# A GENERAL MODELLING SYSTEM AND META-HEURISTIC BASED SOLVER FOR COMBINATORIAL OPTIMISATION PROBLEMS 

## BY MARCUS CHRISTIAN RANDALL, BSc, BBus(Hons)

A Dissertation<br>submitted in fulfillment of the requirements of the degree of Doctor of Philosophy<br>for the School of Environmental and Applied Science,<br>Faculty of Environmental Sciences,<br>Griffith University

5 May, 1999

## Synopsis

There are many real world assignment, scheduling and planning tasks which can be classified as combinatorial optimisation problems (COPs). These are usually formulated as a mathematical problem of minimising or maximising some cost function subject to a number of constraints. Usually, such problems are NP hard, and thus, whilst it is possible to find exact solutions to specific problems, in general only approximate solutions can be found. There are many algorithms that have been proposed for finding approximate solutions to COPs, ranging from special purpose heuristics to general search meta-heuristics such as simulated annealing and tabu search.

General meta-heuristic algorithms like simulated annealing have been applied to a wide range of problems. In most cases, the designer must choose an appropriate data structure and a set of local operators that define a search neighbourhood. The variability in representation techniques, and suitable neighbourhood transition operators, has meant that it is usually necessary to develop new code for each problem. Toolkits like the one developed by Ingber's Adaptive Simulated Annealing (Ingber 1993, 1996) have been applied to assist rapid prototyping of simulated annealing codes, however, these still require the development of new programs for each type of problem. There have been very few attempts to develop a general meta-heuristic solver, with the notable exception being Connolly's General Purpose Simulated Annealing (Connolly 1992).

In this research, a general meta-heuristic based system is presented that is suitable for a wide range of COPs. The main goal of this work is to build an environment in which it is possible to specify a range of COPs using an algebraic formulation, and to produce a tailored solver automatically. This removes the need for the development of specific software, allowing very rapid prototyping. Similar techniques have been available for linear programming based solvers for some years in the form of the GAMS (General Algebraic Modelling System) (Brooke, Kendrick, Meeraus and Raman 1997) and AMPL (Fourer, Gay and Kernighan 1993) interfaces. The new system is based on a novel linked list data structure rather than the more conventional vector notation due to the natural mapping between COPs and lists. In addition, the modelling system is found to be very suitable for processing by metaheuristic search algorithms as it allows the direct application of common local search operators.

A general solver is built that is based on the linked list modelling system. This system is capable of using meta-heuristic search engines such as greedy search, tabu search and simulated annealing. A number of implementation issues such as generating initial solutions, choosing and invoking appropriate local search transition operators and producing suitable incremental cost expressions, are considered. As such, the system can been seen as a good test-bench for model prototypers and those who wish to test various meta-heuristic implementations in a standard way. However, it is not meant as a replacement or substitute for efficient special purpose search algorithms.

The solver shows good performance on a wide range of problems, frequently reaching the optimal and best-known solutions. Where this is not the case, solutions within a few percent deviation are produced. Performance is dependent on the chosen transition operators and the frequency with which each is applied. To a lesser extent, the performance of this implementation is influenced by runtime parameters of the meta-heuristic search engine.

## Table of Contents

Chapter 1: Introduction ..... 1
1.1 Introduction .....  1
1.2 Scope ..... 2
1.3 Aim ..... 3
Chapter 2: A Review of Modelling Methods and Solution Techniques ..... 5
2.1 A Review of Modelling Systems for COPs ..... 5
2.1.1 Integer Linear Programming .....  .5
2.1.2 Connectionist Models ..... 6
2.1.3 Graph-Theoretic Techniques .....  8
2.1.4 Constraint Programming .....  9
2.1.5 A Compacted Integer Vector Approach ..... 10
2.2 Search Algorithms for Combinatorial Optimisation Problems ..... 10
2.2.1 Exact Search Algorithms ..... 10
2.2.1.1 Operations Research Techniques ..... 11
2.2.1.2 A* ..... 11
2.2.1.3 Constraint Programming ..... 12
2.2.2 Heuristics ..... 14
2.2.3 Meta-heuristics ..... 15
2.2.3.1 Greedy Search ..... 17
2.2.3.2 Tabu Search ..... 18
2.2.3.3 Simulated Annealing ..... 20
2.2.3.4 Greedy Randomised Search Procedures (GRASP) ..... 22
2.2.3.5 Genetic Algorithms ..... 23
2.3 General Purpose Meta-heuristic Solvers ..... 24
2.3.1 Algebraic Modelling Approach ..... 25
2.3.1.1 GPSIMAN ..... 25
2.3.2 Code Segment Approach ..... 27
2.3.2.1 Johnson's et al. Generic SA ..... 27
2.3.2.2 ASA ..... 27
2.3.2.3 Skeleton GA Implementations ..... 28
2.4 Summary ..... 28
Chapter 3: A Modelling System Based on Linked Lists ..... 31
3.1 Motivation ..... 31
3.2 Using Dynamic Data Structures to Represent COPs ..... 31
3.3 Expressing COPs Using a List Based Notation ..... 33
3.3.1 General Model ..... 33
3.3.2 Illustrated Examples of the List Modelling System ..... 35
3.4 Search Over Linked Lists ..... 45
3.4.1 List-Based Local Search Transition Operators ..... 46
3.4.1.1 Enforcing Constraints ..... 49
3.4.1.2 Applying the Transition Operators ..... 50
3.4.1.2.1 The Adaptive Probability Model. ..... 51
3.4.2 Calculating Incremental Costs ..... 52
3.5 Summary .....  54
Chapter 4: Implementation of a General Solver. ..... 56
4.1 System Architecture ..... 56
4.2 Algebraic Modelling Language ..... 58
4.3 Feasibility Maintenance and Restoration ..... 61
4.4 Probabilistic Candidate List Strategy ..... 62
4.5 Incremental Cost Templates ..... 63
4.6 Search Engine Implementation. ..... 69
4.6.1 SA ..... 69
4.6.2 GS ..... 70
4.6.3 TS ..... 71
4.6.3.1 Sequential ..... 71
4.6.3.2 A Parallel Implementation of the Solver ..... 71
4.6.3.2.1 A Review of Parallelisation Strategies for Tabu Search ..... 71
4.6.3.2.2 Division of the Neighbourhood ..... 73
4.6.3.2.3 The Parallel Algorithm ..... 74
4.7 Generating an Initial Solution ..... 75
4.8 Practical Limitations of the System. ..... 77
4.9 Summary ..... 78
Chapter 5: Methodology and Results ..... 80
5.1 Experimental Overview and Rationale. ..... 80
5.2 Problems Classes and Instances ..... 81
5.3 Parameter Settings ..... 83
5.4 Termination Conditions. ..... 85
5.5 Benchmarking Against Other Software Packages ..... 85
5.6 Feasibility Restoration Runs ..... 86
5.7 Parallel Runs ..... 86
5.8 Reporting the Results ..... 87
5.8.1 Numerical Data ..... 88
5.8.2 Qualitative Analysis ..... 88
5.8.3 Statistical Analysis ..... 88
5.8.3.1 The Effect of the Transition Operator Sets and Engine-specific Parameters. ..... 88
5.8.3.2 Overall Comparison. ..... 89
5.9 Results ..... 89
5.9.1 Standard Runs ..... 89
5.9.2 Extended Runs ..... 92
5.9.3 Other Software ..... 97
5.9.4 Feasibility Restoration Runs ..... 102
5.9.5 Parallel Runs ..... 104
5.10 Performance Analysis ..... 111
5.10.1 The Effect of the Transition Operator Sets and Engine-specific Parameters ..... 111
5.10.2 Overall Comparison ..... 113
5.10.3 Runtime Differences between the Various Solver Packages ..... 119
5.11 Summary ..... 122
Chapter 6: Conclusions and Further Work ..... 124
6.1 Conclusions ..... 124
6.2 Further Work ..... 126
6.2.1 Direct Hardware Implementation ..... 127
6.2.2 Alternative Dynamic Data Structures ..... 128
6.2.3 Alternative Search Engines ..... 128
6.3 Achievements and Significance ..... 128
Bibliography ..... 131
Glossary of Terms and Acronyms ..... 139
Appendix A: Standard Results for the SA, TS and GS engines ..... 144
Appendix B: List Formulations of Common COPs ..... 198
Appendix C: 0-1 ILP formulations of the Test Problems ..... 206
Appendix D: Algebraic Modelling Language User Manual ..... 211
Appendix E: Overview of the C Problem Description Files ..... 219
Appendix F: "A Simulated Annealing Code for General Integer Linear Programs" ..... 227
Appendix G: Papers Arising from this Study ..... 249

## List of Figures

Figure 1: A graph-theoretic representation of a 5 city TSP. The cities are given by the set $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}$, E\} and form the vertices of the graph. The edge weights represent the distance between each pair of cities. 9
Figure 2: X and Y have the domains of $2,4,5$ and $4,6,8$ respectively. This is the entire search tree. This problem is a demonstration example reproduced from Little and Darby-Dowman (1995, Figure 1,
$\qquad$

Figure 3: Once the constraint $\mathrm{X} \geq \mathrm{Y}$ is added, the search tree is reduced (pruned). This problem is a demonstration example reproduced from Little and Darby-Dowman (1995, Figure 2, p. 3)......... 13
Figure 4: SA implementation choices made by Johnson et al. (1991a, 1991b). Reproduced from
Johnson et al. (1991a, Figure 5, p. 869)........................................................................................ 27
Figure 5: A list and tree structure. A square box is used to denote an element/node while a line is an edge. The traversal of the list structure is indicated by the arrows............................................... 32
Figure 6: A linked list representation of a GAP.................................................................................... 36
Figure 7: The move operator. ................................................................................................................ 46
Figure 8: The swap operator. ................................................................................................................ 47
Figure 9: The inversion operator.......................................................................................................... 47
Figure 10: The reposition operator. ...................................................................................................... 47
Figure 11: The add operator................................................................................................................... 48
Figure 12: The drop operator............................................................................................................... 48
Figure 13: The change operator. ........................................................................................................... 48
Figure 14: An architectural representation of the general solver. ......................................................... 57
Figure 15: Input file representing a GAP............................................................................................... 59
Figure 16: BNF of the algebraic modelling language. .......................................................................... 60
Figure 17: A graphical representation of the SA reheating schedule. .................................................... 69
Figure 18: Block diagram of the communications between the master processor and the slave
processors for the parallel TS algorithm. .................................................................................. 74
Figure 19: Parallel speedup graph for the GPP.................................................................................... 107
Figure 20: Parallel efficiency graph for the GPP. ................................................................................ 107
Figure 21: Parallel speedup graph for the GAP. ................................................................................. 107
Figure 22: Parallel efficiency graph for the GAP................................................................................. 108
Figure 23: Parallel speedup graph for the QAP. ................................................................................. 108
Figure 24: Parallel efficiency graph for the QAP............................................................................... 108
Figure 25: Parallel speedup graph for the BIN. .................................................................................. 109
Figure 26: Parallel efficiency graph for the BIN.................................................................................. 109
Figure 27: Parallel speedup graph for the TSP. .................................................................................. 109
Figure 28: Parallel efficiency graph for the TSP................................................................................. 110
Figure 29: Schematic of the ASC for solving COPs. ........................................................................... 127

## List of Tables

Table 1: Problems that can be expressed using the list modelling system ..... 38
Table 2: The intrinsic functions available to the list modelling system. ..... 39
Table 3: Appropriate transition operators given a problem's list constraints. y represents any list structure while $N$ represents a constant ..... 49
Table 4: Incremental cost expressions according to template and transition operator ..... 68
Table 5: Problem classes and instances that are used in this study. Note: as objective costs are typically reported dimensionless in the literature, units are not shown here ..... 83
Table 6: Transition probability settings for each problem type ..... 84
Table 7: Search engine-specific parameters. ..... 85
Table 8: Optimisation packages that are compared with the general solver. ..... 85
Table 9: Problem classes and instances that are used to test the parallel tabu engine ..... 87
Table 10: Largest clique sizes achieved with the SA engine using the alternative list formulation for the MCP given in Appendix B ..... 92
Table 11: Extended results of the SA engine. ..... 93
Table 12: Extended results of the TS engine ..... 96
Table 13: CSP results (objective costs and runtimes) for SA-CSP ..... 97
Table 14: Results of running tsp_solve on the TSP. ..... 98
Table 15: Results of running dfmax and dfclique on the MCP problem instances. ..... 98
Table 16: Results of running RLF on the GRAPH problem instances ..... 98
Table 17: Results of running QAPBB and QAPSIM on the QAP problem instances ..... 98
Table 18: The results of running Lindo on the problem test suite. ..... 100
Table 19: The results of running OSL on the problem test suite ..... 101
Table 20: Results of branch and bound with the shortest path heuristic for the USApHMP on a DEC 3000/700 with CPLEX (reproduced from Ernst and Krishnamoorthy (1997b, Table 2, p. 15)). . 101
Table 21: Results of branch and bound with the shortest path heuristic for the UMApHMP on a DEC
3000/700 with CPLEX (reproduced from Ernst and Krishnamoorthy (1997b, Table 3, p.16))... 101
Table 22: Feasibility restoration runs using SA. ..... 102
Table 23: Feasibility restoration runs using TS. ..... 103
Table 24: Parallel speedup results. ..... 105
Table 25: Parallel efficiency results ..... 106
Table 26: Parallel speedup and efficiency for gapA10-100 with and without using incremental cost expressions. ..... 111
Table 27: Rank order of the transition sets for each problem type ..... 112
Table 28: Rank order of the different SA cooling lengths for each problem type. ..... 112
Table 29: Rank order of the different TS neighbourhood probabilities for each problem type. ..... 113
Table 30: Performance comparison on the CSP ..... 114
Table 31: Performance comparison on the BIN. ..... 114
Table 32: Performance comparison on GRAPH. ..... 114
Table 33: Performance comparison on QAP ..... 115
Table 34: Performance comparison on TSP ..... 115
Table 35: Performance comparison on MKP ..... 115
Table 36: Performance comparison on GPP. ..... 116
Table 37: Performance comparison on TTP. ..... 116
Table 38: Performance comparison on MCP ..... 116
Table 39: Performance comparison on USApHMP ..... 116
Table 40: Performance comparison on UMApHMP ..... 117
Table 41: Performance comparison on GAP. ..... 117
Table 42: Overall comparison of the performance of the various solvers. ..... 118
Table 43: Ordering of the search engines based on their performances for each problem type. ..... 119
Table 44: The percentage of Optimal and Best-Known solution costs achieved by SA, TS and GS for the standard runs. ..... 125
Table 45: Standard results for the SA engine. ..... 180
Table 46: Standard results for the TS engine. ..... 188
Table 47: Standard results for the GS engine. ..... 197

## List of Algorithms

Algorithm 1: Skeleton pseudocode of an iterative meta-heuristic search algorithm ..... 16
Algorithm 2: Pseudocode for GS ..... 18
Algorithm 3: Pseudocode for TS. ..... 20
Algorithm 4: Pseudocode for SA. ..... 22
Algorithm 5: Pseudocode for GRASP. ..... 23
Algorithm 6: The simulated annealing based 0-1 solver GPSIMAN from Connolly(1990). ..... 26
Algorithm 7: Feasibility restoration procedure for the move operator ..... 62
Algorithm 8: Pseudocode for the master processor ..... 74
Algorithm 9: Pseudocode for the slave processors. ..... 75
Algorithm 10: The algorithm for forming an initial solution that satisfies the list constraints. ..... 76

## Acknowledgements

The undertaking of this PhD would not have been possible without the help and support of numerous people and organisations. I will attempt to give due credit to those who had a major impact on both this study and myself.

## Professor David Abramson

I am extremely fortunate to know and work with Professor Abramson. As my principal supervisor, he has provided me with direction and encouragement throughout the PhD process.

## Dr Clyde Wild

His reading of the thesis has improved its readability considerably. Dr Wild's knowledge of statistics has helped me greatly. I wish to also thank him for his administrative support of this PhD .

## Dr Keith Carter

His reading of the thesis and good-natured support is much appreciated.

## Queensland Parallel Supercomputing Facility

The computer provided by this facility has been the backbone of this work. It would not have been possible to perform the required computations on conventional workstations, especially the testing of the parallel implementation. Sadly, this facility is both underfunded and understaffed. The staff has carried out their difficult task admirably and has assisted me on numerous occasions. In particular, I would like to thank Dr Zdizslaw (Gustav) Meglicki and Mr Andrew Lewis.

## School of Environmental and Applied Science (Griffith University)

This School has supported me financially as well as providing the facilities I needed in order to complete this study.

## School of Computer Science and Software Engineering (Monash University)

The School has always been very welcoming during my frequent visits to Melbourne. In particular I would like to mention Ms Eryn Glover and Mr Paul Logothetis.

## Statement of Original Authorship

This work has not been previously submitted for a degree or diploma at any university or institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Signed:
Date:

## Author's Note

I have endeavoured to write this thesis for a broad scientific audience rather than strictly for those of the Operations Research, Mathematics or Computer Science communities. As part of this approach, I have included a Glossary that explains some of the more discipline-oriented terms and acronyms. I hope and trust that this helps. However, a passing knowledge of statistics is assumed.

Marcus Randall
5 May, 1999

## Chapter 1: Introduction

### 1.1 Introduction

According to Osman and Kelly (1996, p. 2) "Combinatorial optimization problems are normally easy to describe but difficult to solve". Typically these problems have a finite number of alternative solutions that represent different arrangements of discrete objects. As such, each solution state has a different degree of effectiveness for solving a particular problem. The aim is to find the solution that receives the highest overall evaluation while simultaneously satisfying a set of constraints. This is known as the optimal solution. Many real world problems that involve location, arrangement, scheduling and planning tasks can be classified as combinatorial optimisation problems. The importance of these problems to industry cannot be underestimated. For instance, Anbil, Gelman, Patty and Tanga (1991) describe a problem in which crews are scheduled to airline flights. Given that airline operating costs are in the order of billions of dollars per year and that personnel costs are one of the largest components thereof, even slight increases in efficiency can lead to substantial savings.

The number of possible solution states can be large even for relatively small size combinatorial optimisation problems. As a result, many of these problems are intractable and require years of computer processing time to identify optimal solutions. The theory of computational complexity identified by Cook (1971) states that a problem is NP (Nondeterministically Polynomial) hard if the amount of computational time required to find the optimal solution grows exponentially with the size of the problem instance, in the worst case. Consequently, it is improbable that efficient search algorithms exist that will solve such problems to give the optimal solution.

As a result of the fact that many COPs have been shown to be NP hard, combinatorial optimisation has been seen as a challenging field by researchers from a variety of disciplines including mathematics, computer science, engineering, economics and management. Numerous solution techniques exist for combinatorial optimisation problems. The most successful and widely used search algorithms are those that forsake the requirement of obtaining optimal solutions (Osman and Laporte 1996) and these are broadly known as heuristics. In most cases, special purpose algorithms that solve individual problem types are the most efficient way of solving these problems. As such, a great deal of research has focused on the development and refinement of these algorithms. In contrast there has been relatively little investigation to determine if and how more general-purpose search platforms can be built. Therefore it is in this area that the research of the thesis will concentrate.

### 1.2 Scope

Combinatorial optimisation problems may be defined as any optimisation problem that has a finite number of feasible solutions (Winston 1991). Most problems fitting this description have a discrete solution domain that is therefore defined over a subset of integers. The general model for COPs can be expressed using Equations (1) and (2) according to Osman and Kelly (1996).

$$
\begin{align*}
& \text { Optimise } f(x)  \tag{1}\\
& \text { subject to (s.t.) } \\
& \qquad x \in \mathbf{X} \subseteq \Omega \tag{2}
\end{align*}
$$

Where:
$x$ represents a solution to the COP.
$f(x)$ is the evaluation of the arbitrary objective function, $f$, according to $x$.
$\mathbf{X}$ is the feasible space.
$\Omega$ is the entire solution space.

The equation given in (1) states that an arbitrary function, $f$, with argument, $x$ (the solution), is to be optimised. That is, the solution with either the lowest or highest evaluation of $f$ (depending on whether the problem requires minimisation or maximisation) is sought. The feasible space is bound by a set of constraints (which are themselves arbitrary functions) (2). Combinatorial optimisation generally requires that the sets, $\mathbf{X}$ and $\Omega$ consist of discrete values.

The discrete solution space allows the modelling of problems that require some form of assignment or mapping of items to a number of different groups. Therefore much of combinatorial optimisation is primarily concerned with "finding an optimal arrangement, grouping, ordering or selection of discrete objects usually finite in number" (Osman and Kelly 1996, p. 2). There have been varied attempts to model COPs including Integer Linear Programming, graph theoretic approaches and constraint programming. These are discussed in Chapter 2.

As well as modelling these problems, there are subsequently many applicable solution methods and these may be separated into two distinct groups of exact and approximate approaches. Of the former group, the techniques of Operations Research (OR) predominate. The most widely used techniques within this field for solving COPs are "branch and bound" and "cutting plane algorithms" (Taha 1992; Winston 1991). While these techniques guarantee optimal solutions, the effects of NP complexity become apparent. These techniques are discussed in detail in Chapter 2.

The approximate techniques aim to find the best possible solution, not necessarily the proven optimal solution, with a given amount of computational effort. These techniques are referred to as heuristics and are often in the form of specialised algorithms that solve particular problems. Heuristics are generally very efficient. However, it can be expensive and time consuming to alter individual algorithms to suit other problems. Nevertheless, a class of meta-heuristic algorithms exist that can be adapted to suit different problems with less effort. While these algorithms are typically less efficient
than their tailored heuristic counterparts, they have been applied successfully to a variety of difficult and practical COPs.

Although meta-heuristics are general algorithms for solving COPs, there is no universally standard and efficient way of representing problem structure and data. Therefore, the common practice has been to incorporate problem specific information and data within the search engine itself. In contrast, relatively little research has been carried out to produce general and reconfigurable COP search engines, apart from Abramson and Randall (1998), Connolly (1992), Johnson, Aargon, McGeogh and Scheveon (1991a, 1991b) and Ingber (1993, 1996).

The work in this thesis contains the details of a new modelling system that represents COPs in a way that closely reflects the underlying structure of the problem being solved. In this study, it is found that a variety of iterative search techniques coupled with standard local search transition operators can successfully operate within the framework of this modelling system. The modelling system itself is discussed in Chapter 3, while the details of the implementation of a new general-purpose solver are contained in Chapter 4. The general system is tested on a wide variety of COPs over a range of parameters and compared to some benchmark solvers (Chapter 5). Finally, in Chapter 6, conclusions are drawn about the modelling system and its implementation. As well as this, the details of further research projects arising from this work are discussed.

### 1.3 Aim

The aim of this research is to demonstrate that practical size COPs can be solved in a standard way by meta-heuristic search engines. COPs are fundamentally difficult problems to solve and have wide applicability. While Integer Linear Programming (ILP) packages can solve these problems, their performance may be poor especially for moderate to large size problems. They are also particularly concerned with finding the optimal solution. There are also heuristic and meta-heuristic codes, however, they are usually tailored to solve specific problems and efficient solvers may not be available for particular problems. As such, there is no standard test-bench with which COPs can be solved. In this thesis, a practical means of solving arbitrary COPs using a new modelling system and associated meta-heuristic search algorithms is presented. Such a tool is particularly useful for the model prototyper and those who work with numerous problems of this nature.

In particular, the general system developed as part of this research has a number of key features that differentiates it from other solvers in a number of important ways. These are:

- It provides a new modelling data structure for COPs based on linked lists. This eliminates many of the constraints and variables associated with problems formulated using other notations.
- As well as providing a Simulated Annealing (SA), Tabu Search (TS) and Greedy Search (GS) framework, it has an in-built set of neighbourhood transition operators that are commonly used in tailored heuristics. A method has been developed that determines which local search operators are appropriate for a particular problem based on its constraints.
- It incorporates mechanisms that allow the evaluation of incremental objective functions.
- The objective and constraint functions may be arbitrary algebraic expressions.
- It provides an algebraic modelling language, which makes rapid prototyping possible.

The general system was produced so that COPs could be solved with minimal initial development time and effort. While it produces very good solutions to a range of hard problems, it is not meant as a replacement or substitute for efficient special purpose search algorithms.

## Chapter 2: A Review of Modelling Methods and Solution Techniques

This chapter concentrates on two key aspects of solving COPs, namely modelling systems (the way a problem is expressed) and search algorithms (the way a problem is solved). An outline of both of these topics is presented as a basis for developing a new modelling paradigm and subsequently a generalpurpose COP solver system.

### 2.1 A Review of Modelling Systems for COPs

There are a variety of methods that can be used to represent COPs. This section discusses five different approaches and is used as a foundation for the development of alternative modelling systems. The first of the approaches, Integer Linear Programming, expresses COPs in terms of linear equalities and inequalities using binary variables. The graph theoretic approach represents problems as graphs with solution states determined by the connection of vertices with a number of edges. The connectionist approach models COPs in terms of neural network architectures. The CP modelling system formulates problems in terms of finite domain variables rather than binary variables. Finally, the compacted integer vector approach is shown as an extension of ILP notation. In this approach, variables assume a range of integer values rather than being restricted to binary values and constraints can be either linear or non-linear.

### 2.1.1 Integer Linear Programming

As discussed in Section 1.2, COPs require that the solution domain be defined over a subset of discrete values. In terms of a linear programming approach, this domain is usually reduced to the interval $[0,1]$ in order to denote that an object is assigned to a particular group. Equations (3) - (5) give the generic formulation of a 0-1 ILP.

$$
\begin{align*}
& \text { Minimise/Maximise } \quad \sum_{j=1}^{N} C_{j} x_{j}  \tag{3}\\
& \text { s.t. }
\end{align*}
$$

$$
\begin{align*}
& A \cdot x\left\{\begin{array}{l}
\leq \\
\geq \\
\geq
\end{array}\right\} B  \tag{4}\\
& x_{j} \in\{0,1\} \quad \forall j \quad 1 \leq j \leq N \tag{5}
\end{align*}
$$

Where:
$C$ is the cost vector of size $N$.
$x$ is the solution vector of size $N$.
$A$ is the constraint matrix of size $M \times N$.
$B$ is the constraint vector of size $M$.

In the formulation, (3) - (5), the cost is optimised subject to the constraints specified in the matrix expression. The exact nature and values in both the $A$ matrix and $B$ vector depend on the problem being modeled.

Constraints are often used to express assignments of the form required for COPs. For instance, in order to denote that item $i$ is assigned to group $j$, constraints of the form of (6) are used. This constraint ensures that an item cannot belong to more than one group.

$$
\begin{equation*}
\sum_{i=1}^{N} x_{i j}=1 \quad \forall j \tag{6}
\end{equation*}
$$

Constraints of the form (6) are often referred to as encoding constraints as they are required to ensure the integrity of the solution. Similarly, some problems require that artificial variables be created for the same purpose. Encoding constraints and variables enlarge the model size significantly and as a result, the number of potential solution states increases exponentially with problem size, although many of these may not satisfy the constraints. Accordingly with realistic problems, it is usually impractical to simply explore the $2^{N}$ possible states either with explicit or implicit enumeration techniques (discussed in Section 2.2.1.1) in order to obtain the optimal solution.

### 2.1.2 Connectionist Models

Another representation of COPs is based on neural networks. This is referred to as the connectionist approach and models the way that neurons function and interact with one another (Carling 1992) as a paradigm for problem solving. It is typical for neural networks to have a number of nodes (neurons) as well as connections (weights) between them.

There are a number of different implementations and types of neural networks. In general, two models are suitable for representing and solving COPs (Smith, Palaniswami, and Krishnamoorthy 1996a,

1996b); the Hopfield-Tank network (Tank and Hopfield 1985) and the Kohonen Self-Organising Feature Maps (SOFM) (Kohonen 1982).

The Hopfield-Tank (Tank and Hopfield 1985) approach has been the most widely used neural network for COPs (Ramanujam and Sadayappan 1995). It is a single layer, fully interconnected network having an energy function (7) that is minimised (Hopfield 1982).

$$
\begin{equation*}
E=-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} T_{i j} V_{i} V_{j}-\sum_{i=1}^{n} I_{i} V_{i} \tag{7}
\end{equation*}
$$

Where:
$E$ is the resultant energy.
$n$ is the number of neurons.
$T_{i j}$ is the weight between neuron $i$ and neuron $j$.
$V_{i}$ is the activity of neuron $i$.
$I_{i}$ is the external bias or input of neuron $i$.

When modelling COPs using the Hopfield-Tank approach, both the objective function and the constraints are mapped onto (7) (Ramanujam and Sadayappan 1995) by extending the original energy function to include the constraints (8) (Peterson and Södberg 1989).

$$
\begin{equation*}
E=\text { cost }+ \text { constraints } \tag{8}
\end{equation*}
$$

Thus constraints are implemented as penalty terms in the energy function. By applying a learning rule, the state of the network $\left\{\mathrm{V}^{\mathrm{s}}\right\}$ is adapted such that the energy function is minimised. This learning rule is based on a gradient descent (greedy) search and accordingly the network will iterate towards a ground energy state that corresponds to a local optimum for the COP. Recent research by Smith Paliniswami and Krishnamoorthy (1998) has shown that Hopfield-Tank networks can: (a) ensure that optimal solutions are obtained and (b) incorporate hill-climbing search strategies to escape local optima.

Hopfield-Tank networks are more generalisable than the other common approach, SOFM. The SOFM networks take an arbitrarily dimensioned input and transform this into a discrete two dimensional (Euclidean) feature map (Smith et al. 1996a). In addition to adapting weights of the neurons in order to form a solution, the SOFM also organises the neurons on a discrete lattice. As a result, the most popular combinatorial application is to the Travelling Salesman Problem (TSP) (Smith et al. 1996a). An imaginary elastic band is stretched and allowed to move across the lattice until it attaches itself to the nodes. These nodes represent the cities and the distance between the nodes is proportional to the distance between the cities. The positions of the cities on the lattice also reflects their current location in Euclidean space and hence form an ordered ring that represents the tour. Thus the final tour is determined by the position of the nodes in the network and the location of the elastic band at the
network's ground energy state. There has been little effort to adapt this approach to other COPs (Smith et al. 1996a; Osman and Laporte 1996).

