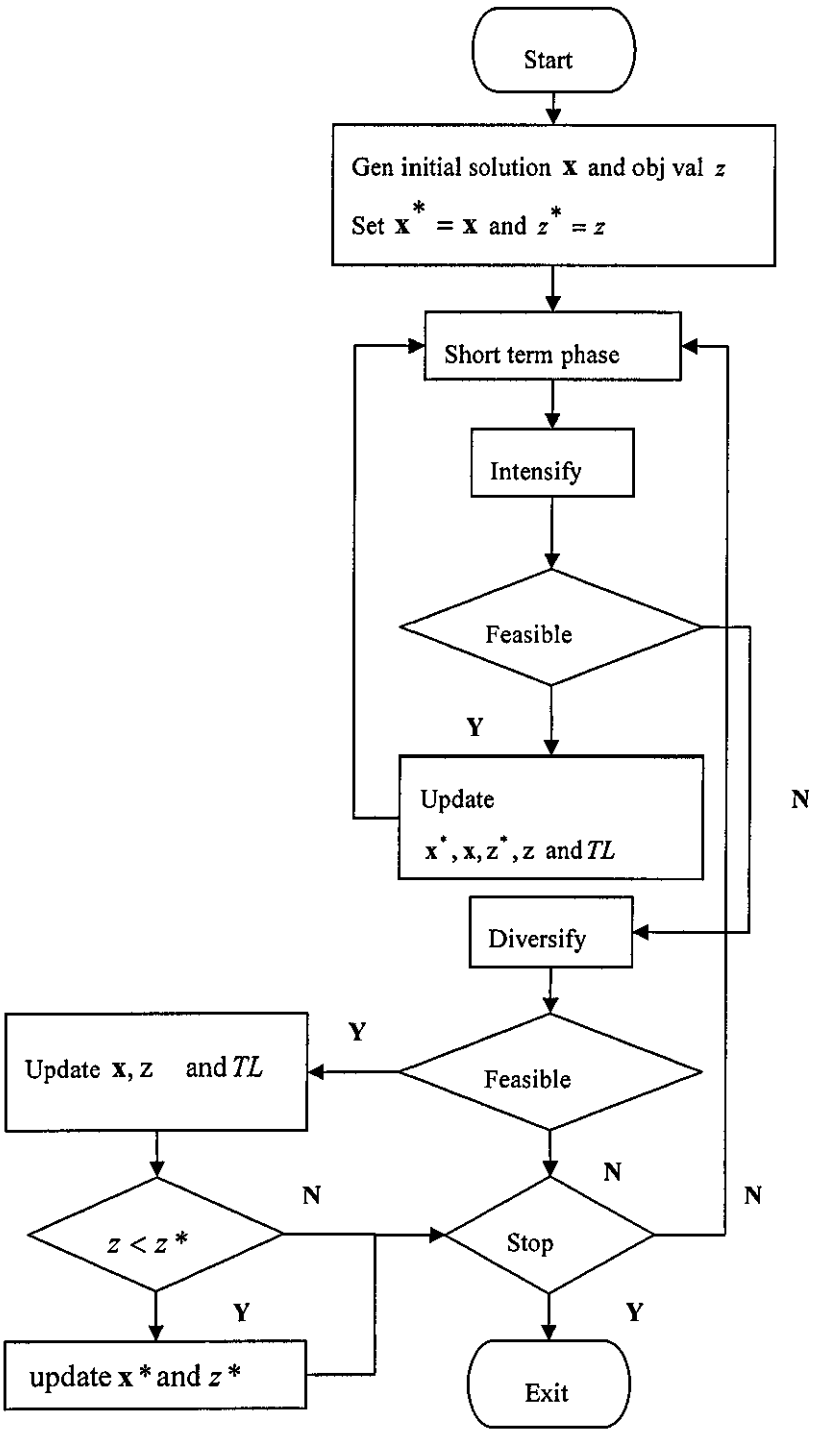
assignment constraints. The TSBB algorithm also takes advantage of a relaxation of GAP, but in contrast to most of the alternative relaxation approaches relaxes the binary constraints and also takes advantage of the fact that the resulting LP relaxation can typically be easily solved using a commercial software package (in this case Xpress-MP). Quite importantly the resulting relaxed solution to the problem contains a relatively large number of variables whose values are binary and thus feasible in relation to the original integer problem. In light of these observations it seems intuitive to consider a referent domain optimization approach since the infeasible portion of the problem, identified from the relaxed solution, can typically be more easily solved as a smaller sub-problem using the exact branch and bound method. The resulting partial solution can subsequently be combined with the integer feasible section of the relaxation to produce a complete integer feasible solution.

The TSBB algorithm proceeds by first applying the Xpress-MP branch and bound solver to an instance of GAP, as defined in chapter 2.2, until the first integer feasible solution is found. The resulting solution is then used to initialize $\mathbf{x}^*$, the best solution found during the search, $\mathbf{x}$ the current and initial solution, and the corresponding objective function values z^* and z respectively. If the branch and bound search completes without finding an integer solution then clearly the problem is infeasible and the search ceases at this point. The benchmark test problems used during the computational testing of TSBB did not pose any significant difficulty for the Xpress software with regard to finding integer feasible solutions very early in the search. These initial solutions were typically of poor quality but provided suitable starting solutions for TSBB. It may be possible to generate problem instances that pose a more difficult test in terms of generating initial solutions and an alternative method for generating initial solutions may be required. Although TSBB does require an integer solution to begin the search this does not necessarily have to be feasible and so initial solutions could either be

generated randomly or by alternative means if necessary. The algorithm proceeds by iteratively performing a short-term search phase, followed by an intensification phase. If a new improved solution is found during the intensification phase then this solution is used as the starting point for the next short term phase. If the intensification phase fails to find an improving solution then the next short-term phase begins from a solution found as a result of implementing a diversification phase. The short-term search phase is described in detail in section 5.4.3 and is itself iterative in that it moves through the feasible solution space from the current solution $\mathbf{x}$ to a neighbouring solution $\mathbf{x}'$. A move from $\mathbf{x}$ to $\mathbf{x}'$ is achieved by re-allocating one or more assignments in $\mathbf{x}$ in order to transform it into $\mathbf{x}'$. The neighbourhoods that are used in order to identify those assignments that are to be re-allocated in order to obtain the new solution are termed *Drop/Add* neighbourhoods and, as the name suggests, have two components that can be thought of as sub-neighbourhoods. The transformation from $\mathbf{x}$ to $\mathbf{x}'$ is a two-stage process that first considers the drop neighbourhood whilst the second stage considers only those assignments that have been dropped and attempts to reallocate the unassigned jobs in order to produce $\mathbf{x}'$. This approach is somewhat different to the more traditional approach of using shift or swap neighbourhoods as described in the following section along with a detailed description of the *Drop/Add* neighbourhood. Information gathered from integer feasible solutions identified during this short-term phase of the search is subsequently used in order to identify a promising sub-region of the solution space that can be searched in order to try to improve the current best solution, where the sub-region is constructed according to frequency information collected from the integer solutions found during the short-term phase. The sole purpose of the intensification phase is to search for solutions that are better than the best solution found up to this point during the search and in this respect follows the branch and bound convention in that a cut-off value is set at the beginning of each intensification phase according to the objective function of the current best solution found. If a feasible solution is

found during this phase then it follows that this must be better than any previous solution found and so $\mathbf{x}^*$ and z^* are updated accordingly, and the new solution is also added to the frequency memory. The diversification phase is implemented as a means to move the search away from the regions of the solution space explored during the preceding short-term and intensification phases. This is carried out in order to identify new and, as yet, unexplored regions that may contain solutions which improve the best solution found during the search to this point. The new solution, if one is found during this diversification, is then utilized to initiate the next short-term phase of the search. If no new solution is found as a result of the diversification phase then the next short-tem phase of the search simply resumes from the last solution found during the previous short-term phase. The stopping condition for the algorithm is simply a time limit that is set dependent upon problem size, details of which are given in chapter 6. The flowchart in figure 5.4.1 depicts the TSBB algorithm.

Figure 5.4.1 Flowchart of the TSBB algorithm.



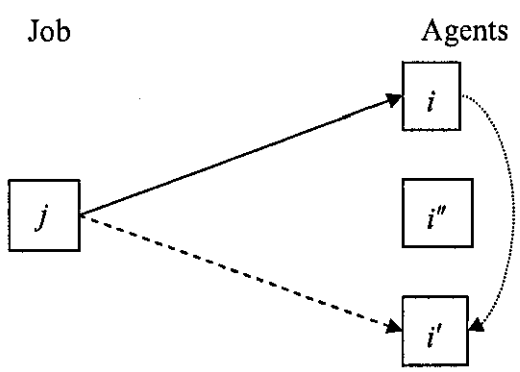
5.4.1 Solution Neighbourhoods

Two commonly used local search neighbourhoods for GAP are the *Shift* neighbourhood and the *Swap* neighbourhood. More recently however more complex neighbourhoods have been used such as *Double Shift* and *Ejection Chains.* Both the *Shift* and *Swap* neighbourhoods, and indeed the *Double Shift* neighbourhood, can be considered to be *Drop/Add* neighbourhoods since they all result in the deletion of at least one assignment and replace the deleted assignments by adding new ones. The *Drop/Add* neighbourhood used by TSBB however is more in keeping with the ejection chain approach in that it allows for more than two jobs to be reassigned during a move from one solution to the next and therefore these neighbourhoods can be considerably larger and consequently may require considerable effort to search. In order to combat excessive amounts of computational effort being applied to the search of a neighbourhood it is common practice to apply some form of stopping criterion that cuts short the scan of the neighbourhood and indeed TSBB follows this convention as described in more detail in subsequent sections.

Shift Neighbourhood

A solution to GAP consists of a set of assignments of jobs to agents such that each job is uniquely assigned to one agent. A shift neighbourhood consists of those moves that can be made by reassigning a job j from its current agent i to a different agent i'' as in figure 5.4.2.

Figure 5.4.2 Shift neighbourhood.

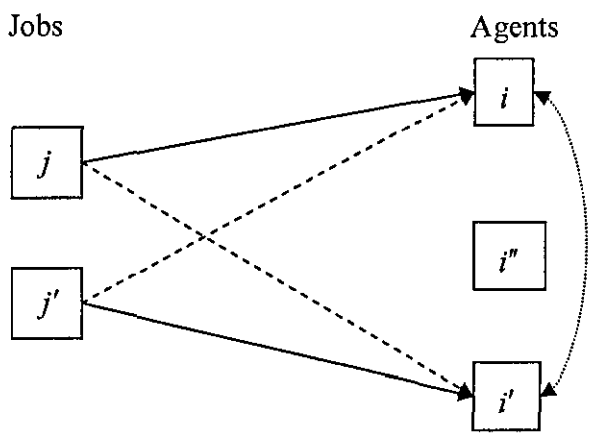


A feasible shift move for a job j can be obtained simply by comparing the available capacity of each agent $s_{i'}, \forall i' \in I$ where $i' \neq i$, with the amount of resource that would be consumed if job j were to be performed by agent i', $a_{i'j}$. Clearly if $a_{i'j} \leq s_{i'}$ then job j can be feasibly reassigned from agent i to agent i'. Infeasible shift moves can also be included in a shift neighbourhood in a process that allows infeasible moves to be made, and are usually evaluated by means of a penalty applied to such a move based on $a_{i'j} - s_{i'}$ when agent i' does not have enough available resource to perform job j.

Swap Neighbourhood

Consider the two assignments (i, j) and (i', j'). A swap neighbourhood consists of those moves that can be made by reassigning job j from agent i to i' and also reassigning job j' from agent i' to i as shown in figure 5.4.3.

Figure 5.4.3 Swap neighbourhood.

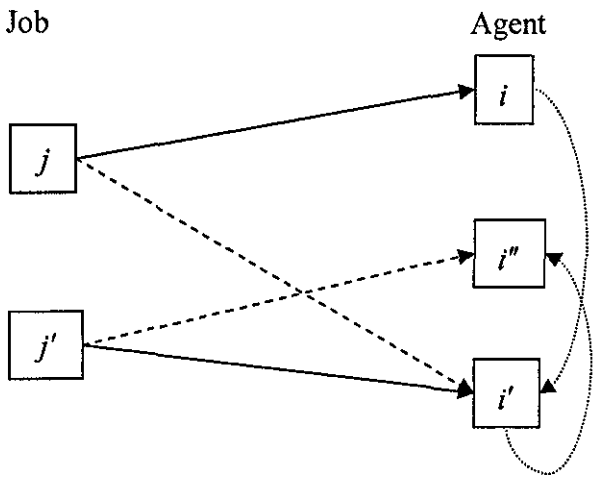


Double-Shift Neighbourhood

This neighbourhood was used by Tai-Hsi et al. (Tai-Hsi Wu., et al. 2004)

and consists of moves that shift a job j from an agent i to an agent i' and then also shift a job j' from agent i' to any other agent i'' as shown in figure 5.4.4.

Figure 5.4.4 Double-shift neighbourhood.



Both components of the swap moves and the double-shift moves are clearly shift moves when treated in isolation however it is important to highlight that in both cases the first shift move is dependent upon the second shift move.

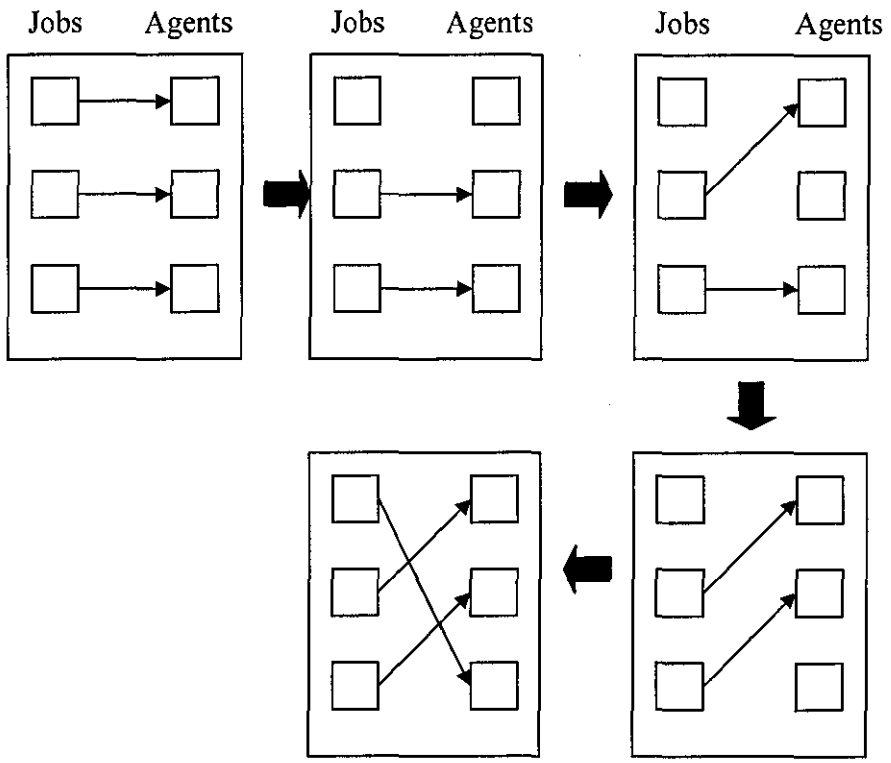
Ejection Chain Neighbourhoods

Ejection chains are used to create complex, powerful moves and have been utilised in by Yagiura et al.(Yagiura, et al. 2004), for instance an ejection move firstly de-allocates an assignment (i, j) from a complete solution so that job j is un-assigned and then the resultant incomplete solution is subjected to a series of attempts to reassign some of the remaining assignments until finally the job j that was initially ejected can be reassigned to another agent to form a new trial solution as depicted in figure 5.4.5. The ejection chain move for GAP can also be viewed as a series of shift moves of length l where each shift component in the chain is dependent upon the previous shift component. In fact the shift move can be considered to be an ejection chain move of length $l = 1$ and the swap move and double–shift move can be

considered to be ejection chain moves of length $l = 2$.

The following section 5.4.1.1 now gives a description of the *Drop/Add* neighbourhood that has been derived for use in the TSBB algorithm. It is important to note that this approach differs from the systematic approaches described previously for de-allocating and re-allocating assignments since the neighbourhoods are defined by feasible regions to both restricted linear programming and integer programming formulations of the original problem.

Figure 5.4.5 Ejection chain neighbourhoods.



5.4.1.1 Drop/Add Neighbourhood

Given an integer feasible solution $\mathbf{x}$ to GAP, a move from $\mathbf{x}$ to $\mathbf{x}'$ can be achieved by first dropping one or more assignments in $\mathbf{x}$ and then reassigning

those jobs that are unassigned as a result of the drop phase. In order to generate $\mathbf{x}'$ at least one assignment in $\mathbf{x}'$ must be different from those assignments in $\mathbf{x}$. In order to make the move from $\mathbf{x}$ to $\mathbf{x}'$ TSBB first drops one or more assignments from the set of assignments S_x, defined by $\mathbf{x}$, where $S_x = \{(i,j) : x_{ij} = 1\}$ to obtain the set of assignments S_y that should be retained for inclusion in $\mathbf{x}'$. As a means of deciding which assignments to drop TSBB takes advantage of the fact that the optimal solution to the linear programming relaxation of GAP will typically contain a relatively large number of integer assignments in terms of the total number of jobs n, m being the number of agents available to do the n jobs, in fact at least $n-m$ jobs will be uniquely assigned in the LP solution and as n is typically much larger than m there will be relatively few assignments to be made in addition to those contained in S_y in order to obtain $\mathbf{x}'$. The construction of S_y is achieved by solving an LP relaxation of GAP, GAPLR where

$$
\begin{aligned}
GAPLR = \quad & \min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} y_{ij} \\
& \text{s.t} \quad \sum_{j=1}^{n} a_{ij} y_{ij} \le b_i \qquad \forall\ i \in I \qquad (5.4) \\
& \qquad \sum_{i=1}^{m} y_{ij} = 1 \qquad \forall\ j \in J \qquad (5.5) \\
& \qquad 0 \le y_{ij} \le 1 \qquad \forall\ i \in I, j \in J \qquad (5.6)
\end{aligned}
$$

and so the drop neighbourhood can be defined by the feasible region of GAPLR. Whilst this can be extremely large and contain many solutions it is fairly straightforward, in most cases, to obtain an optimal solution to GAPLR using the Xpress-MP solver and hence obtain a good set of assignments S_y. The optimal solution $\mathbf{y}$ to GAPLR can now be used to obtain $S_y = \{(i,j) : y_{ij} = 1 \text{ and } (i,j) \in S_x\}$ and subsequently the set of assignments D that have been dropped from S_x is given by

$D = \{(i,j) : (i,j) \in S_x \text{ and } (i,j) \notin S_y\}$. In order to ensure that at least one or more assignments are dropped from S_x and so $|D| \geq 1$, the following constraint is added to each instance of GAPLR

$$\sum_{(i,j)\in S_x} y_{ij} \leq |S_x| - 1 \qquad (5.7).$$

Each instance of GAPLR, is then further restricted by a tabu constraint which prevents those assignments that were added during the move to **x** in the previous iteration, from being dropped in the move to **x'** during the current iteration as is subsequently discussed in section 5.4.2.

Having dropped one or more assignments from S_x in order to obtain S_y by means of solving a linear relaxation of GAP it is then necessary to add assignments to S_y in order to obtain $S_{x'}$. The add neighbourhood can be thought of as the feasible sub-region region of GAP that can be defined by further constraining the original integer programming formulation for GAP to include all assignments (i,j) that are contained in S_y. This leads to the following restricted instance of GAP, RGAP where

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

$$\text{s.t} \quad \sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i \qquad \forall\ i \in I \qquad (5.8)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall\ j \in J \qquad (5.9)$$

$$x_{ij} = 0 \text{ or } 1 \qquad \forall\ i \in I, j \in J \qquad (5.10)$$

$$\sum_{(i,j)\in S_y} x_{ij} \geq |S_y| \qquad (5.11)$$

A key aspect of this approach is due to the fact that typically the number of assignments contained in S_y will be large by comparison to n. The

implications of this with regard to solving *RGAP* are extremely advantageous as constraint (5.11) is quite restrictive in terms of reducing the size of problem to be solved by the branch and bound solver. In the implementation of the TSBB algorithm constraint 5.11 is enforced by fixing the lower bound on those variables representing assignments in S_y to 1 . This approach of fixing variables that are identical in both an integer feasible solution and a solution to an LP relaxation is in keeping with the approach of Danna et al (Danna, et al. 2005), as described in section 5.3, although their approach does not take advantage of the structure of the problem and is only utilized at various intervals during a branch and bound search . Constraint 5.11 also has similarities with a local branching constraint as described by Fischetti and Lodi (Fischetti and A. Lodi. 2003) and discussed in section 5.2. The difference between constraint 5.11 and the local branching constraints however is that constraint 5.11 specifies which problem variables must be fixed, in contrast to the soft-fixing approach of Fischetti and Lodi (Fischetti and A. Lodi. 2003) who specify only the proportion of problem variables to be fixed and allow the branch and bound solver to determine which ones to fix. Constraint 5.11 is therefore more restrictive as the size of the feasible region is more focused. The optimal solution, assuming one exists, to RGAP is clearly the best solution in the neighbourhood and so is defined to be the new solution $\mathbf{x}'$. This new solution $\mathbf{x}'$ has been obtained by first dropping those assignments in D from S_x to obtain S_y and then allowing the Xpress-MP branch and bound solver to add new assignments to S_y in order to obtain $\mathbf{x}'$.

In the following section 5.4.2 the short-term phase of the search is described in detail explaining how the use of short term memory is utilized in conjunction with the *Drop/Add* neighbourhoods in order to generate feasible integer solutions and direct the search through a region of the solution space in the short-term.

5.4.2 Short-Term Phase

Tabu search employs adaptive memory structures in order to direct the search in the short term when the search proceeds from one solution to another solution in the neighbourhood and also in the longer term in order to identify promising regions of the solution space that warrant a more thorough search as well as a means of driving the search into new unexplored regions of the solution space.