Research into neural network architectures for COPs is focusing on hybrid networks as well as refinements to existing techniques (Smith et al. 1996a and Smith et al. 1998). However, as a result of the problems of poor solution quality, lack of generality as well as the performance degradation of implementing neural networks on conventional sequential computer architectures, Osman and Laporte (1996, p. 522) comment of the connectionist approach that:
> "They have not been successful when applied to optimization problems and are not competitive with the best meta-heuristics from the operations research literature, when applied to combinatorial optimization problems."

### 2.1.3 Graph-Theoretic Techniques

A graph-theoretic approach represents COPs as either directed or undirected graphs in Euclidean space (Chartrand 1977; Wilson 1985). A graph $(G)$ consists of vertices ( $V$ ) connected by a set of edges $(E)$ and is given by $G=(V, E)$ (Gould 1988). Local changes can be made in $G$ in a number of ways including partitioning the vertices and adding or dropping the edges between vertices. A range of different problems can be expressed as graphs, especially those arising from transportation and scheduling. For instance, the TSP as a graph theoretic model can be represented as an undirected network ( $G$ ) whose vertices correspond to cities (see Figure 1). Every vertex pair is connected by a weighted edge (representing the distance between the two cities) and the problem becomes one of finding a minimum length Hamiltonian cycle on $G$.

As a result of the spatial characteristics of graphs, it can be difficult to map general graph problems that require optimisation onto a form that is directly solvable in a computational implementation. Despite this, there are a number of special purpose algorithms that exploit graph structure in order to perform combinatorial search for particular problems. Two such examples are the processor allocation problem (Sofianopoulou 1992) and the weighted maximal planar layout problem (Hasan and Osman 1995). The processor allocation problem is a problem in which a number of processes are allocated to a number of processors such that the total communication flow between processes on different processors is minimised. The maximal planar layout problem is a facilities layout problem in which a planar graph is sought that has the qualities: (a) no edges intersect; and (b) the sum of edge weights is maximised. For both problems, graph based algorithms can only solve relatively small problems. More complex problems require specification using algebraic techniques and solutions with local search methods (see Section 2.2.2 for a description of local search).


Figure 1: A graph-theoretic representation of a 5 city TSP. The cities are given by the set $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}$, E \} and form the vertices of the graph. The edge weights represent the distance between each pair of cities.

### 2.1.4 Constraint Programming

An alternative method of modelling and solving COPs is by Constraint Programming (CP). CP is a relatively new field of research belonging primarily to the domain of Artificial Intelligence (AI). It is a merger of declarative language and logic programming (Jaffar and Maher 1997) and is finding use in the modelling and solving both COPs and constraint satisfaction problems (Little and Darby-Dowman 1995).

CP combines an algebraic modelling system with an enumerative tree search algorithm (discussed in 2.2.1.3). Unlike 0-1 ILP notation that uses vector notation, CP can store problem information in terms of lists (Marriot and Stuckey 1998). These listshold integer values bounded within finite domains. In addition to relational constraints, CP incorporates conditional constraints commonly found in logic programming. Complex constraints, such as all elements of a list are different, are also accommodated in CP (Marriot and Stuckey 1998). These features provide for a rich modelling langeaye that is more natural and compact than integer linear programming. As a result, CP based solvers (such as ILOGSOLVER (ILOG 1998)) are finding success in commercial implementations.

Despite the language's richness, it still requires its own form of encoding constraints in order to specify a problem. This can enlarge problem size significantly and is particularly evident for problems such as the TSP. The number of constraints that are required to represent this problem can easily exhaust a CP solver software for sizes below 15 cities (Caseau and Laburthe 1997). This is due to the representation of the sub-tour constraints that ensure the path traveled by the salesman is a Hamiltonian cycle. Many COPs have complex underlying structures and as such, there are difficulties representing and solving common COPs using CP according to Little and Darby-Dowman (1995) and Osman and Laporte (1996).

### 2.1.5 A Compacted Integer Vector Approach

A recent study (Abramson and Randall 1998) examined another representation scheme for COPs ${ }^{1}$. This formulation technique allows variables to take on arbitrary integer values. This is in contrast to the linear modelling of COPs that often require the use of binary variables and result in sparse problem models. This scheme was developed so that a problem model could be written with fewer encoding constraints than the equivalent $0-1$ ILP model. The authors observed that:

- Fewer variables and constraints are required in the new integer formulation,
- The equality and encoding constraints are removed,
- It is necessary to allow inequality $(\nexists)$ constraints, which are non-linear,
- More complex cost functions are required, often involving non-linear operators.

In order to compensate for the lack of encoding constraints, problems are divided into three categories namely: order based, arbitrary linear constraints and assignment problems. Each of these categories has a distinct mathematical structure. Despite obtaining good quality solutions in relatively small amounts of time with the SA meta-heuristic, this modelling technique is difficult to apply to problems that do not fall into any of the categories or that span multiple categories. This study compared the system's performance (called INTSA) against Connolly's (Connolly 1992) General Purpose SIMulated ANnealing (GPSIMAN) and a commercial ILP package, Optimisation Subroutine Library (OSL). The latter two codes accept problems formulated as $0-1$ ILPs. The results demonstrated that solving moderate to large size COPs using 0-1 ILP formulations is impractical.

The work described in Abramson and Randall (1998) acts as a precursor to the material contained in this thesis because it identified that acceptable performance was possible for a range of problems with a common representation. The main disadvantage is that it is not general enough to represent real problems. However, the concept of the representation of COPs using integer values forms a seminal component of the new linked list modelling approach (as presented in Chapter 3).

### 2.2 Search Algorithms for Combinatorial Optimisation Problems

### 2.2.1 Exact Search Algorithms

Exact solution techniques are those that produce solutions to COPs that can be proven to be optimal. In this section, four widely studied approaches from the Operations Research and Artificial Intelligence disciplines are examined.

[^0]
### 2.2.1.1 Operations Research Techniques

Using the 0-1 ILP modelling technique discussed in Section 2.1.1 often gives rise to large and sparse problem models. Techniques belonging to OR have been developed to exploit the structure of these models, using a form of implicit enumeration to find proven optimal solutions. There are two classes of such techniques, namely, branch and bound and cutting plane algorithms (Taha 1992).

Branch and bound (Taha 1992; Winston 1991) is an iterative tree search process in which the problem is first solved by disregarding the constraint that all values must be discrete. This is achieved by using a procedure such as the simplex algorithm. Each variable that contains a continuous value is used in order to form two new sub-problems that exclude this continuous region. For instance, if variable $i$ in solution $x$ contains the value 1.5 , two sub-problems are created by adding the constraints $x_{i} \leq 1$ to the first sub-problem and $x_{i} \geq 2$ to the second sub-problem. The search process will then choose one of the available sub-problems and repeat the process (known as branching). The choice of sub-problem is governed by a branching strategy. Various branches of this tree can be shown not to contain the optimal solution given the lower and upper bounds of other branches (known as bounding), hence they are not explored and said to be pruned. The search is complete when all the branches have been fathomed (i.e. explored or pruned) and the optimal solution is returned.

Cutting plane algorithms are similar to branch and bound, as they will first solve the problem without regard to obtaining pure integer values. However, instead of branching at each step, cutting plane algorithms add a constraint to the system to 'cut' off continuous regions. A characteristic of the algorithm is that the optimal solution is found when an all integer solution is reached.

Techniques such as branch and bound and cutting plane algorithms have been the popular search techniques to solve COPs. However, apart from suffering NP time bounds in the worst case, moderate to large size problems often require many variables and constraints and hence the problem can become intractable even on large supercomputers using sophisticated software packages. As such, Taha (1992, p. 309) states that branch and bound and branch and cut are not "uniformly effective" methods of solving COPs.

### 2.2.1.2 A*

A* is an implicit tree search algorithm that was originally proposed by Hart, Nilsson and Raphael (1968). This algorithm has the characteristic that it considers estimates of the remaining distance until the optimal or goal-state is reached given the current solution-state. The generic objective function for problems solved with $A^{*}$ is:

$$
\begin{equation*}
f^{*}(n)=g(n)+h^{*}(n) \tag{9}
\end{equation*}
$$

Where:
$n$ is the current solution-state.
$f^{*}(n)$ is the modified objective cost of the solution state $n$.
$g(n)$ is the cost of the solution state $n$.
$h^{*}(n)$ is an estimate of $h(n)$, the minimum cost difference between the current state, $n$, and the optimal state.

If $h^{*}(n)$ in Equation (9), is an underestimate of $h(n)$, the $\mathrm{A}^{*}$ search procedure is guaranteed of finding the optimal solution. There are numerous ways that the algorithm can be applied to particular problems. The following description illustrates one such approach for the TSP.

The goal of the TSP is to construct a Hamiltonian tour of minimum length between cities lying on a Euclidean plane. Beginning from a starting city (for example, city 1) and using a best search strategy (as described in Winston (1992)), the next city to add to the solution is chosen so that the $f^{*}(n)$ is the smallest amongst all the alternatives. For this problem, the estimate of the remaining distance can be the distance back to the starting city. The distance from the current city to that starting city is always smaller than the length of the rest of the tour. If the partial tour length is greater than the smallest overall tour length found to date, then this partial tour cannot lead to the optimal solution. Hence, the solution is backtracked to the previous city on the tour and this part of the search space is pruned (or fathomed). Providing that each $h^{*}(n) \leq h(n), \mathrm{A}^{*}$ will always return the optimal solution.

While A* search can be very efficient for some problems, it still suffers from NP time bounds as does branch and bound, and branch and cut. In addition, there may not be enough information to obtain $h^{*}$ or it may be too complex and time consuming to compute (Firebaugh 1989; Luger and Stubblefield 1993; Winston 1992).

### 2.2.1.3 Constraint Programming

Apart from the modelling system discussed in Section 2.1.4, CP provides a proven optimal search technique for COPs. It uses a form of tree search based on implicit enumeration but is different to the branch and bound procedure.

The solution space of a CP problem is represented by a tree structure. Each of the levels of the tree represents a variable and the domain of each variable is given by the set of nodes at that level. A candidate solution is obtained by traversing the tree from the root node to a leaf node. The entire tree is initially pruned by considering the effects of all the constraints (called constraint propagation). The search process is a form of enumeration that proceeds as a depth-wise tree traversal. According to the current place in the tree, the process can disregard values of variables that would make the current solution infeasible (referred to as dynamic pruning). Should the process determine that all values of a
particular variable are infeasible with the current solution, it will backtrack and undo changes at higher levels of the tree. An example of the CP search strategy is shown in Figure 2 and Figure 3. Figure 2 shows a complete CP for a small two variable problem, whilst Figure 3 presents the tree after the constraint, $\mathrm{X} \geq \mathrm{Y}$, is processed (demonstrating constraint propagation).

## [image removed]

Figure 2: X and Y have the domains of $2,4,5$ and $4,6,8$ respectively. This is the entire search tree. This problem is a demonstration example reproduced from Little and Darby-Dowman (1995, Figure 1, p. 3).

## [image removed]

Figure 3: Once the constraint $\mathrm{X} \geq \mathrm{Y}$ is added, the search tree is reduced (pruned). This problem is a demonstration example reproduced from Little and Darby-Dowman (1995, Figure 2, p. 3).

The CP system was originally designed for solving constraint satisfaction problems (Osman and Laporte 1996). Constraint satisfaction problems differ from COPs as they require that only a set of constraints be satisfied rather than the optimisation of a specific objective function. An example of a constraint satisfaction problem is the SEND + MORE $=$ MONEY problem in which numerals are assigned to the letters such that each letter has only one value and the addition is satisfied (Little and Darby-Dowman 1995). In order for CP to evaluate multiple feasible solutions to obtain the optimal solution for a COP, a special constraint is used. This constraint states that in order for a solution to be feasible, it needs to have a cost value better than the currently best-known value.

At the present time, constraint programming systems such as ILOG Solver (ILOG 1998) use the proven optimal search strategy outlined above and as such suffer from the same NP time complexity difficulties as the OR methods. CP problems should also be very carefully modeled as Little and Darby-Dowman (1995, p. 9) note:

> "Generally, the more constraints the greater the search reduction which can take place. However, since each constraint takes time to be woken up and processed, if any are making no significant reduction in the search space, this may use up processing time."

Currently research is being undertaken to allow CP to use other search techniques such as those from the meta-heuristic family (Stuckey and Tam 1996; Barnier 1997). Barnier (1997) discusses a hybrid Genetic Algorithm (GA) - CP implementation capable of solving small Vehicle Routing Problems (VRPs). The problem was modeled using CP notation and the intrinsic CP search engine was replaced by a GA, yielding encouraging initial results.

### 2.2.2 Heuristics

Pearl defines heuristics in the broad sense as "criteria, methods or principals for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal" (Pearl 1984, p. 3). In terms of combinatorial optimisation, heuristic procedures aim to produce good quality solutions in a short amount of computational time. However, this is at the expense of not obtaining proven optimality and is in contrast with the implicit enumeration techniques such as branch and bound, CP and $\mathrm{A}^{*}$. There is a variety of heuristic techniques available for different COPs and such heuristics are tailored to the particular problem that they solve. As a result, specific problem data structures can be directly incorporated in the computer code to produce an efficient solver. Therefore in practice, specialised heuristics are often used for large and complex problems in preference to the exact techniques.

Heuristic techniques can be broadly classified into two groups that describe how the heuristic obtains a solution to a particular problem. These are:

- Improvement Techniques An initial solution to a problem is changed over a number of iterative steps so that solution quality is gradually improved. These changes take the form of altering the solution slightly using any one of a variety of local search transition operators. The set of all possible changes that can be applied to a particular solution state by a particular operator is referred to as its neighbourhood. Iterative improvement techniques are more generalisable than constructive methods and have wider application. Some examples of successful heuristic procedures that are based on iterative
improvement are (Christofides and Eilon 1969; Higgins, Kozan and Ferreira 1997; Lin and Kernighan 1973; Kernighan and Lin 1970; Johnson 1990; Battiti and Protasi 1995).
- Constructive techniques Rather than iteratively adjusting a solution over a number of algorithmic steps (like the improvement techniques), constructive techniques build a final solution from an empty solution by a process of successive augmentation of solution components. As such, constructive techniques rely on a detailed understanding of the problem to allow the development of the solution. For instance, a constructive technique for the VRP builds a solution by successively assigning customers to be serviced by a vehicle until all customers are accounted for (the Clarke-Wright procedure, see Clarke and Wright (1964) and Osman (1993)). Other successful constructive heuristics include the classic nearest neighbour heuristic for the TSP and those works by Crama, Flippo, van de Klundert and Spieksma (1995) and Rosenkrantz, Stearns and Lewis (1977).

In some cases, techniques from the two classes have been combined to form an overall problem solving strategy that will first produce an initial solution using a constructive heuristic and refine this using iterative improvement. An example of this is the Martello and Toth heuristic for the generalised assignment problem (Martello and Toth 1981). The first phase of the algorithm constructs a solution by successively assigning jobs to agents according to a computed desirability measure. The second part consists of an iterative search that attempts to reassign jobs to agents in order to minimise the overall cost.

Local search operators are an integral part of the operation of the improvement techniques. Some local search operators are designed for specific problems such as the Lin-Kernighan heuristic for the TSP (Lin and Kernighan 1973). However, there is a set of operators that have general applicability across problem type. The most well known of these is the Or operator (Or 1976) in which the positions of two elements of the solution structure are exchanged. This transition is generalisable to the $n$-opt exchange as described by Osman $(1993,1995)$ in which more than two elements are involved. Other well known local search operators include add and drop in which items are added to or dropped from the solution respectively as well as the insert operator that changes the position of an item within a solution (Glover and Laguna 1997). The insert and Or operators are particularly appropriate for sequencing problems such as the TSP and Quadratic Assignment Problem (QAP).

### 2.2.3 Meta-heuristics

According to Osman and Kelly (1996), a meta-heuristic may be broadly defined as an "iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space" (Osman and Kelly 1996, p. 3). Like their heuristic counterparts, meta-heuristics are also approximate techniques. The general form of an iterative metaheuristic based on local search is given in Algorithm 1.

```
X = Generate Initial Feasible Solution;
C(X) = Compute initial cost of X;
While (stopping criterion not met)
    Transition = Select a Transition from Neighbourhood (X);
    X' = Apply Transition Operator(X,Transition);
    \DeltaC = Compute Change in Cost (X, X', Transition);
    If (accept)
                X = X';
            C(X) = C(X) + \DeltaC;
    End If;
    If (minimisation problem and C(X) < C ( best ) OR
    (maximisation problem and C(X)> C Cest) )
End While;
Output C Cest;
End.
```

Algorithm 1: Skeleton pseudocode of an iterative meta-heuristic search algorithm.

The key calculations which are performed repeatedly by this algorithm are: the generation of a state transition in the neighbourhood of the current one; application of the transition to compute a new solution, $\mathrm{X}^{\prime}$; the computation of the difference in cost between the new solution and the previous one, and whether to accept the change. This process can be repeated any number of times, and therefore a termination strategy is necessary. Two common approaches are to stop after a fixed number of iterations have occurred or stop after a certain solution quality has been reached. There is no standard stopping criteria in the literature (Barr, Golden, Kelly, Resende and Stewart 1995) and a variety of other methods are possible.

While meta-heuristics are generally less efficient at finding optimal and near optimal solutions than their tailored heuristic counterparts, the advantage of meta-heuristics is that they can be applied (in principle) to solve a wide variety of problems and are subsequently not restricted to particular problems. Despite this, the practice of many researchers has been to tailor meta-heuristic search engines to particular problems (see Beasley and Chu 1997a; Chams, Hertz and de Werra 1987; Chu and Beasley 1997; Connolly 1990; Kampke 1988; Osman 1993, 1995; Taillard 1991 as examples). While this technique can generally produce solutions of good quality in reasonable amounts of computational time, a great deal of effort is often required in order to recode the meta-heuristic program to suit a different problem type. In contrast, relatively little effort has been made to produce a general-purpose meta-heuristic search platform, except for Abramson and Randall (1998), Connolly (1992), Ingber (1993, 1996) and Johnson et al. (1991a, 1991b).

There are a number of different meta-heuristic techniques suitable for solving COPs. This section will describe five of the more common ones, namely Simulated Annealing, Tabu Search, Greedy Search, GRASP (Greedy Randomised Adaptive Search Procedures) and Genetic Algorithms. These techniques can be divided into two categories; those that are able to use local search operators and those that do not. GS, TS, SA and GRASP are in the former group while GAs and Artificial Neural Networks (ANNs) are in the latter.

Most research concerning meta-heuristics and combinatorial optimisation has concentrated on local search techniques. A common issue is that of obtaining and maintaining feasible solutions. For many of the COPs that have been studied, feasible solutions are maintained by the application of an appropriate local search operator. For instance, many SA codes that solve the TSP store the solution as an ordered list of cities. In this case, any local search operator that rearranges the order of the cities will not violate the constraint that each city must be visited only once. However, in the case of other problems, such as the GAP, knapsack problem, VRP and set partitioning problem, there are additional restrictions on the feasible space that cannot be satisfied with the application of a simple local search operator. Examples of this include capacity constraints for the GAP and time window constraints for the VRP. In this case, two main practices have been adopted. First, constraints are incorporated into the objective function as penalty terms and second, the solutions are repaired to a feasible state at each transition. Abramson, Dang and Krishnamoorthy (1996) found that the former approach often returns infeasible solutions and concluded that the second was the most practicable approach (though the processing time for each iteration may be high).

Section 2.2.3 describes each of the chosen meta-heuristic algorithms briefly (apart from ANNs that have been dealt with in Section 2.1.2). Section 2.3 contains a review of existing general meta-heuristic implementations.

### 2.2.3.1 Greedy Search

GS is the simplest of the iterative search techniques. Given an initial feasible solution to a problem, GS will examine the neighbourhood of the current solution for a new solution with a better cost. This process is repeated until an improving transition cannot be made (Algorithm 2). As this process will stop at the first local optimum it encounters, it is often referred to as local optima search.

GS is very easy to implement and unlike its more sophisticated counterparts (namely SA and TS), will return a solution to a problem in a relatively small amount of computational time. It is rarely used as a practical solution technique for COPs as it often produces poor solutions due to its characteristic of settling in a local optimum. However, it is useful as a benchmark to test the performance of other meta-heuristic and heuristic implementations (Barr et al. 1995; Battiti and Tecchiolli 1995).

```
X = Generate Initial Feasible Solution;
C(X) = Compute initial cost of X;
continue = TRUE;
While (continue = TRUE)
    Transition = Select a Transition from Neighbourhood (X);
    X' = Apply Transition(X,Transition);
    \DeltaC = Compute Change in Cost (X, X', Transition);
    If (minimisation problem and \DeltaC < 0) OR (maximisation problem
    and }\DeltaC>0
                X = X';
                C(X) = C(X) + \DeltaC;
    Else
            continue = FALSE;
    End If;
End While;
Output C(X);
End.
```

Algorithm 2: Pseudocode for GS.

### 2.2.3.2 Tabu Search

TS is a relatively new meta-heuristic search method that has been successfully applied to COPs (Glover 1989, 1990; Glover and Laguna 1997). Glover (1989, p. 191) describes TS as:
"Tabu search guides ... a heuristic to continue exploration without being confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged"

The subordinate heuristic can take many forms (Glover and Laguna 1997), though it is usual that it is a local search engine. However, TS has also been used as the controlling strategy for a branch and bound framework for 0-1 ILPs (Aboudi and Jörnsten 1994; Lokketangen, Jörnsten and Storoy 1994). In the context of COPs, TS can be thought of as an enhanced and more general version of GS. The unique characteristics of TS that distinguish it from its simpler counterpart are:

- Tabu Search can escape local optimum traps: Local optima serve as attractors to search techniques. This is inevitable, as search techniques seek the global optimum solution. Often search techniques become trapped in a local optimum, either permanently (like GS) or potentially for a large number of iterations (like SA). TS overcomes this inherent problem by evaluating the neighbourhood of the current solution, $N(x)$ and choosing the best transition from the ones currently available in $N(x)$ regardless of whether it improves the current solution cost ${ }^{2}$. If the transition is non-improving, then the search process has encountered a local optimum and thus begins the process of escape immediately.

[^1]- Tabu Search can effectively sample the search space: After a transition is made in which a non-improving move is accepted, the new neighbourhood, $N^{\prime}(x)$, contains the previous state which is now an improving transition. If this is accepted as the next transition, the search can be said to be cycling. TS has a mechanism that overcomes this problem that is referred to as the tabu list. The tabu list stores information that the search can use in order to avoid previously traversed search routes. A transition is considered tabu if it has been recorded on the tabu list and its tabu tenure has not passed. The tabu tenure is the number of iterations that an item on the list stays tabu. As only limited information is recorded on the tabu list, it is possible that the search process will regard solutions that have not been previously encountered as being tabu. In order to counteract this potentially negative effect of the tabu list, TS makes use of one or more aspiration functions. The simplest and most widely used of these functions is a rule that states that a tabu transition is accepted if it produces a superior quality solution compared to those previously encountered. Glover and Laguna (1997) discuss a wide variety of aspiration functions.

TS is described in Algorithm 3.

```
X = Generate an Initial Feasible Solution;
C(X) = Compute initial cost of X;
best_cost = C(X);
Initialise tabu list T = \varnothing ;
While (stopping criterion not met)
    For (s \in N(X))
                X' = Apply Transition(X, S, Transition);
                            \DeltaC = Compute Change in Cost ( }\textrm{X},\mp@subsup{\textrm{X}}{}{\prime},\textrm{S}\mathrm{ );
        End For;
        While (suitable neighbour not found)
            s \inN(x);
            If (s & T)
                X = Apply Transition(X, s, Transition);
                T = TU S;
                C(X) = Compute cost of X;
                found suitable neighbour = TRUE;
            Else
                If (aspiration(s) = TRUE)
                                    X = Apply Transition(X, s, Transition);
                                    C(X) = Compute cost of X;
                                    found suitable neighbour = TRUE;
                End If;
            End If;
        End While;
        If ((minimisation problem AND C(X) < best_cost) OR
        (maximisation problem AND C(X) > best_cost)) best_cost = C(X);
End While;
Output best_cost;
End.
```

Algorithm 3: Pseudocode for TS.

The TS algorithm described above and in Algorithm 3 corresponds to the short-term implementation that is popular in the literature (Glover and Laguna 1997). Advanced forms of TS use intensification and diversification strategies embodied in long term memory structures (Glover and Laguna 1997). Intensification aims to identify solution attributes that are common to good solutions and to encourage the tabu search (via the tabu and aspiration mechanisms) to seek solutions with these attributes. Diversification is complimentary to this, as it allows the search process to enter unexplored regions of the state space. Diversification can be achieved by using longer tabu tenures for solution attributes that are frequently incorporated into the solution (a form of penalisation), thereby encouraging the use of less frequently used attributes. Another strategy is to form a new starting solution once long-term cyclic behaviour has been detected (Battit and Tecchiolli 1994; Glover and Laguna 1997). However, many implementations of TS use only a limited form of the tabu list that can be thought of as shortterm memory that does not explicitly incorporate intensification and diversification strategies. Some examples of short-term TS implementations are Abada and El-Darzi (1996), Osman (1993, 1995), Taillard (1991), and Taillard, Badeau, Gendreau, Guertin and Potvin (1997).

### 2.2.3.3 Simulated Annealing

SA is derived from the physics of annealing metals. SA seeks to minimise an energy function, which in combinatorial optimisation is the objective function. At the beginning of the annealing there is a
high likelihood of accepting any transition regardless of whether it improves the solution or not, rather than later on in the process. This process is performed in accordance with an exponential acceptance function (called the Boltzman function) based on a parameter called temperature. The temperature is decremented at intervals regulated by a Markov chain length until it is quite small and hence very few uphill transitions, in which a worse solution may replace the current solution, are accepted. As SA can make these non-improving transitions, settling into a local optimum is potentially avoided. The way the temperature is controlled is referred to as the cooling schedule. For further information about SA and its variants, see Connolly (1992), Eglesse (1990), Johnson et al. (1991a, 1991b), Kirpatrick, Gelatt and Vecchi (1983) and van Laarhoven and Aarts (1987). A general algorithm for SA is given in Algorithm 4.

SA has been applied to a wide variety of COPs and there are a great number of articles in the literature, including Abramson (1991), Chams et al. (1987), Connolly (1990, 1992), Johnson (1991a, 1991b), Kampke (1988), Kirpatrick et al. (1983), Kouvelis and Chiang (1992), Koulamas, Antony and Jansen (1994), Osman (1993, 1995), Randall and Abramson (1998) and van Laarhoven and Aarts (1987).

```
X = Generate an Initial Feasible Solution;
C(X) = Compute initial cost of X;
best_cost = C(X);
T = Compute initial temperature;
While (stopping criterion not met)
    Repeat (markov chain length times)
        Transition = Select a Transition from Neighbourhood (X);
        X' = Apply Transition(X,Transition);
        \DeltaC = Compute Change in Cost (X, X', Transition);
        p = generate random number (0,1)
        If (((minimisation problem AND \DeltaC < 0) OR
        (maximisation problem AND \DeltaC > 0)) OR ( (e
                        X = X';
                            C(X) = C(X) + \DeltaC;
        End If;
        If ((minimisation problem AND C(X) < best_cost) OR
        (maximisation problem AND C(X) > best_cost))
        best_cost = C(X);
    End Repeat;
    T = Apply cooling function (T);
End While;
Output best_cost;
End.
```

Algorithm 4: Pseudocode for SA.

### 2.2.3.4 Greedy Randomised Search Procedures (GRASP)

GRASP is a search technique that consists of two distinct phases per iteration, namely a construction and local search phase (Feo and Resende 1995; Mavridou, Pardalos, Pitsoulis and Resende 1995; Glover and Laguna 1997). The first stage builds a new feasible solution from a list of elements, one element at a time. The list itself is produced by ordering the elements with respect to a greedy function based on elements chosen in previous iterations. The next element to be added to the solution is randomly chosen from the best candidates in the list. In many instances, this solution will not correspond to a local optimum and can hence be improved upon. This is achieved by applying a local search procedure (such as GS) to the solution.

GRASP is a new procedure compared with GS, TS and SA. However, it has been successfully applied to a number of problems including set covering (Feo and Resende 1989), VRP (Kontoravdis and Bard 1995), QAP (Li, Pardalos and Resende 1994) and the p-hub median problems (Klincewicz 1992). The general structure of GRASP is given in Algorithm 5.

```
While (stopping criterion not met)
    X = \varnothing;
    S = Construct an ordered element list;
    While (solution not complete)
                s = Greedy random element of s;
                X = XUs;
    End While;
    X = Execute local search (such as Algorithm 2);
    C(X) = Compute cost of X;
    If ((minimisation problem AND C(X) < best_cost) OR
    (maximisation problem AND C(X) > best_cost)) best_cost = C(X);
End While;
Output best_cost;
End.
```

Algorithm 5: Pseudocode for GRASP.

### 2.2.3.5 Genetic Algorithms

Genetic Algorithms (GAs) belong to a broader class of function optimisation techniques known as evolutionary computing. GAs are modeled on the biological selection and reproduction of genetic material, chromosomes. This follows a Darwinian or natural selection approach in which the fittest chromosomes survive and reproduce while the others perish (Goldberg 1989; Randall 1995). In terms of function optimisation, the chromosomes represent potential solution vectors. The algorithm proceeds as follows:
a) Generate an initial population of chromosomes
b) Evaluate each chromosome's objective function (fitness)
c) Select a number of the fittest individuals to form a mating pool
d) Apply mating procedures to the individuals in the mating pool to form the next generation of chromosomes
e) Repeat the procedure from b) onwards until a number of generations have passed or a certain quality of solution has been reached.