Short term or *recency-based* memory is used to record historical information about changes in attributes that have taken place within the previous t iterations of the local search and its purpose is to prevent a return to a solution visited within this period and thus prevent the local search from becoming trapped in a cycle. A short term memory structure would typically take the form of a list containing those attributes that are forbidden from being included in subsequent solutions. In the extreme case this list would be of unlimited size where solution attributes are added to the list at each iteration, however as this list grows there are implications in terms of running time since the list has to be checked at each iteration. This problem can be overcome by the use of fixed size tabu lists where, at each iteration, the most recent tabu attributes are added to the list replacing the oldest tabu attributes. The size of such a list is largely context dependent and takes into account problem size and also the strength of the rule that identifies an attribute as being tabu. The size of such a list is therefore an important parameter of the method and requires some experimentation in order to identify a range of suitable values. Having identified these values a suitable list size can be determined, alternatively the size of the list can be varied dynamically as in (Tai-Hsi Wu., et al. 2004).

There are two short term memory structures used in TSBB, the first identifies those assignments that have been dropped as a result of solving *GAPLR* which are those assignments contained in D. Each entry on this list

T_{add}, of size t_a, is a vector identifying the column indices of those variables x_{ij} where the assignment (i, j) is contained in D. The restricted problem $RGAP$ is constructed at each iteration by the addition of constraint 5.11 in addition to a constraint corresponding to each entry on the list T_{add} of the form

$$\sum_{(i,j)\in D} x_{ij} \leq |D| - 1 \qquad (5.12)$$

to GAP. The purpose of this is to prevent the combination of assignments contained in D from being reinstated in the next t_a iterations. The second structure T_{drop} is a vector containing the column indices of those variables x_{ij} contained in the set $A = \{(i, j) : (i, j) \in S_{x'} \text{ and } (i, j) \notin S_x\}$. The purpose of this tabu restriction is to prevent those assignments that have just been added in the move from $\mathbf{x}$ to $\mathbf{x}'$ from being dropped in the next solution to $GAPLR$. This is achieved by the addition of the following constraint to $GAPLR$

$$\sum_{(i,j)\in A} y_{ij} \geq |A| \qquad (5.13).$$

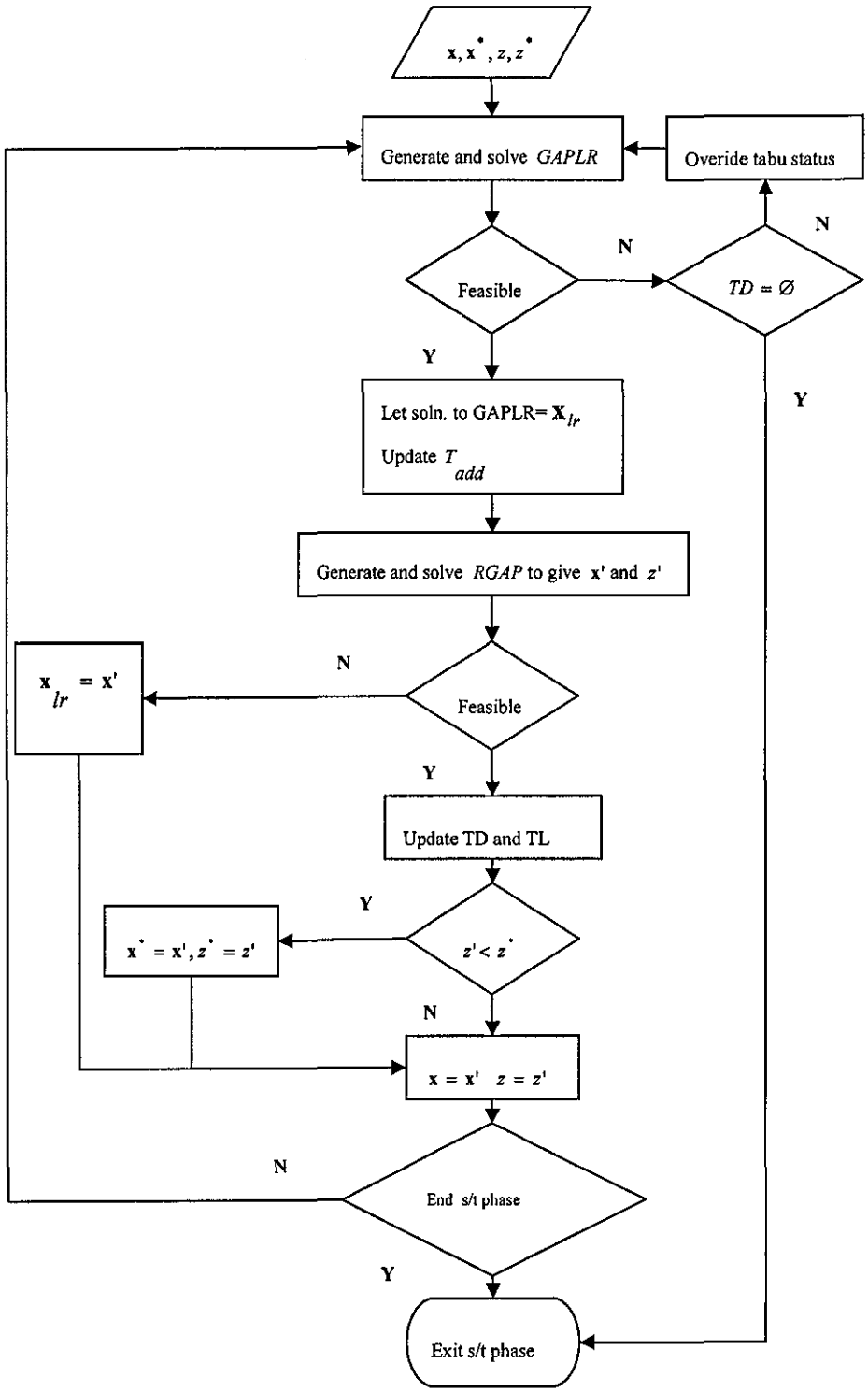
This tabu restriction is considerably stronger than the restriction preventing assignments from being added since there are many more possible assignments excluded from a solution compared with the number of assignments included in a solution and in fact the computational experimentation indicated that applying a single tabu constraint of type 5.13 at each iteration is sufficient.

The short-term phase begins from a solution $\mathbf{x}$, from which an instance of GAPLR can be formulated with the addition of constraints 5.7 and 5.13. In keeping with a conventional tabu search approach the short –term phase of the algorithm is allowed to search for non-improving solutions and as a result no cut-off value is specified for the search of the sub-region defined by the sub-problem at the current iteration. The objective function value of the solution to

the relaxed problem may be greater than the best integer solution found but, since the purpose of the current iteration is to find the best integer feasible solution within the sub-region in the time allowed, is still accepted as a partial move from $\mathbf{x}$ to $\mathbf{x}'$. If a solution to GAPLR is found then this is used to generate the next instance of RGAP and those assignments that have been dropped as a result of the solution to GAPLR are added to the tabu list T_{add}. A constraint of type 5.12 is then added to RGAP for each entry on the tabu list T_{add} prior to solving in an attempt to obtain the next solution $\mathbf{x}'$. If GAPLR is infeasible then it is necessary to override the tabu status of one or more assignments that are tabu from being dropped according to the aspiration criteria described in section 5.4.5 and update constraint 5.13 of GAPLR accordingly. If there are no assignments in $\mathbf{x}$ that are tabu from being dropped then the short term phase ceases at this point. Assuming that a solution $\mathbf{x}'$ has been found by solving RGAP then the tabu list T_{drop} is updated by first deleting those assignments (i, j) that are currently members of T_{drop} and then recording those assignments present in $S_{x'}$ but not in S_x, since these are the set of assignments that have been added during the current iteration and are therefore tabu from being dropped during the move from $\mathbf{x}'$ to the next solution $\mathbf{x}''$. In addition to updating T_{drop} the long term frequency memory T is also updated at this point. Having obtained a solution $\mathbf{x}'$ with corresponding objective function value z', a comparison is made with the objective function value of the best solution found during the search to ascertain whether $z' < z^*$, if so then $\mathbf{x}^*$ and z^* are updated by setting $\mathbf{x}^* = \mathbf{x}'$ and $z^* = z'$. The new solution is set to be the current solution by setting $\mathbf{x} = \mathbf{x}'$ and $z = z'$. The number of iterations performed during a short-term phase of the algorithm is set according to the ratio n/m. If RGAP is infeasible then clearly the add neighbourhood defined by the constraints 5.11 and 5.12 does not contain a feasible solution to GAP and so no *add* move is made in these circumstances, instead the solution to the relaxed problem GAPLR is adopted as the new solution $\mathbf{x}'$. A flowchart of the short-term

phase is given in figure 5.4.6.

Figure 5.4.6 Short term phase of TSBB.



5.4.3 Intensification

Longer term memory uses frequency information as a means for introducing *intensification* and *diversification* strategies. These frequency based memory structures are used to provide information that helps to identify promising regions of the solution space that have been previously visited but which may warrant a more thorough search and also to help identify changes in attributes that would help to drive the search into unexplored regions of the solution space. The longer term memory structure used by TSBB records the number of times that an assignment is contained in an integer solution. These frequencies are recorded in a matrix T of size $m \times n$ which is updated at each iteration by increasing the value t_{ij} by 1 if an assignment $(i, j) \in S_x$. This memory structure is then used as a means for implementing the *intensification* and *diversification* strategies as follows.

The purpose of the intensification phase is to search a region of the feasible solution space that includes solutions for GAP which contain assignments that could be considered attractive since they are

a)Included in the best solution found during the search up to this point and

b)Have been included in a large number of integer feasible solutions that have been visited during the search so far.

The best solution $\mathbf{x}^*$ encountered so far by the search is recovered and the assignments (i, j) contained in S_{x^*} are compared with the corresponding values in t_{ij} in order to construct the set $F = \{(i, j) : (i, j) \in S_{x^*} \text{ and } t_{ij} \geq \alpha k\}$. The parameter α is the threshold for the proportion of time t_{ij} / k, where k is the number of feasible integer solutions generated during the search, that an assignment (i, j) needs to have been included in solutions to date in order to be considered for inclusion in F. At each execution of the intensification phase the value of α is initially set to the value 1 and the set F is subsequently generated. If F is empty i.e. there are no assignments that have

been included in all feasible integer solutions generated thus far, then α is reduced at a rate of 0.1 at each attempt and another attempt to generate F is carried out. This process continues until F contains at least one assignment or until $\alpha = 0$. In practice the value of α did not reach the value 0 and the likelihood of such an event occurring is thought to be small, although in some circumstances it may be necessary to re-evaluate the reduction rate of α in order to avoid the occurrence of such an event. The set F can now be used to generate a restricted instance of GAP that has as its feasible region, only those solutions that contain all assignments in F by adding an intensifying constraint

$$\sum_{(i,j)\in F} x_{ij} \geq |F| \qquad (5.14)$$

to GAP. Constraint 5.14 is implemented within the algorithm by fixing the lower bound of those variables that represent each assignment contained in F to 1. Prior to solving the resulting restricted problem a cut-off value of z^* is given to the branch and bound solver which, in keeping with the branch and bound strategy, restricts the search still further. Clearly the size of F is determined by α and the size of the feasible region of the restricted problem is determined by the size of F and the quality of the current best solution. Any solution obtained as a result of solving the restricted problem must improve on the best solution found and therefore $\mathbf{x}^*, z^*$ and T must be updated accordingly.

5.4.4 Diversification

TSBB attempts to diversify the search by generating a solution that contains one or more assignments that have very low (ideally 0) residence frequency i.e. assignments that have rarely been included in integer feasible solutions found prior to this point in the search. Once such a solution has been generated this is then used to launch the next short term phase of the search process. This is achieved by using the longer term frequency based memory in order to generate a constraint that can be added to *GAP* that will force the

inclusion of low frequency assignments into the diversified solution. In order to achieve this TSBB first attempts to identify those assignments that have rarely been included in any previous solutions i.e. those assignments having the lowest (ideally 0) residence frequency values and constructs the set $H = \{(i,j) : t_{ij} \leq h\}$ where h is the threshold for the maximum proportion of time that an assignment (i,j) has been included in solutions to date in order to become a member of H, and thus become a candidate for inclusion in the diversified solution to be subsequently generated by the addition of the constraint

$$\sum_{(i,j)\in H} x_{ij} \geq d \qquad (5.15)$$

to GAP, where d is the level of required diversification.

If all elements of H have corresponding value $t_{ij} = 0$ then at least one assignment that has never been included in any of the solutions generated will be included in the new diversified solution and hence this new diversified solution will be different to any other solution previously encountered during the search. Clearly the size of H is required to be greater than or equal to d and in order to achieve the required size for H the threshold h is initially set to 0. If $|H| < d$ then h is increased by 0.1 and H is regenerated. This process continues until $|H| \geq d$ at which time the diversification constraint 5.15 can be added to GAP and the diversified solution generated in the same way as the initial solution described earlier and subsequently used to initiate the next short term phase of the search. As in the intensification phase the step size for the change in value of the threshold appeared to work quite well in practice although further consideration should be given to this aspect as necessary. If all elements of H have a value $t_{ij} \geq 1$ then there is no guarantee that the diversified solution will not have been visited previously during the search. Larger values of d will reduce the likelihood of a previously visited solution

be generated during the diversification phase since there should be less chance of a larger number of rarely seen assignments being included together in an integer feasible solution. Whilst the quality of the diversified solution is usually poor in comparison to the best known solution, starting the next short-term phase from a previously unseen solution can be advantageous in terms of providing access to high quality solutions that have previously been inaccessible.

5.4.5 Aspiration Criteria

Aspiration criteria are utilised in tabu search in order to determine when and how to override the tabu status of solution attributes. Implementation of suitable aspiration criteria can greatly affect the performance of the tabu search strategy and can be viewed as a complementary strategy to that of imposing tabu restrictions. There are a number of commonly used aspiration strategies some of which are detailed in (Glover and Laguna 1997).

The move from one solution to another within the search is dependent upon a feasible solution being found to a relaxed version of GAP, either a feasible solution to the LP relaxation GAPLR when dropping assignments, or the feasible integer solution to an instance of RGAP when attempting to add assignments. In the first of these two cases a tabu restriction is applied in order to force the relaxed solution to contain those assignments previously added to the current integer solution. In the situation when no feasible solution to GAPLR exists then clearly the tabu constraint is too restrictive and so must be relaxed in order to solve GAPLR. This is achieved by applying an aspiration criteria which allows the right hand side of the tabu constraint to be reduced by 1. This has the effect of constraining all but 1 of the assignments that are tabu from being dropped to be included in the relaxed solution. If the relaxation is still infeasible then the right hand side is repeatedly reduced until a feasible solution to the relaxation is obtained.

If the solution to an instance of *RGAP* is found to be infeasible i.e. no integer solution is generated then clearly constraint 5.11 is too restrictive in terms of forcing a collection of assignments to be included in an integer solution. Since no new integer solution has been generated the search must come to a halt. In order that the search can continue the restriction is indirectly overridden by adopting the solution to the relaxation as the current solution thus forcing one or more assignments in the relaxation to be dropped on the next iteration and releasing the restriction of constraint 5.11 from the previous iteration.

Example

Given the problem defined below which is taken from the Beasley OR-library

$m = 5$

$n = 15$

$$\mathbf{C} = \begin{pmatrix} 25 & 25 & 18 & 24 & 20 & 19 & 25 & 24 & 23 & 15 & 18 & 18 & 25 & 15 & 22 \\ 25 & 18 & 17 & 22 & 21 & 23 & 20 & 23 & 16 & 19 & 15 & 18 & 16 & 23 & 16 \\ 18 & 16 & 19 & 15 & 15 & 18 & 15 & 20 & 19 & 24 & 22 & 20 & 25 & 16 & 21 \\ 18 & 21 & 16 & 18 & 17 & 24 & 18 & 23 & 22 & 16 & 17 & 22 & 22 & 18 & 16 \\ 17 & 18 & 15 & 21 & 23 & 21 & 24 & 23 & 20 & 22 & 19 & 15 & 22 & 22 & 25 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 16 & 20 & 9 & 22 & 17 & 19 & 20 & 22 & 20 & 13 & 6 & 20 & 23 & 19 & 7 \\ 12 & 22 & 18 & 18 & 6 & 13 & 17 & 17 & 17 & 14 & 20 & 12 & 17 & 14 & 22 \\ 5 & 19 & 19 & 14 & 24 & 16 & 7 & 8 & 9 & 22 & 13 & 23 & 24 & 15 & 20 \\ 20 & 8 & 6 & 9 & 5 & 17 & 23 & 18 & 14 & 12 & 14 & 17 & 15 & 23 & 21 \\ 6 & 6 & 24 & 24 & 8 & 7 & 5 & 25 & 21 & 18 & 12 & 20 & 20 & 7 & 12 \end{pmatrix}$$

$$\mathbf{b} = (40 \quad 38 \quad 38 \quad 35 \quad 34)$$

Firstly generate an initial integer feasible solution

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	1	0	0	1	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	1	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0	0

with objective function value $z = 270$.

From this initial solution generate the set

$S_x = \{(5,1),(5,2),(4,3),(3,4),(5,5),(5,6),(3,7),(3,8),(3,9),(4,10),(2,11),(4,12),(2,13),(1,14),(1,15)\}$

The drop neighbourhood can now be defined by an instance of GAPLR according to the formulation given in 5.4.1.1 with $T_{drop} = \varnothing$ to give the relaxed solution

0	0	0	0	0	0.239555	0	0	0	1	0.574743	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0.2	0	1	0	0
0	0	0	1	0	0.5625	1	1	0	0	0	0	0	0	0
0	0	0.974401	0	1	0	0	0	0	0	0.225257	0	0	0	1
1	1	0.0255993	0	0	0.197945	0	0	0	0	0	1	0	0	0

$S_y = \{(5,1),(5,2),(3,4),(4,5),(3,7),(3,8),(2,9),(1,10),(5,12),(2,13),(1,14),(4,15)\}$

$D = \{(4,3),(5,5),(5,6),(3,9),(4,10),(2,11),(4,12),(1,15)\}$

$T_{add} = D$

An instance of RGAP can now be formulated as described in 5.4.1.1 to define the add neighbourhood. The solution to RGAP is infeasible in this instance and so no move to a solution in the add neighbourhood is made and the solution to GAPLR is defined to be the new solution $\mathbf{x}'$. We now set $\mathbf{x} = \mathbf{x}'$ and have $S_x = S_y$ and $T_{drop} = \varnothing$ since no new assignments have yet been added. The new S_x is now used to construct a new instance of GAPLR by adding the constraint $\sum_{(i,j)\in S_x} y_{ij} \leq |S_x| - 1$ to GAP which is then solved to give a relaxed solution

0	0	0	0	0	0.562276	0	0	0	0.332059	1	0	0	1	0
0	0	0	0	0	0	0	0	0.965231	0	0	0	1	0	0.208685
0	0.0886046	0	1	0	0.437724	1	1	0.034769	0	0	0	0	0	0
0	0	0.894516	0	1	0	0	0	0	0.667941	0	0	0	0	0.791315
1	0.911395	0.105484	0	0	0	0	0	0	0	0	1	0	0	0

Now

$S_y = \{(5,1),(3,4),(4,5),(3,7),(3,8),(1,11),(5,12),(2,13),(1,14)\}$

and

$D = \{(5,2),(2,9),(1,10),(4,15)\}$.

$$T_{add} = \begin{pmatrix} (4,3),(5,5),(5,6),(3,9),(4,10),(2,11),(4,12),(1,15) \\ (5,2),(2,9),(1,10),(4,15) \end{pmatrix}$$

Formulating and solving the new instance of RGAP with the addition of the 2 tabu constraints

$$x(4,3)+x(5,5)+x(5,6)+x(3,9)+x(4,10)+x(2,11)+x(4,12)+x(1,15) \le 7$$
$$x(5,2)+x(2,9)+x(1,10)+x(4,15) \le 3$$

as defined by T_{add} yields the solution

0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	1	1	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	1	0	0	0

with $z = 255$,

$S_x = \{(5,1),(5,2),(4,3),(3,4),(4,5),(2,6),(3,7),(3,8),(3,9),(1,10),(1,11),(5,12),(2,13),(1,14),(4,15)\}$

and

$T_{drop} = \{(5,2),(4,3),(2,6),(3,9),(1,10),(4,15)$.

The frequency based memory has also been updated to record the frequency with which assignments occur in the 2 integer solutions generated thus far to give

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

In order to perform an intensification phase the best solution found so far $\mathbf{x}^*$ is recovered with

$S_{x^*} = \{(5,1),(5,2),(4,3),(3,4),(4,5),(2,6),(3,7),(3,8),(3,9),(1,10),(1,11),(5,12),(2,13),(1,14),(4,15)\}$ and

$F = \{(5,1),(5,2),(4,3),(3,4),(3,7),(3,8),(3,9),(2,13),(1,14)\}$.

The intensification neighbourhood is now defined by adding the constraint

$x(5,1)+x(5,2)+x(4,3)+x(3,4)+x(3,7)+x(3,8)+x(3,9)+x(2,13)+x(1,14) \geq 9$

and an objective function cutoff value of

$z < 255$

to the original GAP problem. Solving the intensification problem informs us that no feasible solution exists in the intensification region and so a diversification is performed to begin the next short term phase. The diversified solution is obtained by formulating, and taking the first integer feasible solution of, GAP with the additional diversifying constraint

$\sum_{(i,j)\in H} x_{ij} \geq m$ where $H = \{(i,j) : t_{ij} = 0\}$.

5.5 Summary

This chapter has given a description of the hybrid algorithm TSBB. The approach of generating neighbourhoods by means of constraining the integer programming formulation of GAP has been adopted, motivated by the local branching and RINS methods for solving general mixed integer programming problems along with the concept of referent domain optimization. The neighbourhoods generated by this approach are dissimilar to the conventional neighbourhoods that are used in local search methods for solving GAP but do still take advantage of the structure of the problem, in contrast to RINS and Local Branching. A tabu search strategy is used to generate constraints that are added to and deleted from the linear programming relaxations and also the integer programming formulations that are generated at each iteration. In the following chapter the performance of TSBB is assessed by means of computational testing performed on a set of benchmark test problems and the results of this experimentation will be described in detail.