The mating process, step d), consists of applying a number of genetic operators to the chromosomes in the mating pool. The most common operators are: crossover - subsections of two or more chromosomes are combined in order to form a new individual, and mutation - an element of a chromosome changes its value to a random value.

Like neural network approaches, GAs have not been as successful as other meta-heuristic techniques in solving COPs, as observed by Osman and Kelly (1996). One problem is that GAs often converge to poor local optima if a population of weak individual chromosomes is dominated by single or small group of fit individuals. This problem can be partially overcome by increasing the mutation or applying a scaling function to the fitness function (Goldberg 1989). Another approach that has been
used in Beasley and Chu (1996, 1997a, 1997b) and Chu and Beasley (1997) to overcome poor local optima, has been to execute a problem specific heuristic after each generation has been produced in order to improve the fitness of the chromosomes. As these studies only report the results of the hybrid implementation and not the GA by itself, it is unclear how much of an impact the specialised heuristic makes.

A more serious problem is that GAs find it difficult to deal with constraints due to their context-free nature (i.e. there is "no reliance on conditions that solutions must obey in a particular problem setting" (Glover and Laguna 1997, p. 309)). This mainly arises from the recombination operator, crossover, in which large portions of different chromosomes are combined so that given feasible parent solutions, there is a high likelihood that the child chromosome is infeasible. To counteract this, three strategies are generally available: a) restore the feasibility of the offspring chromosomes, (b) apply a special purpose operator to ensure feasibility or minimise the amount of infeasibility and (c) trust that at the end of the GA run there is a feasible, highly fit chromosome. Procedure (a) is often computationally expensive, (b) is only available for specific problems such as the TSP, and (c) does not guarantee that a final feasible solution is produced.

### 2.3 General Purpose Meta-heuristic Solvers

As mentioned previously, few attempts have been made in the past to produce a general meta-heuristic solver using local search for COPs. This is due mainly to two reasons:

- Meta-heuristic and heuristic codes that are customised to particular problems (such as the TSP or QAP) are very efficient and produce good quality solutions with relatively little computational effort. However, substantial reprogramming is required in order to build another implementation that will solve a different problem type. Some examples of specific meta-heuristics include Beasley and Chu (1997a), Chu and Beasley (1997), Connolly (1990), Kampke (1988), Osman (1993, 1995), Smith et al. (1996a, 1996b) and Taillard (1991).
- It is a difficult task to design and implement appropriate modelling systems and data structures to support a system that solves arbitrarily formed problems.

This section reviews general meta-heuristic implementations that have been presented in the literature. These implementations can be divided into two approaches: systems that allow the problem to be represented in an algebraic form; and skeleton systems that require the user to produce interfacing computer code that represents the problem.

### 2.3.1 Algebraic Modelling Approach

The only algebraic modelling implementation that is represented in the literature for meta-heuristic based local search is GPSIMAN (Connolly 1992).

### 2.3.1.1 GPSIMAN

Connolly's General Purpose SIMulated ANnealing (GPSIMAN) solves 0-1 ILPs with SA (Connolly 1992). This approach is adequate for very small problems, however, as the size of the problem increases, the number of variables and constraints rapidly increases as a result of the 0-1 encoding. Therefore, making a neighbourhood transition, i.e. changing a variable value from a 1 to a 0 and vice versa, is computationally expensive as many of the constraints are potentially violated and need to be restored to a feasible state before the next transition can be undertaken.

Algorithm 6 shows the overall algorithm used in GPSIMAN. The algorithm operates as follows. First, the problem model and SA parameters are initialised, and an initial solution is generated along with its cost. A number of annealing runs is subsequently performed. Within each run, variables are chosen and altered at random. A transition is characterized by changing the state of a variable (from 0 to 1 or 1 to 0 - referred to as flipping) and measuring the effect in terms of how the cost changes. Also, if the change causes the constraints to become infeasible, then feasibility is restored before the change in cost is evaluated.

The feasibility restoration technique flips variables (other than the original variable) in order to obtain a new feasible solution. The scheme employed in Connolly (1992) is a heuristic technique whereby a score is computed for each of the variables based on how helpful a change in the variable value would be for feasibility restoration. The most helpful variable (the one with the highest score) is flipped and the resulting amount of infeasibility is re-calculated. If feasibility has been restored, the procedure is terminated. However, in many instances, particularly for $0-1$ problems that have many related constraints, this is not the case. The algorithm proceeds to calculate the next most helpful variable. This progresses as a depth-wise tree search, in which the algorithm can backtrack, should it find that the current sequence of flips cannot restore feasibility. This procedure is only useful if feasibility is relatively easy to restore else the search for feasibility can become impracticable. If the process cannot restore feasibility after a fixed number of searches, then the original transition is rejected. After feasibility is restored, the change in the objective cost can be calculated by simply adding or subtracting the appropriate coefficients ( $C_{i}$ ) for all variables that have been flipped. If the cost is positive (for a minimisation problem), then the proposed solution is worse than the current one and it is accepted as the current solution depending on the evaluation of Boltzman's equation. If it is not accepted and a number of consecutive previous solutions have also been rejected, the process is performed at the temperature at which the best solution (in this trial) was found. This continues for the remainder of the annealing run.

GPSIMAN is impractical for problems larger than the original test set that Connolly (1992) proposed. This was demonstrated in Abramson and Randall (1998) (see Appendix F).

```
Get SA parameters (T, T, 的eps_per_trial,max_fails);
Read problem model;
fails=0;
X=Generate initial feasible solution;
C(X)=Compute cost of initial solution;
Repeat (user specified number of annealing trials)
```



```
    T=T;
    While (T > T P
            i=unif_rand(1,sizeof(X));
            X'=X;
            X' =1-X ;
            X'=Restore Feasibility;
            \DeltaC=Compute change in cost(X, X');
            p=unif_rand(0,1);
            If (((minimisation problem AND \DeltaC < 0) OR
            (maximisation problem AND \Delta\overline{C}>0)) OR (e-\Deltac/T}>p)
                    X=X';
                    C(X)=C(X)+\DeltaC;
                    fails=0;
            Else
                    fails=fails+1;
                    If (fails > max_fails)
                        T min}=T
                        \beta=0;
                        T=T Tbes;
                    End If;
            End If;
            If (C(X) < C )
                    C
                    T
            End If;
            T=T/(1+\beta\timesT);
        End While;
        T
        If ( }\beta=0\mathrm{ )
            T}=(\mp@subsup{T}{F}{}+\mp@subsup{T}{min}{m})/2
        Else
            T
        End If;
End Repeat;
Output C best'
End.
Where:
```

    unif_rand \((a, b)\) returns a random uniformly distributed number between \(a\) and \(b\).
    Algorithm 6: The simulated annealing based 0-1 solver GPSIMAN from Connolly(1990).

### 2.3.2 Code Segment Approach

Code segment approaches adopt a different strategy than algebraic representation methods. These systems consist of a core meta-heuristic search engine (such as SA or GS) for which the developer is required to provide an interface (via code modules) for a particular problem. This specifies the cost, constraint and transition functions. These systems are not based on an algebraic modelling language and while code development is a flexible approach to modelling a particular problem, this can require the developer to spend a substantial amount of time defining the problem to be solved. In this section, three implementations are discussed, namely the Johnson et al. Generic SA (Johnson et al. 1991a, 1991b) Ingber's Adaptive Simulated Annealing (ASA) (Ingber 1993, 1996) and skeleton GA codes.

### 2.3.2.1 Johnson's et al. Generic SA

In this implementation (Johnson et al. 1991a, 1991b), the system components are divided into generic and problem specific sections. The generic part consists of a set of core SA subroutines that control the cooling schedule. In these studies (Johnson et al. 1991a, 1991b), the system was tested on the graph partitioning problem, graph colouring problem and number partitioning problem by varying the parameters of the generic part. The problem specific section defines the problem in terms of cost function, constraints and possible transition functions for which the developer must provide compatible computer code. Figure 4 summarises the problem specific and generic components of the system.

## PROBLEM-SPECIFIC

1. What is the solution?
2. What are the neighbors of a solution?
3. What is the cost of a solution?
4. How do we determine an initial solution?

## GENERIC

1. How do we determine an initial temperature?
2. How do we determine the cooling rate $r$ ?
3. How do we determine the temperature length $L$ ?
4. How do we know when we are frozen?

Figure 4: SA implementation choices made by Johnson et al. (1991a, 1991b). Reproduced from Johnson et al. (1991a, Figure 5, p. 869).

### 2.3.2.2 ASA

ASA is a freely available SA toolkit produced by Lester Ingber Research (Ingber 1993, 1996). It uses a different form of annealing schedule from the standard Boltzman function as proposed in the original versions of SA (van Laarhoven and Aarts 1987). This new schedule is much faster than the original as it samples the search space more effectively (Ingber 1996). Unfortunately, the time required to run the program can be lengthy for NP hard problems due to the cooling schedules ASA uses. In order to improve the performance of the system, an option is available that permits rapid annealing ("quenching") and re-annealing in order to find a near optimal solution to the problems being solved,
however, proven optimality cannot be obtained. The search proceeds in feasible space throughout the search process. At each iteration, a number of solution states are generated until one satisfies the problem's constraints.

ASA has a large range of parameter options that can be set. However, the code has the ability to adjust these parameters in a systematic manner throughout the search process (hence the term adaptive). ASA is best suited to solving problems in which the cost function is non-linear, non-convex and has continuous ranging decision variables. As such, ASA has found application in a diverse field of disciplines, including combat analysis, neuroscience and finance (Ingber 1993) rather than pure combinatorial problems.

### 2.3.2.3 Skeleton GA Implementations

Skeleton GA implementations have the components of genetic search inbuilt into a source code library. These components typically include selection, mutation and recombination procedures. Standard parameters such as population size and mutation rate are also adjustable. Two notable skeleton GA codes are GENEsYs by Bäck (1992) and GENESIS by Grefenstette (1987).

As GAs have a standard set of genetic transition operators (i.e. crossover and mutation), only a single objective function and problem data (for instance the distance matrix for the TSP) need to be specified for each problem. However, as GAs are best suited to solving optimisation problems that can be formulated without using constraints, skeleton code implementations generally require that the constraints be incorporated into the objective function as penalty terms. This approach is adopted as the standard genetic operators produce solutions without regard to feasibility (see Section 2.2.3.5). This often leads to the situation in which "finding a feasible point is almost as difficult as finding the best" (Goldberg 1989, p. 85) for COPs. As a result of this, the construction of general GA implementations for solving arbitrary COPs has been hampered and is reflected by Beasley and Chu (1997b) who state that in order to solve COPs effectively, problem specific genetic operators must be incorporated into the GA platform.

### 2.4 Summary

This chapter reviewed modelling systems and search algorithms for COPs as well as examining some existing general meta-heuristic implementations, in order to establish a base for further development of general-purpose meta-heuristic search platforms.

The traditional way of expressing COPs mathematically has been to use a sparse $0-1$ ILP vector formulation, though there are other approaches such as graph theoretic and CP models. In the 0-1 ILP model, variables take on either a value of 0 or 1 to denote assignment of items to groups. This is a very
general and flexible approach that has been quite successful for a range of ILPs, particularly for finding lower and upper bounds on the optimal solution. However, in some cases it is difficult to map the structure of a COP onto a $0-1$ space, resulting in a sparse problem model that contains a large number of variables and constraints. This is especially evident for problems with a complex underlying structure such as the travelling salesman problem (TSP) in which a set of cities are visited but only with a Hamiltonian tour.

Search algorithms may be broadly classified into exact and approximate approaches. The first consists of techniques from OR and AI. These techniques (such as branch and bound and branch and cut for OR; and CP and $\mathrm{A}^{*}$ for AI ) aim to produce solutions that have proven optimality. While COPs can be solved in this way, many are NP hard and as such can require exponential computational time to solve to proven optimality. An alternative group of techniques, known as heuristics, take a different approach to solving COPs. Instead of seeking proven optimality, they endeavour to find good (nearoptimal) solutions in reasonable amounts of computational time. A heuristic is applied to a specific problem and exploits that problem's structural properties in order to find solutions. While these are often very efficient (see for example Lin and Kernighan (1973)), they cannot be readily adapted to other problem types. However, there is a class of heuristic techniques that are applicable across problem types and these are known as the meta-heuristics. Well-known meta-heuristic techniques are Simulated Annealing, Tabu Search, Greedy Search, Greedy Randomised Adaptive Search Procedures, Genetic Algorithms and Artificial Neural Networks.

There have been only a few attempts made in the past to produce implementations of meta-heuristic algorithms that solve arbitrary COPs. This is because it is a difficult task to design and implement appropriate modelling systems and data structures to support general problem solving. It has been determined that there are two basic approaches of building general meta-heuristic search engines, namely: implementations that accept an algebraic model of the problem; and those that require the developer to code information about the problem. The first approach is characterised by Connolly's GPSIMAN (Connolly 1992), while Johnson et al.'s Generic SA (Johnson et al. 1991a, 1991b) and Ingber's ASA (Ingber 1993) are examples of the other category.

It is evident from this chapter that research into meta-heuristic algorithms has concentrated on implementations that solve particular problems, rather than ways of generalising across problem types. This is due to concerns about the efficiency of general implementations and the suitability of existing modelling systems and underlying data structures for the effective representation of these problems. A representation scheme specifically designed for COPs would take into account the fact that these problems require an optimal grouping (or assignment) of discrete objects (Osman and Kelly 1996). The nature of groups, especially using an iterative search algorithm, such as GS, TS or SA, is dynamic and not static. Many existing algebraic representation techniques are based on static memory structures (usually arrays). Therefore, it is believed that dynamic data structures (those that are able to
change with the current state of the solution) could be a viable alternative for modelling COPs. The following chapter investigates this notion.

## Chapter 3: A Modelling System Based on Linked Lists

### 3.1 Motivation

This research is concerned with constructing a general-purpose meta-heuristic solver that deals with a range of problems and delivers good quality solutions in a reasonable amount of computational time. In order to expedite the time it takes to develop appropriate problem descriptions, an algebraic representation for COPs is desirable. The issue then becomes one of choosing an appropriate representation scheme. Each of the methods of representation outlined in Section 2.1 has practical implementation problems associated with it. A suitable representation for COPs should ensure that discrete objects can be grouped, assigned or selected without using a large number of artificial constraints and variables. As any modelling system designed specifically for COPs should allow the evaluation of different assignments of objects to groups, it is believed that dynamic data structures may be suitable for this purpose.

### 3.2 Using Dynamic Data Structures to Represent COPs

As is evident in Section 2.1, the predominant form of representation of solutions to optimisation problems has been with vector and matrix notation. For optimisation problems that have continuous variables (non-COPs), the most efficient form of modelling is the LP (Equations (3) - (5)) as the $A$ matrix and subsequently $B$ vector are dense and hence compact. COPs however, tend to have a sparse $A$ matrix and $B$ vector as a result of the assignment of objects to groups which in turn can require special encoding constraints and variables in conventional notation. As noted in Section 2.1.1, such a sparse representation can lead to extremely large model sizes for relatively small problems, as well as making it difficult for meta-heuristic algorithms to navigate the resulting search space. Therefore, it is believed that COPs can be more effectively represented for meta-heuristics.

Rather than using static data structures to model COPs, this work focuses on the other broad class of data structure, namely the dynamic data structures. These memory structures are not fixed in size and can be altered during the execution of the computer code as the need arises. As such, these data structures are appropriate for systems that dynamically model and test different arrangements of items in a given context (Pohl 1990).

There are a variety of dynamic data structures. The two most common are described below:

- Linked Lists. Lists are dense structures in which elements are linked together in a linear fashion. Lists can shrink and grow by adding and subtracting elements. An element in a list can itself be a list (i.e. a sub-list). Special variations of linked lists include queues, stacks and rings (Pohl 1990). Figure 5 gives a graphical representation of a linked list structure. The list has been shown to be useful for performing general computation as it forms the basis of the computer programming language LISP (LISt Processing) (Barendregt 1984).
- Trees. Like the linked list structure, a tree is a collection of nodes. However, the nodes are arranged in a hierarchical manner in which a node may have a number of branches to child nodes that form sub-trees. A tree is anchored by the root node and terminated at a number of leaf nodes. Figure 5 gives a graphical representation of a tree.


Figure 5: A list and tree structure. A square box is used to denote an element/node while a line is an edge. The traversal of the list structure is indicated by the arrows.

As COPs are primarily concerned with groupings, a structure is needed that ensures group integrity and consistency. From this perspective, a tree structure can be eliminated from consideration as it is used to represent hierarchical information, rather than simple groupings. However, the linked list is appropriate as the use of sub-lists can be used to differentiate groups. Furthermore, each member of a group can therefore be represented directly without the use of encoding constraints (as in the 0-1 ILP and CP systems).

As a result of the mappings being expressed in terms of lists, the elements in the list can take on a range of integer values. Another advantage is that a solution expressed as a list can be altered by using common local search transition operators (these are discussed in Section 2.2.2).

### 3.3 Expressing COPs Using a List Based Notation

### 3.3.1 General Model

In this section, a general model is developed that specifies COPs as dense linked list structures. The list consists of both sub-lists and elements. A sub-list is a list structure within the overall list while an element stores an atomic value. The placement of elements (that contain discrete values) on particular sub-lists or at certain positions on the sub-lists defines the current solution to the problem. To denote that element $e$ is assigned to the $i$ 'th sub-list at the $j$ 'th position, the notation $x(i, j)=e$ is used. For instance, consider a TSP in which city 5 is assigned to position 2 therefore $x(2)=5$. This indicates that city 5 is the second stop on the salesman's tour. In this work, problems that can be expressed with one decision list (denoted by $x$ ) are considered. All other literals in the problem models are considered as scalars, vectors or matrices as appropriate.

The structure of the list is defined for each particular problem and is characterised by:

- number and location of sub-lists within the overall list structure,
- size of the lists (specified in terms of a lower and upper bound),
- the range of values that may appear on a list,
- the number of times that a particular value can appear on a list.

The general list modelling system is stated mathematically in Equations (10) - (16).

$$
\begin{equation*}
\text { Optimise } f(x) \tag{10}
\end{equation*}
$$

s.t.

$$
l h s_{i}(x)\left(\begin{array}{c}
=  \tag{11}\\
< \\
\leq \\
> \\
\geq \\
\neq
\end{array}\right) \text { rhsin} \quad 1 \leq i \leq C
$$

$$
\begin{align*}
& l_{y}\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right)|y|\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right) u_{y}  \tag{12}\\
& \min _{-} \text {count }(y)\left(\begin{array}{l}
= \\
> \\
\geq
\end{array}\right) m_{y}  \tag{13}\\
& \max _{-} \text {count }(y)\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right)  \tag{14}\\
& p_{y} \leq e_{y} \leq q_{y}  \tag{15}\\
& e_{x} \in \mathbb{N} \tag{16}
\end{align*}
$$

Where:
$x$ is the decision list. For instance: $x(i, j)$ is the $j$ 'th element value of the $i$ 'th sub-list. $f(x)$ is the objective function according to decision list $x$.
lhs $i(x)$ is the left-hand side of problem constraint $i$ according to decision list $x$.
$r h s_{i}$ is the right-hand side of problem constraint $i$.
$C$ is the number of problem constraints.
$y$ represents any list structure within list $x$.
$l i l$ is the length of the list defined by $i$.
$l_{y}$ is the minimum length of the list $y$.
$u_{y}$ is the maximum length of list $y$.
min_count $(y)$ and max_count $(y)$ return the number of occurrences of the least frequent and most frequent element value in list $y$ respectively. These functions are discussed in detail below.
$m_{y}$ defines the minimum number of occurrences of each value on list $y$.
$n_{y}$ defines the maximum number of occurrences of each value on list $y$.
$e_{y}$ represents the set of element values contained on list $y$.
$p_{i}$ is the smallest value that can be contained in list $y$.
$q_{i}$ is the largest value that can be contained in list $y$.
$\mathbb{N}$ is the set of Natural Numbers.

The objective function is an arbitrary function over the list $x$ (10). Using this modelling system, constraints are conceptually grouped into two distinct classes, namely the list constraints (constraints that govern the list structure) and the problem constraints (those that define the feasible space of the problem). Equations (12) - (16) in the preceding model refer to the list constraints, while (11) represents the problem constraints (which again are arbitrary functions over the list $x$ ). In general, list constraints are divided into three categories, namely Value Range, Count and List Size.

The legal values of elements in a particular list are discrete and specified between a lower and upper bound. This category is referred to as the Value Range and is in the form of Equations (15) and (16). For instance, if a list stores the order of the cities for an $N$ city TSP, the value range would be defined between 1 and $N$ for list $x$.

The Count constraints regulate the number of occurrences of each value on a list. It is specified using the two functions min_count and max_count (Equations (13) - (14)). min_count(y) returns the cardinality of the element value that appears least on list $y$. For instance, given that a list consists of the elements $\{5,5,6,3,3\}$, min_count returns 1 as the element value 6 appears once in the list. max_count $(y)$ is the opposite is this as it returns the cardinality of the element that appears most on list y. In the previous example, max_count returns 2 . Permutations of values are widely used in COPs and can be specified using the constraints $\min _{-} \operatorname{count}(x)=1$ and max_count $(x)=1$. This combination limits each value to one appearance in the entire list. In the case of the TSP, these constraints only allow tours that do not violate the condition that each city is only visited once. The Count constraint type can eliminate many of the complex encoding constraints required for other systems (especially 0-1 ILPs).

The size and shape of a list structure is specified using Equation of the form (12). This category is referred to as the List Size. Using this notation, an arbitrary list structure can be built as well as allowing the size of a list to be either static or dynamic. If the latter is the case, the size of the list may vary between preset lower and upper limits or without limit on the size.

### 3.3.2 Illustrated Examples of the List Modelling System

In order to demonstrate the representation of COPs using the linked list modelling system, the Generalised Assignment Problem (GAP) (Chu and Beasley 1997; Martello and Toth 1981; Osman 1995) is considered. This problem requires that jobs be assigned to agents subject to capacity constraints. A solution for a 5 agent, 15 job problem appears in Figure 6.


Figure 6: A linked list representation of a GAP.

Therefore job 4 is assigned to agent 3 and job 13 is assigned to agent 5 etc. The objective function is to minimise the total cost of assigning the jobs to the agents and is expressed by (17).

Minimise $\sum_{i=1}^{M} \sum_{j=1}^{\operatorname{Lx(i)}} C(x(i, j), i)$
Where:
$x(i, j)$ is the $j$ 'th job performed by agent $i$.
$C(i, j)$ is the cost of assigning job $i$ to agent $j$.
$M$ is the number of agents.

Whilst this list notation appears similar to the conventional vector form used in standard LPs, each sublist contains a variable number of elements. Thus, the second summation sign in (17) requires a bound which varies depending on the length of each sub-list (i.e. $|x(i)|$ ) and changes according to the current solution state. Similarly, the constraints concerning the capacity of each agent are formed across list space.

$$
\begin{array}{lll}
\sum_{j=1}^{|x(i)|} a(x(i, j), i) \leq b(i) & \forall i & 1 \leq i \leq M \\
& & \\
|x|=M & \forall i & 1 \leq i \leq M \\
1 \leq x(i, j) \leq N & \forall j & 1 \leq j \leq|x(i)| \\
& & \\
\min \_\operatorname{count}(x)=1 & &
\end{array}
$$

Where:
$a(i, j)$ is the resource required by agent $j$ to perform job $i$.
$b(i)$ is the capacity of agent $i$.
$N$ is the number of jobs.

In this example, Equations (19) - (22) are the list constraints while (18) represents the problem constraints.

Using the linked list structure with one level of sub-list as described in the previous section, it is possible to represent a wide range of COPs. Table 1 displays the problems that have been formulated using list notation as part of this research. The set is representative of problems in the literature. The table is divided into four sections, according to the problem's Count constraints. It is found that most COPs fall into one of these categories.

| Count constraint(s) | Problem Name | Reference(s) |
| :---: | :---: | :---: |
| $\begin{aligned} & \min _{-} \operatorname{count}(x)=1 \text { and } \\ & \max _{-} \operatorname{count}(x)=1 \end{aligned}$ | TSP <br> QAP <br> GAP <br> Graph Partitioning Problem <br> (GPP) <br> Graph Colouring Problem <br> (GRAPH) <br> Car Sequencing Problem (CSP) <br> Bin Packing Problem (BIN) <br> VRP <br> Linear Ordering Problem <br> Timetabling Problem (TTP) <br> Uncapacitated Single Allocation <br> p-Hub Median Problem <br> (USApHMP) <br> Capacitated Single Allocation p- <br> Hub Median Problem <br> (CSApHMP) <br> Personnel Time Scheduling <br> Problem <br> Processor Allocation Problem <br> Machine Scheduling / Job <br> Sequencing Problem <br> Single Layout Problems in <br> Flexible Manufacturing | Lawler, Lenstra, Rinnoy and Shmoys (1990) and Reinelt (1991) <br> Burkard, Karisch and Rendl (1997), Connolly (1990) and Nugent, Vollman and Ruml (1968) Chu and Beasley (1997) and Osman (1995) Johnson et al. (1991a) <br> Chams et al. (1987) and Johnson et al. (1991b) <br> Smith et al. (1996a, 1996b) <br> Kampke (1988) <br> Christofides and Eilon (1969), Clarke and Wright (1964), Osman (1993) <br> Chanas and Kobylanski (1995) and Reinelt (1985) <br> Abramson (1991), Abramson and Dang (1993) <br> Ernst and Krishnamoorthy (1996b, 1997b), <br> Skorin-Kapov and Skorin-Kapov (1994) and <br> Campbell (1994) <br> Ernst and Krishnamoorthy (1997a) <br> Krishnamoorthy, Ernst and Beasley (1997) <br> Sofianopoulou (1992) <br> Glover and Laguna (1997) <br> Kouvelis and Chiang (1992) |
| min_count $(x)=0$ and max_count $(x)=1$ | Knapsack / Multiple Knapsack <br> Problem (MKP) <br> Maximum Clique Problem <br> (MCP) <br> Set Partitioning / Covering <br> Problem <br> Aircraft Landing Problem <br> Field Programmable Gate Array placement Problem <br> N Queens Problem <br> Number Partitioning Problem | Beasley and Chu (1997a), Connolly (1992) and Petersen (1967) <br> Battiti and Protasi (1995) and Johnson and Trick (1993) <br> Abramson et al. $(1993,1996)$ and Beasley and Chu (1996, 1997b) <br> Beasley, Krishnamoorthy, Abramson and Sharaihia (1995) and Ernst, Krishnamoorthy and Storer (1997) <br> Chandy and Prithviraj (1996) <br> Sosic and Gu (1991) <br> Johnson et al. (1991b) and Ruml, Ngo, Marks and Shieber (1995) |
| min_count $(x)=1$ and max_count $(x)=$ unbounded | Uncapacitated Multiple Allocation p-Hub Median Problem (UMApHMP) | Ernst and Krishnamoorthy (1996a, 1997b) |
| min_count $(x)=$ unbounded and max_count $(x)=$ unbounded | Cutting Stock Problem | Gilmore and Gommory (1961) and Little and Darby-Dowman (1995) |

Table 1: Problems that can be expressed using the list modelling system.

The following problem models demonstrate the semantics of list notation and are also used to test the performance of the general-purpose system ${ }^{3}$ (see Chapter 5). The problems are a subset of those in Table 1. For the problem types BIN, GRAPH and MCP, alternative list formulations are available and these are given in Appendix B.

A number of auxiliary functions are necessary to express these problems in list notation. These functions are summarised in Table 2.

| Function | Description |
| :---: | :---: |
| $\mathrm{ABS}(i)$ | Returns the absolute value of $i$. |
| $\operatorname{pred}(i, j, k, l)$ | Returns the value $i-l$ unless $i-l<j$ at which $k$ is returned. |
| $\operatorname{succ}(i, j, k, l)$ | Returns the value $i+l$ unless $i+l>k$ at which $j$ is returned. |
| $\min _{i}^{k} f(i)$ | Returns the minimum value of function $f$ between a lowerbound value of $j$ and an upperbound value of $k$. |
| $\max _{i=j}^{k} f(i)$ | Returns the maximum value of function $f$ between a lowerbound value of $j$ and an upperbound value of $k$. |
| $\operatorname{occ}(y, i)$ | Returns the number of times that value $i$ occurs on list $y$. |
| list $(y, i, j)$ | Returns the list number of the $j^{\prime}$ 'th occurrence of the value $i$ on list $y$. |
| $\operatorname{MIN}(a, b)$ | Returns the minimum of $a$ and $b$. |
| $\operatorname{MAX}(a, b)$ | Returns the maximum of $a$ and $b$. |

Table 2: The intrinsic functions available to the list modelling system.

## Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is a facilities assignment problem. Each facility is assigned to a unique location in order to minimise the total intercommunication cost between the facilities. This problem is common in electronics, scheduling, manufacturing and parallel and distributed computing (Burkard et al. 1997; Nugent et al. 1968; Pardalos and Wolkowicz 1994).

```
Minimise \(\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} f(x(i), x(j)) \cdot d(i, j)\)
s.t.
    \(|x|=N\)
    \(1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq N\)
    min_count \((x)=1\)
    max_count \((x)=1\)
```

Where:
$x(i)$ is the facility at location $i$.
$f(i, j)$ is the flow between facility $i$ and $j$.
$d(i, j)$ is the distance between location $i$ and $j$.
$N$ is the number of facilities/locations.

[^2]
## Travelling Salesman Problem

The TSP is a problem in which a salesman visits each of a number of cities exactly once. The salesman starts and ends at a base city and the solution is therefore called a tour. The objective is to minimise the length of the tour that the salesman takes. Whilst there are limited practical applications of the TSP, it is often used as a benchmark COP (Lawler et al. 1990). It is also generalisable to more practical problems such as the vehicle routing problem.

```
Minimise \(\quad \sum_{i=1}^{N} d(x(i), x(\operatorname{pred}(i, 1, N, 1)))\)
```

s.t.

$$
\begin{array}{lll}
|x|=N & & \\
1 \leq x(i) \leq N & \forall i & 1 \leq i \leq N \\
\text { min_count }(x)=1 & & \\
\text { max_count }(x)=1 & &
\end{array}
$$

Where:
$x(i)$ is the $i$ 'th city visited on the tour.
$d(i, j)$ is the distance between city $i$ and $j$.
$N$ is the number of cities.