6 Computational Testing and Results

This chapter of the thesis considers the performance of the algorithm developed and described in the previous chapter. The algorithm was applied to two sets of benchmark test problems. The first set consists of problems of size $n \leq 200$ and the larger set of problems has $400 \leq n \leq 1600$. Section 6.1 describes the different types of problem in relation to how the instances have been generated, how they differ in terms of difficulty, where they can be found and why they can be considered to be a suitable set of problems in terms of testing the performance of the TSBB algorithm. Section 6.2 describes how the TSBB parameter settings were determined and section 6.3 describes the performance of the different phases of the algorithm. Section 6.4 compares the results obtained by TSBB with those obtained by Xpress-MP. Section 6.5 compares TSBB with the ejection chain Tabu Search approach developed by Yagiura et al (Yagiura, et al. 2004) which has been shown to outperform most other approaches in solving the set of benchmark test problems and so can be considered to provide a good comparison in terms of the quality of solutions generated by TSBB. Section 6.6 compares TSBB with a selection of alternative heuristic algorithms on the two sets of benchmark test problems and performs analysis of variance tests to assess any difference in performance between the various algorithms. This chapter then concludes with a summary of the reported results and performance of the TSBB algorithm.

6.1 Benchmark Test Problems

Benchmark instances for testing GAP have been categorised into five different types A, B, C, D and E and there are a number of instances available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/ and http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/gap/. Instances from the first four categories, with $m = 5,10,20$ and $n = 100,200$ were generated by Chu and Beasley (Beasley and P. C. Chu. 1997) and are derived as follows:

Type A: a_{ij} are random integers from the uniform interval [5,25], c_{ij} are uniform random integers from the interval [10,50] and $b_i = 9(n/m) + 0.4R$ where $R = \max_{i \in I} \sum_{j \in J, I_j = i} a_{ij}$ and $I_j = \min[i \mid c_{ij} \leq a_{kj}, \forall k \in I]$

Type B: a_{ij} and c_{ij} are the same as type A and b_i is 70% of the value in type A.

Type C: a_{ij} and c_{ij} are the same as in type A and $b_i = 0.8\sum_{j \in J} a_{ij} / m$.

Type D: a_{ij} are random integers from the interval [1,100], $c_{ij} = 111 - a_{ij} + e_1$ where e_1 are random integers from the interval [-10,10] and $b_i = 0.8\sum_{j \in J} a_{ij} / m$.

Type E instances were generated by Yagiura et al.(Yagiura, et al. 2004) as follows:

Type E: $a_{ij} = 1 - 10 \ln e_2$ where e_2 are random numbers from [0,1], $c_{ij} = 1000 / a_{ij} - 10e_3$ where e_3 are random numbers from[0,1] and $b_i = 0.8\sum_{j \in J} a_{ij} / m$. Larger instances of types C, D and E with $m \leq 80$ and $n \leq 1600$ were also generated .

In terms of difficulty, problems of type A are least difficult to solve and type B and C problems are harder to solve than the preceding category due to tightening of the capacity constraints. Type D and E problems are considerably more difficult due to the inverse relationship that occurs between the cost coefficients and the capacity coefficients. Problem instances of types A and B can quite easily be solved using the standard Xpress-MP solver and so do not provide a suitable level of difficulty with regard to testing TSBB. Testing of the TSBB algorithm therefore focuses on problem types C, D and E. The first problem set consists of six problems of each of these three types

as detailed in table 6.1.1 giving a total of eighteen problems in the first of the benchmark test sets whose problems are deemed in the literature to be of medium size.

Table 6.1.1 Problem dimensions for medium size test instances

m	n
5	100
10	100
20	100
5	200
10	200
20	200

The second set of test problems are deemed to be large problems whose sizes are detailed in table 6.1.2

Table 6.1.2 Problem dimensions for large size test problems

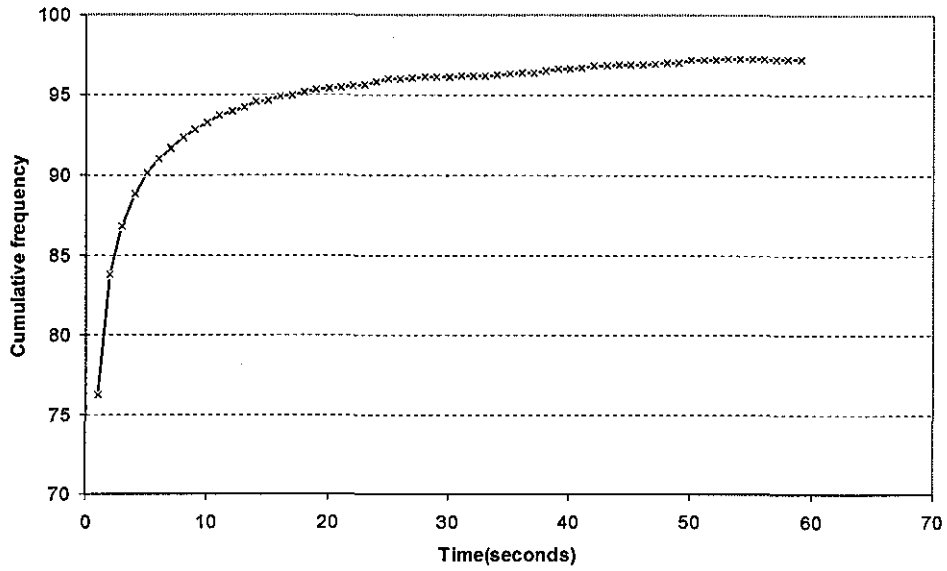
m	n
10	400
15	900
20	400
20	1600
30	900
40	400
40	1600
60	900
80	1600

6.2 Parameter Settings

The branch and bound results obtained from Xpress-MP subsequently presented in 6.3 were all obtained using the default settings of the software which are available from the user manual. These default settings were also used by all calls to the Xpress-MP branch and bound solver from within the TSBB algorithm. For each problem instance TSBB takes the two parameters

t_a and t_d as inputs and all other parameters are set according to problem size where problem size is defined to be mn. Two of the parameters set according to problem size are the maximum time allowed for the solution to a restricted sub-problem in the short-term phase and the time allowed to solve a sub-problem in the intensification phase. Due to the size and complexity of many of the benchmark test problems even highly restricted sub-problems can be difficult and time consuming to solve to optimality. It is therefore necessary to restrict the time allowed for the solution of each sub-problem in order to give the algorithm sufficient opportunity to generate the integer feasible solutions in the short term phase, which subsequently provide information to generate the neighbourhood for the intensification and diversification sub-problems. In the diversification phase the Xpress solver is applied to the diversification sub-problem and ceases on finding the first integer feasible solution and so no time limit is necessary. In practice the Xpress solver typically tends to find integer feasible solutions early in the branch and bound process and quickly reduces the gap between the best known solution and the best bound. After this initial progress it is then common for the branch and bound process to spend large amounts of time and effort attempting to reduce the gap by relatively small amounts or even purely to confirm optimality. This initial period increases with the size and difficulty of the problem instance and so in determining the time restrictions to apply, the short term phase of TSBB was run for 100 iterations for each of the problems in the medium size test set with a time limit of 60 seconds for each sub-problem in the short term phase. Figure 6.2.1 shows the cumulative frequency of sub-problems that were solved to optimality and obtained an integer feasible solution within a 60 second time period, indicating that over 97% of such sub-problems of GAP instances with up to 4000 variables were manageable within this time period. The time limit for each sub-problem in the short-term phase of the algorithm for the medium size problem set was subsequently set to 60 seconds and the time limit for the intensification phase was set to be twice that of the short-term phase time limit.

Figure 6.2.1 Cumulative frequency of solution times for short-term phase



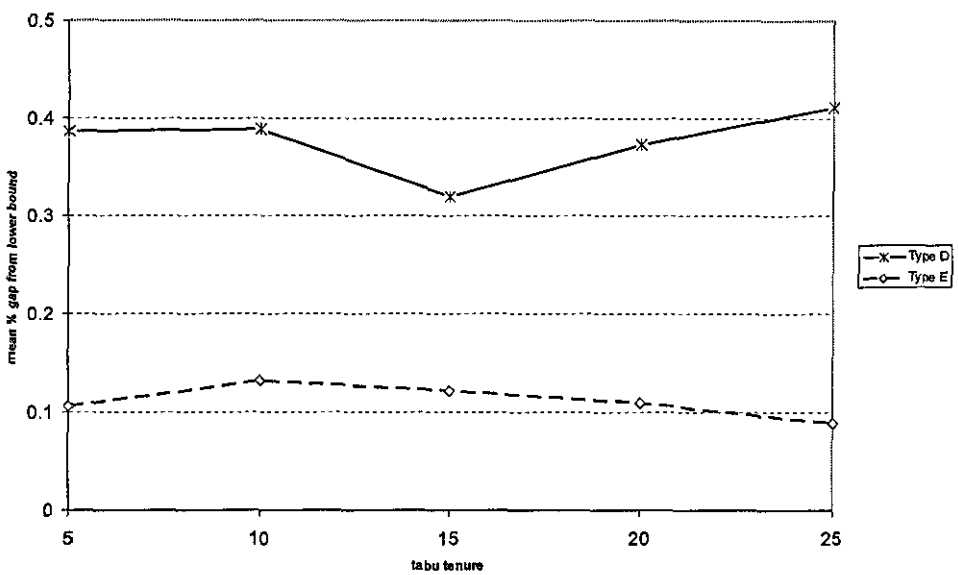
The intensification sub-problems are typically larger than those of the short-term phase and so preliminary experimentation indicated that it was necessary to give the solution attempt more time for these sub-problems. It should be noted however that the objective function value of the best solution found during the search was used as a bound on the intensification sub-problem in an attempt to reduce the size of the branch and bound tree in these situations.

6.2.1 Tabu Tenure

There are two short term tabu memory structures that are used to guide the short term phase of the algorithm. The first of these prevents those assignments that have been brought into the current solution as a result of the add phase from being dropped during the next drop phase. Preliminary experimentation indicated that these assignments tend to be highly restrictive and whilst such a tabu restriction is necessary to avoid cycling there seemed to be no advantage to be gained by using values greater than 1 for this tabu tenure. As a result the computational results detailed in the remainder of this chapter have all been obtained by fixing the value of t_d to 1. The second

short term memory structure is that which prevents those assignments that are excluded as a result of the previous drop phase from being included during the next t_a add phases. The preliminary experimentation also indicated that whilst the value of t_a should not be too small in order to avoid cycling, large values tend to be counter productive in terms of slowing down the algorithm. This seems fairly intuitive when considering that each of the t_a tabu restrictions takes the form of an additional constraint and as such tends to slow down the solution of each of the sub-problems in the short term phase of the algorithm. Testing of the TSBB algorithm therefore considers values of t_a in the range $5 \leq t_a \leq 25$. Each problem in the medium size set of test problems has been solved five times with the value of t_a being set to 5, 10, 15 20 and 25. Figure 6.2.2 shows the mean percentage gap from the best bound for problem types D and E for each value of t_a. The best bound is obtained from the Xpress-MP solution run for each problem. The results for problem type C have been omitted since the percentage gap is zero for all values of t_a for this problem set.

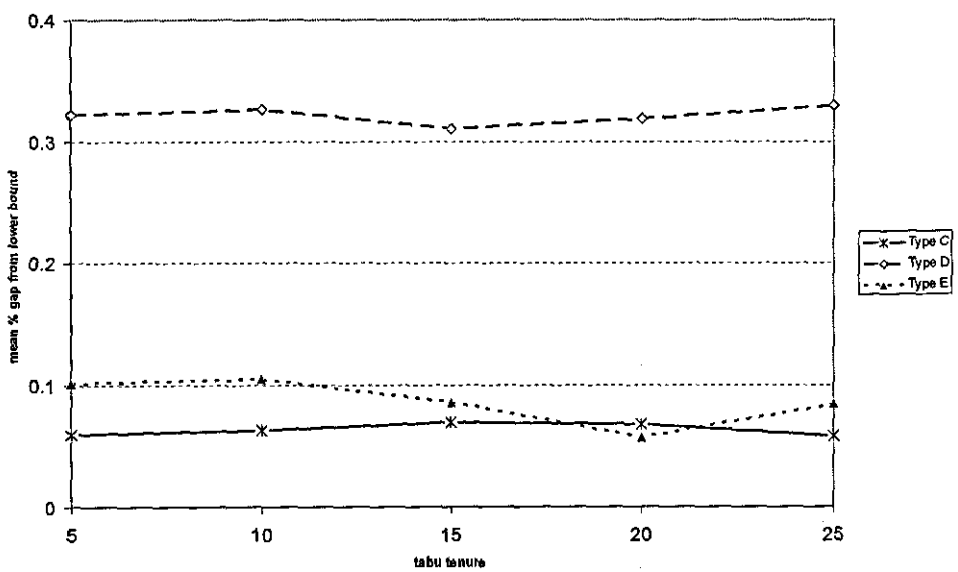
Figure 6.2.2 Mean % gap from best bound for medium size test problems types D and E



For the type D problems the lowest mean % gap was obtained with a value

of $t_a = 15$ and the highest mean % gap was obtained at $t_a = 25$ with the difference between the two being 0.091% (3 d.p). For the type E problems the lowest value was achieved by setting $t_a = 25$ and the highest value was obtained from using $t_a = 10$ with a difference between the two of 0.044% (3 d.p). These results suggest that the algorithm is fairly robust in terms of solution quality for values of t_a in this range. Figure 6.2.3 shows the corresponding results for the large size test problems. For the type C problems the lowest mean % gap is obtained with a value of $t_a = 25$ and the highest gap is achieved with $t_a = 15$ with the difference between these two of 0.012% (3 d.p), for the type D problems we have corresponding values of $t_a = 15$ and $t_a = 25$ with a difference of 0.019% (3 d.p) and the results for the type E problems give $t_a = 20$, $t_a = 10$ with the difference being 0.047% (3 d.p).

Figure 6.2.3 Mean % gap from best bound for large size test problems.



The first point to note from these results is that the quality of solution for the type C problems, in both sets of test instances, does not appear to be sensitive to the value of t_a as is evident from the small difference between the highest and lowest mean %gap over this range of values for t_a, and the mean

% gap for $t_a = 5$ is very close to that of $t_a = 25$, the difference being 0.000001. The results for the type D and E problems also show little variation in terms of the quality of solution obtainable within the range of values for t_a. A second observation to make from these results would be that for the type D problems the quality of solution tends to improve with values of t_a towards the middle of the range with little observable difference between the upper and lower end whilst for the type E problems the better quality solutions tend to be obtained with values of t_a in the upper half of the range, irrespective of problem size. The quality of solution for type C problems however, seems to be consistent across the whole of this range. The two plots also give an indication as to the overall level of difficulty for TSBB of the three different problem types. Type D problems seem to provide most difficulty with regard to the quality of solution obtainable for both medium and large problems.

6.2.2 Intensification and Diversification parameters

The diversification phase of the algorithm uses frequency based memory in order to obtain a solution that is a distance of at least m from any previous solution encountered in the search. The value m was chosen based on preliminary experimentation which indicated that as the size of problem grows due to the number of agents which are able to perform the n jobs then a greater level of diversification is required if the algorithm is to find high quality solutions. This was most evident with regard to harder type D and E problems.

6.3 Performance of short-term and intensification phases

The purpose of the diversification phase in the TSBB algorithm is purely to find an integer feasible solution that is a distance of at least m from any solution visited by the search. The solution found then begins the next short

term phase. The objective of both the short-term phase and the intensification phase are to search the solution space for new improving solutions and so the performance and effectiveness of these two aspects of the method are reported in this section.

Figure 6.3.1 shows the number of new best solutions found during each of the two phases of the search for each of the problems where the number of jobs is 100. This information was taken from the solution run where the best objective function value of the five runs was found. For the type C problems it is noticeable that all of the new best solutions are found during the short-term phase and for the type D and E problems the majority of new best solutions were also found during the short-term phase. Out of all the test problems solved these were the least difficult problems with regard to size and so the short term search phase was able to improve the quality of the best solution quite substantially. For the type D and E problems the short term phase alone was not sufficient and the intensification phase played a more significant role in these situations.

As the size of the problems increases the role of the intensification phase seems to become more significant. Figure 6.3.2 gives the split for each problem where the number of jobs is 200. Generally there are more solutions found during the intensification phase for this subset of problems than for those with 100 jobs although both phases of the algorithm appear to still be contributing towards finding new improved solutions.

Figure 6.3.1 Number of new best solutions found during the search for each of the two phases for problems with 100 jobs.

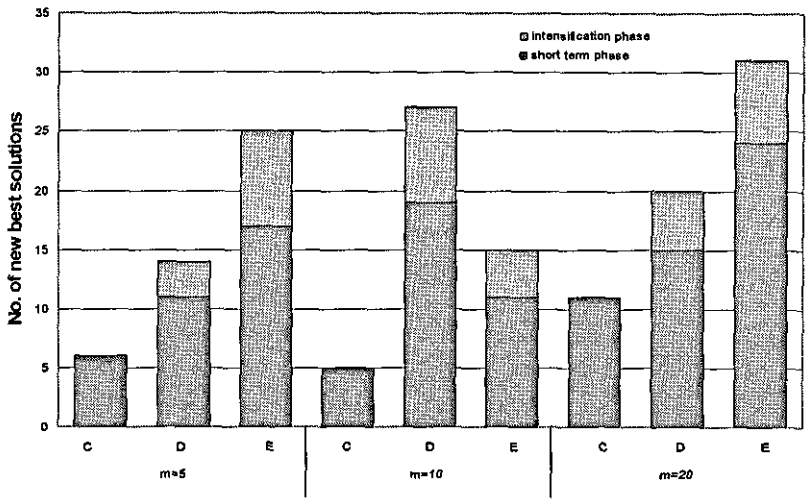
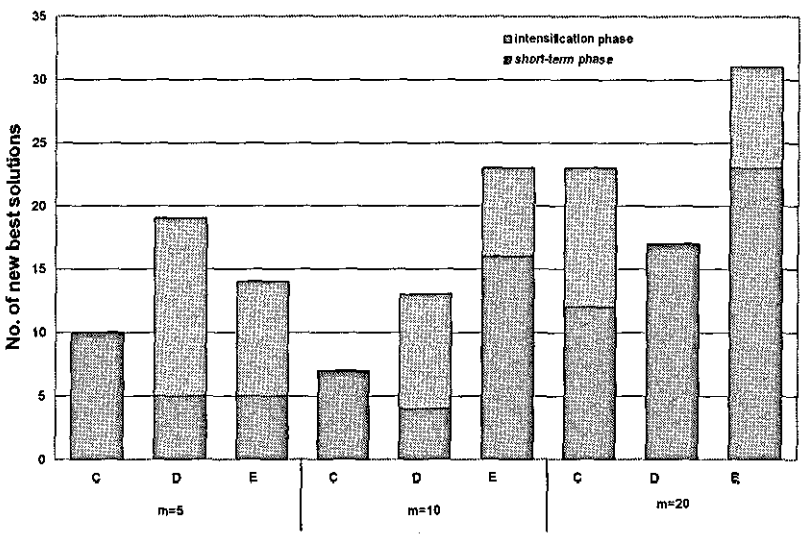


Figure 6.3.2 Number of new best solutions found during the search for each of the two phases for problems with 200 jobs.

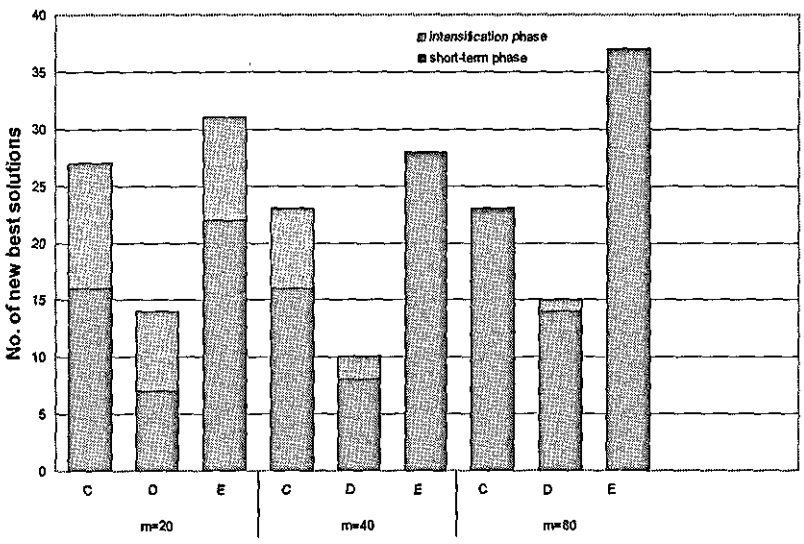


The success of the intensification phase is dependent to some degree upon how well the short-term phase operates. One function of this short-term phase is to find good integer feasible solutions which then provide information to

both define the region to be searched by the intensification phase and restrict the search of this region by providing a good upper bound on the solution value. The information summarized in figures 6.3.1 and 6.3.2 give a reasonably good indication that for these sizes of problems both phases are working effectively in seeking out good solutions in the short-term phase and that the frequency based memory is being constructed and used to good effect in defining good regions of the solution space to explore during the intensification phase.

Figure 6.3.3 gives a summary of new best solutions found during the two search phases for the largest of all the problems tested where the number of jobs is 1600. For problems with this number of jobs it is noticeable that as the number of agents grows up to 80 the contribution made by the intensification phase of the search diminishes.

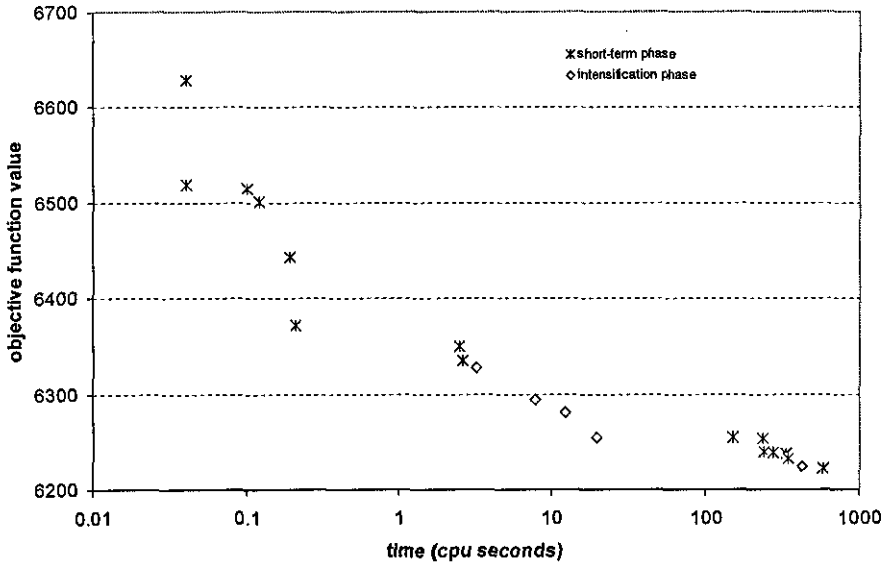
Figure 6.3.3 Number of new best solutions found during the search for each of the two phases for problems with 1600 jobs.



As the solution space grows with problem size it clearly becomes more difficult to identify a region for the intensification phase to find a new improving solution. One reason for this could be that the bound placed on

such a region by the short term phase is strong enough to rule out an improving solution. If this is the case then clearly the short term phase of the algorithm is working effectively. An alternative reason could be that the size of the region to be searched during intensification is just too large and an improving solution cannot be found in the time limit given to this phase of the search. Whatever the reason for this these circumstances require the diversification phase to be working effectively to guide the short term phase towards previously unexplored regions. If this is indeed the case then it will be noticeable that the short term phase continues to find new best solutions deep into the search.

Figure 6.3.4 New best solutions found by search phase for problem type D, m=20, n=100.

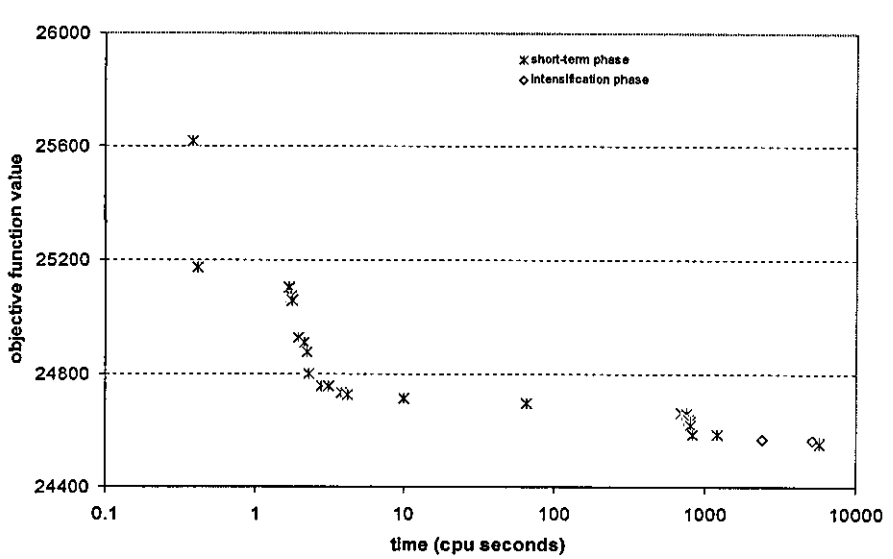


In figure 6.3.4 the progress of new best solutions found during the search is plotted for problem type D with $m = 20$, $n = 100$. The initial improvement in objective function is provided by the short term phase of the search. The intensification phase of the search then improves on the previous best solution 4 times. The fourth improvement yields a solution which launches a subsequent short-term search where clearly there is no better solution found in the region of this new best solution and there follows a period of time without

improvement before the short-term phase once again finds new improving solutions. Prior to the discovery of these new improvements the search diversifies to a new unseen solution. In doing so the diversification has launched the next short-term search in a new and fruitful region. The discovery of the new improving solution by the short-term phase provides new frequency information which contributes to the definition of a region of intensification where the best solution is improved still further. The best solution is finally improved once more by the short-term phase. This is a good example of how the three phases of the algorithm work in conjunction with each other in order to guide the search quite successfully towards new improving solutions.

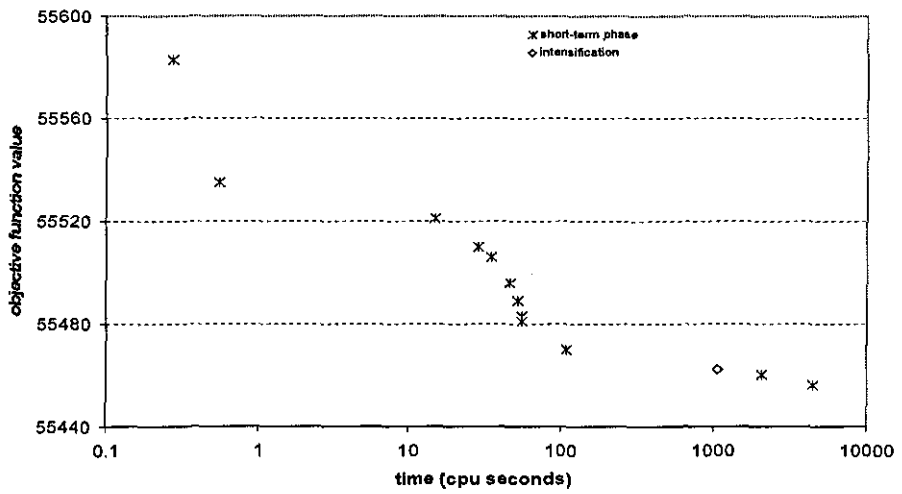
For the type D problem where $m = 40$ and $n = 400$ all but 2 of the improving best solutions are discovered by the short-term phase. The profile in figure 6.3.5 shows that the intensification phase does not find any improving best solutions until much further into the search. The diversification and short-term phases however contribute to the frequency memory that defines the regions within which the improving best solutions are found during the latter stages of the search.

Figure 6.3.5 New best solutions found by search phase for problem type D, m=40, n=400.



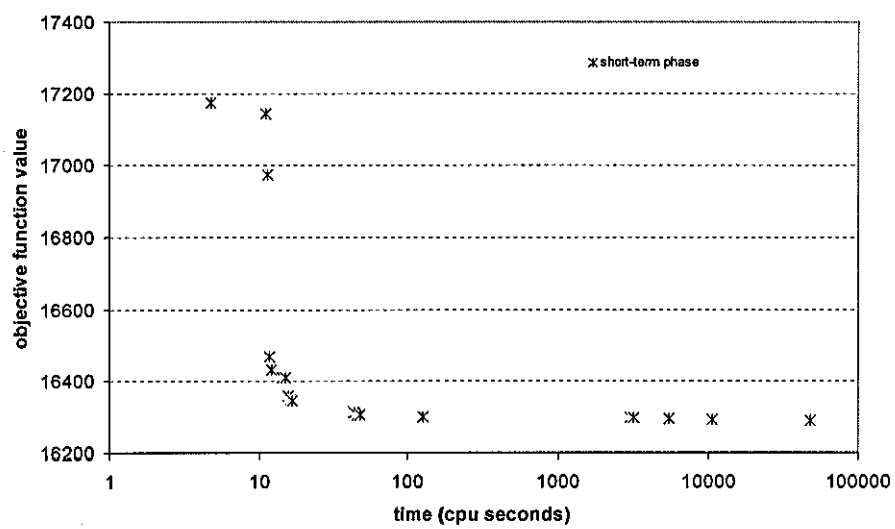
This result gives a good indication that the memory structures and the three different phases of the search are working together to exploit the feasible solution space throughout the whole of the search. This is an important aspect to highlight since it is in contrast to the effectiveness of the standard branch and bound approach which tends to improve the solution early during the search but often struggles to discover regions of the branch and bound tree containing these new improved solutions as the search progresses. In this respect the tabu search aspect of TSBB helps to overcome this situation and to explore areas of the tree that branch and bound would not otherwise reach. The best solution found during the search for this problem is discovered by the short-term phase following a new best solution found during a previous intensification phase. The best solution found during intensification has been used to launch a subsequent short-term phase which has found the best solution in the neighbourhood of the intensification solution. This is encouraging and a further indication of how each phase of the algorithm works in conjunction with each of the others. This behaviour can also be seen with the larger problems also as depicted in Figure 6.3.6 which indicates how the three separate search phases have combined in a similar way when solving a type D problem with 900 jobs.

Figure 6.3.6 New best solutions found by search phase for problem type D, m=15, n=900



For the problem instance of type C where $m = 80$ and $n = 1600$ there are no new improved best solutions found during the search by the intensification phase. The regions defined for intensification during the search for a problem of this size can be large and the branch and bound solver encounters the same problems as it would when operating by itself on a large problem. TSBB can still be very effective under these conditions since it becomes more important for the diversification and short-term phase to work in conjunction with each other. Figure 6.3.7 indicates that after a significant improvement in the best solution during the early part of the search there is a period during the middle part of the search where no improvement to the best solution is made. The move to unexplored regions and the short-term search phase are still able to combine and find improvements to the best solution deep into the latter stage of the search indicating that the long-term memory structures work well in providing information about where to find these more promising regions.

Figure 6.3.7 New best solutions found by search phase for problem type C, m=80, n=1600.



The results presented in this section have provided insight into the effectiveness of TSBB to be able to effectively solve both the medium and large size test problems. Examination of the results for these problems gives a

good indication as to how the components of the method work together and complement each other in the overall aim of finding high quality solutions. The results show how, with the use of the short and long term memory structures, the solution space can be intelligently searched with the aim of discovering these high quality solutions. The remaining sections of this chapter compare the TSBB algorithm with the standard branch and bound approach using the Xpress-MP solver, the ejection chain tabu search approach of (Yagiura, et al. 2004) and other algorithms from the literature. A comprehensive comparison and analysis of the performance of TSBB with these methods is presented.

6.4 Comparison with Xpress-MP

The results presented in this section compare the TSBB algorithm with the Xpress-MP branch and bound solver. The results obtained by the Xpress-MP solver were all achieved using the default settings of the software with regard to the branch and bound strategy for solving each problem. The same default settings were also used within TSBB whenever the Xpress solver is called to solve the generated sub-problems. Firstly, following solution of the LP relaxation the Xpress solver attempts to strengthen the lower bound on the problem by generating a series of *cuts.* There are varying degrees of aggressiveness that can be used during this cut generation phase although it is important to strike a balance between a very aggressive strategy which increases computational time and effort, due to the increased number of cuts generated, and a less aggressive strategy which generates fewer cuts but takes less time and effort but will normally result in a weaker initial bound and increased time and effort during the tree search. The default strategy allows the Xpress solver to select the most appropriate strategy. Having attempted to increase the lower bound on the problem Xpress then attempts to set an initial cut-off value for the tree search by generating an integer feasible solution heuristically, typically using a rounding approach. Once again this is left to the default heuristic strategy of the software. The solver then progresses into

the branch and bound tree search where the following strategic aspects need to be set in order to minimize the time and effort of the solution process.

Node selection: The strategy of selecting outstanding nodes for processing is based on an assessment of the characteristics of the problem matrix. The set of nodes from which one will be chosen for processing is identified by this strategy.

Backtrack: Having identified the set of nodes from which to choose, the backtrack approach specifies which node should be processed next. The default here is to select the node which provides the best bound on the solution.

Variable selection: The variable chosen for branching at the selected node is determined by means of calculating up and down pseudo-costs for each variable. The default approach is to select the variable with the smallest up and down pseudo costs.

6.4.1 Medium size problem set

The results given in table 6.4.1 are for the medium size problem set where the maximum solution time allowed for each problem with both methods was set at 3000 and 6000 cpu seconds for $n = 100$ and $n = 200$ respectively. These time restrictions were set according to the time allowed for solutions to these problems as defined in (Yagiura, et al. 2002) and (Yagiura, et al. 2004). All experimental testing was carried out on a Linux machine with an Intel P4 Xeon 3.0 GHZ processor and 1.0 GB of RAM.

Columns 1, 2 and 3 define the problem type and size, column 4 gives the best bound for each problem that was found during the Xpress-MP solution run, columns 5 and 8 give the best objective function values obtained during the specified time limits from each method whilst columns 6 and 9 give the time in cpu seconds that each method took to find its best objective function value. Column 11 indicates whether or not the solution found by Xpress-MP was proved to be optimal during the solution run. From table 6.4.1 it can be

seen that Xpress was able to solve all of the type C problems and 4 out of the 6 type E problems to proven optimality. The type D problems appear to pose a much stiffer test where only one out of the 6 problems of this type was solved to optimality by Xpress-MP and this was the smallest of the 6. It can be seen from table 6.4.1 that optimal solutions found by Xpress-MP were also found by TSBB. For the problems that were not solved to optimality by Xpress-MP the solution values obtained by TSBB were better than those obtained by Xpress-MP in all instances.

Table 6.4.1 Comparison of the best solutions obtained by TSBB and Xpress-MP for medium size problems

				TSBB			Xpress-MP			
Type	m	n	Best Bound	Best solution value	Time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	5	100	1931.00	1931	0.79	0.00	1931	0.00	0.00	y
c	10	100	1402.00	1402	4.03	0.00	1402	6.00	0.00	y
c	20	100	1243.00	1243	0.69	0.00	1243	1.00	0.00	y
c	5	200	3456.00	3456	7.85	0.00	3456	4.00	0.00	y
c	10	200	2806.00	2806	23.96	0.00	2806	233.00	0.00	y
c	20	200	2391.00	2391	137.92	0.00	2391	871.00	0.00	y
d	5	100	6351.94	6353	702.54	0.02	6353	1571.00	0.02	y
d	10	100	6335.03	6350	2581.01	0.24	6359	2453.00	0.38	n
d	20	100	6160.76	6222	574.74	0.99	6277	1385.00	1.89	n
d	5	200	12739.87	12743	5114.75	0.02	12746	4775.00	0.05	n
d	10	200	12422.19	12440	399.45	0.14	12460	122.00	0.30	n
d	20	200	12224.29	12280	3400.98	0.46	12318	5907.00	0.77	n
e	5	100	12681.00	12681	18.5	0.00	12681	84.00	0.00	y
e	10	100	11577.00	11577	106.1	0.00	11577	327.00	0.00	y
e	20	100	8423.77	8467	1942.65	0.51	8597	3751.00	2.06	n
e	5	200	24930.00	24930	30.81	0.00	24930	20.00	0.00	y
e	10	200	23307.00	23307	110.83	0.00	23307	1156.00	0.00	y
e	20	200	22376.28	22380	202.25	0.02	22658	75.00	1.26	n
					Mean	0.13			0.37	

A scan of table 6.4.1 also reveals that the TSBB algorithm tends to find solutions of equally high quality or better and for most instances in substantially shorter times, suggesting that TSBB is able to outperform the Xpress-MP branch and bound solver for problems of this size and level of difficulty. In order to substantiate this claim a comparison between the two methods can be made by calculating the percentage gap from the best bound as provided by Xpress during its solution attempts as detailed in columns 7 and 10 of the results table.

Across all 18 problems in this test set the mean % gap from the best bound for TSBB is 0.13% compared to a mean % gap of 0.37% achieved by Xpress-MP, representing a difference between the two means of 0.24%. This suggests that TSBB performs better than Xpress-MP across this problem set. In order to ascertain whether TSBB is significantly better than Xpress-MP statistical significance tests have been conducted which consider each of the problem types in isolation and also the problem set as a whole. Since the number of data values available for these tests is small there seemed to be little justification to make the assumption of normality with regard to the distribution of the difference in the % gaps of TSBB and Xpress-MP for each problem type. It was therefore necessary to consider a non-parametric test, in this case the Wilcoxon signed rank test to test the hypothesis

$$H_0 : M = 0$$

where M is the median difference in % gap from the best bound of each of the two methods. Table 6.4.2 gives the results of these significance tests for each problem type in addition to the overall set of medium size problems. The table gives a test statistic in column 2 calculated using the statistical software package SPSS and column 3 gives a probability value indicating the two-tail significance of the test statistic.

Table 6.4.2 Significance tests for the difference in % gap between TSBB and Xpress-MP for best solutions from the medium size problem set.

Type	Test statistic	significance
c	n/a	n/a
d	-2.02	0.043
e	-1.34	0.18
all	-2.366	0.018

There is no difference between the two methods for the type C problems from this problem set since both are able to find the optimal solutions for all six problems of this type. For the type D problems from this set the results of the hypothesis test shows that at the 95 % confidence level TSBB is able to obtain a significantly better quality of solution compared to Xpress-MP where quality of solution is measured by the % gap from the best bound. When considering the type E solutions in isolation there is no significant evidence to suggest that the hypothesis H_0 can be rejected at the 90 % confidence level although when we consider the problem set as a whole we see that there is sufficient evidence to suggest that TSBB is significantly better at finding high quality solutions at the 95 % confidence level.

Section 6.2.1 described how each problem was solved by TSBB with 5 different tabu tenures and although there was some variance in the quality of solution obtained with the different values the overall quality of solution seemed to be reasonably good across the whole range. Table 6.4.3 compares the average solution value for each problem across the five different values of t_a with the best solution found by Xpress. Column 7 in table 6.4.3 now gives the mean percentage gap from the best bound for the average of the five runs of TSBB. The results show that for the solutions solved to proven optimality by Xpress, TSBB was able to obtain the same optimal solution on all of the five runs for each of these problems. For the remaining problems in this set

the percentage gap from the best bound for the average solution value obtained by TSBB was smaller than the percentage gap from the best bound of the best solution obtained by Xpress in all cases. The overall mean percentage gap for TSBB is therefore smaller than that of Xpress. If we repeat the hypothesis tests using these new values we obtain the results in table 6.4.4

Table 6.4.3 Comparison of average solution value for TSBB over 5 runs with best solution obtained by Xpress-MP for medium size problems.

			TSBB				Xpress-MP			
Type	m	n	Best Bound	Average solution value	Average time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	5	100	1931.00	1931.00	0.80	0.00	1931.00	0.00	0.00	y
c	10	100	1402.00	1402.00	6.99	0.00	1402.00	6.00	0.00	y
c	20	100	1243.00	1243.00	0.70	0.00	1243.00	1.00	0.00	y
c	5	200	3456.00	3456.00	8.63	0.00	3456.00	4.00	0.00	y
c	10	200	2806.00	2806.00	91.81	0.00	2806.00	233.00	0.00	y
c	20	200	2391.00	2391.00	275.79	0.00	2391.00	871.00	0.00	y
d	5	100	6351.94	6353.00	1441.69	0.02	6353.00	1571.00	0.02	y
d	10	100	6335.03	6356.80	940.98	0.34	6359.00	2453.00	0.38	n
d	20	100	6160.76	6230.00	924.50	1.12	6277.00	1385.00	1.89	n
d	5	200	12739.87	12744.00	2499.42	0.03	12746.00	4775.00	0.05	n
d	10	200	12422.19	12445.20	268.49	0.19	12460.00	122.00	0.30	n
d	20	200	12224.29	12292.00	2680.01	0.55	12318.00	5907.00	0.77	n
e	5	100	12681.00	12681.00	175.74	0.00	12681.00	84.00	0.00	y
e	10	100	11577.00	11577.00	351.66	0.00	11577.00	327.00	0.00	y
e	20	100	8423.77	8475.60	689.33	0.62	8597.00	3751.00	2.06	n
e	5	200	24930.00	24930.00	39.88	0.00	24930.00	20.00	0.00	y
e	10	200	23307.00	23307.00	265.80	0.00	23307.00	1156.00	0.00	y
e	20	200	22376.28	22388.00	1584.69	0.05	22658.00	75.00	1.26	n
					Mean	0.16			0.37	

Clearly there is no significant difference in average solution quality for the type C problems. For the type D and type E problems the results are identical

to those presented in table 6.4.2. indicating that there is evidence to suggest the performance of TSBB compared with Xpress-MP for this problem set across the range of tabu tenures tested is significantly better.

Table 6.4.4 Significance tests for the difference in % gap for the average of 5 TSBB runs and Xpress-MP for medium size problems

Type	Test statistic	significance
c	n/a	n/a
d	-2.023	0.043
e	-1.342	0.18
all	-2.366	0.018

Considering the 5 TSBB runs, a measure of solution quality is given by taking the average of the 5 solution values. If the average time taken to find the 5 solution values is considered to be a measure of solution time, i.e. the time taken to find the average solution value, then it seems fair to compare the average solution time for TSBB with the solution time of the Xpress run. Such a comparison shows that TSBB perform well particularly on the harder type D and E problems, where in many cases the solution times are a fraction of those for Xpress. Section 6.4.2 now considers the larger problem set.

6.4.2 Large size problem set

For the large size problems the time limits for each solution run were set at 10000 cpu seconds for problems where $n = 400$ and $n = 900$ and 50000cpu seconds for $n = 1600$. Table 6.4.5 compares the best solutions obtained by TSBB and Xpress-MP for the large size problems. The Xpress branch and bound solver was only able to solve 2 out of the 27 problems in this set to proven optimality and both of these optimal solutions were also found by TSBB. In both of these cases TSBB was able to find the optimal solution in a much shorter time than the Xpress solver. Of the remaining 25 problems in this set Xpress was able to find a better objective function value for two of these instances, both of which were type D problems. In all of the remaining 23 instances for this problem set TSBB was able to find objective function

values better than Xpress.

It is noticeable from the results presented in table 6.4.5 that in some instances the time taken by Xpress to obtain its best solution value is considerably shorter than the time taken for TSBB to find its best solution. In every one of these cases the objective function value of the best solution found by TSBB is smaller than that found by Xpress. If the 12 instances where this occurs are isolated, as presented in table 6.4.6, it can be seen that in 10 out of the 12 instances TSBB is able to find a solution at least as good in shorter times. This behaviour in the Xpress branch and bound approach is typical of how the branch and bound approach can struggle to find improving solutions particularly for large and difficult problems. In large trees consisting of many thousands of nodes the tree search process can spend large amounts of time searching areas of the tree that subsequently turn out to be unproductive. One of the objectives of the TSBB algorithm is to attempt to overcome this issue by using tabu search memory structures and strategies to guide the search towards areas of the solution space that the normal branch and bound process may never reach. The results presented in this section certainly seem to suggest that TSBB achieves reasonable success in achieving this aim since it is able to find new improving solutions in stages of the search where the branch and bound search cannot.