## Bin Packing

A set of items, each of which has a particular weight, is packed into a number of bins. Each bin has the same weight capacity. The problem can be formulated so that the number of bins is minimised or the excess weight of each bin is minimised. The following model is of the latter.

$$
\left.\begin{array}{lll}
\text { Minimise } & \sum_{i=1}^{M} \operatorname{MAX}\left(0, \sum_{j=1}^{|x(i)|} w(x(i, j))-W_{\max }\right.
\end{array}\right)
$$

Where:
$x(i, j)$ is the $j$ 'th item assigned to bin $i$.
$W_{\max }$ is the maximum bin weight.
$w(i)$ is the weight of item $i$.
$M$ is the number of bins.
$N$ is the number of items.

## Graph Colouring

Given a graph $G=(V, E)$, a colour is assigned to each vertex, such that the colour of the vertex is different to that of its neighbours (those vertices to which it is connected by an edge). The problem can be formulated so that either the number of colours is minimised or the number of neighbour violations for a given set of colours is minimised. The following model is of the latter.

```
Minimise \(\sum_{i=1}^{N} \sum_{j=1}^{|x(i)-1|} \sum_{k=j+1}^{|x(i)|} e d g e(x(i, j), x(i, k))\)
s.t.
    \(|x|=N\)
    \(|x(i)| \geq 1 \quad \forall i \quad 1 \leq i \leq N\)
    \(1 \leq x(i, j) \leq M \quad \forall i \quad 1 \leq i \leq N\)
    \(\forall j \quad 1 \leq j \leq|x(i)|\)
    min_count \((x)=1\)
    max_count \((x)=1\)
```

Where:
$x(i, j)$ is the $j$ 'th node assigned to colour $i$.
edge $(i, j)$ is 1 if there is an edge between vertex $i$ and $j, 0$ otherwise.
$N$ is the number of colours available.
$M$ is the number of vertices.

## Uncapacitated Single Allocation p-Hub Median Problem

The USApHMP is a member of p-median hub allocation problems (Ernst and Krishnamoorthy 1996b, 1997b; Skorin-Kapov and Skorin-Kapov 1994). In this problem, a routing network needs to be designed to allow a commodity flow between each pair of nodes. As it is expensive to route commodities directly from the source node to the destination node, a subset of nodes (called hubs) are used to consolidate the flows into larger flows that can be handled more economically. Each non-hub node is connected to a single hub node and the hubs are fully interconnected. The aim is to find a configuration of hub and non-hub nodes that minimises the total cost of the flows between every pair of nodes. This problem has applications in the design of telecommunications, airline passenger and postal delivery networks.

## Minimise

$$
\begin{aligned}
& \sum_{l=1}^{P} \sum_{k=1}^{P} \sum_{j=1}^{\left|x_{l}\right|} \sum_{i=1}^{\left|x_{k}\right|} W(x(k, i), x(l, j)) \cdot(\chi d(x(k, i), x(k, 1))+\tau d(x(k, 1), x(l, 1))+\delta d(x(l, 1), x(l, j))) \\
& \text { s.t. } \\
& |x|=P \\
& |x(i)| \geq 1 \quad \forall i \quad 1 \leq i \leq P \\
& \begin{array}{lll}
1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq P \\
& \forall j & 1 \leq j \leq 1 x(i)
\end{array} \\
& \text { min_count }(x)=1 \\
& \text { max_count }(x)=1
\end{aligned}
$$

Where:
$x(i, j)$ is the $j^{\prime}$ th node on the $i$ 'th hub (Note: $x(i, 1)$ is a hub).
$P$ is the number of hubs.
$N$ is the number of nodes.
$W(i, j)$ is the flow from node $i$ to node $j$.
$d(i, j)$ is the distance from node $i$ to node $j$.
$\chi$ is the collection cost coefficient.
$\delta$ is the distribution cost coefficient.
$\tau$ is the transfer cost coefficient.

## Uncapacitated Multiple Allocation p-Hub Median Problem

Like the USApHMP, the UMApHMP is a member of $p$-median hub allocation problems (Ernst and Krishnamoorthy 1996a, 1997b). In this particular version of the problem, non-hub nodes are allowed to be assigned to more than one hub.

Minimise

$$
\begin{gathered}
\sum_{i=1}^{N} \sum_{j=1}^{N} \min _{k=1}^{\operatorname{crc}(x, i) \operatorname{cec}(x, j)} \min _{l=1} W(i, j) \cdot(\chi d(i, x(l i s t(x, i, k), 1))+\tau d(x(l i s t(x, i, k), 1), x(l i s t(x, j, l), 1))+\delta d(x(l i s t(x, j, l), 1), j)) \\
\\
\text { s.t. } \\
\\
|x|=P \\
|x(i)| \geq 1 \\
1 \leq x(i, j) \leq N \\
\\
\\
\text { min_count }(x)=1
\end{gathered}
$$

Where:
$x(i, j)$ is the $j$ th node on the $i$ 'th hub (Note: $x(i, 1)$ is a hub).
$P$ is the number of hubs.
$N$ is the number of nodes.
$W(i, j)$ is the flow from node $i$ to node $j$.
$d(i, j)$ is the distance from node $i$ to node $j$.
$\chi$ is the collection cost coefficient.
$\delta$ is the distribution cost coefficient.
$\tau$ is the transfer cost coefficient.

## Multiple Knapsack Problem

The multiple knapsack problem frequently arises in resource allocation situations (Beasley and Chu 1997a). Given a project $i$, with profit $c_{i}$, the overall profit of the inclusion of a number of projects is maximised subject to a number of budgetary constraints. This problem is a generalisation of the knapsack problem.

Maximise $\sum_{i=1}^{|x|} c(x(i))$
s.t.

$$
\begin{array}{lll}
\sum_{i=1}^{|x|} a(x(i), j) \leq b(j) & \forall j & 1 \leq j \leq M \\
0 \leq|x| \leq N & & \\
1 \leq x(i) \leq N & \forall i & 1 \leq i \leq|x| \\
\text { max_count }(x)=1 & &
\end{array}
$$

Where:
$x(i)$ is the $i$ 'th project in the project mix.
$c(i)$ is the profit of including project $i$ in the project mix.
$a(i, j)$ is the number of units of resource $j$ required by project $i$.
$b(j)$ is the maximum number of resource units available from resource $j$.
$N$ is the number of projects.
$M$ is the number of different resources.

## School Time Tabling Problem

In this problem, a number of tuples (consisting of teacher, class and room attributes) are scheduled in a fixed number of time slots (periods). The aim is to minimise the number of clashes between tuples. This problem has particular application in primary/elementary school time tabling (Abramson 1991; Abramson and Dang 1993).

```
Minimise \(\sum_{i=1}^{P} \sum_{l=1}^{C} \sum_{j=1}^{|x(i)|-1} \sum_{k=j+1}^{|x(i)|} \operatorname{clash}(t(x(i, j), l)-t(x(i, k), l))\)
s.t.
    \(|x|=P\)
    \(\begin{array}{lll}1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq P \\ & \forall j & 1 \leq j \leq|x(i)|\end{array}\)
    min_count \((x)=1\)
    max_count \((x)=1\)
```

Where:
$x(i, j)$ is the $j$ th tuple in period $i$.
$t(i, j)$ is the $i$ 'th tuple. $j$ refers to the different aspects of the tuple, namely:

- class number $(j=1)$
- teacher number $(j=2)$
- room number ( $j=3$ ).
$N$ is the number of available tuples that are to be scheduled.
$C$ is the number of items in the tuple (three).
$P$ is the number of periods.
clash $(i)=\left\{\begin{array}{l}1, \text { if }(i=0) \\ 0, \text { otherwise }\end{array}\right.$


## Graph Partitioning Problem

Given the graph, $G=(V, E)$, the aim is to find two equal partitions of nodes $(V)$ such that the number of interconnections between the partitions $(E)$ is minimised. This problem has many applications particularly in circuit board design in which off chip connections must be kept to a minimum (Johnson et al. 1991a).

Minimise $\sum_{i=1}^{\frac{N}{2}} \sum_{j=1}^{\frac{N}{2}} \operatorname{edge}(x(1, i), x(2, j))$
s.t.
$|x|=2$
$\begin{array}{lll}|x(i)|=\frac{N}{2} & \forall i & 1 \leq i \leq 2 \\ 1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq 2 \\ & \forall j & 1 \leq j \leq \frac{N}{2}\end{array}$
min_count $(x)=1$
$\max$ _count $(x)=1$
Where:
$x(i, j)$ is the $j$ 'th node in partition $i$.
edge $(i, j)$ is 1 if nodes $i$ and $j$ are connected by an edge, 0 otherwise.
$N$ is the number of nodes.

## Car Sequencing Problem

The CSP is a common problem in the car manufacturing industry and has been studied by Smith et al. (1996a, 1996b). In this problem, a number of different car models are sequenced on an assembly line. The objective is to separate cars of the same model type as much as possible in order to evenly distribute the manufacturing workload.

Minimise $\sum_{i=1}^{M} \sum_{j=1}^{D(i)-1} \sum_{k=j+1}^{D(i)} P(A B S(x(i, k)-x(i, j)), i)$
s.t.
$|x|=M$
$\begin{array}{lll}|x(i)|=D(i) & \forall i & 1 \leq i \leq M \\ 1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq M \\ & \forall j & 1 \leq j \leq D(i)\end{array}$
min_count $(x)=1$
max_count $(x)=1$
Where:
$x(i, j)$ is the order in the sequence of the $j$ 'th car of model $i$.
$P(i, j)$ is the separation penalty for the $j$ 'th model separated by $i$ places in the sequence.
$N$ is the number of cars.
$M$ is the number of models.
$D(i)$ is the number of cars of model $i$ in the sequence.

## Maximum Clique Problem

Given a graph, $G=(V, E)$, the aim is to find the largest clique $V_{c}\left(V_{c} \subseteq V\right.$ ) such that each member of the clique has a common edge with every other member of the clique. The problem has applications in fault tolerance diagnosis, timetabling and printed circuit board testing (Glover and Laguna 1997). The problem can be formulated so that either the number of nodes in the clique is maximised or the number of edge violations is minimised. The following model is of the latter.

Minimise $\frac{M^{2}-M}{2}-\sum_{i=1}^{M-1} \sum_{j=i+1}^{M} e d g e(x(i), x(j))$ s.t.

$$
\begin{array}{lll}
|x|=M & \\
1 \leq x(i) \leq N \\
\text { max_count }(x)=1
\end{array} \quad \forall i \quad 1 \leq i \leq M
$$

Where:
$x(i)$ is the $i$ 'th node assigned to the clique.
$N$ is the number of nodes.
edge( $i, j)$ is 1 if there is an edge between nodes $i$ and $j, 0$ otherwise.
$M$ is the clique size.

From these few examples, it can be seen that the linked list modelling system is capable of representing a variety of common COPs. Whilst not as general as either the ILP or CP languages, the dense linked list structure makes it possible to directly apply well known local search operators (as discussed next).

### 3.4 Search Over Linked Lists

There are a number of possible ways that a list can be altered in order to form a new solution to a particular problem. Portions of the list structure can be shifted to new locations within the list, deleted from the list, or added to the list. A portion may represent an element, group of elements, a sub-list or a group of sub-lists. As well as this, individual values of elements in the list may be altered. Accordingly, there is a great deal of scope and flexibility in the way that new solutions are created.

An attractive feature of using the linked list representation is that well-known local search operators form a subset of all possible transition operators that are available for lists. This is due to the dense nature of the list structure. Therefore, direct application of these operators to the problem model is possible. However, this in turn restricts the type of meta-heuristic search algorithm that can be used with the linked list modelling system to those that are based on local search (i.e. GS, TS, SA and

GRASP). ANNs and GAs cannot directly utilise the linked list modelling systems as they rely on specialised methods of generating solutions to problems (see Sections 2.1.2 and 2.2.3.5 respectively).

### 3.4.1 List-Based Local Search Transition Operators

As a result of using an integer representation, it is possible to apply to the list structure well known local search operators that have successfully been used in other heuristic and meta-heuristic codes, for example: Abramson (1991), Connolly (1990), Chams et al. (1987), Johnson et al. (1991a, 1991b), Kampke (1988), Osman (1993, 1995) and Taillard (1991). There are a variety of ways that lists can be altered in order to form different solutions. However, seven different transition operators that correspond to common local search operators as used in the literature, have been identified as sufficient to navigate the search space of the list-based formulations. Given the wide variety of standard COPs that have converted into this notation (see Table 1), this set appears to be sufficient to find good quality solutions. The operators are:

- Move: An element is moved from one list to the end of another list. This is similar to Osman's shift process (Osman 1993, 1995). See Figure 7.
- Swap: The positions of two elements, from the same or different lists, are swapped. This is equivalent to the Or operator (2-opt) and Osman's interchange process (Osman 1993, 1995) (which is generalisable to the $n$-opt). See Figure 8.
- Inversion: The sequence between two elements on the same sub-list is reversed. See Figure 9.
- Reposition: The position of an element in a list is changed. See Figure 10.
- Add: An element is added to a list. See Figure 11.
- Drop: An element is removed from a list. See Figure 12.
- Change: The value of a list element is changed to another value. See Figure 13.


Figure 7: The move operator.


Figure 8: The swap operator.


Figure 9: The inversion operator.


Figure 10: The reposition operator.


Figure 11: The add operator.


Figure 12: The drop operator.


Figure 13: The change operator.

The transition operators that can be applied to a particular problem are determined by the characteristics of the problem's list based model. The use of appropriate transition operators ensures that the problem's list constraints are not violated. For instance, in a permutation problem, operators that only perturb the ordering of the elements in the list (such as swap, inversion and reposition) are appropriate. A set of rules can be formalised that determine appropriate transition operators given the list constraints (Table 3). Note, as in Section 3.3.1, $y$ represents any list structure within list $x$.

| List Constraints |  | Transition Operators |
| :---: | :---: | :---: |
| Count | List Size Conditions |  |
| min_count $(y)=1$ and max_count $(y)=1$ | $\begin{aligned} & \|y(i)\|=N \quad \forall i 1 \leq i \leq\|y\| \text { or } \\ & \|y\|=N \\ & \|y\|>1 \end{aligned}$ | swap, inversion, reposition <br> swap, inversion, reposition, move |
| max_count $(y)=1$ | $\begin{aligned} & \|y\|=1 \\ & \|y\|>1 \end{aligned}$ | swap, inversion, reposition, change, add, drop <br> swap, inversion, reposition, change, add, drop, move |
| min_count $(y)=1$ or unspecified | $\|y\|=1$ <br> $\|y(i)\|=N \quad \forall i 1 \leq i \leq\|y\|$ $\|y\|>1$ | swap, inversion, reposition, change, add, drop swap, inversion, reposition, change swap, inversion, reposition, change, add, drop, move |

Table 3: Appropriate transition operators given a problem's list constraints. $y$ represents any list structure while $N$ represents a constant.

A transition is made in accordance with the rules of the particular meta-heuristic technique, subject to the list constraints being satisfied. For instance, if an element appears once on list $y$, the constraint min_count $(y)=1$ makes it infeasible to drop this element.

### 3.4.1.1 Enforcing Constraints

Search may proceed either through feasible or infeasible space in order to visit solutions. For list based problems, there are effectively two spaces that are bound by the list and problem constraints respectively. Table 3 in the previous section presented a set of rules for choosing transition operators that do not violate list constraints. However, there is no guarantee that the operators will also preserve the feasibility of the problem constraints. Two strategies are in place in order to ensure problem feasibility is satisfied:

- Feasibility Maintenance
- Feasibility Restoration

The first method only allows transitions that do not violate the problem constraints. For instance, in the GAP, a job is only moved to a new sub-list (i.e. an agent) if that agent has the enough spare capacity to perform that job. Feasibility restoration on the other hand initially performs a transition without regard to the problem constraints and then employs a number of feasibility maintaining transitions in order to obtain problem feasibility once more. The advantage of the former technique is that it is computationally inexpensive to implement. However, feasibility restoration can traverse infeasible space and is required if the problem is tightly constrained and a straight feasibility maintaining transition is not possible.

### 3.4.1.2 Applying the Transition Operators

Using the local search operators described in Section 3.4.1, it is possible to explore multiple neighbourhoods in the course of solving a particular problem. In many cases, not all of the operators (as per Table 3) are required. For instance, one may be interested in the effect of performing only swaps on a TSP. Therefore the unwanted transition operators can be eliminated. The meta-heuristic technique is then performed using only those operators that were not excluded, referred to as the active operators. At each iteration of the search, an operator is chosen from those in the active set. If all but one of the operators is eliminated, this operator is used at each iteration. In the case that there are two or more active operators, each operator is assigned a probability of being selected at each transition ${ }^{4}$ (totaling 1). Equations (23) - (26) can be used to select a neighbourhood operator.

$$
\begin{equation*}
p_{i}=\frac{1}{N} \quad \forall i \quad 1 \leq i \leq N \quad \text { (default) } \tag{23}
\end{equation*}
$$

(alternatively $p_{i}$ can be explicitly specified, however $\sum_{i=1}^{N} p_{i}=1$ )

$$
\begin{equation*}
o p=\left\{i ; l b_{i} \leq c \leq u b_{i}\right\} \quad 1 \leq i \leq N \tag{24}
\end{equation*}
$$

$$
\begin{equation*}
l b_{i}=\sum_{j=1}^{i-1} p_{j} \quad \forall i \quad 1 \leq i \leq N \tag{25}
\end{equation*}
$$

$$
\begin{equation*}
u b_{i}=\sum_{j=1}^{i} p_{j} \quad \forall i \quad 1 \leq i \leq N \tag{26}
\end{equation*}
$$

Where:
$N$ is the number of active transition operators.
$o p$ is the selected transition operator.
$c$ is a uniform random number in the range $[0,1]$.
$l b_{i}$ is the lower probability bound of the active transition operator $i$.
$u b_{i}$ is the upper probability bound of the active transition operator $i$.
$p_{i}$ is the probability of the active transition operator $i$ being selected.

Equation (23) assigns an equal probability to each of the active transition operators. For instance, if there are two operators, each one receives a probability of 0.5 . Alternatively, the probability of each of the operators may be individually specified. The probability space is partitioned into a number of sections (using these probabilities as bounds) that represent each of the transition operators (Equation (25) - (26)). A uniform random number is then generated in order to select the next transition operator (Equation (24)). This is like the weighted probability wheel used in roulette wheel selection by Goldberg (1989). For example, if the operator move is given the probability region $[0,0.5]$ while swap

[^3]is assigned $(0.5,1]$ and 0.6 is the uniform random number ( $c$ ), swap is chosen as the next transition operator.

### 3.4.1.2.1 The Adaptive Probability Model

The set of transition operator probabilities can be either static throughout the search or vary dynamically according to the relative performances of the active transition operators (in terms of optimising the objective function). Thus if an operator is producing solutions of high quality, its probability is raised. Similarly, the probability is dropped for a poorly performing operator. The amount of the change is dependent on a parameter known as the adaptive rate. This is similar to the learning rate employed by ANNs (Carling 1992). If the adaptive rate is set low, the change in probabilities will also be conservative and vice versa.

The probabilities are adapted at set intervals throughout the search process. Equations (27) - (30) show how the probabilities are adapted.

$$
\begin{array}{ll}
d c_{i}=\sum_{j=k}^{N+k-1}\left\{\begin{array}{cc}
i f(\operatorname{tran}(j)=i), \text { cost }_{j}-\operatorname{cost}_{j-1} \\
0, \text { otherwise }
\end{array}\right. & \forall i 1 \leq i \leq M \\
r d_{i}=\frac{d c_{i}}{\sum_{j=1}^{M}\left|d c_{j}\right|} & \forall i 1 \leq i \leq M \\
p a_{i}=p_{i}\left\{\begin{array}{l}
- \\
+
\end{array}\right\} p_{i} \cdot r d_{i} \cdot \alpha & \forall i 1 \leq i \leq M \\
p_{i}=\frac{p a_{i}}{\sum_{j=1}^{M} p a_{j}} & \forall i 1 \leq i \leq M
\end{array}
$$

Where:
$|i|$ is the absolute value of $i$.
$M$ is the number of active transition operators.
$N$ is the number of iterations between adaptations.
$\alpha$ is the adaptation rate (i.e. $\alpha=0.01$ (very slow adaptation), $\alpha=0.9$ (very fast adaptation)).
$d c_{i}$ is the summed incremental cost of the $i$ 'th transition operator.
$k$ is the iteration number of the first iteration in the adaptation period. For instance, if $N=100$,
$k$ would take on successive values of $1,101,201, \ldots$.
$\operatorname{tran}(i)$ is the transition operator used at iteration $i$.
$r d_{i}$ is the ratio incremental cost of the $i$ 'th transition operator.
$p a_{i}$ is the new raw probability of the $i$ 'th transition operator.
$p_{i}$ is the scaled probability of the $i$ 'th transition operator.
cost $t_{i}$ is the objective cost obtained at iteration $i$.

Equation (27) calculates the total change in cost for each of the active transition operators and (28) normalises these values between 0 and 1 . The new probability set is computed based on the adaptive rate (29) and this is in turn converted to a new set of probability values (30).

### 3.4.2 Calculating Incremental Costs

The objective function of many COPs is costly to compute, especially for realistic size problems. The linked list structure makes it possible to compute an incremental cost based on the elements that have changed in the list. This can result in substantial computational savings, particularly for iterative algorithms such as GS, TS and SA.

Algebraic incremental cost expressions can be developed for problems modeled in linked list notation. As each of the local search operators alters the solution in a different way, a number of incremental cost expressions may be required for the one problem type. There are two methods of making use of incremental costs in a list-modelling environment:

- Automatically generate incremental cost expressions given an objective function and set of transition operators.
- Manually develop incremental cost expressions.

It can be difficult to automatically generate incremental cost expressions for problems modeled in a list notation using the first approach. This is due in part to a) incremental cost expressions being dependent on the transition operator used, b) many objective functions in a list based modelling environment are non-linear. In contrast, to calculate an incremental cost for a 0-1 ILP does not require a specific incremental cost expression, as the coefficients of the variables that are changed need only to be either subtracted or added appropriately. This is a disadvantage of a general modelling solver system based on a compacted representation such as linked lists. As a result, it is difficult to produce a general algorithm that generates incremental cost expressions from an arbitrary objective function.

In order to produce a practical means of automatically generating incremental cost expressions, a slightly less general approach is adopted. This method uses a system that matches the objective function against one of a number of templates. The rationale behind this approach is that many list based objective functions are very similar. For instance, consider the objective functions of graph colouring with the processor allocation problem (see Appendix B) and the knapsack (see Appendix B) / multiple knapsack with the set partitioning problem / set covering problems (see Appendix B) as examples of this. The template method therefore attempts to fit an objective function to a known template and produce incremental cost expressions for the set of feasible transition operators. Equations (31) - (36) present a set of templates that are typical of objective functions formulated in list notation.

$$
\begin{align*}
& \sum_{i=1}^{|x|} f(x(i))  \tag{31}\\
& \sum_{i=1}^{|x|} \sum_{j=1}^{|x(i)|} f(x(i, j), i)  \tag{32}\\
& \sum_{i=1}^{|x|-c} f(x(i), x(i+c))  \tag{33}\\
& \sum_{i=1}^{|x|-1} \sum_{j=i+1}^{|x|} f(x(i), x(j))  \tag{34}\\
& \sum_{i=1}^{|x|-1} \sum_{j=i+1}^{|x|} f(x(i), x(j)) \cdot d(i, j)  \tag{35}\\
& \sum_{i=1}^{|x|} \sum_{j=1}^{|x|=1} \sum_{k=j+1}^{|x|} f(x(i, j), x(i, k)) \tag{36}
\end{align*}
$$

In order to accommodate a wide variety of objective functions, simple transformations of these functions are also admissible (Equations (37) - (41)). For instance, Equation (37) indicates that the system can derive suitable incremental cost expressions for a function of a list that is multiplied by some constant. An example of this might be a transformed GAP having the objective function $\sum_{i=1}^{M} \sum_{j=1}^{|x(i)|} C(x(i, j), i) \cdot c$ where $c$ is a constant. Similarly, objective functions
$\sum_{i=1}^{M} \sum_{j=1}^{|x(i)|} C(x(i, j), i)+g(i)$ (given by Equation (39)) can also be processed by the system.

$$
\begin{align*}
& f(x(\ldots)) \cdot c  \tag{37}\\
& f(x(\ldots)) \pm c  \tag{38}\\
& f(x(\ldots)) \pm g(\ldots)  \tag{39}\\
& f(x(\ldots)) \pm g(x(\ldots))  \tag{40}\\
& f(x(i \pm c)) \tag{41}
\end{align*}
$$

A given set of templates cannot support every conceivable objective function. This set of templates covers all of the problems from Section 3.3 .2 except USApHMP, UMApHMP and the timetabling
problem. Incremental cost expressions can be calculated manually by considering the effect of the application of the transition operator to the cost function and list structure. In some instances, it may not be possible to calculate an appropriate expression because of the complexity of the cost function, in which case the original cost function is the only expression that can be used. Section 4.5 outlines how these templates ( $31-35$ ) can be applied to produce appropriate incremental cost expressions.

### 3.5 Summary

This chapter has described a general modelling system for COPs that allows solution-states to be expressed using dynamic list structures. Unlike more conventional modelling systems (such as ILPs and CP ), a list-based solution consists of a densely packed set of integers that dynamically changes according to the current solution-state. The feasible solution space of problems formulated using the list notation is governed by two classes of constraints; the list constraints and the problem constraints. The list constraints consist of restrictions to solutions that include: the range of values that can occur on the list; the number of times that a particular value can appear on a list; and the shape and size of the list structure. In many cases, these constraints alone are sufficient to express a wide range of COPs, some of which include the well-known TSP and QAP. Additional problem-specific constraints (like the capacity constraints for the GAP and the budgetary constraints for the MKP) can also be modeled via the problem constraints.

Based on the configuration of the list constraints for a particular problem model, suitable local search operators can be deduced automatically. As more that one operator may be suitable for a problem, a probabilistic procedure for choosing a transition operator at each iteration of a meta-heuristic search algorithm has been proposed. In this method, each operator is assigned a probability of being chosen and is subsequently selected using a variation of the weighted roulette wheel notion introduced by Goldberg (1989). The system can also adaptively set the probabilities according to the performance of each of the operators.

Objective functions for practical size COPs can be costly to evaluate. However, using the linked list modelling system, appropriate incremental cost expressions can be produced. These expressions are calculated based on the elements that have recently changed in the solution and their use can represent a substantial efficiency increase for the meta-heuristic search algorithm. An automatic system that matches an objective function to one of a set of templates to produce incremental cost expressions has been devised. While not entirely general, this system can generate appropriate incremental cost expressions for a wide range of COPs. This is due to the similarity of the objective functions of various COPs and the fact that simple transformations of the template set can be accommodated.

In order to gauge the benefits of modelling COPs using linked lists, the following chapter describes the issues relating to a software implementation of the system. As many COPs can be expressed with one

## Chapter 4: Implementation of a General Solver

This chapter discusses a number of issues related to producing a general-purpose solver for COPs based on the linked list modelling system. These topics include: the design of a text-based language for describing arbitrary list models, a method of producing initial feasible solutions, a feasibility restoration scheme, a candidate list strategy for TS and GS, appropriate incremental cost templates; and the implementation of the SA, TS and GS search engines. These techniques are designed to operate within a restricted general list modelling system that is described in Section 4.8.

The work on the linked list modelling system has culminated in the design of a software implementation named COSULOM (Combinatorial Optimisation System Using List Oriented Metaheuristics). COSULOM is a real world solver written in ANSI C so that it is available to a wide range of computer platforms. The TS component is also available as a parallel program for MIMD (Multiple Instructions, Multiple Data) computers supporting the MPI (Message Passing Interface) library.

### 4.1 System Architecture

The solver performs a number of tasks in order to progress from an initial problem model to a final solution. Figure 14 gives an architectural description of the implementation of the system.


Figure 14: An architectural representation of the general solver.

A file containing a problem model is read by the compiler that converts this into a set of C files. These C files contain the code that represents the objective function, list model, constraints and problem data. The algebraic language that describes the list models is given in Section 4.2. The compiler also
produces a mathematical list-based model in LaTeX 2 e format (using the algebraic notation outlined in Section 3.3). The files are then linked with the core code modules to produce an executable file based on the selected meta-heuristic technique.

The runtime system has two main tasks: to form an initial solution to the problem as well as running the appropriate meta-heuristic search in order to produce a final solution to the problem being solved. The process of forming an initial solution is described in Section 4.7.

### 4.2 Algebraic Modelling Language

It can take considerable time to develop either a meta-heuristic based computer program or a suitable code segment (as discussed in Section 2.3.2) compared with an algebraic model. This is because code development is a complex process that requires planning, implementation and testing phases. However, a high level algebraic description (such as the one described herein) is typically less costly to develop and not as prone to error.

The general system accepts text files that contain a list based algebraic description of the problem and automatically converts these to computer code. This approach is more efficient than interpreting data structures that represent the objective and constraint functions. In addition, customised code modules can be used instead of the code generated from the input file compiler (a description of these modules is given in Appendix E). This flexibility allows the system to accept arbitrary functions that are difficult to write algebraically.

The algebraic modelling language is based loosely on the GAMS syntax, which is widely used and well understood. The reader unfamiliar with the GAMS syntax is referred to Brooke et al. (1997). The language can be used to express list models as described in Chapter 3. While the syntax has been borrowed, semantic modifications have had to be made in order to account for the list modelling system. This is particularly apparent in the SOLUTION section that incorporates the list constraints. Files can incorporate both data (for instance the distance matrix for the TSP) as well as objective and constraint functions. Alternatively, data and expressions can be contained in separate files.