The mean percentage gap from the best bound for TSBB for the whole of the large problem set is 0.14%, as detailed in table 6.4.5, and the corresponding value for the solutions obtained by Xpress is 0.64%. As for the medium size problem set the distribution of the difference between the percentage gaps obtained by each method is used to provide a test of whether there can be considered to be any significant difference between the two approaches for solving problems of this size and type.

Table 6.4.5 Comparison of the best solutions obtained for TSBB and Xpress-MP for large size test problems

				TSBB			Xpress-MP			
Type	m	n	Best Bound	Best solution value	Time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	10	400	5597.00	5597	58.62	0.00	5597	255.00	0.00	y
c	20	400	4779.33	4782	536.74	0.06	4791	8819.00	0.24	n
c	40	400	4243.04	4245	156.17	0.05	4254	6054.00	0.26	n
c	15	900	11338.38	11341	866.89	0.02	11346	4869.00	0.07	n
c	30	900	9979.12	9989	3485.02	0.10	9998	3975.00	0.19	n
c	60	900	9320.43	9331	1231.45	0.11	9459	28.00	1.49	n
c	20	1600	18800.30	18805	4179.67	0.02	18811	13.00	0.06	n
c	40	1600	17141.35	17150	3509.96	0.05	17178	29.00	0.21	n
c	80	1600	16283.00	16289	48178.8	0.04	16729	47804.00	2.74	n
d	10	400	24957.38	24976	656.17	0.07	24983	12076.00	0.10	n
d	20	400	24556.18	24620	4901.229	0.26	24668	5607.00	0.46	n
d	40	400	24348.00	24552	5650.05	0.84	24574	4158.00	0.93	n
d	15	900	55401.16	55456	4443.55	0.10	55461	2973.00	0.11	n
d	30	900	54830.23	55012	666.76	0.33	54979	8436.00	0.27	n
d	60	900	54551.00	54785	3008.51	0.43	54963	41.00	0.76	n
d	20	1600	97821.97	97906	6322.73	0.09	97908	6476.00	0.09	n
d	40	1600	97105.00	97328	31823.87	0.23	97311	42646.00	0.21	n
d	80	1600	97034.00	97449	33495.4	0.43	97459	24017.00	0.44	n
e	10	400	45746.00	45746	240.27	0.00	45746	8293.00	0.00	y
e	20	400	44873.07	44877	2175.536	0.01	45484	3951.00	1.36	n
e	40	400	44548.93	44609	3433.93	0.13	45484	37.00	2.10	n
e	15	900	102419.27	102421	1560.29	0.00	102686	163.00	0.26	n
e	30	900	100423.18	100430	8890.69	0.01	101312	18.00	0.89	n
e	60	900	100119.91	100363	4962	0.24	101954	6492.00	1.83	n
e	20	1600	180642.64	180645	7697	0.00	181060	30095.00	0.23	n
e	40	1600	178286.69	178391	18579.62	0.06	180071	17982.00	1.00	n
e	80	1600	176792.84	177035	34624	0.14	178727	840.00	1.09	n
					Mean	0.14			0.64	

The results for these tests are presented in table 6.4.7 and suggest that for

the large type C problems there is a significant difference between the % gap achievable by the two approaches at the 95% confidence level. The difference between the quality of solutions obtained for the type D problems is not significant but for the large type E problems the result suggests that the difference between the % gaps is also significant at the 95% confidence level.

Table 6.4.6 Time taken by TSBB to find solutions at least as good as Xpress-MP

				TSBB		Xpress	
Type	m	n	Best Bound	Obj Value	Time	Obj Value	Time
c	60	900	9320.43	9446	3.86	9459	28.00
c	20	1600	18800.30	18811	185.64	18811	13.00
c	40	1600	17141.35	17176	23.1	17178	29.00
c	80	1600	16283.00	16468	11.08	16729	47804.00
d	40	400	24348.00	24567	2397.87	24574	4158.00
d	15	900	55401.16	55461	1332.89	55461	2973.00
d	80	1600	97034.00	97449	33495.37	97459	24017.00
e	40	400	44548.93	45383	10.93	45484	37.00
e	15	900	102419.27	102571	13.2	102686	163.00
e	30	900	100423.18	101025	4.76	101312	18.00
e	40	1600	178286.69	179250	11.66	180071	17982.00
e	80	1600	176792.84	178631	37.85	178727	840.00

Considering the entire set of large problems suggests that TSBB is able to produce significantly better results than the default Xpress branch and bound solver for problems of this size and type since the result of the significance test comparing the overall difference in the % gaps is significant at the 99% confidence level.

Table 6.4.7 Significance tests for the difference in % gap between TSBB and Xpress-MP for large problems.

Type	Test statistic	significance
c	-2.2521	0.012
d	-1.402	0.161
e	-2.2521	0.012
all	-4	0

As for the medium size set of problems the large problems were each run a total of 5 times with the different values of the tabu tenure t_a. The average solution value for each problem over the 5 runs is presented in table 6.4.8 in order to compare this value with the best solution obtained by the Xpress-MP branch and bound solver. The average time taken to obtain the best solution in each case is given in column 6 of the table. The average solution values obtained across the five different values of t_a compare favourably with the best solutions found by Xpress, reinforcing the fact that the TSBB algorithm performs well compared to Xpress branch and bound across this particular range of values for t_a.

The results of the significance tests in table 6.4.9 show that the difference in % gap from the best bound of the 5 TSBB solutions for the type C and E problems yields a significant test statistic at the 95% confidence level. There is no evidence to suggest that there is any significant difference for the type D problems however there is evidence to suggest that for all problems combined TSBB is able to find better solutions on average for t_a in the range 5 to 25 than Xpress at the 99% confidence level.

If the two problem sets are combined into one larger set of problems then a comparison can be made of the overall effectiveness of TSBB compared with the Xpress branch and bound approach.

Table 6.4.8 Comparison of average solution for TSBB over 5 runs with best solution obtained by Xpress-MP for large size problems.

				TSBB			Xpress-MP			
Type	m	n	Best Bound	Average solution value	Average time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	10	400	5597.00	5597.00	281.14	0.00	5597	255.00	0.00	y
c	20	400	4779.33	4782.20	1336.68	0.06	4791	8819.00	0.24	n
c	40	400	4243.04	4245.00	1010.81	0.05	4254	6054.00	0.26	n
c	15	900	11338.38	11341.40	3444.62	0.03	11346	4869.00	0.07	n
c	30	900	9979.12	9991.40	1657.99	0.12	9998	3975.00	0.19	n
c	60	900	9320.43	9333.60	1763.68	0.14	9459	28.00	1.49	n
c	20	1600	18800.30	18807.00	4932.39	0.04	18811	13.00	0.06	n
c	40	1600	17141.35	17151.80	1091.61	0.06	17178	29.00	0.21	n
c	80	1600	16283.00	16290.8	26748.59	0.05	16729	47804.00	2.74	n
d	10	400	24957.38	24977.60	2346.78	0.08	24983	12076.00	0.10	n
d	20	400	24556.18	24630.20	5469.28	0.30	24668	5607.00	0.46	n
d	40	400	24348.00	24563.60	5040.72	0.89	24574	4158.00	0.93	n
d	15	900	55401.16	55460.20	5831.54	0.11	55461	2973.00	0.11	n
d	30	900	54830.23	55008.6	6671.97	0.33	54979	8436.00	0.27	n
d	60	900	54551.00	54833.40	2914.67	0.52	54963	41.00	0.76	n
d	20	1600	97821.97	97918.20	36113.54	0.10	97908	6476.00	0.09	n
d	40	1600	97105.00	97355.40	27647.99	0.26	97311	42646.00	0.21	n
d	80	1600	97034.00	97500.20	16064.41	0.48	97459	24017.00	0.44	n
e	10	400	45746.00	45746.00	426.09	0.00	45746	8293.00	0.00	y
e	20	400	44873.07	44893.40	1665.03	0.05	45484	3951.00	1.36	n
e	40	400	44548.93	44633.60	4582.97	0.19	45484	37.00	2.10	n
e	15	900	102419.27	102421.00	2394.00	0.00	102686	163.00	0.26	n
e	30	900	100423.18	100504.40	6544.70	0.08	101312	18.00	0.89	n
e	60	900	100119.91				101954	6492.00	1.83	n
e	20	1600	180642.64	180646.40	7729.54	0.00	181060	30095.00	0.23	n
e	40	1600	178286.69	178408.6	20299.16	0.07	180071	17982.00	1.00	n
e	80	1600	176792.84	177058.80	28657.86	0.15	178727	840.00	1.09	n
					Mean	0.16				0.64

The results for these hypothesis tests are given in table 6.4.10 where it can be seen that for all type C problems from the two problem sets the quality of solution obtained by TSBB can be considered to be significantly better than those obtained by Xpress at the 95% confidence level. For the type D problems the difference in the % gap of solutions obtained by the two methods is also significant at the 95% confidence level and for the type E problems the level of significance is 99%.

Table 6.4.9 Significance tests for the difference in % gap for the average of 5 TSBB runs and Xpress-MP for large size problems

Type	Test statistic	significance
c	-2.2521	0.012
d	0.35	0.726
e	-2.2521	0.012
all	-3.687	0

When considering all problems in the two sets combined TSBB appears to significantly outperform branch and bound over all 45 test problems at the 99% level of significance. When considering all 45 problems the number of observations is large enough for the test statistic to be considered approximately distributed as a standard normal. Comparing the test statistic value of -4.645 against the against a critical value of -2.57 confirms that the difference between the two methods is highly significant.

Table 6.4.10 Significance tests for the difference in % gap between TSBB and Xpress-MP for both test sets combined

Type	Test statistic	significance
c	-2.2521	0.012
d	-2.552	0.011
e	-2.803	0.005
all	-4.645	0

The results of these hypothesis tests reflect the performance of the two algorithms on the three different problem types. The type C problems are relatively easy to solve and so the Xpress branch and bound solver was able to handle these problems better than the harder type D and E problems thus reducing the difference in solution quality between the two methods. The type D problems were generally found to be the most difficult to solve for TSBB and whilst it was able to outperform Xpress on many of these difficult problems the degree to which it was able to find better solutions than Xpress was less than that for the type E problems. The TSBB algorithm generally performed better on the type E problems than the type D problems and was able to obtain significantly better quality of solutions overall.

In the following section the TSBB algorithm will be compared with what could be considered to be the most highly effective tabu search approach to solving GAP and comparison of TSBB with the ejection chain tabu search provides another good indication of the performance of the TSBB algorithm.

6.5 Comparison with Ejection Chain TS

A review of the ejection chain tabu search approach (Yagiura, et al. 2004) is given in the literature review of chapter three of this thesis. Its application to the medium and large size problems is highly effective and it is considered to be a good indication as to the performance of the TSBB algorithm.

6.5.1 Comparison of the medium size test set

The first comparison of the two algorithms, both for these medium size problems and in section 6.5.2 for the large size problems, considers the objective functions of the best solution found by each method. Table 6.5.1 gives the values for both methods along with the respective % gap from the best bound obtained from the Xpress-MP branch and bound solution runs. It is noticeable from the results table that there is no difference between the two

methods for any of the type C problems. For the 12 type D and E problems ECTS obtained solutions with a smaller % gap from the best bound in 5 instances. The mean difference for these 5 problems however is 0.13%. The difference between the overall means for the % gap from the best bound is just 0.03%. In order to establish whether this difference is significant enough to suggest that the ECTS is better at finding high quality solutions than TSBB the hypothesis tests carried out in section 6.4 are applied here as well.

Table 6.5.1 Comparison of the best solution run obtained by TSBB and ECTS for the medium size problem set.

				TSBB		Ejection Chain TS	
Type	m	n	Best Bound	Best solution value	% deviation from best bound	Best solution value	% deviation from best bound
c	5	100	1931.00	1931	0.00	1931	0.00
c	10	100	1402.00	1402	0.00	1402	0.00
c	20	100	1243.00	1243	0.00	1243	0.00
c	5	200	3456.00	3456	0.00	3456	0.00
c	10	200	2806.00	2806	0.00	2806	0.00
c	20	200	2391.00	2391	0.00	2391	0.00
d	5	100	6351.94	6353	0.02	6353	0.02
d	10	100	6335.03	6350	0.24	6349	0.22
d	20	100	6160.76	6222	0.99	6206	0.73
d	5	200	12739.87	12743	0.02	12743	0.02
d	10	200	12422.19	12440	0.14	12440	0.14
d	20	200	12224.29	12280	0.46	12277	0.43
e	5	100	12681.00	12681	0.00	12681	0.00
e	10	100	11577.00	11577	0.00	11577	0.00
e	20	100	8423.77	8467	0.51	8436	0.15
e	5	200	24930.00	24930	0.00	24930	0.00
e	10	200	23307.00	23307	0.00	23307	0.00
e	20	200	22376.28	22380	0.02	22379	0.01
				Mean	0.13		0.10

Table 6.5.2 gives the results for each problem type along with a result for

the entire problem set. There is no difference for the type C problems since both methods are able to find the optimal solutions as found by the Xpress branch and bound solver. For the type D problems alone the results suggest that there is no significant difference between the two methods for solving this type and size of problem since the value of the test statistic does not suggest that there is evidence that the null hypothesis can be rejected. This result is also true for the type E problems only. The test for all problem types of this size considered together shows that there is some evidence at the 95% confidence level to suggest that ECTS is able to find solutions with a smaller % gap from the best bound compared with TSBB

Table 6.5.2 Significance tests for the best solution of TSBB and ECTS for medium size problems.

Type	Test statistic	significance
c	n/a	n/a
d	-1.604	0.109
e	-1.342	0.18
all	-2.023	0.043

The detailed results reported in (Yagiura, et al. 2004) give average solution values for 5 runs of each problem. Table 6.5.3 compares the average of these 5 runs with the average of the 5 runs performed by TSBB and table 6.5.4 give the results of tests with regard to whether the difference in the mean % gap from the best bound is significant.

The results of the hypothesis tests presented in table 6.5.4 show that the mean difference in the quality of the average solution values for each problem are not significantly different for the type C and E problems although the results for the type D problems seem to indicate that the average solution values attainable by ECTS are significantly better than those from the TSBB solutions at the 90% confidence level. This level of confidence increases to 95% for the entire problem set.

Table 6.5.3 Comparison of the average of 5 solution runs obtained by TSBB and ECTS for the medium size problem set.

				TSBB		Ejection Chain TS	
Type	m	n	Best Bound	Average solution value	% deviation from best bound	Average solution value	% deviation from best bound
c	5	100	1931.00	1931.00	0.00	1931	0.00
c	10	100	1402.00	1402.00	0.00	1402	0.00
c	20	100	1243.00	1243.00	0.00	1243	0.00
c	5	200	3456.00	3456.00	0.00	3456	0.00
c	10	200	2806.00	2806.00	0.00	2806	0.00
c	20	200	2391.00	2391.00	0.00	2391	0.00
d	5	100	6351.94	6353.00	0.02	6353	0.02
d	10	100	6335.03	6356.80	0.34	6351.8	0.26
d	20	100	6160.76	6230.00	1.12	6210.6	0.81
d	5	200	12739.87	12744.00	0.03	12743.2	0.03
d	10	200	12422.19	12445.20	0.19	12441.6	0.16
d	20	200	12224.29	12292.00	0.55	12278.6	0.44
e	5	100	12681.00	12681.00	0.00	12681	0.00
e	10	100	11577.00	11577.00	0.00	11577	0.00
e	20	100	8423.77	8475.60	0.62	8438.4	0.17
e	5	200	24930.00	24930.00	0.00	24930	0.00
e	10	200	23307.00	23307.00	0.00	23307	0.00
e	20	200	22376.28	22388.00	0.05	22379	0.01
				Mean	0.16		0.11

The evidence would seem to suggest that the ECTS algorithm is able to outperform TSBB for this problem set although it would seem that this is largely due to the difference between the solutions for the type D problems which were the hardest of the three problem types for TSBB. For the type C and E problems the results suggest that there is little or no difference between the two methods of problems of medium size.

Table 6.5.4 Hypothesis test results for the difference of 5 solutions runs for each problem by TSBB and ECTS

Type	Test statistic	significance
c	n/a	n/a
d	-1.826	0.068
e	-1.342	0.18
all	-2.201	0.028

6.5.2 Comparison of the large size test set

The results given in the previous section indicated that TSBB was competitive compared to the ECTS method for the types C and E problems of medium size and the results that follow seem to indicate that this is indeed the case for the large problems also. Table 6.5.5 details the solution values for the large set of problems along with the % gap from the best bound found by the Xpress-MP branch and bound solver. The results of the hypothesis tests given in table 6.5.6 suggest that the ECTS performs better than the TSBB method for this problem set. There are however some positive aspects of the performance of TSBB for this problem set. Of the 27 problems ECTS is able to find better solution values for 19 out of the 27, of the remaining 8 problems TSBB is able to match the solutions found by ECTS for 3 instances and for the other 5 problems TSBB is able to find better solutions than ECTS. Of the 5 instances where TSBB was able to improve on the best solutions found by ECTS, 1 instance was for a type C problem and the remaining 4 were type E problems. The 1 type C problem was in fact the largest of the 9 sizes of problem and for the type E problems improved solution values were found across all three values for n.

The results presented in table 6.5.7 give a comparison of the average of 5 solution runs for each problem instance.

Table 6.5.5 Comparison of the best solution run obtained by TSBB and ECTS for the large size problem set.

				TSBB		Ejection Chain TS	
Type	m	n	Best Bound	Best solution value	% deviation from best bound	Best solution value	% deviation from best bound
c	10	400	5597.00	5597	0.00	5597	0.00
c	20	400	4779.33	4782	0.06	4782	0.06
c	40	400	4243.04	4245	0.05	4244	0.02
c	15	900	11338.38	11341	0.02	11340	0.01
c	30	900	9979.12	9989	0.10	9984	0.05
c	60	900	9320.43	9331	0.11	9328	0.08
c	20	1600	18800.30	18805	0.02	18803	0.01
c	40	1600	17141.35	17150	0.05	17147	0.03
c	80	1600	16283.00	**16289**	0.04	16291	0.05
d	10	400	24957.38	24976	0.07	24974	0.07
d	20	400	24556.18	24620	0.26	24604	0.19
d	40	400	24348.00	24552	0.84	24456	0.44
d	15	900	55401.16	55456	0.10	55425	0.04
d	30	900	54830.23	55012	0.33	54983	0.28
d	60	900	54551.00	54785	0.43	54656	0.19
d	20	1600	97821.97	97906	0.09	97867	0.05
d	40	1600	97105.00	97328	0.23	97160	0.06
d	80	1600	97034.00	97449	0.43	97097	0.06
e	10	400	45746.00	45746	0.00	45746	0.00
e	20	400	44873.07	**44877**	0.01	44882	0.02
e	40	400	44548.93	44609	0.13	44579	0.07
e	15	900	102419.27	**102421**	0.00	102422	0.00
e	30	900	100423.18	**100430**	0.01	100438	0.01
e	60	900	100119.91	100363	0.24	100177	0.06
e	20	1600	180642.64	**180645**	0.00	180647	0.00
e	40	1600	178286.69	178391	0.06	178311	0.01
e	80	1600	176792.84	177035	0.14	176856	0.04
				Mean	0.14		0.07

These average values follower a similar pattern to the one observed when

analysing the best of the 5 solution runs for each problem. The results of the hypothesis tests detailed in table 6.5.8 reach the same conclusions as the tests conducted for the difference in % gap for the best solutions also. In 4 instances the average of the 5 different runs for TSBB the average solution values are smaller than the average of the 5 solution values from the ECTS runs. Whilst this is not sufficient to be able to say that TSBB performs better it does indicate that in certain situations there are some instances where TSBB will perform as well as, or even better than ECTS. Since ECTS is known to outperform most other algorithms for these types and sizes of problems it is encouraging that the TSBB algorithm is able to surpass this performance in some circumstances. A significant point of note is that the 5 TSBB runs were implemented with five different settings of the tabu tenure reinforcing the observation that the algorithm is particularly robust across this range of settings.

Table 6.5.6 Hypothesis test for the difference in the % gap for best solution values from the large size problems.

Type	Test statistic	significance
c	-1.859	0.063
d	-2.666	0.008
e	-1.12	0.263
all	-3.686	0

In the following section 6.6 the TSBB algorithm will be assessed against several other algorithms reported in the literature to be reasonably successful at solving the two problems sets which have been used to assess TSBB.

Table 6.5.7 Comparison of the average of 5 runs by TSBB and ECTS for the large size problem set.

				TSBB		Ejection Chain TS	
Type	m	n	Best Bound	Average solution value	% deviation from best bound	Average solution value	% deviation from best bound
c	10	400	5597.00	5597.00	0.00	5597	0.00
c	20	400	4779.33	4782.20	0.06	4782.4	0.06
c	40	400	4243.04	4245.00	0.05	4244.6	0.04
c	15	900	11338.38	11341.40	0.03	11340.4	0.02
c	30	900	9979.12	9991.40	0.12	9984.6	0.05
c	60	900	9320.43	9333.60	0.14	9329	0.09
c	20	1600	18800.30	18807.00	0.04	18803.2	0.02
c	40	1600	17141.35	17151.80	0.06	17147.4	0.04
c	80	1600	16283.00	16290.8	0.05	16292.4	0.06
d	10	400	24957.38	24977.60	0.08	24976.4	0.08
d	20	400	24556.18	24630.20	0.30	24609	0.22
d	40	400	24348.00	24563.60	0.89	24461.2	0.46
d	15	900	55401.16	55460.20	0.11	55433.4	0.06
d	30	900	54830.23	55008.6	0.33	54908.8	0.14
d	60	900	54551.00	54833.40	0.52	54666.6	0.21
d	20	1600	97821.97	97918.20	0.10	97872.4	0.05
d	40	1600	97105.00	97355.40	0.26	97166	0.06
d	80	1600	97034.00	97500.20	0.48	97103	0.07
e	10	400	45746.00	45746.00	0.00	45746	0.00
e	20	400	44873.07	44893.40	0.05	44883.4	0.02
e	40	400	44548.93	44633.60	0.19	44584.6	0.08
e	15	900	102419.27	102421.00	0.00	102423	0.00
e	30	900	100423.18	100504.40	0.08	100440.6	0.02
e	60	900	100119.91	100375.40	0.26	100181.2	0.06
e	20	1600	180642.64	180646.40	0.00	180647.4	0.00
e	40	1600	178286.69	178408.6	0.07	178313.2	0.01
e	80	1600	176792.84	177058.80	0.15	176862.8	0.04
				Mean	0.16		0.07

Table 6.5.8 Hypothesis test for the difference in the % gap for the average of 5 runs for the large size problems.