A user manual describing the syntax, layout and available functions of the language is given in Appendix D. Nevertheless, an example of a file that describes a small GAP is presented in Figure 15. This file is divided into a number of distinct sections. The SCALAR section specifies problem constants typically related to problem size. In this case, there are 5 agents (denoted M ) and 15 jobs (denoted N). The SETS section allows range variables to be specified, i.e. $J$ ranges in value from 1 to $M$ (across the agents). The PARAMETER and TABLE sections contain the data for the problem instance. For the GAP, a cost matrix $C$, resource matrix $A$ and an agent capacity vector $B$ are required. The SOLUTION section is used to describe the list model and constraints. $\operatorname{SIZE}(X)=M$ denotes that there
are $M$ sub-lists while MIN_COUNT $(X)=1$ and MAX_COUNT $(X)=1$ indicates that each element can appear only once on the list. To specify that the value range is between 1 and N , the statement $1<=\mathrm{X}<=\mathrm{N}$ is used. Finally the EQUATIONS section allows the objective (specified using the COST keyword), constraints and incremental costs to be specified. The capacity constraints are formed for each agent ( J ). It is important to note that in this system, the user specifies whether the objective function is minimised or maximised at runtime.

```
SCALAR
    N /15/
    M /5/;
SETS
    J /1-M/;
PARAMETER
    B(M) / 36 34 38 27 33 /;
TABLE
    C(M,N)
        / 17 21 22 18 24 15 20 18 19 18 18 16 22 24 24 16
        23}16
```



```
        19}19192222 20 16 19 17 21 19 25 23 25 25 25,
        18}191915 15 21 25 16 16 23 15 22 17 19 22 24 /
    A (M,N)
    / 8 15 14 23 8 16 8 25 9 17 25 15 10 8 24
        15}7723~22 111 11 12 10 17 16 7% 16 10 18 22,
        21}2006[\begin{array}{llllllllllllllll}{124}&{24}&{10}&{24}&{9}&{21}&{14}&{11}&{14}&{11}&{19}&{16}
        20}11
        8 13 13 13 10 20 25 16 16 17 10 10 5 5 12 23 /;
SOLUTION
        SIZE(X)=M
        MIN_COUNT (X)=1
        MAX_COUNT (X)=1
        1<=\overline{X}<=N;
EQUATIONS
    COST.. SUM(SUM(C(J,X(J,K)),K=1,SIZEOF(X(J))),J)
    CAPACITY: SUM(A(J,X(J,K)),K=1,SIZEOF(X(J)))<=B(J);
SOLVE USING SA
```

Figure 15: Input file representing a GAP.

In order to describe the algebraic modelling formally, a BNF (Backaus-Naur Form) description is provided in Figure 16.

```
<problem_description>}\leftarrow{<\mathrm{ scalar_section> > 0-1 {<set_section>} 0-1 {<parameter_section>} }0-1
{<table_section>} {-1 <solution_section> {<equations_section>} }-1< < solve_section>
<scalar_section>\leftarrow SCALAR {<scalar>} } ; ;
<scalar>& <identifier> / <number> /
<sets_section>\leftarrow SETS {<set>} \mp@subsup{1+}{+}{*}
<set>\leftarrow<<identifier> / <constant> - <constant> /
<parameter_section>\leftarrow PARAMETER {<parameter>} } ; ;
<parameter>}\leftarrow<\mathrm{ <identifier> (<constant> )/ <number_set>/
<table_section>\leftarrow TABLE {<table> } 1+ ;
<table>\leftarrow <identifier> ( <constant> , <constant> ) / <number_set> /
<number_set>\leftarrow <number> {,<number> } O+
<solution_section>\leftarrow SOLUTION <list_constraints> ;
<list_constraints>\leftarrow{<<size_constraints> } }-1 {<count_constraints>} (0-1 <value_range_constraint>
<size_constraint>}\leftarrow{<constant><<relational_operator>\mp@subsup{}}{0-1}{}\mathrm{ SIZE (<indentifier> {<constant>} }0-1 )
{<constant><relational_operator> }0-1
                                    Continued...
<count_constraints>\leftarrow {<count_type> (<identifier> ) <count_rhs> } }-2
<count_type>\leftarrow MIN_COUNT | MAX_COUNT
<count_rhs>\leftarrow=1
```

```
<value_range_constraint>\leftarrow<constant> <relational_operator><identifier><relational_operator>
<constant>
<equations_section>\leftarrow<<objective_function>{<incremental_cost_objective> > } <+
{<constraint_function> } } ;
<objective_function>\leftarrow COST.. <expression>
<expression>\leftarrow<expression_component> | (<expression> ) | <expression><arithmetic_operator>
<expression>
<expression_component>\leftarrow <sum> | <constant> | <list_or_array_reference> | <list_size> | <min> 1
<max> <<minimum>|<maximum>| <abs>
<sum>\leftarrowSUM (<expression>,<range>)
<min>\leftarrow MIN (<expression> , <range>)
<max>\leftarrow MAX (<expression> , <range>)
<minimum>\leftarrowMINIMUM(<expression>, <expression>)
<maximum>\leftarrowMAXIMUM( <expression> , <expression> )
<range>\leftarrow <identifier> | <indentifier> = <expression> , <expression>
<abs>\leftarrow ABS (<expression>)
<list_size>\leftarrow SIZEOF ( <identifier>{ (<constant>) } } )
<incremental_cost_objective>\leftarrow DCOST (<transition_operator> ).. <incremental_cost_expression>
<transition_operator>\leftarrow move | swap | inversion | reposition | add | drop | change
<incremental_cost_expression>}\leftarrow<\mathrm{ <expression> | <pred_statement> | <succ_statement> |
<incremental_cost_attribute> | (<incremental_cost_expression> ) | <incremental_cost_expression>
<arithmetic_operator> <incremental_cost_expression>
<pred_statement>\leftarrow PRED (<identifier> ,<constant> ,<constant> ,<constant> )
<succ_statement>\leftarrow SUCC (<identifier>, <constant> ,<constant> , <constant> )
<occurrence_statement>\leftarrowOCC(<identifier>,<constant>)
\ll l i s t \_ s t a t e m e n t > \leftarrow L I S T ( < i d e n t i f i e r > , < c o n s t a n t > , < c o n s t a n t > ) ~
<incremental_cost_attribute>\leftarrow & E1 |E2|L1 | L2 |P1 |P2
<constraint_functions>\leftarrow<<identifier> : <constraint>
<constraint>\leftarrow <expression><relational_operator><expression>
<solve_section>\leftarrow SOLVE USING <search_engine>
<search_engine>\leftarrowSA | GS | TS
<relational_operator>\leftarrow<= |< = |> |>= |<>
<constan\>\leftarrow <identifier> | <number>
<number>\leftarrow <float> | <integer>
<float>\leftarrow{<<digit> } + . {<digit> } }\mp@subsup{1}{+}{
<integer>}\leftarrow{<\mathrm{ digit> } }\mp@subsup{1}{+}{
<digit>\leftarrow0|1|2|3|4|5|6|7|8|9
<identifier>\leftarrow < any alphabetic character }}\mp@subsup{}}{1+}{
Where:
\(\boldsymbol{x}\) is a terminal.
\(<x>\) is a non-terminal.
\(\{x\}_{a-b} x\) is present between \(a\) and \(b\) times.
\(\{x\}_{a+} x\) is present \(a\) times or greater.
```

Figure 16: BNF of the algebraic modelling language.

### 4.3 Feasibility Maintenance and Restoration

A feasibility maintenance operation consists of a simple application of the transition operator. For neighbourhood search techniques such as GS and TS, all possible applications of the operator are examined and only the ones that satisfy both the list and problem constraints are considered. The feasibility maintenance scheme for SA uses a first admissible strategy (Osman 1993, 1995) in which a number of transitions of the same type are attempted until one satisfies both the list and problem constraints. Consider the GAP example; each element is moved from its sub-list to another sub-list in turn until one of these transitions produces a new feasible solution. The order in which candidate transitions are performed is permuted so that the search process sufficiently samples the neighbour space. This technique has been found to be effective by Connolly (1990, 1992), Johnson et al. (1991a, 199b) and Abramson and Randall (1998).

As discussed in Section 3.4.1.1, the aim of feasibility restoration is to traverse infeasible space in order to find another feasible pocket of solutions that may be inaccessible otherwise. It is possible to apply feasibility restoration techniques to both the list constraints and problem constraints. In this implementation, only the operation regarding the latter type is considered.

The process of feasibility restoration consists of first breaking and then repairing feasibility. A transition is made that potentially violates some or all of the problem constraints. A series of feasibility maintenance transitions are subsequently undertaken in order to return to a feasible state.

Feasibility restoration schemes differ for each transition operator. The change operator can be seen as a simple implementation of feasibility restoration for add/drop transitions as an element is added to a list and another is dropped. Performing feasibility restoration for the swap, inversion and reposition operators can be more complex as the original transition will perturb multiple elements of the list and hence may require a great deal of computational effort to restore to a feasible state. The principal of feasibility restoration is therefore most effectively demonstrated with a less complex operator, the move (Algorithm 7). In the context of this operator, the algorithm is useful for problems such as the GAP and VRP in which elements can be moved between lists.

```
Move an element to another sub-list;
Check the constraints associated with the old sub-list;
If (these constraints are violated)
    Attempt to add a combination of elements on this sub-list;
    If (this fails) try to do the same thing except remove the
    elements off the old sub-list;
    If (the old sub-list is now feasible) perform
    feasibility maintenance with the displaced elements;
    If (this fails) abort the restoration, reinstate the
    original solution and exit;
End if;
Check the constraints associated with the new sub-list;
If (these constraints are violated)
    Attempt to add a combination of elements from this list;
    if (the new sub-list is now feasible) perform
    feasibility maintenance with the displaced elements;
    If (this fails) try to add to this sub-list a combination of
    elements;
    If (this fails) abort the restoration, reinstate the
    original solution;
End If;
End.
```

Algorithm 7: Feasibility restoration procedure for the move operator.

In the above algorithm, a combination of elements is either removed from the sub-list or added to the sub-list from other sub-lists in order to make the constraints associated with the original sub-list feasible. For practical purposes, the number of elements that constitute the combination is bounded by a constant value. As with feasibility maintenance, the search for a combination terminates when one that satisfies the problem constraints associated with the sub-list is found.

### 4.4 Probabilistic Candidate List Strategy

Unlike SA, both GS and TS systematically examine the neighbourhood of the current solution to find a replacement solution. Many practical size problems have large neighbourhoods. For instance, if a swap is performed on a TSP with $n$ cities, each solution would have $\frac{n(n-1)}{2}$ neighbours. Therefore the search process can become slow if the neighbourhood is large. In order that large problems can be solved in reasonable amounts of time, the system has a parameter that controls the amount of the neighbourhood that is searched. This is achieved by using a subset $N_{s}$ of the neighbourhood $N$ (i.e. $N_{s}$ $\subseteq N$ ) and is referred to as the candidate list strategy (as outlined by Glover and Laguna (1997)). There are a number of ways in which this strategy can be defined (Glover and Laguna 1997) such as choosing every $k$ 'th neighbour so that $\left|N_{s}\right|=N / k$. This implementation uses a probabilistic approach. For instance, if the probability is set to 1 , then all neighbours are tested, however, if the probability is set to 0.5 , then each neighbour has a $50 \%$ chance of being evaluated. This is accomplished by using the rule in (42).

Where:
$c$ is a uniform random number in the range $[0,1]$.
$p$ is the probability that a neighbour is evaluated. $p$ is in the range $[0,1]$.

### 4.5 Incremental Cost Templates

As discussed in Section 3.4.2, the template method matches an arbitrarily formed objective function with one of a number of available templates in order to produce an algebraic expression to calculate the incremental cost. The set of possible templates for COPs is large and therefore this implementation only uses a subset of these (as specified previously in Equations (30) - (36)).

If the system cannot recognise a given combination of template and transformation, it reports that it is unable to produce an incremental cost expression. The user can nevertheless manually calculate the appropriate incremental cost expression and code it in the algebraic modelling language described in Section 4.2 or directly in the C programming language. Descriptions of how incremental cost expressions are coded in the algebraic modelling language as well as directly in the C description files are given in Appendix D and Appendix E respectively. The advantage of this system is that templates can be easily added, should the need arise.

In order to produce an incremental cost expression from a given template and transition operator, information about the type of change that has occurred to the list structure is needed. For instance, if an item is added to the solution list and the objective function is matched to template 1 (Equation (31)), the value of the item $(e)$ is used to form the incremental expression, $+f(e)$. Apart from element value, sub-list and position number are also important indicators of change. The following description shows how this information is used for each of the transition operators (taking into account Figure 7 to Figure 13).

Move: An element with value $e_{l}$ is removed from $x\left(l_{l}, p_{l}\right)$ and placed at $x\left(l_{2}, p_{2}\right)$.
Swap: An element with value $e_{l}$ located at $x\left(l_{l}, p_{l}\right)$ is swapped with an element with value $e_{2}$ located at $x\left(l_{2}, p_{2}\right)$.
Inversion: The elements between $x\left(l_{1}, p_{1}\right)$ (with value $e_{I}$ ) and $x\left(l_{2}, p_{2}\right)$ (with value $e_{2}$ ) are inverted.
Reposition: The element with value $e_{I}$ is removed from $x\left(l_{l}, p_{l}\right)$ and inserted at $x\left(l_{1}, p_{2}\right)$.
Add: An element with value $e_{l}$ is added at $x\left(l_{l}, p_{l}\right)$.
Drop: An element with value $e_{l}$ is dropped from $x\left(l_{l}, p_{l}\right)$.
Change: An element with value $e_{1}$ located at $x\left(l_{l}, p_{1}\right)$ changes its value to $e_{2}$.

Table 4 gives the output of templates with transition operators using the notation described above. For example, consider Template 2 given by Equation (32) in Section 3.4.2. Template 2 represents objective functions that sum all of the elements in the list structure according to a function $f$. Each transition operator affects the solution differently and subsequently a unique incremental cost
expression is produced. For the move operator, an element is deleted from one sub-list and placed at the end of another. Hence the incremental cost expression becomes $c_{i}=c_{i-1}-f\left(e_{l}, l_{l}\right)+f\left(e_{l}, l_{2}\right)$ where $c_{i}$ is the cost of the objective functions at the current iteration and $c_{i-1}$ denotes the previous iteration. The swap operator involves more calculations as two elements are dropped and again added to new sublists. Conversely, transition operators that change the order of elements within a sub-list do not affect the overall cost (as addition is commutative). Hence for the inversion and reposition operator slots in the table, None is given (i.e. $c_{i}=c_{i-1}$ ).

The effect of each of the transition operators on each of the templates is considered in Table 4. The abbreviations used in this table are:

- $N P$ - it is not possible to perform the indicated transition on the template. For example, if a template has only one sub-list (such as Template 1), it is not possible to use a transition operator that involves two sub-lists (such as move).
- $N W$ - the resulting incremental cost expression has a greater complexity that the original expression.
- None - there is no resulting change in the objective cost for this template with this transition operator.

|  | Table of Incremental Cost Expressions |  | inversion | reposition | add | drop | change |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | move | swap |  |  |  |  |  |
| Template 1  <br> (31)  | $N P$ | None | None | None | $+f\left(e_{1}\right)$ | $-f\left(e_{1}\right)$ | $-f\left(e_{1}\right)+f\left(e_{2}\right)$ |
| $\begin{array}{ll} \text { Template } \\ \text { (32) } \end{array}$ | $-f\left(e_{l}, l_{l}\right)+f\left(e_{l}, l_{2}\right)$ | $\begin{aligned} & -f\left(e_{1}, l_{1}\right)+f\left(e_{1}, l_{2}\right)- \\ & f\left(e_{2}, l_{2}\right)+f\left(e_{2}, l_{1}\right) \end{aligned}$ | None | None | $+f\left(e_{l}, l_{l}\right)$ | - $f\left(e_{1}, l_{l}\right)$ | $-f\left(e_{1}, l_{l}\right)+f\left(e_{2}, l_{l}\right)$ |
| Template $\quad 3$ (33) (Note: $p_{1}<p_{2}$ ) | None | $\begin{aligned} & -f\left(x \left(\operatorname { p r e d } \left(p_{l}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{l}\right)- \\ & f\left(e_{l}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\|x\|,\right.\right.\right. \\ & c)))- \\ & f\left(x \left(p r e d \left(p_{2}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{2}\right)- \\ & f\left(e_{2}, x\left(\operatorname { s u c c } \left(p_{2}, 1,\right.\right.\right. \\ & \|x\|, c)))+ \\ & f\left(x \left(p r e d \left(p_{2}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{l}\right)+ \\ & f\left(e_{l}, x\left(\operatorname { s u c c } \left(p_{2}, 1,\right.\right.\right. \\ & \|x\|, c)))+ \\ & f\left(x \left(\operatorname { p r e d } \left(p_{l}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{2}\right)+ \\ & f\left(e_{2}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\right.\right.\right. \\ & \|x\|, c))) \end{aligned}$ | $\begin{aligned} & -f\left(x \left(p r e d \left(p_{1}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{D}\right)- \\ & f\left(e_{2}, x\left(\operatorname { s u c c } \left(p_{2}, 1,\right.\right.\right. \\ & \|x\|, c)),)+ \\ & f\left(x \left(p r e d \left(p_{1}, 1,\right.\right.\right. \\ & \left.\|x\|, c)), e_{2}\right)+ \\ & f\left(e_{1}, x\left(\operatorname { s u c c } \left(p_{2}, 1,\right.\right.\right. \\ & \|x\|, c))) \end{aligned}$ | $\begin{aligned} & -f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{l}\right)- \\ & f\left(e_{1}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\|x\|,\right.\right.\right. \\ & c-1)))- \\ & f\left(x\left(p r e d\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., x\left(\operatorname{succ}\left(p_{2}, 1,\|x\|, c\right)\right)\right) \\ & + \\ & f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & , x\left(\operatorname { s u c c c } \left(p_{l}, 1,\|x\|, c-\right.\right. \\ & 1)))+ \\ & f\left(x\left(\operatorname{pred}\left(p_{2}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{l}\right)+ \\ & f\left(e_{l}, x\left(\operatorname { s u c c } \left(p_{2}, 1,\|x\|, c\right.\right.\right. \\ & 1)) \end{aligned}$ | $\begin{aligned} & + \\ & f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{t}\right) \end{aligned}$ | $\begin{aligned} & f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{l}\right)- \\ & f\left(e_{l}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\|x\|,\right.\right.\right. \\ & c-1)))+f\left(x\left(p_{l}\right),\right. \\ & x\left(s u c c \left(p_{l}, 1,\|x\|, c-\right.\right. \\ & 1))) \end{aligned}$ | $\begin{aligned} & - \\ & f\left(e_{1}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\|x\|,\right.\right.\right. \\ & c)))- \\ & f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{I}\right)+ \\ & f\left(e_{2}, x\left(\operatorname { s u c c } \left(p_{l}, 1,\|x\|,\right.\right.\right. \\ & c)))+ \\ & f\left(x\left(\operatorname{pred}\left(p_{l}, 1,\|x\|, c\right)\right)\right. \\ & \left., e_{2}\right) \end{aligned}$ |
| $\text { Template } 4$ (34) | $N P$ | $\begin{aligned} & -\sum_{i=1}^{p_{1}-1} f\left(x(i), e_{1}\right)- \\ & \sum_{i=p_{1}+1}^{\|x\|} f\left(e_{1}, x(i)\right)- \\ & \sum_{i=1}^{p_{2}-1} f\left(x(i), e_{2}\right)- \end{aligned}$ | NW | $N W$ | $+\sum_{i=1}^{\|x\|-1} f\left(x(i), e_{l}\right)$ | $\begin{aligned} & -\sum_{i=1}^{p_{1}-1} f\left(x(i), e_{I}\right)- \\ & \sum_{i=p_{1}}^{\|x\| 1} f\left(e_{l}, x(i)\right) \end{aligned}$ | $\begin{aligned} & -\sum_{i=1}^{p_{1}-1} f\left(x(i), e_{1}\right)- \\ & \sum_{i=p_{1}+1}^{\|x\| 1} f\left(e_{l}, x(i)\right)- \\ & \sum_{i=1}^{p_{1}-1} f\left(x(i), e_{2}\right)+ \end{aligned}$ |



Table 4: Incremental cost expressions according to template and transition operator.

### 4.6 Search Engine Implementation

The following describes how each of the search engines is implemented within the framework of the general solver system. Due to the opportunities that exist to create parallel implementations of TS, both a serial and parallel version are described.

### 4.6.1 SA

The SA search engine implements the cooling schedule called Q8-7 (Connolly 1990, 1992) as it has shown to be quite successful (Abramson and Randall 1998; Connolly 1990, 1992) (Algorithm 6). The cooling schedule is based on reheating the temperature a number of times throughout the search (see Figure 17). The temperature to which it reheats is always less than the previous reheating point. The number of times this reheating occurs as well as the interval between reheats can be altered. Therefore, if there is a long interval between reheating points, fewer reheatings will occur within the fixed runtimes (as used in this study) than if the interval between reheating points is short. The SA engine also has an option that allows it to perform as many reheats as possible in order to try to reach a given solution quality.


Figure 17: A graphical representation of the SA reheating schedule.

[^4]
### 4.6.2 GS

The GS engine follows the pseudocode given in Algorithm 2. Unlike SA, GS searches the neighbourhood of the current solution in order to find a suitable replacement. As a result of using different transition operators, there are a number of possible neighbourhood structures (described below).

- Move: The neighbourhood consists of moving each element from its sub-list to the end of each other sub-list in the solution. For instance, if an element at $x(1,1)$ is in a solution having three sub-lists, two moves involving this element would be to place it at the end of sub-list 2 and sub-list 3.
- Swap: The swap neighbourhood is composed of a set of pairs of elements. A pair consists of two different elements of the list. This set contains all possible combinations of elements. Given $n$ elements in a list, this equates to $\frac{n(n-1)}{2}$ neighbours.
- Inversion: The inversion neighbourhood consists of pairs of elements between which the sequence can be reversed. Therefore, both elements of a pair must be located on the same sub-list. The neighbourhood is a subset of the swap neighbourhood.
- Reposition: The reposition neighbourhood consists of each combination of element and position on that element's sub-list. For instance, if sub-list 1 has a length of 4, the possible transitions available to the element at $x(1,1)$ are to reposition it at positions 2,3 and 4 on the sub-list.
- Add: The neighbourhood consists of adding each element value in the legal range to the end of each sub-list. For instance, if sub-list 1 contains two elements and the value range is given by $1 \leq x \leq 4$, element $x(1,3)$ can take on the values $\{1,2,3,4\}$.
- Drop: The neighbourhood consists of dropping each element from its place in the current solution. Therefore the size of the neighbourhood is the number of elements in the list structure.
- Change: The neighbourhood consists of the set of transitions in which each element in the solution structure changes its value to a different value in the legal range. For instance, if the value range of the solution is $1 \leq x \leq 4$ and $x(1,1)=2$, the values that $x(1,1)$ can change to are $\{1,3,4\}$.

Note: Neighbourhoods only incorporate members that do not violate the list and problem constraints.

GS terminates once it encounters a local optimum. The existence of a local optimum is determined in the following way. If the current transition operator fails to find an improved solution, each of the other active transition operators are tested instead. If each of these cannot produce a better solution, then a local optimum has been encountered.

### 4.6.3 TS

Both the sequential and parallel version of the TS engine follows the pseudocode given in Algorithm 3. The neighbourhood structures are the same as those described in Section 4.6.2.

### 4.6.3.1 Sequential

The TS engine allows the tabu list size to be set ${ }^{6}$. This parameter governs the number of iterations that a particular transition will stay tabu. The tabu list is implemented as a matrix. This approach has been adopted by a number of other researchers (Osman 1993,1995; Randall and Abramson 1998; Taillard 1991) and has been shown to be quite effective. The tabu list in this system records whether a particular element has been placed on a certain sub-list. If the position of an element on a sub-list is important (for instance, ordering problems such as the TSP and QAP), the tabu list records the element against the position number as well. Hashing vectors are an alternative means of representing a tabu list and can easily be generalised across problem type (Woodruff and Zemel 1993). However, while hashing vectors require only a small amount of memory, their use as the tabu list mechanism can degrade the performance of the search process (Glover and Laguna 1997).

### 4.6.3.2 A Parallel Implementation of the Solver

In recent years, parallel processing hardware platforms have become increasingly more available. This has meant potential increases in the performance of search algorithms for COPs. TS is particularly well suited for parallel implementation and the performance is generally scalable (Glover and Laguna 1997). Therefore, efficient parallelisation strategies can lead to significant savings in the amount of computational time required to solve large size problems. This is unlike SA which is an inherently sequential algorithm, however some parallelisation strategies exist; see van Laarhoven and Aarts (1987) and Ramanujam and Sadayaappan (1995) for a description of these. While it is possible to implement a parallel GS, the benefit would not justify the development effort. This is due to the fact that GS is a limited search technique that cannot overcome the first local optimum it encounters.

This following sub-section reviews a number of parallelisation strategies suitable for TS and describes the implementation of the one that has been adopted for the general system. The parameters of the parallel TS engine (such as tabu list size and neighbour selection probability) are the same as in the sequential version.

### 4.6.3.2.1 A Review of Parallelisation Strategies for Tabu Search

There are a number of strategies available and these can be organised in a taxonomy as in Crainic, Toulouse and Gendreau (1997) and Glover and Laguna (1997). Combining parallelisation strategies

[^5]has also proven effective, as demonstrated by Badeau et al. (1997). Some of the more common may be summarised as:

- Parallel Evaluation of Neighbours: This method is an implementation of the master slave model and works on the premise that the most computationally expensive part of tabu search is the evaluation of the neighbours (corresponding to low level parallelism). Given $P$ slave processors, the neighbourhood is divided equally among the processors. Each processor performs the transitions along with the evaluation of the cost function and constraints and sends its best neighbour back to the master processor. The master processor then determines the best neighbour among all that it has received (according to the tabu rules) and establishes this as the next solution. This approach can require a substantial amount of communication time because of the master slave configuration, but can be easily applied to a range of COPs. This method can be illustrated by considering the QAP. Given that a possible transition operator consists of swapping pairs of facilities, each processor can be assigned a subset of these pairs to evaluate. See Taillard (1993) and Garcia, Potvin and Rousseau (1994) as examples of the implementation of this method.
- Parallel Independent Tabu Searches: In this approach, a number of sequential tabu searches are run simultaneously across processors on a particular problem. Each search is different as key parameters such as random seed, initial solution or tabu list size are varied. This method is particularly suitable to parallel architectures in which each node behaves as an independent system, such as MIMD computers. Because of the independence of the searches, no communication is required between the processors. See Taillard (1991) for an example of the implementation of this method.
- Parallel Interacting Tabu Searches: This approach is similar to the previous method, except that an interaction between the searches occurs at given intervals, (De Falco, Del Balio and Tarantino 1996). This consists of determining which search has been the most successful and transferring its solution to the other search processes. Each search then continues with an empty tabu list from this solution. This approach can have quite a large communication overhead due to the necessity of broadcasting entire solution structures.
- Search Space Division: Each processor is assigned a subsection of the search space. A tabu search subsequently explores its subsection and sends back the partial solution to the master process once it has finished (De Falco et al. 1996). These partial solutions are combined into a final solution. While this method has low communication costs, the process of dividing the search space is very problem specific and may not be possible for all problems. An example of this method deals with the VRP (Taillard 1993). Each processor is assigned a unique vehicle tour to optimise. At the end of the search, each
processor sends its tour to the master processor that assembles these tours and computes the overall cost.

Due to the solver's capacity to process general problem formulations, methods 1 and 2 are implemented. Because each processor is an independent system, method 2 can be easily achieved by running a number of sequential tabu searches on different nodes of an MIMD computer. Despite the communication cost overhead associated with method 1, this approach has been adopted because its applicability across problem type is required for the general list modelling system. This decision is also influenced by the fact that the machine used here has an MIMD architecture (coarse grain and loosely coupled) that is suitable for this parallelisation approach. The parallel code is implemented using the MPI library version 1.0.

### 4.6.3.2 2 Division of the Neighbourhood

At each transition, the master processor calculates an appropriate sub-neighbourhood to delegate to each of the slaves based on the transition to perform, the current solution and the number of available processors. The members of the neighbourhood are divided evenly amongst the processors (see Section 4.6 .2 for a description of each of the neighbourhoods of the transition operators). In the event that $\frac{N}{P}$ (where $N$ is the number of neighbours and $P$ is the total number of processors) is non-integral, the last processor is assigned a different size neighbourhood subset to evaluate than the other processors. This is calculated by the following rule:

$$
\begin{aligned}
& \text { if }\left(\left(\frac{N}{P}-\operatorname{trunc}\left(\frac{N}{P}\right)\right)<0.5\right) \\
& \qquad S=\operatorname{trunc}\left(\frac{N}{P}\right) \\
& \text { else } \\
& \qquad S=\operatorname{trunc}\left(\frac{N}{P}\right)+1 \\
& S_{\mathrm{p}}=N-(P \cdot S)
\end{aligned}
$$

Where:
trunc ( $x$ ) returns the integer component of $x$. $S$ is the number of neighbours the processors 1 through $P-1$ receives. $S_{p}$ is the number of neighbours that processor $P$ receives.

This procedure can lead to a slightly uneven distribution of neighbours to processors. However, as the numerator in $\frac{N}{P}$ becomes large (i.e. more neighbours), the ratio of left over neighbours to allocated neighbours becomes small. Hence, for large size problems, this method of allocation should make little difference to the performance of the TS engine.

### 4.6.3.2 3 The Parallel Algorithm

Figure 18 shows the inter-processor communication needed for the parallel evaluation of neighbours method. The master and slave processes are synchronised by the communication messages. The master's task is to coordinate the entire TS process as well as to control key data structures such as the tabu list and solution memory. It is responsible for delegating the neighbourhood evaluation tasks to the slaves as well as determining which neighbour will form the next solution. In addition, the master also acts as a slave as it evaluates a subsection of the neighbourhood. The master's and slaves' activities are described fully in Algorithm 8 and Algorithm 9 respectively. The communication messages in Figure 18 are noted in these algorithms. Note: the TS mechanics are exactly the same as those outlined in Algorithm 3.