Type	Test statistic	significance
c	-2.047	0.041
d	-2.524	0.012
e	-2.214	0.027
all	-3.949	0

6.6 Comparison with other Heuristics

In the two previous sections a detailed comparison of the results and performance of the TSBB algorithm with firstly a commercial integer programming branch and bound solver and secondly with the ejection chain tabu search approach. This section provides analysis and comparison of the TSBB algorithm with several other heuristic approaches that have been applied to the two problem sets for GAP. The first of these comparisons in section 6.6.1 compares the TSBB with 9 alternative algorithms that have all been tested on the medium size problem set while section 6.6.2 compares TSBB with 4 other approaches whose performance has been tested on the large problem set.

6.6.1 Comparison of alternative algorithms for medium size problems

The TSBB algorithm is compared with 9 other heuristics for the medium size problems and the solution values for each of these is given in table 6.6.1 where solution values in bold type indicates the best solution found out of all the methods. The values for the 9 heuristics are taken from results reported in (Yagiura, et al. 2006), where the problems with $n = 100$ were limited to 150 seconds of solution time and the problems with $n = 200$ were limited to 300 seconds of solution time. In order to make a fair comparison the values for

TSBB were taken from the best solution runs with the same time limits. The 9 other heuristic methods are:

- Path relinking with ejection chains (PREC) (Yagiura, et al. 2006)
- Ejection chain tabu search (ECTS) (Yagiura, et al. 2004)
- Two branching variable depth search methods (BVDS-l and BVDS-j) (Yagiura, et al. 1998)
- A variable depth search method due to Yagiura (VDS) (Yagiura, Yamaguchi et al. 1999)
- A variable depth search approach by Racer and Amini (RA) (Amini and M. Racer. 1994)
- A tabu search approach Laguna et al (LKGG) (Laguna, et al. 1995)
- A genetic algorithm approach due to Chu and Beasley (CB) (Beasley and P. C. Chu. 1997)
- The tabu search method by Diaz and Fernandez (DF) (Diaz and E. Fernandez. 2001).

Although a detailed comparison for TSBB with ECTS has already been presented in section 6.5 this was performed with the longer time limits as detailed in that section. The best bound for each problem given in table 6.6.1 is the one found by Xpress-MP during its longer run as specified in section 6.4. The summary of results in table 6.6.1 show that the results obtained by TSBB in this short time limit compare very favourably against all of the other heuristics for these much shorter runs with regard to the actual solution values. In keeping with the results presented in sections 6.4 and 6.5 the % gap from the best bound has been calculated for each algorithm in order to make a fair comparison and are detailed in table 6.6.2. The comparisons made in sections 6.4 and 6.5 were carried out using t-tests for the difference in % gaps between two methods. In order to make comparisons between these multiple methods it is more appropriate to use ANOVA analysis in order to determine significant differences between approaches.

Table 6.6.1 Solution values of the different heuristics for the medium size problems.

Type	m	n	Best bound	TSBB	PREC	ECTS	BVDS-I	BVDS-J	VDS	RA	LKGG	CB	DF
C	5	100	1931.00	**1931**	**1931**	**1931**	**1931**	**1931**	**1931**	**1938**	**1931**	**1931**	**1931**
	10	100	1402.00	**1402**	**1402**	**1402**	**1402**	1403	**1402**	1405	1403	1403	**1402**
	20	100	1243.00	**1243**	**1243**	**1243**	1244	1244	1246	1250	1245	1244	**1243**
	5	200	3456.00	**3456**	**3456**	**3456**	**3456**	3457	3457	3469	3457	3458	3457
	10	200	2806.00	**2806**	2807	**2806**	2809	2808	2809	2835	2812	2814	2807
	20	200	2391.00	**2391**	**2391**	2392	2401	2400	2405	2419	2396	2397	**2391**
D	5	100	6351.94	6355	**6353**	6357	6358	6362	6365	n.a	6386	6373	6357
	10	100	6335.03	6366	6356	6358	6367	6370	6380	6532	6406	6379	**6355**
	20	100	6160.76	6254	**6211**	6221	6275	6245	6284	6428	6297	6269	6220
	5	200	12739.87	12745	**12744**	12746	12755	12755	12778	n.a	12788	12796	12747
	10	200	12422.19	12449	**12438**	12446	12480	12473	12496	12799	12537	12601	12457
	20	200	12224.29	12332	**12269**	12284	12440	12318	12335	12665	12436	12452	12351
E	5	100	12681.00	**12681**	**12681**	12682	**12681**	12682	12685	12917	12687	n.a	**12681**
	10	100	11577.00	**11577**	**11577**	**11577**	11585	11599	11585	12047	**11641**	n.a	11581
	20	100	8423.77	8488	8444	**8443**	8499	8484	8490	9004	8522	n.a	8460
	5	200	24930.00	**24930**	**24930**	**24930**	24942	24933	24948	25649	25147	n.a	24931
	10	200	23307.00	**23307**	23310	**23307**	23346	23348	23340	24717	23567	n.a	23318
	20	200	22376.28	22380	**22379**	22391	22475	22437	22452	24117	22659	n.a	22422

The difference in performance between the algorithms, measured by % gap from the best bound, is assessed for each of the three problem types and also for the entire set of 18 problems combined. The ANOVA tables for each of these four situations are given in table 6.6.3. and clearly show that there is a significant difference between the mean % gap of at least two of the methods for each of the problem types and for the whole set of problems. In order to identify which of the methods differ significantly from others Tukey's method is used to create pairwise confidence intervals and thus identify which methods are significantly better than others.

Table 6.6.2 % gap from the best bound for the 9 comparative heuristics.

Type	m	n	Best bound	TSBB	PREC	ECTS	BVDS-I	BVDS-J	VDS	RA	LKGG	CB	DF
C	5	100	1931.00	0	0	0	0	0	0	0.363	0	0	0
	10	100	1402.00	0	0	0	0	0.071	0	0.214	0.071	0.071	0
	20	100	1243.00	0	0	0	0.08	0.08	0.241	0.563	0.161	0.08	0
	5	200	3456.00	0	0	0	0	0.029	0.029	0.376	0.029	0.058	0.029
	10	200	2806.00	0	0.036	0	0.107	0.071	0.107	1.033	0.214	0.285	0.036
	20	200	2391.00	0	0	0.042	0.418	0.376	0.586	1.171	0.209	0.251	0
	Mean			0	0.006	0.007	0.101	0.105	0.16	0.62	0.114	0.124	0.011
D	5	100	6351.94	0.046	0.017	0.08	0.095	0.158	0.206	n.a	0.536	0.331	0.08
	10	100	6335.03	0.489	0.331	0.363	0.505	0.552	0.71	3.109	1.12	0.694	0.315
	20	100	6160.76	1.513	0.815	0.978	1.854	1.367	2	4.338	2.211	1.757	0.962
	5	200	12739.87	0.04	0.032	0.048	0.119	0.119	0.299	n.a	0.378	0.441	0.056
	10	200	12422.19	0.216	0.127	0.192	0.465	0.409	0.594	3.033	0.924	1.439	0.28
	20	200	12224.29	0.881	0.366	0.488	1.765	0.767	0.906	3.605	1.732	1.863	1.037
	Mean			0.531	0.281	0.358	0.801	0.562	0.786	3.521	1.15	1.088	0.455
E	5	100	12681.00	0	0	0.008	0	0.008	0.032	1.861	0.047	n.a	0
	10	100	11577.00	0	0	0	0.069	0.19	0.069	4.06	0.553	n.a	0.035
	20	100	8423.77	0.763	0.24	0.228	0.893	0.715	0.786	6.868	1.166	n.a	0.43
	5	200	24930.00	0	0	0	0.048	0.012	0.072	2.884	0.87	n.a	0.004
	10	200	23307.00	0	0.013	0	0.167	0.176	0.142	6.05	1.116	n.a	0.047
	20	200	22376.28	0.017	0.012	0.066	0.441	0.271	0.338	7.779	1.263	n.a	0.204
	Mean			0.13	0.044	0.05	0.27	0.229	0.24	4.92	0.836	n.a	0.12
Overall Mean				0.22	0.111	0.138	0.39	0.298	0.395	2.958	0.7	0.606	0.195

The results show that the TSBB algorithm is significantly better than RA for each of the three problem types and for the whole of the problem set. The confidence level of these tests was set at 95 % and shows that TSBB is at least as good as 8 out of the 9 other heuristics in terms of quality of solution for this set with these time limits and performs significantly better than RA in these circumstances.

Table 6.6.3 ANOVA tables for difference in %gap of the 9 methods for medium size problems.

Type C

Source	DF	SS	MS	F	P
Method (ind)	9	1.8254	0.2028	7.42	0.000
Error	50	1.3661	0.0273		
Total	59	3.1915			

Type D

Source	DF	SS	MS	F	P
Method (ind)	9	34.887	3.876	11.68	0.000
Error	48	15.930	0.332		
Total	57	50.817			

Type E

Source	DF	SS	MS	F	P
Method (ind)	8	119.579	14.947	21.91	0.000
Error	45	30.700	0.682		
Total	53	150.279			

All

Source	DF	SS	MS	F	P
Method (Tot)	9	105.895	11.766	16.25	0.000
Error	162	117.268	0.724		
Total	171	223.163			

Further analysis of these results also shows that there are no significant differences between 8 of the 9 algorithms in terms of quality of solution and the only method which performs significantly worse than the others is the RA method. The results of a similar analysis for the large set of test problems is now presented in section 6.6.2.

6.6.2 Comparison of alternative algorithms for large size problems

There are fewer methods in the literature that have been tested on the large set of test problems and so the comparison is restricted to 4 alternative algorithms in this instance. These are the path relinking with ejection chain, branching variable depth search, the tabu search LKGG and the Diaz and Fernandez tabu search. No comparison is given here with the ejection chain tabu search since this has already been given in section 6.5.

Table 6.6.4 Solution values of the different heuristics for the large size problems.

Type	m	n	Best bound	TSBB	PREC	BVDS	LKGG	DF
	10	400	5597.00	**5597**	**5597**	5605	5608	5598
	20	400	4779.33	**4782**	**4782**	4795	4792	4786
	40	400	4243.04	**4245**	**4245**	4259	4251	4248
	15	900	11338.38	**11341**	**11341**	11368	11362	n.a
C	30	900	9979.12	9989	**9984**	10022	10007	n.a
	60	900	9320.43	9331	**9328**	9386	9341	n.a
	20	1600	18800.30	18805	**18803**	18892	18831	n.a
	40	1600	17141.35	17150	**17145**	17262	17170	n.a
	80	1600	16283.00	**16289**	**16289**	16380	16303	n.a
	10	400	24957.38	24976	**24969**	25032	25145	25039
	20	400	24556.18	24620	**24587**	24780	24872	24747
	40	400	24348.00	24552	**24417**	24724	24726	24707
	15	900	55401.16	55456	**55414**	55614	56423	n.a
D	30	900	54830.23	55012	**54868**	55210	55918	n.a
	60	900	54551.00	54785	**54606**	55123	55379	n.a
	20	1600	97821.97	97906	**97837**	98248	100171	n.a
	40	1600	97105.00	97328	**97113**	97721	99290	n.a
	80	1600	97034.00	97449	**97052**	98146	98439	n.a
	10	400	45746.00	**45746**	**45746**	45878	172185	45781
	20	400	44873.07	**44877**	44879	45079	137153	45007
	40	400	44548.93	44609	**44574**	44898	63669	44921
	15	900	102419.27	**102421**	102422	102755	463142	n.a
E	30	900	100423.18	**100430**	100434	100956	527451	n.a
	60	900	100119.91	100363	**100169**	100917	479650	n.a
	20	1600	180642.64	**180645**	180646	181143	936609	n.a
	40	1600	178286.69	178391	**178302**	179036	1026259	n.a
	80	1600	176792.84	177035	**176857**	178205	1026417	n.a

Table 6.6.4 gives the solution values for each of the 5 algorithms. The results for TSBB are the best of the five solution runs as detailed in section 6.4 whilst the solution values for the other 4 algorithms are those reported in (Yagiura, et al. 2006) .

Table 6.6.5 % gap from the best bound for the 5 comparative heuristics.

Type	m	n	Best bound	TSBB	PREC	BVDS	LKGG	DF
	10	400	5597.00	0	0	0.1429337	0.1965339	0.0178667
	20	400	4779.33	0.055915	0.055915	0.3279198	0.2651495	0.1396088
	40	400	4243.04	0.0460887	0.0460887	0.3760405	0.1874966	0.1167927
	15	900	11338.38	0.0231342	0.0231342	0.2612636	0.2083459	n.a
C	30	900	9979.12	0.0989958	0.0488911	0.4296862	0.2793724	n.a
	60	900	9320.43	0.1134311	0.0812438	0.7035328	-96.34137	n.a
	20	1600	18800.30	0.0249996	0.0143615	0.4877582	0.1632953	n.a
	40	1600	17141.35	0.0504651	0.0212958	0.7038558	0.167142	n.a
	80	1600	16283.00	0.0368482	0.0368482	0.5957133	0.1228275	n.a
			Mean	0.0499864	0.0364198	0.4476338	-10.52791	0.0914227
	10	400	24957.38	0.0746194	0.0465716	0.299002	0.7517739	0.3270498
	20	400	24556.18	0.2598792	0.1254935	0.9114462	1.2860973	0.7770605
	40	400	24348.00	0.8378512	0.2833908	1.5442747	1.5524889	1.4744538
	15	900	55401.16	0.0989939	0.0231832	0.3841865	1.8444448	n.a
D	30	900	54830.23	0.3315205	0.0688916	0.6926352	1.9838938	n.a
	60	900	54551.00	0.4289564	0.1008231	1.0485601	1.5178457	n.a
	20	1600	97821.97	0.0859022	0.0153659	0.4355169	2.401333	n.a
	40	1600	97105.00	0.2296483	0.0082385	0.6343649	2.2501416	n.a
	80	1600	97034.00	0.4276851	0.0185502	1.1459901	1.4479461	n.a
			Mean	0.3083396	0.0767232	0.7884418	1.6706628	0.8595213
	10	400	45746.00	0	0	0.2885498	276.39356	0.0765094
	20	400	44873.07	0.0087486	0.0132056	0.4589072	205.64654	0.2984547
	40	400	44548.93	0.1348412	0.0562759	0.7835661	42.919259	0.8351947
	15	900	102419.27	0.0016858	0.0026622	0.3277963	352.202	n.a
E	30	900	100423.18	0.0067916	0.0107747	0.530575	425.22834	n.a
	60	900	100119.91	0.2428026	0.049035	0.7961391	379.07556	n.a
	20	1600	180642.64	0.0013061	0.0018597	0.2769885	418.48722	n.a
	40	1600	178286.69	0.0585083	0.0085887	0.4202852	475.6229	n.a
	80	1600	176792.84	0.1369717	0.0362889	0.7987632	480.5761	n.a
			Mean	0.0657395	0.0198545	0.5201745	339.57239	0.4033863
			Overall mean	0.1413552	0.0443325	0.5854167	110.23838	0.4514435

As with all previous results the % gap from the best bound has been calculated and these are summarized in table 6.6.5. These values are once again used to perform a comparison of the alternative algorithms with an ANOVA analysis to identify significant differences between the quality of solution obtained by each using the % gap from the best bound as the indicator of performance. The ANOVA tables have been calculated for each of the three problem types for this large set as well as an ANOVA table comparing the entire set of problems. These results are summarized in table 6.6.6. There are no reported results for the DF algorithm for problems where $n = 900$ and $n = 1600$.

Table 6.6.6 ANOVA tables for difference in %gap of the 5 methods for large problems.

Type C

Source	DF	SS	MS	F	P
Method	4	1.0095	0.2524	24.85	0.000
Error	34	0.3453	0.0102		
Total	38	1.3548			

Type D

Source	DF	SS	MS	F	P
Method	4	13.484	3.371	24.83	0.000
Error	34	4.616	0.136		
Total	38	18.099			

Type E

Source	DF	SS	MS	F	P
Method	4	797254	199313	41.42	0.000
Error	34	163608	4812		
Total	38	960861			

All

Source	DF	SS	MS	F	P
Method	4	267740	66935	8.80	0.000
Error	112	851672	7604		
Total	116	1119412			

The ANOVA tables clearly show that there is a significant difference between at least two of the algorithms and so further pair wise comparisons

are performed using Tukey's method to identify which of the algorithms differ significantly. The analysis shows that the % gaps for the type C and D solutions found by TSBB are significantly better at the 95% confidence level than both the LKGG algorithm and the BVDS algorithm. The results also indicate that the TSBB performs better on the type E problems than LKGG as it does for the entire set of large problems. The results also indicate that the % gaps for the TSBB solutions form the best bound are at least as good as those for PREC and DF. Although the comparison between TSBB and DF is not significantly different it should be noted that DF only provides solutions for three problems of each type with $n = 400$ and the mean % gap over all problems sizes for TSBB for each type is lower than that of DF for the problems with $n = 400$ indicating the ability of TSBB to find better solutions than DF.

6.7 Summary

This chapter has given a detailed description of the computational experimentation and presented detailed results of the experimentation. The functionality of the different aspects of the algorithm have been analysed and an explanation of its parameter settings given. The performance of the algorithm has been assessed in detail by comparing the results obtained during computational testing on two sets of benchmark test problems with those obtained by the branch and bound solver in the Xpress-MP software and other high quality heuristic algorithms that have been reported in the literature. By measuring the quality of performance of an algorithm by comparing the % gap between the solution value and the best bound found by the Xpress solver statistical tests of significant differences have been conducted. The TSBB algorithm appears to work effectively at solving a range of problems of differing size and difficulty, some of which prove to be highly challenging to even the most powerful of heuristic approaches. Whilst TSBB clearly is able to outperform the default branch and bound strategy of Xpress for these two sets of test problems the detailed comparison with the ejection chain tabu

search shows this to be very difficult to outperform. Encouragingly TSBB is able to find improved solutions compared to those found by ECTS on some large and difficult problems. The ANOVA analysis presented in section 6.6 shows that the TSBB algorithm is able to significantly outperform some of the alternative algorithms both on the medium size problems with short running times and also for the larger problems with longer run times. The results of this analysis also show that none of the alternative algorithms are able to significantly outperform TSBB with any degree of confidence suggesting that TSBB is at least as good as all the alternative algorithms under these circumstances. Noticeably algorithm PREC seems to perform particularly well on the type D problems due to the importance it places on the diversification aspect of the search. Yagiura et. al (Yagiura, et al. 2006) give some experimental results that indicate that the distance between locally optimal solutions for the type D problems are further apart than those for type C and E problems and so the strong diversification element of PREC attempts to overcome this.

7 Conclusions and discussion

The research presented in this thesis has described the development of a hybrid approach to solving the generalized assignment problem (GAP). GAP is one of a class of combinatorial problems that are known to be NP-Hard and as a result solution approaches for solving it have been widely researched within the OR community over the last thirty years due to its theoretical importance. Its importance is further enhanced since GAP is used to model a variety of real world applications as described in chapter 2. During this thirty year period some of these practical situations that can be modelled by GAP have become larger, more complex and thus much harder to solve. Also during this time there have been considerable developments in approaches for solving GAP and other combinatorial optimization problems and so much larger and more difficult instances of GAP have been generated and used as benchmark test instances for the ever more powerful heuristic and meta-heuristic approaches that have been developed including Genetic Algorithms, Tabu Search and Simulated Annealing. More recently however researchers have attempted to combine different methods in order to construct even more powerful hybrid methods and as described earlier this is the approach taken during this research. The focus of the discussion in this chapter is firstly on the motivation, development and performance of the TSBB algorithm and secondly section 7.2 is devoted to suggestions for further work and research.

7.1 TSBB development and performance

A key aspect of the hybrid TS/Branch and Bound algorithm, developed during this research and presented in chapter 5, is to search the solution space of the problem by identifying promising regions of that space, generating sub-problems to represent the smaller more focused region and then solving the sub-problem prior to defining a different sub-region to be examined during the next iteration of the algorithm. The tabu search aspect of the algorithm is used to generate these sub-regions using both recency and frequency based memory

structures. This approach of generating sub-problems as a means of obtaining solutions to large and difficult problems has been applied in a number of ways by taking advantage of the structure of GAP. This is possible due to the fact that GAP can be viewed as a series of related Knapsack or Assignment problems and whose relevance to the GAP has been described in chapter 2. This divide and conquer approach has been adopted quite successfully in some methods although the increase in difficulty with modest increase in the sizes of GAP instances proves testing even for the appropriate sub-problems. Another successful approach has been to relax some of the problem constraints in order to generate an infeasible solution and then for feasibility to be restored in a heuristic manner. This approach is indeed central to the conventional branch and bound method which generates LP relaxations to perform a tree search in order to identify integer feasible solutions. In this respect TSBB takes advantage of the structure of GAP since the solution to the LP relaxation contains a large number of binary assignments to variables leaving a relatively small number of infeasible fractional assignments of values to variables and TSBB takes advantage of this in the short-term phase of the algorithm to generate integer feasible solutions. All of these and other approaches to solving GAP have been comprehensively reviewed in chapter 3.