Figure 18: Block diagram of the communications between the master processor and the slave processors for the parallel TS algorithm.

```
Generate an initial feasible solution;
Broadcast the initial solution to the slaves (send message F1);
While (termination condition not met)
    Partition the neighbourhood into equal sizes and send partition
    details to the slaves (send message F2);
    Evaluate each neighbour from own neighbourhood and retain the
    best neighbour;
    Collect each slave's best neighbour (receive messages F3);
    Determine the most suitable neighbour (using the tabu list
    and aspiration rules) and use it to form the next solution;
    Broadcast the attributes of the chosen transition to the slaves
    (send message F4);
    Update the tabu list;
    Determine if the termination condition has been met;
    Broadcast the termination information to the slaves
    (send message F5);
End While;
Report best obtained solution;
End.
```

Algorithm 8: Pseudocode for the master processor.

```
Receive Initial solution (receive message Fl);
While (terminate signal is not "stop")
    Receive details of neighbourhood partition to evaluate
    (receive message F2);
```

```
    Evaluate the neighbours in this partition;
    Send the transition attributes of the best neighbour to the
    master processor (send message F3);
    Receive the attributes of the chosen transition and update
    the local copy of the solution (receive message F4);
    Receive the termination information signal
    (receive message F5);
End While;
End.
```

Algorithm 9: Pseudocode for the slave processors.

### 4.7 Generating an Initial Solution

The generation of initial feasible solutions to COPs can be a difficult task according to Goldberg (1989). This is especially so for solver systems (such as this one) that are able to solve a variety of problems. An alternative to automatic generation is to allow some problem specific means of generating initial solutions such as using constructive heuristics or the requirement that solutions be produced manually. While these methods may produce efficient initial solutions, a more general approach that takes advantage of list structures is proposed here. The following procedure is capable of finding initial feasible solutions to all of the problems described in Section 3.3 .2 with a relatively small amount of computational effort (typically < 1 CPU second).

The method uses a two-stage process in order to form initial feasible solutions. The first stage attempts to satisfy the list constraints while the second is concerned with obtaining the feasibility of the problem constraints. Once the two sets of feasibility have been established, the system invokes the metaheuristic search algorithm.

The first stage of the algorithm consists of the following process (Algorithm 10). Initially, a set of feasible element values is produced, according to the Count and Value Range constraints. Following this, the element values are distributed throughout the list structure to ensure the length bounds on the lists are valid. Accordingly, a solution that satisfies the list constraints is established. In many cases such as the TSP and QAP that have no problem constraints, this initial solution is fully feasible and the search process can begin immediately.

```
/* Create the vector of element values to place in the solution
list */
Case ((min_count (x)=1) AND (max_count (x)=1))
    For i=list_value_lowerbound, list_value_upperbound
            elements_to_place(i-list_value_lowerbound + 1)=i;
        End For;
Case (max_count (x)=1)
        j=list_value_lowerbound;
        For i=list_value_lowerbound, list_value_upperbound
            If (randomly_select()=true)
                                    elements_to_place(j -list_value_lowerbound + 1)=i;
                                    j=j+1;
                End If;
        End For;
Case ((min_count (x)=1) OR (unspecified))
        For }\overline{i}=list_value_lowerbound, list_value_upperbound
            elements_to_place(i - list_value_lowerbound + 1)=i;
        End For;
        j=list_value_lowerbound;
        For i=list_value_lowerbound, list_value_upperbound
                        If (randomly_select()=true)
                        elements_to_place(j -list_value_lowerbound + 1)=i;
                            j=j+1;
                End If;
        End For;
/* Place the elements */
Randomly rearrange the elements contained in the elements_to_place
vector;
For i=1,length(elements_to_place)
    sublist=succ(x,sublist);
    position=|x(sublist)|;
    While (x(sublist,position) does not violate the list
    length bounds)
                    sublist=succ(x,sublist);
                    position=|x(sublist)|;
                    If (each sublist has been tried without success) exit and
                    report failure;
    End While;
    x(sublist,position)=elements_to_place(i);
End For;
End.
Where:
\(x\) is the solution list.
elements_to_place is a vector containing the element values to place on the solution list.
randomly_select () is a function that generates a random state from the set \{true, false\}.
length (i) returns the length of vector \(i\).
\(\operatorname{succ}(x, i)\) returns \(i+1\) unless \(i=|x|\) at which \(i=1\).
```

Algorithm 10: The algorithm for forming an initial solution that satisfies the list constraints.

If problem constraints are present, the next stage is performed. As the constraints are arbitrary algebraic equations, it is difficult to devise a general algorithm that modifies the generated solution so that it is feasible. However, a meta-heuristic search engine can be applied to establish a feasible solution by allowing it to minimise the sum of the constraints' violation (49). When this sum is equal to 0 , a feasible solution has been obtained. Constraint violation is calculated according to the relational operators that are present in the constraints. For instance, if the sign of a constraint is $\leq$ and the left-
hand side is larger than the right-hand side, the net difference is the amount of constraint violation. This is shown in Equation (43). The constraint violations (44) - (48) for the other signs are calculated in a similar manner.

$$
\begin{array}{ll}
(\leq) & c v_{i}=M A X\left(0, l h s_{i}-r h s_{i}\right) \\
(<) & c v_{i}=\operatorname{MAX}\left(0, l h s_{i}-r h s_{i}+1\right) \\
(\geq) & c v_{i}=\operatorname{MAX}\left(0, r h s_{i}-l h s_{i}\right) \\
(>) & c v_{i}=M A X\left(0, r h s_{i}-l h s_{i}+1\right) \\
(=) & c v_{i}=\left|l h s_{i}-r h s_{i}\right| \\
(\neq) & c v_{i}=\left\{\begin{array}{c}
i f\left(r h s_{i}=l h s_{i}\right), 1 \\
0, \text { otherwise }
\end{array}\right. \tag{48}
\end{array}
$$

Minimise $\sum_{i=1}^{N} c v_{i}$
Where:
$N$ is the number of constraints.
$|a|$ is the absolute value of $a$. $\operatorname{MAX}(a, b)$ returns the larger value of $a$ and $b$. $c v_{i}$ is the constraint violation of constraint $i(1 \leq i \leq N)$.
$l h s_{i}$ is the evaluation of the left hand side of constraint $i(1 \leq i \leq N)$.
$r h s_{i}$ is the evaluation of the right hand side of constraint $i(1 \leq i \leq N)$.

### 4.8 Practical Limitations of the System

The general list model for COPs is flexible and broad in its approach. Due to its generality, implementing a practical solver based on the modelling system presented in Equations (10) - (16) would be both expensive to develop and slow to run. In order to produce an efficient system that is still able to solve a wide range of problems (like those outlined in Table 1), certain restrictions have been placed on the general model. These are summarised below:

- The list structure is limited to one level of sub-lists as this is adequate to express the assignment of objects to groups.
- There are four categories of Count constraints. The most common of these is (I) (min_count $(x)=1$ and max_count $(x)=1$ ) that indicates that each value appears exactly once on the list structure (i.e. a permutation). There is also (2) max_count $(x)=1$ (each value can appear once or not at all) and (3) min_count $(x)=1$ (each value appears at least once or many times). This attribute may be omitted and hence (4) there is no restriction on how many times a value can appear on the list.
- Count and value range statements apply across the entire list structures rather than to individual sub-lists.

The more limited list model that the general system implements is given by Equations (50) - (57). Despite its restricted nature, this modelling system can express all of the problems given in Table 1.

$$
\begin{equation*}
\text { Optimise } f(x) \tag{50}
\end{equation*}
$$

s.t.

$$
l h s_{i}(x)\left(\begin{array}{c}
=  \tag{51}\\
< \\
\leq \\
> \\
\geq \\
\geq \\
\neq
\end{array}\right)
$$

$$
l b l\left(\begin{array}{l}
=  \tag{52}\\
< \\
\leq
\end{array}\right)|x|\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right) u b l
$$

$$
l b l_{i}\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right)|x(i)|\left(\begin{array}{l}
= \\
< \\
\leq
\end{array}\right) \quad 1 \leq b l_{i} \quad 1 \leq i \leq|x|
$$

$$
\begin{equation*}
\text { min_count }(x)=1 \tag{54}
\end{equation*}
$$

and/or

$$
\begin{equation*}
\operatorname{max\_ count}(x)=1 \tag{55}
\end{equation*}
$$

$$
\begin{align*}
& 1 \leq i \leq|x|  \tag{56}\\
& 1 \leq j \leq|x(i)|
\end{align*}
$$

$$
x(i, j) \in \mathbf{N}
$$

$$
l b v \leq x(i, j) \leq u b v \quad l \begin{array}{ll}
l \leq i x \mid  \tag{57}\\
& 1 \leq j \leq|x(i)|
\end{array}
$$

Where:
$l b l$ is the lower bound of the main list's length. $u b l$ is the upper bound of the main list's length. $l b l_{i}$ is the lower bound of sub-list $i$ 's length. $u b l_{i}$ is the upper bound of sub-list $i$ 's length. $l b v$ is the lower bound value that can be contained on the list. $u b v$ is the upper bound value that can be contained on the list.

### 4.9 Summary

The implementation of the list modelling system as a practical solver has been described in this chapter. The system allows algebraic descriptions of COPs to be compiled in order to produce an
individual solver. The syntax of description files is similar to the well-known GAMS language with some modifications to account for the list modelling notation. Other important issues related to the production of the general solver are; how a transition is made in the search space (feasibility maintenance and restoration); how to select neighbours to evaluate at each iteration for GS and TS; the available incremental cost templates and their output; and how to generate initial feasible solutions to a range of problems. In addition, the mechanics of the SA, GS and TS search engines are described. The TS engine has also been implemented as a parallel program that is suitable for MIMD machine architectures.

In the interests of efficiency, the current system accepts problems that have restricted list models. For instance, list models can contain only one level of sub-lists. Despite the restrictions, many COPs can be successfully modeled using this approach. In order to test the effectiveness of the system, a range of experiments are proposed and performed. These experiments and the results are described in Chapter 5.

## Chapter 5: Methodology and Results

### 5.1 Experimental Overview and Rationale

The system is tested on many standard problems from the literature. As there are a large number of parameters that can be set, a full test of its performance is impracticable. The main objective is to choose a suitable subset of problems, problem instances and parameters to determine the effectiveness of the system. However, there are many variations of SA and TS algorithms that could also be used and these may give different performances than recorded here. Nevertheless, the general system is compared against widely available COP solver packages to gauge its performance as well as to draw some conclusions about different parametric choices.

There are a number of ways that problems can be modeled using list notation (as is evident by Appendix B). In this work, list models that are best suited to iterative search are used. These are models that provide undulating cost landscapes rather than flatter surfaces. This is quite a common approach that has been used in many studies, including Abramson and Randall (1998), Chams et al. (1987) and Johnson et al. (1991a, 1991b). For instance, there are two different formulations of the bin packing problem. The formulation adopted here minimises the excess weight for a given number of bins rather than minimising the actual number of bins. This is because the latter model has many equivalent packings (effectively flattening the search surface) that can lead to unproductive wandering through search space. Another example is the graph colouring problem in which a feasible colouring is sought using a fixed number of colour classes. The common formulation requires that the number of colour classes is minimised. The other problem formulated in this manner is the maximum clique problem.

The computing platform that is used to conduct all of the experiments discussed herein is an IBM SP2 consisting of 22 RS6000 model 590 nodes connected by an MIMD architecture ${ }^{7}$. Each node has a peak performance of 266 MFLOPs (Millions of Floating Point Operations Per second). This computer is operated by Queensland Parallel Supercomputing Facility (QPSF) which is a consortium funded by Griffith University, University of Queensland, Queensland University of Technology, University of Southern Queensland, James Cook University, University of Central Queensland, Bond University and the Commonwealth Government of Australia.

[^6]
### 5.2 Problems Classes and Instances

The problems used to test the general system are representative of those given in Table 1. In some cases, all published instances have been used (such as for the Car Sequencing Problem), otherwise a few instances of each problem type are used.

Table 5 describes each of the COPs. The Problem heading gives the broad name of the problem while Instance is the name that is used to refer to a particular problem. The Description column provides a brief description of the size of the problem instance. The last two columns indicate the Optimal Cost (if proven optimality has previously been obtained for that problem in the literature) and the Best Known Cost (otherwise).

| Problem | Problem Sources | Instance | Description | Optimal Cost | $\begin{gathered} \hline \text { Best Known } \\ \text { Cost } \\ \hline \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Car Sequencing Problem (CSP) | Smith et al. (1996b) | n20tI <br> n20t2 <br> n20t3 <br> n20t4 <br> n20t5 <br> n40tI <br> n40t2 <br> n40t3 <br> n40t4 <br> n40t5 <br> n60tl <br> n60t2 <br> n60t3 <br> n60t4 <br> n60t5 <br> n80tl <br> n80t2 <br> n80t3 <br> n80t4 <br> n80ts | 20 cars, class 1 20 cars, class 2 20 cars, class 3 20 cars, class 4 20 cars, class 5 40 cars, class 1 40 cars, class 2 40 cars, class 3 40 cars, class 4 40 cars, class 5 60 cars, class 1 60 cars , class 2 60 cars, class 3 60 cars, class 4 60 cars, class 5 80 cars, class 1 80 cars, class 2 80 cars, class 3 80 cars, class 4 80 cars, class 5 |  | 58 40 29 10 150 146 94 66 33 352 238 152 105 58 562 330 215 146 82 772 |
| Bin Packing Problem (BIN) | Beasley (1990) | binlal binla2 binla4 binla6 bin2al bin2a2 bin2a3 bin2a4 bin3al bin3a2 bin3a3 bin3a4 | 120 items 120 items 120 items 120 items 250 items 250 items 250 items 250 items 500 items 500 items 500 items 500 items | 48 49 49 48 99 100 102 100 198 201 202 204 |  |
| Graph Colouring Problems (GRAPH) | Johnson et <br> al. (1991b) | C 125.1 C 125.5 C 125.9 C 250.1 C 250.5 C 250.9 | 125 nodes, 736 edges 125 nodes, 3891 edges 125 nodes, 6961 edges 250 nodes, 3218 edges 250 nodes, 15668 edges 250 nodes, 27897 edges |  | $\begin{gathered} 5 \\ 17 \\ 43 \\ 8 \\ 89 \\ 29 \\ 71 \end{gathered}$ |
| Uncapacitated Single Allocation P Hub Problem(USApHMP) | Ernst and Krishnamoorthy (1996b) | $\begin{aligned} & \text { ap20a2 } \\ & \text { ap20a3 } \\ & \text { ap20a4 } \\ & \text { ap20a5 } \\ & \text { ap25a2 } \\ & \text { ap25a3 } \\ & \text { ap25a4 } \\ & \text { ap25a5 } \end{aligned}$ | 20 nodes, 2 hubs 20 nodes, 3 hubs 20 nodes, 4 hubs 20 nodes, 5 hubs 25 nodes, 2 hubs 25 nodes, 3 hubs 25 nodes, 4 hubs 25 nodes, 5 hubs | 172816.7 151533.1 135624.9 123130.1 175542.0 155256.3 139197.2 123574.3 |  |


| Problem | Problem Sources | Instance | Description | Optimal Cost | Best Known Cost |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Uncapacitated Multiple Allocation P Hub Problem(UMApHMP) | Ernst and Krishnamoorthy (1996a) | ap10a2 <br> ap10a3 <br> apl0a4 <br> apl0a5 | 20 nodes, 2 hubs 20 nodes, 3 hubs 20 nodes, 4 hubs 20 nodes, 5 hubs | $\begin{gathered} \hline 163603.94 \\ 131581.79 \\ 107354.73 \\ 86028.88 \\ \hline \end{gathered}$ |  |
| Quadratic Assignment Problem(QAP) | Burkard et al. (1997) | $\begin{gathered} \text { nug08 } \\ \text { nug12 } \\ \text { nug15 } \\ \text { nug20 } \\ \text { nug30 } \\ \text { ste36a } \\ \text { tho40 } \\ \text { esc64a } \\ \text { sko72 } \\ \text { will00 } \\ \hline \end{gathered}$ | 8 facilities/locations 12 facilities/locations 15 facilities/locations 20 facilities/locations 30 facilities/locations 36 facilities/locations 40 facilities/locations 64 facilities/locations 72 facilities/locations 100 facilities/locations | 107 289 575 1285 3062 | $\begin{gathered} 4763 \\ 120258 \\ 58 \\ 33158 \\ 136522 \end{gathered}$ |
| Traveling Salesman Problem(TSP) | $\begin{aligned} & \text { Reinelt } \\ & \text { (1991) } \end{aligned}$ | gr24 <br> swiss42 <br> hk48 <br> eil51 <br> brazil58 <br> st70 <br> kroA100 <br> ch130 <br> a280 | 24 cities <br> 42 cities <br> 48 cities <br> 51 cities <br> 58 cities <br> 70 cities <br> 100 cities <br> 130 cities <br> 280 cities | $\begin{gathered} 1272 \\ 1273 \\ 11461 \\ 426 \\ 25395 \\ 675 \\ 21282 \\ 6110 \\ 2579 \end{gathered}$ |  |
| Generalised Assignment Problem(GAP) ${ }^{8}$ | Beasley (1990) | gap1-1 gap2-1 gap3-1 gap4-1 gap5-1 gap6-1 gap7-1 gap8-1 gapA5-100 gapA5-200 gapA10-100 gapA10-200 gapA20-100 gapA20-200 | 15 jobs, 5 agents 20 jobs, 5 agents 25 jobs, 5 agents 30 jobs, 5 agents 24 jobs, 8 agents 32 jobs, 8 agents 40 jobs, 8 agents 48 jobs, 8 agents 100 jobs, 5 agents 200 jobs, 5 agents 100 jobs, 10 agents 200 jobs, 10 agents 100 jobs, 20 agents 200 jobs, 20 agents | 336 434 580 656 563 761 942 1133 1698 3235 1360 2623 1158 2339 |  |
| Graph Partitioning Problem(GPP) | Johnson et <br> al. (1991a) | G124.02 G124.04 G124.08 G124.16 G250.01 G250.02 G250.04 G250.08 | 124 nodes, 298 edges 124 nodes, 635 edges 124 nodes, 1240 edges 124 nodes, 2542 edges 250 nodes, 662 edges 250 nodes, 1224 edges 250 nodes, 2566 edges 250 nodes, 4842 edges |  | $\begin{gathered} 13 \\ 63 \\ 178 \\ 449 \\ 29 \\ 114 \\ 357 \\ 828 \\ \hline \end{gathered}$ |
| Multiple Knapsack Problem(MKP) | Beasley (1990) | $\begin{gathered} \hline \text { weing1 } \\ \text { pb6 } \\ \text { weish12 } \\ \text { sent01 } \\ \text { sent02 } \\ \text { weish15 } \\ \text { weish25 } \\ \text { weish29 } \\ \hline \end{gathered}$ | 28 items, 2 knapsacks 40 items, 30 knapsacks 50 items, 5 knapsacks 60 items, 30 knapsacks 60 items, 30 knapsacks 60 items, 5 knapsacks 80 items, 5 knapsacks 90 items, 5 knapsacks | $\begin{gathered} 141278 \\ 776 \\ 6339 \\ 7772 \\ 8722 \\ 7486 \\ 9939 \\ 9410 \\ \hline \end{gathered}$ |  |
| Maximum Clique Problem(MCP) | Battiti and Protasi (1995) | $\begin{gathered} \hline \text { johnson8-2-4 } \\ \text { johnson16-2-4 } \\ \text { keller4 } \\ \text { c-fat200-1 } \\ \text { brock200_2 } \\ \text { brock200_3 } \\ \text { brock200_1 } \\ \hline \end{gathered}$ | 28 nodes, 210 edges 120 nodes, 5460 edges 171 nodes, 9435 edges 200 nodes, 1534 edges 200 nodes, 9876 edges 200 nodes, 12048 edges 200 nodes, 14834 edges | $\begin{gathered} 4 \\ 8 \\ 12 \end{gathered}$ | $\begin{aligned} & 11 \\ & 12 \\ & 15 \\ & 21 \\ & \hline \end{aligned}$ |
| Time Tabling Problem (TTP) | Abramson and Dang (1993) | hdtt4 <br> hdtt5 <br> hdtt6 <br> hdtt7 <br> hdtt8 | 120 tuples, 30 periods 150 tuples, 30 periods 180 tuples, 30 periods 210 tuples, 30 periods 240 tuples, 30 periods | $\begin{aligned} & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \end{aligned}$ |  |

[^7]| Problem | Problem <br> Sources | Instance | Description | Optimal <br> Cost | Best Known <br> Cost |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | ttgen1 | 275 tuples, 30 periods | 0 |  |

Table 5: Problem classes and instances that are used in this study. Note: as objective costs are typically reported dimensionless in the literature, units are not shown here.

### 5.3 Parameter Settings

The performance of the system is tested by varying the following key groups of parameters:

- The transition operators and the probability of applying these operators (see Table 6)
- The search engine specific parameters (see Table 7)

The parameters are set relatively coarsely. This is because the amount of computing time that would be involved in testing all combinations of parameter settings would be prohibitive. Therefore, the parameter settings outlined here are used to conduct tests that consist of runs involving small amounts of Central Processor Unit (CPU) time (referred to as the standard tests). Based on the outcome of these experiments, suitable parameter settings for different problem types can be derived.

The most time consuming set of experiments is the variation of the transition operator probabilities. A discussion of probability settings can be found in Section 3.4.1.2. As such, a test set for each problem type has been devised and these are given in Table 6. For instance, the GAP has five settings in which the probabilities of the move and swap operators are $\{\{1,0\},\{0.5,0.5\},\{0.2,0.8\},\{0.8,0.2\}$, adaptive $\}$ respectively. Note: for the adaptive setting, $\alpha=0.001$ and $N=10$ is used.

Each set of probabilities for a particular problem type is given a TRANSITION SET REFERENCE NUMBER in order to uniquely identify the set in subsequent tables. Due to the large amount of computer time that could be required, a full exploration of transition probability parameter space (according to Table 6) will only occur for the SA runs. The best transition probability set obtained for a particular problem type with SA is used for the GS and TS runs (this process is described in Section 5.8.3.1).

| $\begin{aligned} & \text { PROBLEM } \\ & \text { CLASS } \\ & \hline \end{aligned}$ | SET REFERENCENUMBER | PROBABILITY OF SELECTING TRANSITION OPERATORS |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Move | Swap | Inversion | Reposition | Add | Drop | Change | Adaptive |
| CSP | 1 |  | 1 |  |  |  |  |  |  |
| GPP | 1 |  | 1 |  |  |  |  |  |  |
| GAP | 1 | 1 |  |  |  |  |  |  |  |
|  | 2 | 0.2 | 0.8 |  |  |  |  |  |  |
|  | 3 | 0.8 | 0.2 |  |  |  |  |  |  |
|  | 4 | 0.5 | 0.5 |  |  |  |  |  |  |
|  | 5 |  |  |  |  |  |  |  | 1 |
| BIN | 1 | 1 |  |  |  |  |  |  |  |
|  | 2 | 0.2 | 0.8 |  |  |  |  |  |  |
|  | 3 | 0.8 | 0.2 |  |  |  |  |  |  |
|  | 4 | 0.5 | 0.5 |  |  |  |  |  |  |
|  | 5 |  |  |  |  |  |  |  | 1 |
| GRAPH | 1 | 1 |  |  |  |  |  |  |  |
|  | 2 | 0.2 | 0.8 |  |  |  |  |  |  |
|  | 3 | 0.8 | 0.2 |  |  |  |  |  |  |
|  | 4 | 0.5 | 0.5 |  |  |  |  |  |  |
|  | 5 |  |  |  |  |  |  |  | 1 |
| USAPHMP | 1 | 1 |  |  |  |  |  |  |  |
|  | 2 | 0.5 | 0.5 |  |  |  |  |  |  |
|  | 3 | 0.2 | 0.8 |  |  |  |  |  |  |
|  | 4 | 0.8 | 0.2 |  |  |  |  |  |  |
|  | 5 | 0.4 | 0.4 |  | 0.2 |  |  |  |  |
|  | 6 |  |  |  |  |  |  |  | 1 |
| TTP | 1 | 1 |  |  |  |  |  |  |  |
|  | 2 | 0.2 | 0.8 |  |  |  |  |  |  |
|  | 3 | 0.8 | 0.2 |  |  |  |  |  |  |
|  | 4 | 0.5 | 0.5 |  |  |  |  |  |  |
|  | 5 |  |  |  |  |  |  |  | 1 |
| TSP | 1 |  | 1 |  |  |  |  |  |  |
|  | 2 |  |  | 1 |  |  |  |  |  |
|  | 3 |  |  |  | 1 |  |  |  |  |
|  | 4 |  | 0.33 | 0.33 | 0.33 |  |  |  | - |
|  | 5 |  | 0.8 | 0.1 | 0.1 |  |  |  |  |
|  | 6 |  | 0.1 | 0.8 | 0.1 |  |  |  |  |
|  | 7 |  | 0.1 | 0.1 | 0.8 |  |  |  |  |
|  | 8 |  |  |  |  |  |  |  | 1 |
| QAP | 1 |  | 1 |  |  |  |  |  |  |
|  | 2 |  |  | 1 |  |  |  |  |  |
|  | 3 |  |  |  | 1 |  |  |  |  |
|  | 4 |  | 0.33 | 0.33 | 0.33 |  |  |  |  |
|  | 5 |  | 0.8 | 0.1 | 0.1 |  |  |  |  |
|  | 6 |  | 0.1 | 0.8 | 0.1 |  |  |  |  |
|  | 7 |  | 0.1 | 0.1 | 0.8 |  |  |  |  |
|  | 8 |  |  |  |  |  |  |  | 1 |
| MCP | 1 |  |  |  |  |  |  | 1 |  |
| MKP | 1 |  |  |  |  | 0.33 | 0.33 | 0.33 |  |
|  | 2 |  |  |  |  | 0.5 | 0.5 | 0 |  |
|  | 3 |  |  |  |  | 0.25 | 0.25 | 0.5 |  |
|  | 4 |  |  |  |  | 0.1 | 0.1 | 0.8 |  |
|  | 5 |  |  |  |  | 0.4 | 0.4 | 0.2 |  |
|  | 6 |  |  |  |  |  |  |  | 1 |
| UMAPHMP | 1 | 0.2 | 0.2 |  |  | 0.2 | 0.2 | 0.2 |  |
|  | 2 | 0.4 | 0.4 |  |  | 0.1 | 0.1 |  |  |
|  | 3 | 0.8 |  |  |  | 0.1 | 0.1 |  |  |
|  | 4 | 0.3 | 0.3 |  |  | 0.1 | 0.1 | 0.2 |  |
|  | 5 6 |  |  |  |  | 0.33 | 0.33 | 0.33 |  |
|  | 6 |  |  |  |  |  |  |  | 1 |

Table 6: Transition probability settings for each problem type.

Table 7 gives a brief description of the parameters required for the SA and TS engines (Note: GS needs no engine-specific parameters). A description of the implementation of each search method is given in Section 4.6.

| Search Engine | Parameters |
| :--- | :--- |
| SA | Reheating run length $=\{1000,5000,10000\} \times(u b v-l b v)$ <br> TS |
| $\left.\begin{array}{l}\text { Tabu list length }=\{\text { Problem Size }\end{array}\right\}$ |  |
| NS | Neighbourhood evaluation probability $=\{0.1,0.5,1.0\}$ |
| Neighbourhood evaluation probability $=\{0.1,0.5,1.0\}$ |  |

Table 7: Search engine-specific parameters.

### 5.4 Termination Conditions

For the standard trials, each run on the SP2 platform receives 1600 seconds of CPU time (chosen due to the constraints of the computer's job scheduling system). If the optimal solution is known for a particular problem instance, the run is terminated once this has been reached. As mentioned in Section 4.6.2, GS terminates once it encounters a local optimum.

For those problems that: (a) the system has not reached the optimal cost in the trial runs; or (b) the optimal is not known and the best known has not been reached or improved upon in the trial runs; further tests are carried out with the best parameter set. These are referred to as the extended runs and each terminates after 6400 seconds of CPU time has elapsed.

### 5.5 Benchmarking Against Other Software Packages

In order to evaluate the performance of the general system, it is compared with existing optimisation packages. A variety of software packages, ranging from commercial OR codes to special-purpose heuristics for particular problems, have been obtained for this purpose. While it is not possible to test all alternative packages, it is believed that these will give a good indication of the general system's performance. Table 8 gives a brief description of each package.

| Name | Problem(s) | Description |
| :---: | :---: | :---: |
| Optimisation Subroutine Library (OSL) | all (except TTP and BIN (a)) | A commercial OR package by IBM (IBM 1990). |
| LINDO | all (except TTP and BIN (a)) | A commercial OR package by Lindo Systems (Schrage 1997). |
| tsp_solve | TSP | A collection of heuristic solvers for the TSP. This product is from the GNU Free Software Foundation and selects the most appropriate solver for each type of TSP (Hurwitz 1994). |
| QAPSIM | QAP | An SA solver for the QAP (Burkard et al. 1997). |
| QAPBB | QAP | A Branch and Bound code for the QAP (Burkard et al. 1997). |
| SA-CSP | CSP | A special purpose SA code provided by K. Smith and used in Smith et al. (1996b). |
| dfmax, dfclique | MCP | Special purpose branch and bound codes used in Battiti and Protasi (1995). |
| Recursive Largest First (RLF) | GRAPH | A heuristic used by Johnson et al. (199.1b). |

Table 8: Optimisation packages that are compared with the general solver.

[^8]These packages are run according to the specifications given by their technical documentation. Both quality of solution, and time to reach the best solution, are recorded for each package on the problems that are run. The 0-1 ILP formulations of the test problems used by Lindo and OSL are given in Appendix C.

### 5.6 Feasibility Restoration Runs

In order to test the effectiveness of the feasibility restoration implementation outlined in Section 4.3, the GAP instances from Table 5 are used. The GAP was chosen as the test-bed as it is the only problem type in the test collection that has problem constraints and can utilise the move transition operator. Feasibility restoration will be tested with the SA and TS engines.

### 5.7 Parallel Runs

The primary concern of testing the parallel TS engine is the evaluation of the parallel speedup and efficiency of the TS engine, rather than solution quality (as this is tested extensively with the sequential version of the code). Table 9 lists the problems and problem instances that are used to test the parallel tabu engine (a subset of those in Table 5). The best transition operator set for each problem type is used. This is determined from the standard runs using the process outlined in Section 5.8.3.1. Each problem is run for a fixed number of iterations on dedicated nodes. Wall clock time as opposed to CPU time is used to measure the time taken for each run. The number of processors is also varied from 1 node to 12 nodes in order to determine the effectiveness of the parallel code.

The guidelines for reporting parallel experiments as outlined in Barr and Hickman (1993) are followed. The most common measure of effectiveness of a parallel algorithm is given by speedup. Speedup is defined by (58) (Barr and Hickman 1993).

Time to solve a problem with the fastest serial

$$
\begin{align*}
\text { speedup }= & \frac{\text { code on a specific parallel computer }}{\text { Time to solve the same problem with the parallel }}  \tag{58}\\
& \text { code using } P \text { processors on the same computer }
\end{align*}
$$

According to Barr and Hickman (1993), average values should not be used in (58) and would require a new definition of speedup. As a result of this, only one seed per combination of problem type and processor grouping is used.

In order to ascertain the effectiveness of the parallel code, the efficiency for each problem is computed by Equation (59).
efficiency $=\frac{\text { speedup }}{P}$

Speedup and efficiency tables and graphs are produced from the collected data.

| Problem | Instance |
| :--- | :--- |
| GPP | G250.01 <br> G250.02 <br> G250.04 <br> G250.08 |
| QAP | tho40 <br> esc64a <br> sko72 <br> wil00 |
| GAP | gapA10-100 <br> gapA10-200 <br> gapA20-100 <br> gapA20-200 |
| BIN | bin3a1 <br> bin3a2 <br> bin3a3 <br> bin3a4 |
| TSP | st70 <br> kroA100 <br> ch130 <br> a280 |

Table 9: Problem classes and instances that are used to test the parallel tabu engine.

### 5.8 Reporting the Results

A large amount of raw data is gathered as a result of these experiments. In order that any valid conclusions are reached, the results are analysed and reported in a succinct manner that is recognised by the operational research and broader scientific community. To accomplish this, the reporting of the results is divided into three categories:

- Numerical Data
- Qualitative Analysis
- Statistical Analysis

As it is expected that the data will be highly non-normally distributed, non-parametric descriptions and analyses are predominately used.

### 5.8.1 Numerical Data

Each combination of problem instance, search engine and parameter set is run across 10 random seeds (referred to as a trial set). The random seeds control probabilistic search engine choices, such as which neighbour to evaluate in TS and GS, as well as the generation of the initial solutions. Because many problem instances are used, the application of different random seeds will reduce variability in the results (Barr et al. 1995). For each run, the best solution cost and the corresponding CPU time ${ }^{10}$ is recorded. This information is presented in a series of tables in the form of summary statistics. Each trial set is described using:

- Min - The minimum result of the trial set.
- Med - The median (mid-point) of the trial set. This is a measure of the central tendency of the data.
- Max - The maximum of the trial set.
- IQR - The Inter Quartile Range of the trial set. IQR is a measure of the dispersion of the results. It is defined as the difference between the first and third quartile points of the trial set (Emory and Cooper 1991).


### 5.8.2 Qualitative Analysis

The overall trends present in the result tables are highlighted. The qualitative analyses are made in terms of the solution quality achieved by the general solver as well as the amount of runtime required to obtain these solutions.

### 5.8.3 Statistical Analysis

Statistical analyses are performed in order to determine the effect of the transition operator sets and engine-specific parameters. As well as this, statistical analyses are used to calculate the best overall software package for each problem type. An $\alpha$ of 0.05 is used throughout.

### 5.8.3.1 The Effect of the Transition Operator Sets and Engine-specific Parameters

As well as the descriptive statistics mentioned previously, tests for significant differences in solution quality (gained from varying the transition operator sets and the engine-specific parameters for each problem) are conducted. This takes the form of the Kruskal-Wallis procedure (discussed in Sprent (1993)) with post-hoc analysis in order to detect individual differences.

[^9]
### 5.8.3.2 Overall Comparison

Statistical analysis can also be used to obtain a quantitative measure of the performance of the general system compared with other software packages. The analysis consists of two phases.

The first phase rank-orders the performance of each applicable code on every problem instance. This is achieved by using the Kruskal-Wallis procedure. If a significant difference is detected on a particular problem instance, a post-hoc test is performed in order to determine where the differences lie. From this, the solvers can be ordered. Performance is calculated primarily on objective cost. If two algorithms are indistinguishable using the objective cost, runtime is used instead. For some problems (such as GRAPH and MCP), direct comparison between the competing solvers is not possible as the objective functions measure different quantities.

The second phase determines the best overall solver for each problem type. This is achieved by analysing the ranks for each solver (generated in the first phase) again using the Kruskal-Wallis test. If a significant difference is detected, post-hoc procedures are carried out to determine where these differences lie. From this, the most effective solvers for a given problem type can be deduced. As RLF, dfmax and dfclique utilise a different objective function to the one used by the general solver, a ranking of their performance is made using a subjective analysis.

### 5.9 Results

### 5.9.1 Standard Runs

The standard run tables (Table 45-Table 47) are presented in Appendix A due to their large size. The following descriptions are qualitative analyses of the performances of the SA, TS and GS engines for each problem type based on a review of these tables. These analyses highlight the main trends present in the data with particular reference to the effects of the transition operator sets. A quantitative analysis using statistical methods can be found in Section 5.10 in which the performances of the solvers for each problem type are rank ordered.

Note: in all cases except CSP and some GAP instances, the GS engine performed poorly. Typically GS returned inferior quality solutions in very short runtimes, compared with SA and TS.

## CSP

The search engines (in particular SA and TS) produced very good solutions, frequently finding better solutions than reported in Smith et al. (1996b). There was only one problem instance (n40t5) for which the engines could not obtain the best-known solution. SA and TS consistently produced a minimum solution cost of 354 whereas the Hopfield-Tank neural network used by Smith et al. (1996b) reached a cost of 352 .

## BIN

Using SA, transition operator sets that incorporated the swap operator (sets $2-5$ ) were all very effective in finding the prescribed bin packings. The use of the move operator in isolation (set 1) produced costs that were considerably worse than those obtained for the other sets. The TS engine found it difficult to produce the required bin packings for the large problem instances (i.e. bin3al, bin3a2, bin3a3 and bin3a4).

## GRAPH

The smallest colouring for each problem instance attempted in Johnson et al. (1991b) is also used here (see Table 5 for these). These colourings could only be obtained by Johnson et al. (1991b) after many hours of computer processing time (and not at all for C125.9 and C250.9). As the final costs achieved by the SA and TS engines are relatively close to 0 , it is not unreasonable to conclude that these solutions are of good quality.

## USApHMP

The optimal solutions were frequently found using SA and TS. While transition operator set 1 performed consistently well in terms of solution quality, the runtimes were typically longer than for the other sets (sets $2-6$ ). It is interesting to note that for this problem type, the use of the move operator in isolation (set 1) outperforms the other sets, while for the problems BIN, GAP and TTP, the same transition operator set produced the worst results.

## UMApHMP

For SA, transition operator sets $2,3,4$ and 6 performed equally well in terms of solution quality. TS could not produce the optimal solution for ap10a4 and ap 10a5, though it was at most $4 \%$ away in the worst case.

## QAP

For SA, the use of the swap operator helped to produce more efficient solutions than inversion or reposition. It is noted that the application of swap transition perturbs the current solution less than
reposition and inversion. In general, the system performs extremely well on QAPs. This is characterised by the fact that TS finds solutions for the largest problem instance (will00) that are at most $1.5 \%$ away from the best-known solution cost. This is also achieved with a small amount of computational effort.

## TSP

For SA, the use of the inversion operator helped to produce more efficient solutions than swap or reposition. Like the QAP, it is noted that the application of inversion transition perturbs the current solution less than reposition and swap. The solver produced very poor solutions for a280. As this is a very large problem instance, it is expected that substantially more computer processing time would be required in order to find reasonable quality solutions.

## GAP

The SA engine frequently produces optimal solutions using transition operator sets $2-5$. Very poor performance is recorded for SA using the move operator in isolation (transition operator set 1). The best results obtained by the SA and TS engine are comparable with that of the specialised GA metaheuristic implementation of Chu and Beasley (1997) in terms of both solution quality and runtime. It is interesting to note that GS performed well on the larger GAPs (gapA5-100 - gapA20-200) as it regularly found the optimal solution.

## GPP

The SA engine is a very efficient solver for this problem type. The only instance that it could not solve as well as Johnson et al. (1991a) was G250.01. In this case, it produced a solution that was $3 \%$ from the best-known cost. TS found the optimal solution for the problem instances G124.02-G124.16. However, for the problems with 250 vertices, TS produced near optimal solutions. It is believed that was the case as it had to evaluate a large number of neighbours at each step (up to 31125). Longer TS runs should give improved solutions (see Section 5.9.2).

## MKP

The SA engine solved all the MKP problem instances to optimality using transition operator set 4 . The performance of the TS engine (using this set) was similar except that the optimal solution could not be produced for weish25 and weish29.

## MCP

Both SA and TS could not obtain the optimal solutions for the problem instances as recorded in Battiti and Protasi (1995) (except for keller4). However, both search engines typically achieved the same solution cost for each problem instance. As a result of this, further investigation was undertaken. The

SA engine was run using the alternative formulation that maximised the number of nodes in the clique (the list formulation is provided in Appendix B). These results show that the best-known result could only be gained for the problem instances keller4 and johnson16-2-4 (Table 10).

| MCP Problem Instance | Optimal/Best <br> Known Solution Cost | Maximum Cliques <br> Achieved using SA Engine |
| :--- | :--- | :--- |
| johnson8-2-4 | 4 | 3 |
| johnson16-2-4 | 8 | 8 |
| keller4 | 11 | 11 |
| c-fat200-1 | 12 | 11 |
| brock200_2 | 12 | 10 |
| brock200_1 | 21 | 18 |
| brock200_3 | 15 | 13 |

Table 10: Largest clique sizes achieved with the SA engine using the alternative list formulation for the MCP given in Appendix B.

## TTP

The SA engine is extremely efficient for this problem type as it records better solutions than Abramson and Dang (1993) for the largest problem instances (hdtt7 and hdtt8). It is possible that this is due to the fact that Abramson and Dang (1993) only used the equivalent of the move operator whereas the general solver employed this operator in conjunction with swap transitions. As the problem instances were generated to have solutions that represent clash-free time tables (i.e. a cost of 0 ), it can be concluded the SA engine finds the optimal solution to these difficult (Abramson and Dang 1993) problems. TS could not find the optimal solutions for the problem instances hdtt6 - hdtt8.

### 5.9.2 Extended Runs

Some of the test problem instances were run for longer periods of time using SA and TS in order to determine whether improved results could be obtained. Each problem instance was chosen as the bestknown or optimal cost could not be found in the standard runs for it. The amount of CPU time for each run is increased from 1600 seconds to 6400 seconds. Table 11 and Table 12 show these results. As observed in Section 5.9.1, GS tends to return poor locally optimal solutions after short amounts of runtime. It is for this reason that the Extended Runs were only be performed for SA and TS.

| Problem | Instance | Optimal/ Best Known Cost | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| GRAPH | C125.5 | 0 | 1000 | 8 | 9 | 10 | 1 | 8 | 16.5 | 24.25 | 8.75 |
|  |  |  | 5000 | 5 | 6.5 | 10 | 2.5 | 24 | 65.5 | 110.25 | 61 |
|  |  |  | 10000 | 3 | 6 | 10 | 2 | 24 | 81 | 231.5 | 105 |
|  | C125.9 | 0 | 1000 | 4 | 5 | 8 | 1 | 16 | 22 | 27.25 | 5.25 |
|  |  |  | 5000 | 2 | 3 | 4 | 0 | 23 | 55.5 | 124.25 | 22.5 |
|  |  |  | 10000 | 1 | 2 | 3 | 0 | 46 | 179.5 | 253.25 | 26.25 |
|  | C250.1 | 0 | 1000 | 4 | 7 | 12 | 3.75 | 104 | 121.5 | 177 | 39.5 |
|  |  |  | 5000 | 2 | 4 | 11 | 2.75 | 157 | 443 | 873.5 | 393.75 |
|  |  |  | 10000 | 2 | 3.5 | 11 | 1.75 | 157 | 727 | 1430 | 597.25 |
|  | C250.5 | 0 | 1000 | 15 | 20 | 25 | 3.25 | 60 | 76 | 79.75 | 13.25 |
|  |  |  | 5000 | 7 | 10.5 | 13 | 2 | 193 | 319 | 376.25 | 35 |
|  |  |  | 10000 | 7 | 10 | 13 | 1 | 194 | 344.5 | 789.5 | 212 |
|  | C250.9 | 0 | 1000 | 17 | 24 | 26 | 2.75 | 76 | 82 | 88.25 | 4.25 |
|  |  |  | 5000 | 11 | 12 | 15 | 2.5 | 197 | 371.5 | 442.25 | 103 |
|  |  |  | 10000 | 7 | 10 | 13 | 1.5 | 472 | 728 | 885.75 | 262.25 |
| QAP | tho40 | 120258 | 1000 | 120316 | 120713.5 | 121903 | 405.75 | 163 | 2456 | 3834.5 | 2741.5 |
|  |  |  | 5000 | 120316 | 120527 | 121140 | 334.5 | 290 | 2837.5 | 6096.5 | 4125.25 |
|  |  |  | 10000 | 120271 | 120316 | 121292 | 0 | 365 | 2304 | 6376.75 | 2046.25 |
|  | esc64a | 58 | 1000 | 58 | 58 | 58 | 0 | 1 | 3 | 7.25 | 2.5 |
|  | sko72 | 33158 | 1000 | 33209 | 33209 | 33311 | 50 | 150 | 1966.5 | 4057.25 | 2117.75 |
|  | will00 | 136522 | 1000 | 136773 | 136773 | 136978 | 58.25 | 869 | 1038.5 | 3939.5 | 803 |
| TSP | st70 | 675 | 1000 | 678 | 686.5 | 695 | 6.75 | 11 | 111 | 215.25 | 47.25 |
|  |  |  | 5000 | 680 | 682.5 | 688 | 4.25 | 11 | 773.5 | 3034 | 199.75 |
|  |  |  | 10000 | 676 | 681.5 | 685 | 3.5 | 874 | 1021 | 2213.25 | 21.5 |
|  | kroA100 | 21282 | 1000 | 21366 | 21519.5 | 21668 | 235.75 | 43 | 2118 | 3367.25 | 747.5 |
|  | ch 130 | 6110 | 1000 | 6193 | 6271 | 6343 | 75.75 | 842 | 2645 | 3653 | 1253.75 |
|  | a280 | 2579 | 1000 | 2842 | 2886.5 | 3057 | 72.25 | 1794 | 2291.5 | 3225 | 582 |
| GPP | G250.01 | 29 | 1000 | 29 | 31 | 33 | 1.75 | 1744 | 1876.5 | 2204.5 | 168 |

Table 11: Extended results of the SA engine.

| Problem | Instance | Optimal/ Best Known Cost | Neighbour Probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| BIN | bin3a2 | 0 | 0.1 | 0 | 1.5 | 3 | 1 | 2113 | 4738.5 | 6206 | 2185.75 |
|  |  |  | 0.5 | 3 | 5.5 | 14 | 2.5 | 2378 | 4088 | 6219 | 1434.25 |
|  |  |  | 1 | 7 | 15.5 | 33 | 8 | 3722 | 4679.5 | 5713 | 917.5 |
|  | bin3a4 | 0 | 0.1 | 2 | 4.5 | 5 | 2 | 1874 | 4897.5 | 6162 | 2016 |
|  |  |  | 0.5 | 5 | 14 | 39 | 7.75 | 2705 | 5631 | 6339 | 1741 |
|  |  |  | 1 | 13 | 18 | 27 | 7.5 | 3792 | 5394 | 6280 | 761.75 |
| GRAPH | C125.1 | 0 | 0.1 | 6 | 7 | 9 | 0.75 | 225 | 1791 | 4922 | 2536.75 |
|  |  |  | 0.5 | 2 | 3 | 5 | 0.75 | 261 | 2154.5 | 6275 | 2203.75 |
|  |  |  | 1 | 2 | 3 | 5 | 0.75 | 726 | 1610.5 | 5488 | 2456 |
|  | C125.5 | 0 | 0.1 | 9 | 10 | 12 | 1.75 | 379 | 1988.5 | 6126 | 3698 |
|  |  |  | 0.5 | 4 | 5 | 7 | 0.75 | 774 | 2619.5 | 6212 | 3949 |
|  |  |  | 1 | 4 | 6 | 7 | 1 | 760 | 2386.5 | 5536 | 1655.5 |
|  | C125.9 | 0 | 0.1 | 6 | 6 | 7 | 1 | 295 | 2397 | 5834 | 2759.5 |
|  |  |  | 0.5 | 2 | 2 | 3 | 0.75 | 1283 | 4288 | 5969 | 1377.5 |
|  |  |  | 1 | 2 | 3 | 4 | 0.75 | 871 | 2869.5 | 5968 | 3561.25 |
|  | C250.1 | 0 | 0.1 | 17 | 19 | 22 | 2.5 | 1227 | 3372 | 5001 | 1849.75 |
|  |  |  | 0.5 | 13 | 16 | 18 | 2.5 | 908 | 2923.5 | 5537 | 2266.75 |
|  |  |  | 1 | 16 | 19 | 23 | 2 | 1331 | 4016.5 | 5501 | 1378.5 |
|  | C250.5 | 0 | 0.1 | 14 | 21 | 23 | 0.75 | 360 | 2241 | 5490 | 3035 |
|  |  |  | 0.5 | 14 | 17.5 | 18 | 1.75 | 1978 | 5004.5 | 5868 | 1632.25 |
|  |  |  | 1 | 18 | 20.5 | 23 | 2.5 | 3121 | 4963 | 6168 | 690.25 |
|  | C250.9 | 0 | 0.1 |  | 16.5 | 19 |  | 740 | 3002 | 4997 | 851.5 |
|  |  |  | 0.5 | 11 | 15.5 | 19 | 2 | 2988 | 5554 | 6329 | 711.25 |
|  |  |  | 1 | 16 | 19 | 22 | 1.75 | 4319 | 5214 | 6051 | 773.25 |
| USAPHMP | ap25a5 | 123574.3 | 0.1 | 123659.7 | 123766.1 | 124683.5 | 916.83 | 36 | 2992.5 | 5660 | 2278.25 |
|  |  |  | 0.5 | 123659.7 | 123659.7 | 123659.7 | 0 | 267 | 3095 | 6203 | 2572.5 |
|  |  |  | 1 | 123659.7 | 128319.8 | 135708.2 | 5322.22 | 23 | 368.5 | 4020 | 805.5 |
| UMAPHMP | apl0a4 | 107354.7 | 0.1 | 107612.1 | 108262.5 | 109510.7 | 260.37 | 61 | 2091.5 | 5561 | 4132 |
|  |  |  | 0.5 | 108262.5 | 109260.1 | 110508.3 | 1020.41 | 20 | 2065 | 6088 | 4226.5 |
|  |  |  | 1 | 109391.4 | 110956.9 | 111845.8 | 1403.38 | 698 | 2696.5 | 5903 | 2367.75 |
|  | ap10a5 | 86028.9 | 0.1 | 86794.76 | 87469.62 | 88572.91 | 1031.99 | 396 | 3062.5 | 6309 | 3268 |
|  |  |  | 0.5 | 87761.12 | 88132.31 | 88645.64 | 352.14 | 717 | 3854 | 6275 | 2994 |
|  |  |  | 1 | 86794.76 | 89183.2 | 89643.23 | 1179.22 | 242 | 1569 | 4463 | 2682 |
| QAP | ste36a | 4763 | 0.1 | 4763 | 4765 | 4862 | 11.75 | 680 | 2260 | 3447 | 791.25 |
|  |  |  | 0.5 | 4873 | 5001 | 5270 | 169.25 | 34 | 716.5 | 2941 | 1098 |
|  |  |  | 1 | 4944 | 5215.5 | 5485 | 228.5 | 15 | 95.5 | 1902 | 343.25 |
|  | tho40 | 120258 | 0.1 | 120329 | 120403 | 120620 | 75.75 | 665 | 1983 | 3356 | 547.5 |
|  |  |  | 0.5 | 120316 | 120823.5 | 121615 | 773 | 220 | 2746 | 6372 | 2805 |
|  |  |  | 1 | 120803 | 123163 | 125789 | 1530 | 15 | 87 | 324 | 179.75 |
|  | sko72 | 33158 | 0.1 | 33196 | 33273 | 33428 | 39 | 3694 | 6082.5 | 6343 | 435.25 |


| Problem | Instance | Optimal/ <br> Best Known <br> Cost | Neighbour Probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 0.5 | 33344 | 33541 | 33834 | 149.25 | 2951 | 5101 | 6071 | 589.25 |
|  |  |  | 1 | 33272 | 33590 | 33845 | 203.5 | 1813 | 4956 | 6379 | 2371.75 |
|  | will 100 | 136522 | 0.1 | 136825 | 137134 | 137381 | 261.25 | 2864 | 5425.5 | 6320 | 1112.25 |
|  |  |  | 0.5 | 136859 | 137440.5 | 137976 | 422.25 | 2961 | 6052 | 6365 | 838.5 |
|  |  |  | 1 | 137315 | 137594 | 137813 | 269 | 5256 | 6069 | 6387 | 474.75 |
| TSP | kroA100 | 21282 | 0.1 | 21415 | 21968.5 | 22380 | 2705.25 | 491 | 1472 | 5329 | 2890.25 |
|  |  |  | 0.5 | 21370 | 21691.5 | 22341 | 362.75 | 841 | 3708 | 6226 | 3741.75 |
|  |  |  | 1 | 21363 | 21952.5 | 22954 | 597.5 | 1467 | 3620.5 | 5791 | 1976 |
|  | ch 130 | 6110 | 0.1 | 6187 | 6290 | 6407 | 43.5 | 707 | 3157 | 6175 | 2065 |
|  |  |  | 0.5 | 6209 | 6320.5 | 6453 | 77.5 | 1278 | 3924 | 6392 | 1278 |
|  |  |  | 1 | 6164 | 6334.5 | 6443 | 114.75 | 1921 | 3803 | 4949 | 2029.5 |
|  | a280 | 2579 | 0.1 | 3527 | 3607 | 3801 | 102.5 | 6337 | 6362.5 | 6380 | 18.75 |
|  |  |  | 0.5 | 15729 | 16806.5 | 17439 | 326 | 6355 | 6393 | 6400 | 7 |
|  |  |  | 1 | 22475 | 23247 | 24268 | 722.75 | 6361 | 6398.5 | 6400 | 2 |
| GAP | gapA10-200 | 2623 | 0.1 | 2630 | 2631 | 2633 | 1 | 144 | 1323.5 | 6001 | 1446.75 |
|  |  |  | 0.5 | 2625 | 2626 | 2629 | 2.5 | 236 | 397.5 | 3364 | 1533.25 |
|  |  |  | 1 | 2624 | 2624.5 | 2626 | 1 | 638 | 1039 | 9271 | 3103.25 |
|  | gapA20-200 | 2339 | 0.1 | 2348 | 2350 | 2351 | 1.75 | 143 | 1425.5 | 3808 | 2684 |
|  |  |  | 0.5 | 2341 | 2342.5 | 2345 | 2 | 316 | 2208 | 4804 | 1616.25 |
|  |  |  | 1 | 2341 | 2343 | 2346 | 1.75 | 504 | 838 | 6066 | 318 |
| GPP | G250.1 | 29 | 0.1 | 32 | 37.5 | 44 | 5.25 | 1650 | 3332 | 6297 | 2863.5 |
|  |  |  | 0.5 | 47 | 51 | 62 | 4.25 | 771 | 5474 | 6352 | 3208 |
|  |  |  | 1 | 53 | 56 | 65 | 5.25 | 1599 | 3845 | 5742 | 2857 |
|  | G250.2 | 114 | 0.1 | 117 | 122 | 126 | 5.25 | 613 | 3684 | 6152 | 2223 |
|  |  |  | 0.5 | 127 | 130 | 152 | 7 | 1622 | 5318 | 6226 | 1060 |
|  |  |  | 1 | 127 | 138 | 151 | 12.5 | 2969 | 4490.5 | 6407 | 1886.5 |
|  | G250.4 | 357 | 0.1 | 357 | 370 | 380 | 6.5 | 2189 | 4209 | 5508 | 1496.5 |
|  |  |  | 0.5 | 368 | 380.5 | 391 | 8.25 | 2009 | 4808 | 6054 | 1702.25 |
|  |  |  | 1 | 379 | 386.5 | 396 | 8 | 2969 | 5810 | 6322 | 2157.25 |
|  | G250.8 | 828 | 0.1 | 830 | 834.5 | 847 | 9.75 | 1865 | 5121.5 | 5859 | 2196 |
|  |  |  | 0.5 | 838 | 858 | 877 | 14.25 | 2381 | 4768 | 6181 | 2286.5 |
|  |  |  | 1 | 855 | 869.5 | 901 | 9.75 | 3141 | 4455 | 5853 | 1564.25 |
| MKP | weish25 | 9939 | 0.1 | 9837 | 9857 | 9889 | 24.75 | 679 | 2892.5 | 6152 | 876.25 |
|  |  |  | 0.5 | 9832 | 9832 | 9883 | 23.25 | 192 | 1491.5 | 4284 | 2449.5 |
|  |  |  | 1 | 9832 | 9832 | 9832 | 0 | 165 | 326 | 1128 | 314.25 |
|  | weish29 | 9410 | 0.1 | 9207 | 9266 | 9345 | 86.75 | 377 | 4171.5 | 6032 | 3420.25 |
|  |  |  | 0.5 | 9130 | 9206.5 | 9345 | 207.75 | 484 | 2838 | 6368 | 2785 |
|  |  |  | 1 | 9023 | 9023 | 9023 | 0 | 75 | 267 | 630 | 230 |
| TTP | hdtt6 | 0 | 0.1 | 9 | 11.5 | 13 | 1 | 337 | 3123.5 | 5944 | 2314.25 |
|  |  |  | 0.5 | 2 | 3.5 | 5 | 1 | 1360 | 2982 | 5490 | 1872 |
|  |  |  | 1 | 0 | 2 | 4 | 1.75 | 1384 | 4187 | 6235 | 1608.5 |


| Problem | Instance | Optimal / <br> Best Known <br> Cost | Neighbour Probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | hdtt 7 | 0 | 0.1 | 14 | 15.5 | 18 | 2 | 436 | 1727 | 6333 | 4015.5 |
|  |  |  | 0.5 | 4 | 7 | 8 | 0 | 1312 | 3302 | 6104 | 1527.75 |
|  |  |  | 1 | 6 | 7.5 | 10 | 2.5 | 3248 | 4291 | 6163 | 1199 |
|  | hdtt8 | 0 | 0.1 | 17 | 20 | 21 | 1.75 | 1544 | 3041.5 | 6214 | 1759 |
|  |  |  | 0.5 | 8 | 11 | 12 | 1 | 2683 | 5109 | 6392 | 1074.5 |
|  |  |  | 1 | 8 | 14 | 19 | 2 | 3096 | 5479 | 6289 | 937.5 |

Table 12: Extended results of the TS engine.

Table 11 and Table 12 show that the TS engine has benefited most from the extra runtime as it frequently found the optimal, best-known or improved cost within the extra time. In nearly all cases, SA finds the same solution as it did in the standard runs and spends the rest of the time trying to find better quality solutions. This may be due to the use of the Q8-7 temperature schedule (Connolly 1990, 1992). It is possible that the schedule is not sufficiently reheating the temperature to adequately escape deep local optima in the final stages of the search process. Further investigation would be required to verify and remedy this situation.

### 5.9.3 Other Software

Table 13 to Table 17 are produced as a result of running the heuristic codes while Table 18 and Table 19 show the results of running Lindo and OSL respectively. For those software programs that can produce proven optimal solutions, the column "Runtime to Proven Opt" is included. In the case of dfmax, the output displays the time at which each new solution cost is recorded, therefore the amount of CPU time required to find the first instance of the optimal solution is shown. This column is labeled "Runtime to Opt". There are some abbreviations used in these tables. These are:

- CNS indicates that the algorithm Could Not Solve the problem with the given amount of computational time ( 6400 seconds).
- TB indicates that the resulting model exceeded the memory and disk space storage capacity of the hardware/software platform.
-     - indicates that a result could not be obtained.

| Problem instance | Best Known <br> cost | Min | Med | Max | IQR | Total Runtime <br> (seconds) $^{\mathbf{1 1}}$ |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| n20tl | 58 | 58 | 61 | 63 | 3.25 | 10 |
| n20t2 | 40 | 40 | 42 | 43 | 2 | 10 |
| n20t3 | 29 | 30 | 31 | 31 | 1 | 9 |
| n20t4 | 10 | 10 | 10.5 | 12 | 1 | 8 |
| n20t5 | 150 | 150 | 150 | 150 | 0 | 8 |
| n40tl | 146 | 146 | 150 | 158 | 4 | 12 |
| n40t2 | 94 | 94 | 100.5 | 105 | 3 | 11 |
| n40t3 | 66 | 67 | 70 | 73 | 1.25 | 11 |
| n40t4 | 33 | 33 | 35 | 39 | 2.5 | 10 |
| n40t5 | 352 | 354 | 362 | 362 | 1 | 10 |
| n60t1 | 238 | 238 | 242.5 | 252 | 6 | 14 |
| n60t2 | 152 | 155 | 171 | 177 | 13 | 14 |
| n60t3 | 105 | 108 | 112 | 115 | 1.25 | 14 |
| n60t4 | 58 | 60 | 63 | 67 | 2.5 | 13 |
| n60t5 | 562 | 566 | 568 | 578 | 8.5 | 12 |
| n80tl | 330 | 331 | 341 | 358 | 6.25 | 18 |
| n80t2 | 215 | 215 | 231 | 252 | 5.75 | 17 |
| n80t3 | 146 | 150 | 154 | 158 | 2.25 | 17 |
| n80t4 | n80t5 | 772 | 86 | 89 | 96 | 3 |

Table 13: CSP results (objective costs and runtimes) for SA-CSP.