One advantage to the TSBB method is that it utilises a commercial mathematical programming software application, the Xpress-MP solver, to implement the standard branch and bound aspect of the algorithm. There are alternative mathematical programming software applications that could be used to implement TSBB and indeed these modern solvers tend to be very powerful and sophisticated implementations of mathematical programming techniques. The Xpress-MP software provides the facility to interact with the branch and bound solver in order to customize approaches to different problem types whilst also providing a highly effective set of default settings which are able to solve a wide variety of problems. In-built into the software are a set of libraries that can be called from within the user's own program

which provides a high degree of flexibility for the programmer. This has been important in terms of implementation of the TSBB algorithm since it has been coded in C thus allowing the generated sub-problems to be easily called from within the program at the relevant stages of the algorithm. This ease of implementation is considered to be a positive aspect of the TSBB algorithm.

There are three phases to the TSBB algorithm that perform separate tasks and yet are integrative in the sense that they must work together in order to be successful in finding good integer solutions. The short-term phase of the algorithm moves from one integer solution to a neighbouring solution by means of dropping assignments from the current solution and then adding new assignments to the resulting partial solution in order to produce a new solution. The dropping of assignments from the current solution is achieved by solving a linear relaxation with an added *tabu* constraint and is advantageous in that it is quickly and easily achieved by a call to Xpress once the relaxed LP problem has been formulated from within the program. The integer assignments that occur naturally within the resulting solution are then fixed and another call to Xpress generates a new integer solution. The objective of this short term phase is to look for good integer solutions with the intention of

a) Improving the objective function value of the best solution found during the search since this will assist the intensification phase of the algorithm by providing a cut-off value which is intended to reduce the amount of effort spent searching the region identified for the intensification phase and
b) Providing the frequency memory with information about attributes that tend to be found in good integer solutions so that the intensification phase can focus its search in an attempt to find solutions containing such assignments.

The results presented in chapter 6.3 are evidence of the fact that the short-term phase does achieve both of these objectives very well in most situations since

many new improving best solutions are found during this search phase even during the latter stages of the time allowed for solving the problem. This is in contrast to the performance of the standard branch and bound approach of Xpress-MP which tends to reduce the gap between the best integer solution and the best bound early on in the solution process but can then be drawn into large unproductive areas of the branch and bound tree where it fails to improve the solution any further, which is particularly evident in the larger and harder type D and E problems.

The intensification phase of the TSBB algorithm is in keeping with the standard tabu search strategy of using frequency based memory in order to identify promising solution attributes that can then be incorporated into a more focused search on regions of the solution space where these solution attributes occur. The frequency based memory employed within TSBB is a long term memory structure that is driven by the short-term phase of the algorithm and is used to generate sub-problems where the integer feasible solution space contains only integer solutions that have assignments that are contained in a very high proportion of the integer solutions found during the short-term phases of the search. These regions are easily identified from the frequency based memory and are just as easily implemented by the addition of a constraint to the sub-problem that is given to Xpress to solve. This more focused search can then be performed using the standard branch and bound approach and in keeping with this strategy a cut off value is given to the solver as a result of improvements made during the short-term phase in order to further reduce the computational effort in searching this region. An additional aspect to this intensification phase is that as the search progresses the number of assignments that appear in a high proportion of the total number of integer solutions reduces. This could be perceived as the area to be searched by branch and bound is growing in size since the restriction on assignments that must be contained in any new integer solution is fewer. As this occurs however it is usual that the objective function value of the best

solution is at such a level that it becomes extremely difficult to find any further improving solutions and in such situations fixing fewer assignments during the intensification phase could in fact be advantageous as it allows more scope for the introduction of new assignments. This could effectively be viewed as a widening of the intensification search area similar in fact to the variable neighbourhood approach which tends to widen the search when there is no improvement to be found within the current neighbourhood. The results presented in section 6.3 show once again that this strategy does appear to work quite effectively as solutions are improved during intensification across all the three different types of problem and also across the different sizes of problem although its effect does seem to diminish for the very large very difficult instances.

The third phase of the algorithm is the diversification phase and whilst this once again follows a fairly straight forward tabu search strategy that uses frequency based memory in order to guide the search to new and unexplored areas of the solution space its role is still an important aspect of the algorithm. This is once again easily implemented by means of the addition of a tabu constraint to the original problem. This restricted problem is once again given to Xpress to solve and the first integer solution found during the tree search is accepted as the new solution for the next short-term phase of the algorithm. Since the purpose of the tabu constraint is to exclude assignments from the next solution it is less restrictive than both the short-term and intensification sub-problems and as such an integer solution is usually found quite quickly by Xpress, the timing of this phase therefore has not been a practical issue. Whilst the aim of this phase of the search is not to find an improving solution its strategic importance cannot be overlooked and its contribution in terms of guiding the search to areas of the solution space where the possibility of new improvements can be made is crucial to the overall performance of the algorithm. There is evidence of its success in terms of achieving this guidance in the search profiles of some of the problems presented in section 6.3. This

can be seen in the periods of time where there is no improvement in the best solution and then a new period of improving solutions are found by the short term phase. Since the solution found during diversification is used to implement the next short-term phase then such an occurrence is due to the move to a new region by means of diversification.

In section 6.3 detailed results of a comparison of TSBB with the Xpress-MP branch and bound solver on two sets of benchmark test problems are presented. Analysis of these results clearly shows that the TSBB strategy used to guide the branch and bound aspect does indeed lead to areas of the solution space that the standard branch and bound approach would otherwise not encounter. The objective function values of the best obtainable solutions by TSBB generally tend to be lower than those found by the Xpress solver. By focusing the search based on information gained from previous integer feasible solutions the neighbourhoods defined by the subsequent sub-problems tend to be more fruitful than the areas of the branch and bound tree that are chosen to be searched by the default settings of the Xpress software. In the instances where Xpress is able to obtain optimal solutions to a problem TSBB is also able to find the optimal solution in every case and usually in significantly shorter periods of time than Xpress. By using the % gap from the best bound found by the Xpress solver as a measure of the quality of the solution found by each method it was possible to confirm, by testing whether there was any difference between the mean % gap of the two approaches, that TSBB is significantly better than Xpress at solving the types and sizes of the GAP instances contained within the two sets of benchmark problems. These results also reinforce the fact that the memory structures utilised within the different phases of the TSBB algorithm are effective at exploiting the solution space and defining sub-trees that provide high quality solutions.

In section 6.4 further comparisons were made with an ejection chain tabu search approach that is widely accepted as one of the most effective tabu

search approaches for solving the instances of GAP from the two sets of test problems. This situation provided a much tougher test of the quality of solution found by TSBB although the comparison shows that TSBB is quite competitive across the range of problems. The significance tests for the medium size problems show that when considering each of the three problem types in isolation TSBB is able to match the quality of solution found by ECTS although when considering the test set as a whole there is evidence to show that ECTS can outperform TSBB for this problem set. There are however 2 problem instances, one of type D and one of type E, where ECTS is able to find a solution significantly better than those found by TSBB. Removal of these two problems would suggest that the difference between the two methods is much less significant.

The significance tests for the large problems indicate that ECTS can indeed outperform TSBB for this set of test problems. Further examination of the objective function values for this set of results indicate that in general TSBB is able to find solutions with objective function values close to those obtainable by ECTS and in some instances is able to find solutions with better objective function values than those found by ECTS. It could be argued therefore that for some instances TSBB will outperform ECTS and on this basis TSBB should be considered as a valid approach to apply to these types and sizes of problem particularly in view of its ease of implementation and utilisation of existing commercially available software.

The final set of comparisons in section 6.6 also give a good indication that TSBB can be competitive against a range of different approaches that have been developed and reported in the literature in recent years. In contrast to the comparisons of 6.4 and 6.5 a comparison with alternative algorithms with a much shorter time limit is performed. The results of these comparisons show that TSBB is competitive against all of the other algorithms and can therefore be considered as a valid approach for solving large hard instances of GAP

over longer time periods as well as shorter ones.

The TSBB algorithm developed during this research and presented here in this thesis has practical relevance since it has been shown to perform well on benchmark instances of varying size and difficulty and there is evidence to show that it is extremely competitive with regard to the quality of solutions obtained. TSBB also has theoretical relevance since it shows how it is possible to use tabu search memory structures to guide a standard branch and bound process by defining neighbourhoods that are represented by sub-problems that can be passed to an existing commercial solver to be solved, and that using the default settings can produce high quality solutions. Chapter six also shows how the different tabu search strategies are effectively integrated and co-operate with each other to search the solution space effectively. With regard to implementation tabu search restrictions are applied by means of *tabu constraints* which use the information stored in the memory structures to define the sub-problems and hence the region of the solution space where the next phase of the search should be focussed. The following section 7.2 now presents some suggestions for further work that may be carried out in order to develop the hybrid approach and its applications.

7.2 Further work

There are two aspects to the development of the TSBB algorithm. The first is the development of the algorithm itself to perhaps include additional strategies that may, after additional experimentation, provide scope for developing some aspects of the algorithm. The second aspect presented in this chapter gives consideration to the suitability and application of TSBB to extensions to the GAP problem.

7.2.1 Algorithm development

Whilst this research has produced a hybrid tabu search and branch and bound algorithm and performed considerable experimentation and testing on large and difficult sets of benchmark test problems to assess its effectiveness

for solving these types of problem it is thought that there remains some scope for developing additional and perhaps slightly more sophisticated memory structures and strategies that may enhance the algorithm in some way. Essentially TSBB uses both a short term recency memory and a longer term frequency memory in order to guide the search process towards regions of the solution space that may prove productive with regard to identifying new improved best solutions. This strategy seems to work quite effectively as has been discussed in chapter 6. With some of the largest and most tightly constrained problems however the regions defined by the sub-problems generated by reference to the memory structures can still prove extremely challenging to solve within reasonable time limits. One approach to tackling this situation might be to embed a separate heuristic to further guide the branch and bound search within the region of such a sub-problem. This could involve fixing still more assignments in a heuristic manner to narrow the sub-region further or, as is suggested by Glover in Tabu Search part II (Glover. 1990), using separate tabu memory structures at different levels of the search which can then be purged and discarded when the search moves up a layer. For example suppose that during the search process a sub-problem has been defined by fixing a large number of assignments according to the appropriate memory structure, this could be defined to be a top level problem. Prior to attempting to solve the sub-problem new tabu memory structures could be initialized that are considered to be one level down from the top level and apply only to the search of the region defined by the sub-problem. An attempt to solve this sub-problem during the specified time limit using either the branch and bound solver or some other heuristic approach similar to the short-term phase of TSBB, or intensification within the sub region can be made. When the search returns to the top level following its attempt to solve the sub-problem the memory structures created for searching the previous sub-problem can then be discarded. This approach might involve searching down to several layers which would require corresponding levels of tabu memory and so consideration of how such a layered approach might be implemented

could be of interest to this approach.

An alternative form of memory that may be of interest might be to record the frequency of assignments appearing in an elite solution list containing the best n solutions say. This information could then be used as a means of implementing an alternative intensification strategy for example.

The implementation of TSBB has used the default settings of Xpress-MP for the branch and bound aspect of the algorithm and it may be interesting to adjust some of these settings either dynamically within the search process or statically prior to attempting to solve problems of different types and sizes and perhaps the structure of the problem could be further exploited in doing so.

As well as adapting TSBB by means of differing strategies with regard to its implementation a further aspect that would be worthy of further investigation would be to attempt to apply TSBB to extensions of the GAP problem and this is discussed in the following section.

7.2.2 Application to extensions of GAP

The generalized assignment problem with special ordered sets of type 2 extends the standard GAP problem by replacing the binary variables $x_{ij} \in (0,1)$ with $0 \leq x_{ij} \leq 1$. With reference to GAP where each job must be allocated entirely to one agent the special ordered set of type 2 allows each job to be split between two agents providing that they are adjacent in some way. The special ordered sets of type 2 were first introduced by Beale and Tomlin (Beale, Tomlin 1970) and then further developed by Beale and Forrest (Beale and J. J. H. Forrest. 1976). The TSBB strategy of fixing the value of variables that satisfy feasibility and occur naturally in the solution of the LP relaxation could also be employed in the problem with these special ordered sets. This would still allow sub-regions of the solution space to be defined that could then be passed to the Xpress-MP solver to be solved. Since the Xpress solver

also has the capability to solve problems with special ordered sets of type two this would assist with ease of implementation. Implementation of the tabu structures, both short and long-term, could prove to be more challenging since it would now be necessary to consider pairs of variables to include or exclude during the short term phase and also the frequency with which jobs are split between adjacent variables in the longer term for intensification and diversification purposes.

The approach therefore would also prove interesting and challenging by extending the special ordered sets aspect. In the original GAP the assignment constraints have a right hand side value of 1 which can be split between two adjacent variables in GAPS2. If the right hand side of the assignment constraints were constrained to have value k say, where k is some integer greater than or equal to 1 then this might provide additional challenges.

7.2.3 Application to other non-GAP problems

One area of future research that may prove interesting and challenging would be to attempt to solve other types of non-GAP combinatorial optimization problems. It is evident from the research reported in this thesis that the GAP suits the approach taken in terms of the structure of the problem. This may not necessarily be the case with other types of problem and may therefore require additional strategies, for example with regard to determining the problem variables to fix in order to generate the sub-regions that represent the neighbourhoods and in terms of generating relevant tabu search memory structures that can be used in order to formulate suitable tabu constraints to be added to the problem.

Appendix: C code for Algorithm TSBB

```
/*******************************Include Files*************************/
#include<stdlib.h>
#include<math.h>
#include<limits.h>
#include<stdio.h>
#include "xprs.h"
#include<time.h>
#include<ctype.h>
#include<string.h>
/*********************************************************************/
void init_x(char *fname);
void XPRS_CC intsol(XPRSprob my_prob, void *my_object);
void XPRS_CC getRsol(XPRSprob my_prob, void *my_object);
void get_Total(void);
void alloc_mem(void);
void drop_x(char *fname);
void add_drop_con(void);
void solve_IPR(char *fname);
void fix(void);
void update_T(void);
void update_Ta(void);
void relax_T(void);
void add_T_con(void);
void add_Ta_con(void);
void get_x(void);
void update_TL(void);
void intensify(char *fname);
void diversify(char *fname);
void init_LP(char *fname);
/************************Global
Variables*************************************/
int m, n, iter=0, **T, *Tstatus, t, **Ta, *Tastatus, ta, *TL, s=0, tlim;
double  Total=0, subTotal=0, elapsed=0;
double *x, *xnew, *xlp, *xbest, z, znew, zlp, zbest;
clock_t start,end;
XPRSprob IP, LP;
char *resfile;
/*********************************************************************/
/*****************************Main*********************************************/
int main(int argc, char *argv[])
{
       int i, j, mipstatus, lpstatus;

       m=atoi(argv[1]);       n=atoi(argv[2]);
       resfile=argv[4];       t=atoi(argv[5]);       ta=atoi(argv[6]);
       tlim=atoi(argv[7]);

       start=clock(); //start the clock
       alloc_mem();   //allocate memory to arrays

       //generate initial solution
       init_x(argv[3]);
       init_LP(argv[3]);

       //check for integer solution
       XPRSgetintattrib(IP,XPRS_MIPSTATUS,&mipstatus);

       if(mipstatus==4)//if an integer solution has been found
       {
              //get the solution and objective function value
              XPRSgetsol(IP,x,NULL,NULL,NULL);
              XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&z);

              update_TL();   //update frequency memory
              ++s;           //increase number of integer solutions found

              XPRSdestroyprob(IP);
              do{//main loop
```

```
for(i=0;i<(int)(n/m);++i)//short term phase
{
        //drop assignments
        drop_x(argv[3]);

        //check feasibility
        XPRSgetintattrib(LP,XPRS_LPSTATUS,&lpstatus);
        if(lpstatus!=1)//relaxation infeasible
        {
                //overide tabu status
                do{
                        relax_T();
                        drop_x(argv[3]);
        XPRSgetintattrib(LP,XPRS_LPSTATUS,&lpstatus);


                }while(lpstatus!=1);
                //until relaxed solution found
        }
        //get relaxed solution and objective function
        XPRSgetsol(LP,xlp,NULL,NULL,NULL);
        XPRSgetdblattrib(LP,XPRS_LPOBJVAL,&zlp);

        //sve the basis
        XPRSwritebasis(LP,"","");

        //update tabu add memory
        update_Ta();
        //XPRSdestroyprob(LP);

        //generate and solve restricted integer problem
        solve_IPR(argv[3]);

        //check feasibility
        XPRSgetintattrib(IP,XPRS_MIPSTATUS,&mipstatus);
        if(mipstatus==4 || mipstatus==6)
        //integer solution found
        {
                //get solution and objective value
                XPRSgetsol(IP,xnew,NULL,NULL,NULL);
        XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&znew);

                get_Total();//update total time
                update_T();//update tabu drop memory


        //update current solution and objective value
                for(j=0;j<m*n;++j){ x[j]=xnew[j]; }
                z=znew;

                if(z<zbest)//if new best solution found
                {
                XPRSgetsol(IP,xbest,NULL,NULL,NULL);
                        zbest=z;
                }
                XPRSdestroyprob(IP);
                update_TL();//update frequency memory
                ++s;//update number of integer solutions
        }
        else//if no integer solution found
        {
                get_x();//get relaxed solution
                XPRSdestroyprob(IP);
        }
}//end of short term phase

intensify(argv[3]);//perform intensification

//check feasibility
XPRSgetintattrib(IP,XPRS_MIPSTATUS,&mipstatus);
if(mipstatus==4 || mipstatus==6)
```

```
                        //if new best solution found
                        {
                                //update best solution
                                XPRSgetsol(IP,xbest,NULL,NULL,NULL);
                                XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&zbest);

                                //record new solution
                                  XPRSgetsol(IP,xnew,NULL,NULL,NULL);
                                  XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&znew);
                                XPRSdestroyprob(IP);

                                update_T();//update tabu drop memory

                                //update current solution and objective value
                                for(j=0;j<m*n;++j){ x[j]=xnew[j]; }
                                z=znew;

                                update_TL();//update frequency memory
                                ++s;//update number of integer solutions
                        }
                        else    //if no new best solution found
                        {
                                XPRSdestroyprob(IP);
                                diversify(argv[3]);//perform diversification

                                //check feasibility
                                XPRSgetintattrib(IP,XPRS_MIPSTATUS,&mipstatus);
                                if(mipstatus==4 || mipstatus==6)
                                {//if diversified solution found

                                        //update new and current solutions
                                        XPRSgetsol(IP,xnew,NULL,NULL,NULL);
                                  XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&znew);
                                        update_T();
                                        for(j=0;j<m*n;++j){ x[j]=xnew[j]; }
                                        z=znew;
                                }
                                  XPRSdestroyprob(IP);
                        }
                        get_Total();//update elapsed time
                }while(Total<tlim);//end of main loop
        }
        else if(mipstatus==5)//if problem is integer infeasible
        {
                printf("problem is integer infeasible\n");
        }
        else if(mipstatus==6)//if solution is optimal
        {
                printf("optimal integer solution found\n");
        }
        XPRSdestroyprob(LP);

        //free memory
        XPRSfree();     free(x);        free(xlp);
        free(xnew);     free(xbest);    free(T);        free(Ta);
        free(TL);       free(Tstatus);  free(Tastatus);
      return 1;
}
/*****************************************************************************/
void solve_IPR(char *fname)
{
        int status;

        //initialize problem
        XPRScreateprob(&IP);
        XPRSsetintcontrol(IP,XPRS_OUTPUTLOG,0);
        XPRSreadprob(IP,fname,"l");
        XPRSsetintcontrol(IP,XPRS_SOLUTIONFILE,0);
        XPRSsetintcontrol(IP,XPRS_PRESOLVE,0);
        XPRSsetintcontrol(IP,XPRS_MIPLOG,-100);
```

```
        //function to record integer solution
        XPRSsetcbintsol(IP,getRsol,NULL);
        //set time limit
        XPRSsetintcontrol(IP,XPRS_MAXTIME,-60);
        //fix assignments
        fix();
        //add tabu constraints
        add_Ta_con();
        //solve
        XPRSreadbasis(IP,fname,"");
        XPRSminim(IP,"g");
}
/*****************************************************************************/
void alloc_mem(void)
{
        int i;
        FILE *fp;

        x=calloc(m*n,sizeof(double));
        xnew=calloc(m*n,sizeof(double));
        xlp=calloc(m*n,sizeof(double));
        xbest=calloc(m*n,sizeof(double));
        T=malloc(sizeof(int*)*t);
        for(i=0;i<t;++i)
        {
                T[i]=calloc(1,sizeof(int));
        }
        Tstatus=calloc(t,sizeof(int));

        Ta=malloc(sizeof(int*)*ta);
        for(i=0;i<ta;++i)
        {
                Ta[i]=calloc(1,sizeof(int));
        }
        Tastatus=calloc(ta,sizeof(int));

         TL=calloc(m*n,sizeof(int));

        if((fp=fopen(resfile,"a"))==NULL)
        {
                printf("Cannot open file!! \n");
                exit(1);
        }
        fprintf(fp,"t=%d\tta=%d\n",t,ta);
        fclose(fp);
         zbest=XPRS_PLUSINFINITY;
}
/*****************************************************************************/
void update_TL(void)
{
        int i;

        for(i=0;i<m*n;++i)
        {
                if(x[i]>1.0-1.0E-09)
                {
                        ++TL[i];
                }
        }
}
/*****************************************************************************/
void diversify(char *fname)
{
        int i, j=0, *mclind, mstart[2],status;
        double *dmatval, rhs[1], f=0.0;
        char qrtype[1];

        //array to store column indices
        mclind=calloc(m*n,sizeof(int));
        //array to store matrix values
        dmatval=calloc(m*n,sizeof(double));
```

```
        //initialize problem
        XPRScreateprob(&IP);
        XPRSsetintcontrol(IP,XPRS_OUTPUTLOG,1);
        XPRSreadprob(IP,fname,"l");
        XPRSsetintcontrol(IP,XPRS_PRESOLVE,0);
         XPRSsetintcontrol(IP,XPRS_MAXTIME,1);

        //analyse frequency memory
        do{
                j=0;
                for(i=0;i<m*n;++i)
                {
                        if(TL[i]<=f*s)
                        {
                                mclind[j]=i;
                                dmatval[j]=1.0;
                                ++j;
                        }
                }
                f=f+0.1;
        }while(j<m);

        //add diversifying constraint
        if(j>0)
        {
               mstart[0]=0;   mstart[1]=j;
               rhs[0]=m;
               qrtype[0]='G';
            XPRSaddrows(IP,1,j,qrtype,rhs,NULL,mstart,mclind,dmatval);
        }
        XPRSminim(IP,"g");//solve

        free(mclind);   free(dmatval);
}
/*************************************************************************/
void intensify(char *fname)
{
        int i, j=0, *mindex, status;
        double *bnd, f=1.0;
        char *qbtype;
          FILE *fp;

        //initialize arrays
        mindex=calloc(n,sizeof(int));
        bnd=calloc(n,sizeof(double));
        qbtype=malloc(sizeof(char)*n);

        //initialize problem
        XPRScreateprob(&IP);
        XPRSsetintcontrol(IP,XPRS_OUTPUTLOG,1);
        XPRSreadprob(IP,fname,"l");
        XPRSsetintcontrol(IP,XPRS_PRESOLVE,0);
         XPRSsetintcontrol(IP,XPRS_MIPLOG,-1000);
        XPRSsetdblcontrol(IP,XPRS_MIPABSCUTOFF,zbest-0.9999);
        XPRSsetintcontrol(IP,XPRS_MAXTIME,-120);
        //function to record integer solutions
        XPRSsetcbintsol(IP,intsol,NULL);

        //identify bounds to be fixed
        do{
                j=0;
                for(i=0;i<m*n;++i)
                {
                        if(xbest[i]>1.0-1.0E-09 && TL[i]>=f*s)
                        {
                                mindex[j]=i;
                                bnd[j]=1.0;
                                qbtype[j]='B';
                                ++j;
                        }
```

```
            }
            f=f-0.1;
        }while(f>=0.1 && j<=0);

        //adjust size of arrays
        mindex=realloc(mindex,sizeof(int)*j);
        bnd=realloc(bnd,sizeof(double)*j);
        qbtype=realloc(qbtype,sizeof(char)*j);

        //fix bounds
        if(j>0)
        {
                XPRSchgbounds(IP,j,mindex,qbtype,bnd);
        }

        if((fp=fopen(resfile,"a"))==NULL)
        {
                printf("Cannot open file!!(function intsol) \n");
                exit(1);
        }
        fprintf(fp,"int,%d\n",j);
        fclose(fp);
        XPRSminim(IP,"g");//solve

        //check feasibility
        XPRSgetintattrib(IP,XPRS_MIPSTATUS,&status);
        if(status==4 || status==6)//if new best solution found
        {
                //update current and best solutions
                XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&z);
                XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&zbest);
                XPRSgetsol(IP,x,NULL,NULL,NULL);
                XPRSgetsol(IP,xbest,NULL,NULL,NULL);
        }

        free(mindex);   free(bnd);      free(qbtype);
}
/*************************************************************************/
void get_x(void)
{
        int i;

        for(i=0;i<m*n;++i)
        {
                if(x[i]>1.0-1.0E-09 && xlp[i]>1.0-1.0E-09)
                {
                        x[i]=1.0;
                }
                else
                {
                        x[i]=0.0;
                }
        }
}
/*************************************************************************/
void relax_T(void)
{
        int i;

        for(i=t-1;i>=0;--i)
        {
                if(T[i][0]>0 && Tstatus[i]>1)
                {
                        --Tstatus[i];
                        break;
                }
                else if(T[i][0]>0 && Tstatus[i]==1)
                {
                        T[i]=realloc(T[i],sizeof(int));
                        T[i][0]=0;
                        Tstatus[i]=0;
```

```
                        break;
                }
        }
}
/*****************************************************************************/
void update_T(void)
{
        int i, j;

        for(i=t-1;i>0;--i)
        {
                T[i][0]=T[i-1][0];
                T[i]=realloc(T[i],sizeof(int)*T[i][0]+1);
                for(j=1;j<=T[i][0];++j)
                {
                        T[i][j]=T[i-1][j];
                }
                Tstatus[i]=Tstatus[i-1];
        }
        T[0][0]=0;
        for(i=0;i<m*n;++i)
        {
                if(xnew[i]>=1.0-1.0E-09 && x[i]<1.0-1.0E-09)
                {
                        ++T[0][0];
                        T[0]=realloc(T[0],sizeof(int)*(T[0][0])+1);
                        T[0][T[0][0]]=i;
                }
        }
        Tstatus[0]=T[0][0];
}
/*****************************************************************************/
void update_Ta(void)
{
        int i, j;

        for(i=ta-1;i>0;--i)
        {
                Ta[i][0]=Ta[i-1][0];
                Ta[i]=realloc(Ta[i],sizeof(int)*Ta[i][0]+1);
                for(j=1;j<=Ta[i][0];++j)
                {
                        Ta[i][j]=Ta[i-1][j];
                }
                Tastatus[i]=Tastatus[i-1];
        }
        Ta[0][0]=0;
        for(i=0;i<m*n;++i)
        {
                if(x[i]>1.0-1.0E-09 && xlp[i]<=1.0-1.0E-09)
                {
                        ++Ta[0][0];
                        Ta[0]=realloc(Ta[0],sizeof(int)*(Ta[0][0])+1);
                        Ta[0][Ta[0][0]]=i;
                }
        }
        Tastatus[0]=Ta[0][0];
}
/*****************************************************************************/
void fix(void)
{
        int i, j=0, *mindex;
        double *bnd;
        char *qbtype;
        FILE *fp;

        //intialize arrays
        mindex=calloc(n,sizeof(int));
        bnd=calloc(n,sizeof(double));
        qbtype=malloc(sizeof(char)*n);
```

```
        //identify variables to be fixed
        for(i=0;i<m*n;++i)
        {
                if(x[i]>1.0-1.0E-09 && xlp[i]>1.0-1.0E-09)
                {
                        mindex[j]=i;
                        bnd[j]=1.0;
                        qbtype[j]='B';
                        ++j;
                }
        }

        //resize arrays
        mindex=realloc(mindex,sizeof(int)*j);
        bnd=realloc(bnd,sizeof(double)*j);
        qbtype=realloc(qbtype,sizeof(char)*j);

        //fix bounds on variables
        if(j>0)
        {
                XPRSchgbounds(IP,j,mindex,qbtype,bnd);
        }

        if((fp=fopen(resfile,"a"))==NULL)
        {
                printf("Cannot open file!!(function intsol) \n");
                exit(1);
        }
        fprintf(fp,"st,%d\n",j);
        fclose(fp);
        free(mindex); free(bnd);      free(qbtype);
}
/*****************************************************************************/
void drop_x(char *fname)
{
        int i, rows, mindex[1];

        XPRSgetintattrib(LP,XPRS_ROWS,&rows);
        if(rows>m+n)
        {
                for(i=m+n;i<rows;++i)
                {
                        mindex[0]=i;
                        XPRSdelrows(LP,1,mindex);
                }
        }
        /*XPRScreateprob(&LP);
          XPRSsetintcontrol(LP,XPRS_OUTPUTLOG,1);
        if(XPRSreadprob(LP,fname,"l")!=0)
        {
                exit(1);
        }
        XPRSsetintcontrol(LP,XPRS_PRESOLVE,0);*/
        //XPRSsetdblcontrol(LP,XPRS_MIPABSCUTOFF,zbest-0.9999);

        //add the drop constraint
        add_drop_con();

        //add the tabu constraint
        add_T_con();
        //solve
        XPRSminim(LP,"");
}
/*****************************************************************************/
void init_LP(char *fname)
{
        //initialize the problem
        XPRScreateprob(&LP);
        XPRSsetintcontrol(LP,XPRS_OUTPUTLOG,0);
        XPRSreadprob(LP,fname,"l");
        XPRSsetintcontrol(LP,XPRS_PRESOLVE,0);
```

```
        //solve
        XPRSminim(LP,"");
}
/*************************************************************************/
void add_Ta_con(void)
{
        int i, j, k=0, mstart[2], *mclind;
        double *dmatval, rhs[1];
        char qrtype[1];

        //initialize arrays
        mclind=calloc(n,sizeof(int));
        dmatval=calloc(n,sizeof(double));

        //add constraint
        for(i=0;i<ta;++i)
        {
                for(j=1;j<=Ta[i][0];++j)
                {
                        mclind[k]=Ta[i][j];
                        dmatval[k]=1.0;
                        ++k;
                }
                if(k>0)
                {
                        mstart[0]=0;    mstart[1]=k;
                        rhs[0]=(double)(Tastatus[i]-1);
                        qrtype[0]='L';
                    XPRSaddrows(IP,1,k,qrtype,rhs,NULL,mstart,mclind,dmatval);
                }
                k=0;
        }

        free(mclind);   free(dmatval);
}
/*************************************************************************/
void add_T_con(void)
{
        int i, j, k=0, mstart[2], *mclind;
        double *dmatval, rhs[1];
        char qrtype[1];

        //initialize arrays
        mclind=calloc(n,sizeof(int));
        dmatval=calloc(n,sizeof(double));

        //add constraint
        for(i=0;i<t;++i)
        {
                for(j=1;j<=T[i][0];++j)
                {
                        mclind[k]=T[i][j];
                        dmatval[k]=1.0;
                        ++k;
                }
                if(k>0)
                {
                        mstart[0]=0;   mstart[1]=k;
                        rhs[0]=(double)Tstatus[i];
                        qrtype[0]='G';

        XPRSaddrows(LP,1,k,qrtype,rhs,NULL,mstart,mclind,dmatval);
                }
                k=0;
        }

        free(mclind);  free(dmatval);
}
/*************************************************************************/
void add_drop_con(void)
{
```

```
        int i, j=0, *mclind, mstart[2];
        double *dmatval, rhs[1];
        char qrtype[1];

        //initialize arrays
        mclind=calloc(n,sizeof(int));
        dmatval=calloc(n,sizeof(double));

        //add constraint
        for(i=0;i<m*n;++i)
        {
                if(x[i]>1.0-1.0E-09)
                {
                        mclind[j]=i;
                        dmatval[j]=1.0;
                        ++j;
                }
        }
        if(j>0)
        {
                mstart[0]=0;    mstart[1]=j;
                rhs[0]=j-1;
                qrtype[0]='L';
                XPRSaddrows(LP,1,j,qrtype,rhs,NULL,mstart,mclind,dmatval);
        }

        free(mclind);  free(dmatval);
}
/*****************************************************************************/
void get_Total(void)
{
        end=clock();//(double)tcurr)/CLOCKS_PER_SEC;

        if(end<start)
        {
                elapsed=(double)((INT_MAX-start)+(INT_MAX+end))/CLOCKS_PER_SEC;
        }
        else
        {
                elapsed=(double)(end-start)/CLOCKS_PER_SEC;
        }
        Total=Total+elapsed;
      start=clock();//((double)tcurr)/CLOCKS_PER_SEC;//start the clock
}
/*****************************************************************************/
void init_x(char *fname)
{
        int i, j, status;

        //initialize the solver
        XPRSinit(NULL);

        //initialize the problem
        XPRScreateprob(&IP);
        XPRSsetintcontrol(IP,XPRS_OUTPUTLOG,1);
        XPRSreadprob(IP,fname,"l");
        XPRSsetintcontrol(IP,XPRS_PRESOLVE,0);
          XPRSsetintcontrol(IP,XPRS_MIPLOG,-1000);
        /*XPRSsetcbintsol(IP,intsol,NULL);*/
          XPRSsetintcontrol(IP,XPRS_MAXTIME,1);
        XPRSsetcbintsol(IP,intsol,NULL);//record integer solution

        //solve
        XPRSminim(IP,"g");

        //check feasibility
        XPRSgetintattrib(IP,XPRS_MIPSTATUS,&status);
        if(status==4 || status==6)
        {//if feasible solution found
                //update solution
                XPRSgetsol(IP,xbest,NULL,NULL,NULL);
```

```
                XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&zbest);
        }
}
/*****************************************************************************/
void XPRS_CC getRsol(XPRSprob my_prob, void *my_object)
{
        FILE *fp;

         XPRSgetsol(IP,xnew,NULL,NULL,NULL);
        XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&znew);

        get_Total();
        if((fp=fopen(resfile,"a"))==NULL)
        {
                printf("Cannot open results file(function Rsol)!! \n");
                exit(1);
        }
        if(znew<zbest)
        {
                fprintf(fp,"%f\t%d\tsterm\n",Total,(int)znew);
        }
        fclose(fp);
}
/*****************************************************************************/
void XPRS_CC intsol(XPRSprob my_prob, void *my_object)
{
        FILE *fp;
        double obj;

        XPRSgetdblattrib(IP,XPRS_MIPOBJVAL,&obj);

        get_Total();
        if((fp=fopen(resfile,"a"))==NULL)
        {
                printf("Cannot open file!!(function intsol) \n");
                exit(1);
        }
        if(obj<zbest)
        {
                fprintf(fp,"%f\t%d\tintensify\n",Total,(int)obj);
        }
        fclose(fp);
}
/*****************************************************************************/
```

Bibliography

Amini,M.M. and Racer, M., 1995. A hybrid heuristic for the generalized assignment problem. *European Journal of Operational Research,* **87**, pp. 343-348.

Amini,M.M. and Racer, M., 1994. A rigorous computational comparison of alternative solution methods for the generalised assignment problem. *Management Science,* **40**, pp. 868-890.

Balachandran,V., 1976. An integer generalized transportation model for optimal job assignment in computer networks. *Operations Research,* **24**(4), pp. 868-890.

Beale,E.M.L. and Forrest, J.J.H., 1976. Global Optimization using special ordered sets. *Mathematical Programming,* **10**(1), pp. 52-69.

Beale, E.M.L. and Tomlin, J.A., 1970. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. J. LAWRENCE, ed. In: *5th International Conference on Operational Research.* 1970, Tavistock Publications pp447-454.

Beasley,J.E. and Chu, P.C., 1997. A Genetic Algorithm for the generalised assignment problem. *Computers and Operations Research,* **27**, pp. 17-23.

Budenbender,K., Grunert, T. and Sebastian, H., 2000. A Hybrid Tabu Search/Branch-and-Bound Algorithm for the Direct Flight Network Design Problem. *TRANSPORTATION SCIENCE,* **34**(4), pp. 364-380.

Cattrysse,D.G., Degraeve, Z. and Tistaert, J., 1998. Solving the generalized assignment problem using polyhedral results. *European Journal of Operational Research,* **108**, pp. 618-628.

Dammeyer, F. and Voss, S., 1993 . Dynamic Tabu list management using the reverse elimination method. *Annals of Operations Research,* **41**, pp. 31-46.

Danna,E., Rothberg, E. and Le Pape, C., 2005. Exploring relaxation induced neighbourhoods to improve MIP solutions. *Mathematical Programming,* **A**(102), pp. 71-90.

Diaz,J.A. and Fernandez, E., 2001. A Tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research,* **132**, pp. 22-38.

Dorigo, M. and Di Caro, G., 1999. The ant colony optimization meta-heuristic. In: F. GLOVER, ed, *New Ideas in Optimization.* McGraw-Hill, .

Fischetti,M. and Lodi, A., 2003. Local Branching. *Mathematical Programming,* **B**(98), pp. 23-47.

Fisher,M.L. and Jaikumar, R., 1981. A Generalized Assignment Heuristic for Vehicle Routing. *Networks,* **11**, pp. 109-124.

Fisher,M.L., Jaikumar, R. and Van Wassenhove, L.N., 1986. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Management Science,* **32**(9), pp. 1095.

Foulds,L.R. and Wilson, J.M., 1997. A variation of the generalized assignment problem arising in the New Zealand dairy industry. *Annals of Operations Research,* **69**, pp. 105-114.

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 Francisco: .

Gavish,B. and Pirkul, H., 1985. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming,* **31**, pp. 78-105.

Gilmore,P.C. and Gomory, R.E., 1966. The theory and computation of knapsack functions. *Operations Research,* **14**(5), pp. 1045-1074.

Glover, F., 1997. A template for scatter search and path relinking. In: J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, eds, *Lecture Notes in Computer Science.* 1363 edn. Springer, pp. 13-54.

Glover,F., 1989. Tabu Search-PartI. *ORSA Journal on Computing,* **1**(3), pp. 190-205.

Glover,F., 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research,* **13**(5), pp. 533-549.

Glover,F., 1977. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences,* **8**(1), pp. 156-166.

Glover,F., 1990. Tabu Search-PartII. *ORSA Journal on Computing,* **2**(1), pp. 4-32.

Glover, F. and Laguna, M., 1997. Tabu search. Kluwer Academic Publishers. Boston: .

Guignard,M. and Rosenwein, M.B., 1989. An improved dual based algorithm for the generalized assignment problem. *Operations Research,* **37**(4), pp. 658-663.

Haddadi,S. and Ouzia, H., 2004. Effective algorithm and heuristic for the generalized assignment problem. *European Journal of Operational Research,* **153**, pp. 184-190.

Higgins,A.J., 2001. A dynamic Tabu search for large-scale generalised assignment problems. *Computers and Operations Research,* **28**, pp. 1039-1048.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. 1st edn. MIT. Michigan: .

Karmarkar,N., 1984. A new polynomial-time algorithm for linear programming. *Combinatorica,* **4**(4), pp. 373-395.

Kellerer, H., Pferschy, U. and Pisinger, D., 2004. Knapsack Problems. 1st edn. Springer-Verlag. Berlin: .

Kolesar,P.J., 1967. A branch and bound algorithm for the knapsack problem. *Management Science,* **13**(9), pp. 723-735.

Laguna,M., Kelly, J.P., Gonzalez-Verlade, J.L. and Glover, F., 1995. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research,* **82**, pp. 176-189.

Land,A.H. and Doig, A.G., 1960. An Automatic Method of Solving Discrete programming Problems. *Econometrica,* **28**(3), pp. 497-520.

Løkketangen, A. and Glover, F. Probabilistic move selection in Tabu Search for zero-one mixed integer programming problems. In I.H. Osman & J.P. Kelly, eds, *Meta-Heuristics: Theory & Applications*, 1996, 467-487, Kluwer Academic Publishers.

Lourenco,H.R. and Serra, D., 2002. Adaptive search heuristics for the generalized assignment problem. *Mathware and Soft Computing,* **9**, pp. 209-234.

Martello, S. and Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. Wiley. Chichester: .

Martello, S. and Toth, P., 1981. An algorithm for the Generalized Assignment Problem. J.P. Brans, ed. In: *Ninth IFORSInternational Conferenceon Operational Research,* July 1981 1981, North-Holland pp589-603.

Martello,S. and Toth, P., 1980. Solution of the zero-one multiple knapsack problem. *European Journal of Operational Research,* **4**(4), pp. 276-283.

Martello,S. and Toth, P., 1977. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research,* **1**(3), pp. 169-175.

Nauss,R.M., 2003. Solving the Generalised Assignment Problem: An Optimizing and Heuristic Approach. *INFORMS Journal on Computing,* **15**(3), pp. 249-266.

Nemhauser,G.L. and Ullmann, Z., 1969. Discrete dynamic programming and capital allocation. *Management Science,* **15**(9), pp. 494-505.

Nemhauser, G.L. and Wolsey, L.A., 1998. Integer and combinatorial optimisation. Wiley. Chichester: .

Osman,I.H., 1995. Heuristics for the Generalised assignment problem: Simulated Annealing and Tabu Search approaches. *OR Spektrum,* **17**, pp. 211-215.

Pisinger,D., 1999. An exact algorithm for largemultiple knapsack problems. *European Journal of Operational Research,* **114**(3), pp. 528-541.

Puchinger, G. and Raidl, G., 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation,* 2005, Springer pp41-53.

Raidl, G.R. and Feltl, H., 2004. An improved hybrid genetic algorithm for the generalized assignment problem, . In: Haddadd, H.M. et al., ed. *Proceedings of the 2003 ACM Symposium on Applied Computing,* 2003 2004, ACM Press pp990-995.

Ross,G.T. and Soland, P., 1977. Modelling Facility Location problems as Generalized Assignment Problems. *Management Science,* **24**(3), pp. 354-357.

Ross,G.T. and Soland, P., 1975. A Branch and Bound based algorithm for the generalised assignment problem. *Mathematical programming,* **8**, pp. 91-103.

Sahni,S., 1975. Approximate algorithms for the 0-1 knapsack problem. *Journal of the ACM,* **22**, pp. 115-134.

Tai-Hsi Wu., Jinn-Yi Yeh. and Yu Ru Syau., 2004. ATabu Search Approach To The Generalised Assignment Problem. *Journal of the Chinese Institute of Industrial Engineers.,* **21**(3), pp. 301-311.

Trick,M.A., 1992. A Linear Relaxation Heuristic for the Generalized Assignment Problem. *Naval Research Logistics,* **39**, pp. 137-151.

Weingartner,H.M. and Ness, D.N., 1967. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research,* **15**(1), pp. 83-103.

Wilson,J.M., 1997. A Genetic Algorithm for the generalised assignment problem. *Journal of the Operational Research Society,* **48**, pp. 804-809.

Yagiura,M., Ibaraki, T. and Glover, F., 2006. A path relinking approach with ejections chains for the generalized assignment problem. *European Journal of Operational Research,* **169**(2), pp. 548-569.

Yagiura,M., Ibaraki, T. and Glover, F., 2004. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing,* **16**(2), pp. 131-151.

Yagiura,M., Ibaraki, T. and Glover, F., A Path Relinking Approach for the generalized assignment problem in: *Proceedings of the International Symposium on Scheduling,* Japan, June 4-6, 2002, pp. 105-108.

Yagiura,M., Yamaguchi, T. and Ibaraki, T., 1998. A variable depth search algorithm with branching search for the generalised assignment problem. *Optimisation Methods & Software,* **10**, pp. 419-441.

Yagiura, M., Yamaguchi, T. and Ibaraki, T., 1999.
A variable depth search algorithm for the generalized assignment problem. In: S. Voss, S. Martello, I.H. Osman and C. Roucairol, eds, *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Boston: Kluwer Academic Publishers., pp. 459-471.

Xpress-MP, Dash Optimization Ltd., Blisworth, Northamptonshire, United Kingdom.