[^10]| Problem Instance | Optimal Cost | Cost | Runtime (seconds) |
| :--- | :--- | :--- | :--- |
| gr24 | 1272 | 1272 | 0 |
| swiss42 | 1273 | 1273 | 0 |
| hk48 | 11461 | 11461 | 0 |
| eil51 | 426 | 426 | 46 |
| brazil58 | 25395 | 25395 | 5 |
| st70 | 675 | 675 | 559 |
| kroA100 | 21282 | CNS | CNS |
| ch130 | 6110 | CNS | CNS |
| a280 | 2579 | CNS | CNS |

Table 14: Results of running tsp_solve on the TSP.

|  |  | dfmax |  |  | dfclique |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem Instance | Optimal/ <br> Best Known <br> Cost | Nodes in Clique | Runtime to Opt (seconds) | Runtime to proven Opt (seconds) | Nodes in Clique | Runtime to proven Opt (seconds) |
| brock200_1 <br> brock200_2 <br> brock200_3 <br> c-fat200-1 <br> johnson8-2-4 ${ }^{12}$ <br> johnson16-2-4 <br> keller4 | $\begin{array}{\|l\|} \hline 21 \\ 12 \\ 15 \\ 12 \\ 4 \\ 8 \\ 11 \\ \hline \end{array}$ | $\begin{aligned} & 21 \\ & 12 \\ & 15 \\ & 12 \\ & 4 \\ & 8 \\ & 11 \end{aligned}$ | $\begin{array}{\|l} \hline 176 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hline \end{array}$ | $\begin{array}{\|l} 665 \\ 1 \\ 8 \\ 0 \\ 0 \\ 28 \\ 15 \end{array}$ | $\begin{aligned} & 21 \\ & 12 \\ & 15 \\ & 12 \\ & 4 \\ & 8 \\ & 11 \end{aligned}$ | $\left\lvert\, \begin{aligned} & 671 \\ & 1 \\ & 8 \\ & 0 \\ & 0 \\ & 28 \\ & 15 \end{aligned}\right.$ |

Table 15: Results of running dfmax and dfclique on the MCP problem instances.

| Problem Instance | Best Known <br> Colouring | Number of <br> Colours obtained | Runtime <br> (seconds) |
| :--- | :--- | :--- | :--- |
| C125.1 | 5 | 6 | 0 |
| C125.5 | 17 | 21 | 0 |
| C125.9 | 43 | 51 | 0 |
| C250.1 | 8 | 10 | 0 |
| C250.5 | 29 | 36 | 0 |
| C250.9 | 71 | 85 | 0 |

Table 16: Results of running RLF on the GRAPH problem instances.

|  |  | QAPBB |  | QAPSIM |  |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Problem | Optimal /Best <br> Known Cost | Cost | Runtime to Proven <br> Opt (seconds) | Cost | Runtime <br> (seconds) |
| nug08 | 107 | 107 | 0 | 109 | 0 |
| nug12 | 289 | 289 | 16 | 300 | 0 |
| nug15 | 575 | CNS | CNS | 585 | 0 |
| nug20 | 1285 | CNS | CNS | 1299 | 0 |
| nug30 | 3062 | CNS | CNS | 3168 | 1 |
| ste36a | 4763 | TB | TB | 5050 | 1 |
| tho40 | 120258 | TB | TB | 124219 | 1 |
| esc64a | 58 | TB | TB | 58 | 1 |
| sko72 | 33158 | TB | TB | 33617 | 10 |
| will00 | 136522 | TB | TB | 137539 | 40 |

Table 17: Results of running QAPBB and QAPSIM on the QAP problem instances.

[^11]Table 18 and Table 19 display the results of solving the test COPs with commercial ILP packages. There are a number of additional abbreviations used in these tables. These are:

- (a) refers to the penalty formulation, while (b) refers to the formulations that either minimise or maximise groups (see Section 5.1).
- CNFLM indicates that a linear model could not be formulated.
- opt denotes that the optimal solution cost was reached.
- all represents all the problem instances of a problem type.
- a-b indicates problem instance a through problem instance $\mathbf{b}$ (according to the order given in Table 5).

USApHMP and UMApHMP were not solved with Lindo and OSL because comparable results exist in the literature (Ernst and Krishnamoorthy 1997b). Table 20 and Table 21 give the results for these two problem types using highly refined MILP (Mixed Integer Linear Program) formulations. Note: the standard formulations of these problems as used by Campbell (1994) were not solved in Ernst and Krishnamoorthy (1997b) because of poor performance with the increase in problem size beyond 10 nodes.

| Problem | Problem Instance | Cost | Runtime to Proven Opt (seconds) |
| :---: | :---: | :---: | :---: |
| CSP | all | CNS | CNS |
| $\mathrm{BIN}(\mathrm{a})$ | all | CNFLM | CNFLM |
| BIN (b) | all | TB | TB |
| GRAPH (a) | all | TB | TB |
| GRAPH (b) | all | TB | TB |
| QAP | all | TB | TB |
| TSP | all | TB | TB |
| GAP | gap 1-1 | opt | 1 |
|  | gap2-1 | opt | 6 |
|  | gap3-1 | opt | 62 |
|  | gap4-1 | opt | 9 |
|  | gap5-1 | opt | 17 |
|  | gap6-1 | opt | 73 |
|  | gap7-1 | opt | 207 |
|  | gap8-1 | CNS | CNS |
|  | gapA5-100 | opt | 2 |
|  | gapA5-200 | opt | 4 |
|  | gapA10-100 | opt | 16 |
|  | gapA10-200 | opt | 8 |
|  | gapA20-100 | opt | 18 |
|  | gapA20-200 | CNS | CNS |
| GPP | all | TB | TB |
| MKP | weing I | opt | 1 |
|  | weish12 | opt | 2 |
|  | weish 15 | opt | , |
|  | weish25 | opt | 2 |
|  | weish29 | opt | 1 |


| Problem | Problem Instance | Cost | $\begin{array}{\|l} \hline \begin{array}{l} \text { Runtime to } \\ \text { Proven Opt } \end{array} \\ \text { (seconds) } \\ \hline \end{array}$ |
| :---: | :---: | :---: | :---: |
|  | $\begin{aligned} & \text { sent01 } \\ & \text { sent02 } \\ & \text { pb6 } \end{aligned}$ | $\left\lvert\, \begin{aligned} & \mathrm{opt} \\ & \mathrm{opt} \\ & \mathrm{opt} \end{aligned}\right.$ | $\begin{aligned} & 29 \\ & 44 \\ & 19 \end{aligned}$ |
| $\begin{aligned} & \mathrm{MCP}(\mathrm{a}) \\ & \mathrm{MCP}(\mathrm{~b}) \end{aligned}$ | brock200_1 - <br> keller4 <br> johnson8-2-4 (4 <br> node clique) <br> brock200_1- <br> keller4 <br> johnson8-2-4 | $\left\{\begin{array}{l} \mathrm{TB} \\ 1 \\ \mathrm{~TB} \\ 3 \end{array}\right.$ | $\begin{aligned} & \mathrm{TB} \\ & 10 \\ & \mathrm{~TB} \\ & 37 \end{aligned}$ |
| TTP | all | CNFLM | CNFLM |

Table 18: The results of running Lindo on the problem test suite.

| Problem | Problem Instance | Cost | $\begin{array}{\|l} \begin{array}{l} \text { Runtime to } \\ \text { Proven Opt } \\ \text { (seconds) } \end{array} \\ \hline \end{array}$ |
| :---: | :---: | :---: | :---: |
| CSP | $\begin{aligned} & n 20 t 1-n 20 t 5 \\ & n 40 t 1-n 80 t 5 \end{aligned}$ | $\begin{aligned} & \mathrm{CNS} \\ & \mathrm{~TB} \end{aligned}$ | $\begin{aligned} & \mathrm{CNS} \\ & \mathrm{~TB} \end{aligned}$ |
| BIN (a) <br> BIN (b) | $\begin{aligned} & \text { binlal-binla6 } \\ & \text { bin2al-bin3a4 } \\ & \text { all } \end{aligned}$ | $\begin{aligned} & \text { CNS } \\ & \text { TB } \\ & \text { CNFLM } \end{aligned}$ | $\begin{aligned} & \text { CNS } \\ & \text { TB } \\ & \text { CNFLM } \end{aligned}$ |
| GRAPH (a) <br> GRAPH (b) | C 125.1 $\mathrm{C} 125.5-\mathrm{C} 250.9$ C 125.1 $\mathrm{C} 125.5-\mathrm{C} 250.9$ | $\begin{aligned} & \mathrm{CNS} \\ & \mathrm{~TB} \\ & \mathrm{CNS} \\ & \mathrm{~TB} \end{aligned}$ | CNS TB CNS TB |
| QAP | all | TB | TB |
| TSP | all | TB | TB |
| GAP | gap1-1 <br> gap2-1 <br> gap3-1 <br> gap4-1 <br> gap5-1 <br> gap6-1 <br> gap7-1 <br> gap8-1 <br> gapA5-100 <br> gapA5-200 <br> gapA10-100 <br> gapA10-200 <br> gapA20-100 <br> gapA20-200 | opt opt opt opt opt opt CNS CNS opt opt opt opt opt opt | 2 6 16 10 34 2361 $C N S$ $C N S$ 1 2 4 3 5 11 |
| GPP | $\begin{aligned} & \mathrm{G} 124.2 \\ & \mathrm{G} 124.4-\mathrm{G} 250.8 \end{aligned}$ | $\begin{aligned} & \mathrm{CNS} \\ & \mathrm{~TB} \end{aligned}$ | $\begin{aligned} & \mathrm{CNS} \\ & \mathrm{~TB} \end{aligned}$ |
| MKP | weing1 weish12 weish15 weish25 weish29 sent01 sent02 pb6 | opt <br> opt <br> opt <br> opt <br> opt <br> opt <br> opt <br> opt | $\begin{aligned} & 0 \\ & 1 \\ & 1 \\ & 1 \\ & 1 \\ & 3 \\ & 8 \\ & 8 \end{aligned}$ |


| Problem | Problem Instance | Cost | Runtime to <br> Proven Opt <br> (seconds) |
| :--- | :--- | :--- | :--- |
| MCP (a) | brock200_1 - <br> keller4 <br> johnson8-2-4 (4 <br> node clique <br> brock200_1 - <br> keller4 <br> johnson8-2-4 | 1 | CNS |
| MCP (b) | 3 | 3 |  |
| TTP | all | CNFLM | CNFLM |

Table 19: The results of running OSL on the problem test suite.

| $n$ | p | Optimal Solution | MILP CPU <br> (seconds) |
| :--- | :--- | :--- | :--- |
| 20 | 2 | 172816.69 | 4.03 |
| 20 | 3 | 151533.08 | 0.8 |
| 20 | 4 | 135624.88 | 1.24 |
| 20 | 5 | 123130.09 | 1.73 |
| 25 | 2 | 175541.98 | 4.03 |
| 25 | 3 | 155256.32 | 7.18 |
| 25 | 4 | 139197.17 | 8.44 |
| 25 | 5 | 123574.29 | 13.76 |

Table 20: Results of branch and bound with the shortest path heuristic for the USApHMP on a DEC 3000/700 with CPLEX (reproduced from Ernst and Krishnamoorthy (1997b, Table 2, p. 15)).

| $\mathbf{n}$ | $\mathbf{p}$ | Optimal Solution | MILP CPU <br> (seconds) |
| :--- | :--- | :--- | :--- |
| 10 | 2 | 163603.94 | 1.6 |
| 10 | 3 | 131581.79 | 1.52 |
| 10 | 4 | 107354.73 | 1.62 |
| 10 | 5 | 86028.88 | 0.8 |

Table 21: Results of branch and bound with the shortest path heuristic for the UMApHMP on a DEC 3000/700 with CPLEX (reproduced from Ernst and Krishnamoorthy (1997b, Table 3, p.16)).

The ILP packages performed well on problem types that required few encoding constraints such as GAP and MKP. Proven optimal solutions were obtained (in most cases) for these problems in runtimes that are comparable with the general system. Many of the other problems (especially those with complex encoding constraints) produced 0-1 ILP models that exceeded the capacity of the hardware/software platform and subsequently could not be run.

For the most part, the general system outperformed the specialised heuristics in terms of solution quality, particularly in the case of QAP (QAPBB and QAPSIM), GRAPH (RLF) and CSP (SA-CSP). The statistical analysis in Section 5.10.2 also confirms this. The heuristics for MCP produced the bestknown costs with relatively little CPU time. However, there is a question about the validity of dfmax and dfclique, especially in reference to the problem johnson8-2-4. Both Lindo and OSL record a proven optimal solution of 3 nodes using a 0-1 ILP formulation while the two heuristic codes produce a value of 4 nodes.

### 5.9.4 Feasibility Restoration Runs

Table 22 and Table 23 display the results of the feasibility restoration procedure for the SA and TS engines respectively.

| Problem Instance | Optimal Cost | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| gap 1-1 | 336 | 1000 | 363 | 363 | 363 | 0 | 0 | 27 | 34 | 6 |
|  |  | 5000 | 363 | 363 | 363 | 0 | 0 | 124 | 144 | 30 |
|  |  | 10000 | 363 | 363 | 363 | 0 | 0 | 245 | 282 | 50.25 |
| gap2-1 | 434 | 1000 | 434 | 434 | 434 | 0 | 28 | 40 | 51 | 5.25 |
|  |  | 5000 | 434 | 434 | 434 | 0 | 135 | 193 | 250 | 32.75 |
|  |  | 10000 | 434 | 434 | 434 | 0 | 275 | 381.5 | 501 | 59 |
| gap3-1 | 580 | 1000 | 580 | 580 | 580 | 0 | 76 | 101.5 | 198 | 33 |
|  |  | 5000 | 580 | 580 | 580 | 0 | 286 | 409.5 | 523 | 91.5 |
|  |  | 10000 | 580 | 580 | 580 | 0 | 4 | 788 | 946 | 174.25 |
| gap4-1 | 656 | 1000 | 656 | 656 | 656 | 0 | 86 | 135 | 174 | 37.5 |
|  |  | 5000 | 656 | 656 | 656 | 0 | 381 | 560.5 | 678 | 76.5 |
|  |  | 10000 | 656 | 656 | 656 | 0 | 764 | 853.5 | 1287 | 324.75 |
| gap5-1 | 563 | 1000 | 563 | 563 | 563 | 0 | 0 | 56 | 142 | 43.5 |
|  |  | 5000 | 563 | 563 | 563 | 0 | 0 | 232 | 365 | 58.25 |
|  |  | 10000 | 563 | 563 | 563 | 0 | 0 | 471 | 520 | 72.25 |
| gap6-1 | 761 | 1000 | 758 | 759.5 | 761 | 1 | 2 | 494.5 | 987 | 349.5 |
|  |  | 5000 | 759 | 759 | 761 | 1 | 0 | 515 | 1255 | 420.25 |
|  |  | 10000 | 758 | 760 | 761 | 1 | 2 | 722.5 | 1303 | 684 |
| gap 7-1 | 942 | 1000 | 933 | 936.5 | 938 | 1 | 1 | 9.5 | 246 | 110 |
|  |  | 5000 | 935 | 937 | 938 | 1 | 1 | 9.5 | 975 | 576 |
|  |  | 10000 | 935 | 937 | 938 | 1 | 1 | 450.5 | 1415 | 1209.75 |
| gap8-1 | 1133 | 1000 | 1109 | 1116.5 | 1123 | 6 | 275 | 878 | 1398 | 874 |
|  |  | 5000 | 1109 | 1116.5 | 1123 | 6 | 275 | 878 | 1530 | 874 |
|  |  | 10000 | 1109 | 1116.5 | 1123 | 6 | 275 | 878.5 | 1542 | 874.25 |
| gapA5-100 | 1698 | 1000 | 1702 | 1725 | 1754 | 17 | 683 | 956.5 | 1588 | 291 |
|  |  | 5000 | 1702 | 1725 | 1754 | 17 | 683 | 956.5 | 1588 | 291 |
|  |  | 10000 | 1702 | 1725 | 1754 | 17 | 683 | 956.5 | 1588 | 291.5 |
| gapA5-200 | 3235 | 1000 | 3235 | 3716 | 4168 | 200.5 | 1484 | 1580 | 1593 |  |
|  |  | 5000 | 3235 | 3716 | 4168 | 192.75 | 1484 | 1570 | 1591 | 25.75 |
|  |  | 10000 | 3235 | 3716 | 4168 | 200.5 | 1484 | 1580 | 1591 | 25.75 |
| gapA10-100 | 1360 | 1000 | 1361 | 1363 | 1371 | 2.75 | 2 | 7.5 | 37 | 10.75 |
|  |  | 5000 | 1361 | 1363 | 1371 | 2.75 | 2 | 9.5 | 450 | 13.5 |
|  |  | 10000 | 1361 | 1363 | 1371 | 2.75 | 2 | 7.5 | 37 | 10 |
| gapA10-200 | 2623 | 1000 | 3729 | 4031 | 4407 | 476 | 1489 | 1568 | 1591 | 51.75 |
|  |  | 5000 | 3733 | 4031 | 4407 | 476 | 1489 | 1557 | 1590 | 49.75 |
|  |  | 10000 | 3729 | 4031 | 4407 | 476 | 1489 | 1568 | 1591 | 52.25 |
| gapA20-100 | 1158 | 1000 | 1158 | 1158 | 1158 | 0 | 8 | 13 | 36 | 4 |
|  |  | 5000 | 1158 | 1158 | 1158 | 0 | 8 | 13 | 36 | 4 |
|  |  | 10000 | 1158 | 1158 | 1158 | 0 | 8 | 13 | 36 | 3.75 |
| gapA20-200 | 2339 | 1000 | 2339 | 2339 | 2342 | 1.5 | 25 | 52 | 119 | 59 |
|  |  | 5000 | 2339 | 2339 | 2341 | 0 | 25 | 77.5 | 197 | 58.25 |
|  |  | 10000 | 2339 | 2339 | 2341 | 0 | 25 | 77.5 | 197 | 58.25 |

Table 22: Feasibility restoration runs using SA.

| Problem <br> Instance | Optimal Cost | Neighbour Probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| gap 1-1 | 336 | 0.1 | 363 | 363 | 363 | 0 | 5 | 33 | 76 | 37 |
|  |  | 0.5 | 363 | 363 | 363 | 0 | 1 | 5 | 12 | 6 |
|  |  | 1.0 | 363 | 363 | 363 | 0 | 0 | 2 | 10 | 4.25 |
| gap2-1 | 434 | 0.1 | 428 | 432.5 | 434 | 4.75 | 150 | 859 | 1499 | 7125.5 |
|  |  | 0.5 | 434 | 434 | 434 | 0 | 5 | 44 | 218 | 109.25 |
|  |  | 1.0 | 434 | 434 | 434 | 0 | 0 | 15.5 | 23 | 6.5 |
| gap 3-1 | 580 | 0.1 | 570 | 572.5 | 577 | 1.75 | 48 | 389 | 1060 | 375.5 |
|  |  | 0.5 | 580 | 580 | 580 | 0 | 28 | 296.5 | 1281 | 770.5 |
|  |  | 1.0 | 580 | 580 | 580 | 0 | 24 | 128 | 398 | 167.25 |
| gap4-1 | 656 | 0.1 | 634 | 639.5 | 644 | 2.75 | 14 | 769.5 | 1443 | 577 |
|  |  | 0.5 | 653 | 655.5 | 656 | 2.5 | 317 | 530 | 985 | 222.25 |
|  |  | 1.0 | 656 | 656 | 656 | 0 | 10 | 373.5 | 1197 | 454.75 |
| gap5-1 | 563 | 0.1 | 550 | 551 | 555 | 1.75 | 26 | 553 | 1136 | 813.25 |
|  |  | 0.5 | 561 | 563 | 563 | 0 | 30 | 297.5 | 1581 | 685.25 |
|  |  | 1.0 | 563 | 563 | 563 | 0 | 2 | 32 | 316 | 65.25 |
| gap6-1 | 761 | 0.1 | 738 | 742 | 750 | 1.75 | 5 | 1085.5 | 1485 | 706.5 |
|  |  | 0.5 | 757 | 758 | 760 | 0.75 | 180 | 608.5 | 1493 | 930 |
|  |  | 1.0 | 759 | 760.5 | 761 | 1 | 14 | 332 | 1197 | 631.5 |
| gap 7-1 | 942 | 0.1 | 914 | 916.5 | 921 | 2.5 | 43 | 638.5 | 1225 | 506 |
|  |  | 0.5 | 936 | 937.5 | 938 | 1.75 | 2 | 913.5 | 1450 | 748 |
|  |  | 1.0 | 938 | 939.5 | 941 | 1.75 | 61 | 888.5 | 1349 | 101.5 |
| gap8-1 | 1133 | 0.1 | 1073 | 1081.5 | 1089 | 7.5 | 65 | 589 | 1526 | 879.75 |
|  |  | 0.5 | 1108 | 1112 | 1119 | 3.5 | 342 | 921 | 1382 | 757.75 |
|  |  | 1.0 | 1114 | 1118.5 | 1123 | 2.75 | 439 | 946 | 1400 | 272.75 |
| gapA5-100 | 1698 | 0.1 | 1748 | 1809 | 1862 | 70.25 | 1192 | 1499 | 1583 |  |
|  |  | 0.5 | 1821 | 1910.5 | 2421 | 222 | 1539 | 1568 | 1597 | 18.5 |
|  |  | 1.0 | 1969 | 2193 | 2674 | 216.75 | 1498 | 1577 | 1599 | 41.5 |
| gapA5-200 | 3235 | 0.1 | 4177 | 4777.5 | 5278 | 603 | 1488 | 1550.5 | 1597 | 69 |
|  |  | 0.5 | 4658 | 5337.5 | 5677 | 418.75 | 1231 | 1370.5 | 1527 | 120 |
|  |  | 1.0 | 4931 | 5512.5 | 5937 | 452 | 816 | 1350 | 1588 | 294 |
| gapA10-100 | 1360 | 0.1 | 1418 | 1430.5 | 1436 | 5 | 23 | 962.5 | 1483 | 568.5 |
|  |  | 0.5 | 1360 | 1360 | 1361 | 1 | 9 | 237.5 | 1493 | 249.5 |
|  |  | 1.0 | 1360 | 1360 | 1360 | 0 | 7 | 22.5 | 32 | 15.5 |
| gapAl0-200 | 2623 | 0.1 | 4210 | 5064.5 | 5631 | 482.5 | 1480 | 1522 | 1582 | 21 |
|  |  | 0.5 | 5086 | 5650 | 5886 | 422.75 | 1030 | 1493 | 1572 | 56.5 |
|  |  | 1.0 | 5674 | 5758.5 | 5966 | 129.25 | 821 | 1329.5 | 1598 | 187.5 |
| gapA20-100 | 1158 | 0.1 | 1307 | 1321.5 | 1325 | 6.5 | 78 | 648.5 | 1436 | 926.25 |
|  |  | 0.5 | 1167 | 1170 | 1172 | 0.75 | 160 | 838.5 | 1568 | $1235$ |
|  |  | 1.0 | 1158 | 1159 | 1159 | 0.75 | 56 | 580.5 | 1292 | 664.25 |
| gapA20-200 | 2339 | 0.1 | 2449 | 2455.5 | 2466 | 6.75 | 46 | 406 | 1495 | 511.5 |
|  |  | 0.5 | 2343 | 2344 | 2345 | 0 | 4 | 528.5 | 1260 | 351 |
|  |  | 1.0 | 2339 | 2339 | 2339 | 0 | 106 | 354 | 1194 | 199.5 |

Table 23: Feasibility restoration runs using TS.

Table 22 and Table 23 show that the use of feasibility restoration moves gives better quality solutions than feasibility maintaining moves (see Table 45 and Table 46 , transition probability set 1 , Appendix A) for both the SA and TS engines. However, it is clear that the combination of feasibility maintaining moves and swaps (transition probability sets $2-5$ in Table 45 and Table 46, Appendix A) result in similar or better quality solutions in considerably shorter runtimes than feasibility restoration.

On some of the larger problems (in particular GapA10-200), both SA and TS produce solutions that are very far from optimal. This is due to the large computational requirements of feasibility restoration compared with feasibility maintenance. Apart from the complex mechanics of the restoration algorithm, it does not make use of incremental cost expressions. This is because a large number of individual list transition operations are performed before the cost function can be evaluated. It is envisaged that techniques can be developed to overcome this problem.

### 5.9.5 Parallel Runs

Table 24 and Table 25 show the parallel speedup and efficiency gained by running the test problems.
Parallel speedup and efficiency graphs for each problem type are given in Figure 19-Figure 28.

| Problem | Instance | Processors |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| GPP | G250.01 | 0.63 | 0.95 | 1.7 | 2.43 | 3.12 | 3.72 | 4.42 | 5.22 | 5.71 | 6.2 | 6.71 | 7.18 |
|  | G250.02 | 0.65 | 1.02 | 1.79 | 2.66 | 3.32 | 3.96 | 4.69 | 5.55 | 6.23 | 6.65 | 7.02 | 7.82 |
|  | G250.04 | 0.67 | 1.15 | 1.85 | 2.63 | 3.56 | 4.45 | 5.1 | 6.01 | 6.57 | 7.04 | 7.57 | 8.73 |
|  | G250.08 | 0.62 | 1.1 | 1.88 | 2.88 | 3.69 | 4.39 | 5.08 | 5.74 | 6.46 | 7.19 | 8.07 | 8.17 |
| QAP | tho40 | 0.96 | 1.9 | 2.77 | 3.57 | 3.83 | 5.01 | 5.69 | 6.15 | 6.41 | 6.91 | 6.79 | 7.29 |
|  | esc64a | 0.95 | 1.61 | 2.32 | 3.06 | 3.69 | 4.4 | 4.96 | 5.4 | 5.54 | 6.19 | 6.58 | 6.85 |
|  | sko72 | 0.98 | 1.95 | 2.9 | 3.9 | 4.83 | 5.65 | 6.46 | 7.53 | 8.44 | 8.97 | 10.07 | 10.93 |
|  | willo | 0.99 | 1.96 | 2.94 | 3.89 | 4.88 | 5.83 | 6.78 | 7.68 | 8.63 | 9.43 | 10.44 | 11.24 |
| GAP | gapA10-100 | 0.97 | 1.42 | 2.15 | 2.69 | 3.3 | 3.52 | 4.01 | 4.16 | 4.56 | 4.22 | 4.77 | 4.46 |
|  | gapA10-200 | 0.99 | 1.78 | 2.62 | 3.78 | 4.18 | 4.94 | 5.83 | 6.97 | 7.12 | 8.22 | 8.69 | 9.47 |
|  | gapA20-100 | 0.97 | 1.59 | 2.56 | 2.94 | 3.36 | 3.78 | 4.16 | 4.56 | 4.72 | 5.15 | 5.03 | 5.86 |
|  | gapA20-200 | 0.98 | 1.62 | 2.49 | 3.23 | 3.99 | 4.74 | 5.62 | 5.4 | 6 | 6.94 | 7.68 | 8.31 |
| BIN | bin3al | 0.99 | 1.71 | 2.45 | 3.16 | 3.84 | 4.65 | 5.2 | 6.02 | 6.59 | 7.4 | 7.89 | 8.57 |
|  | bin3a2 | 0.99 | 1.78 | 2.55 | 3.29 | 4.06 | 4.88 | 5.51 | 6.32 | 7.09 | 7.77 | 8.42 | 9.07 |
|  | bin3a3 | 0.99 | 1.71 | 2.48 | 3.20 | 3.91 | 4.65 | 5.36 | 6.01 | 6.73 | 7.68 | 8.15 | 8.79 |
|  | bin3a4 | 0.99 | 1.74 | 2.46 | 3.28 | 4.01 | 4.72 | 5.41 | 6.13 | 6.75 | 7.76 | 8.19 | 8.68 |
| TSP | st70 | 0.99 | 1.76 | 2.58 | 3.62 | 4.4 | 5.2 | 5.79 | 6.1 | 7.04 | 7.82 | 8.13 | 8.69 |
|  | kroAl00 | 0.69 | 1.32 | 1.96 | 2.42 | 2.96 | 3.35 | 3.47 | 4.22 | 4.57 | 4.95 | 5.21 | 5.73 |
|  | ch130 | 0.99 0.88 | 1.94 1.78 | 2.83 | 3.72 3.37 | 4.62 | 5.31 | 6.22 | 7.04 | 7.8 | 8.55 | 9.49 | 10.05 |
|  | a280 | 0.88 | 1.78 | 2.57 | 3.37 | 4.18 | 5.05 | 5.83 | 6.53 | 7.47 | 8.11 | 8.95 | 9.77 |

Table 24: Parallel speedup results.

| Problem | Instance | Processors |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| GPP | G250.01 | 0.63 | 0.48 | 0.57 | 0.61 | 0.62 | 0.62 | 0.63 | 0.65 | 0.63 | 0.62 | 0.61 | 0.6 |
|  | G250.02 | 0.65 | 0.51 | 0.6 | 0.67 | 0.66 | 0.66 | 0.67 | 0.69 | 0.69 | 0.67 | 0.64 | 0.65 |
|  | G250.04 | 0.67 | 0.57 | 0.62 | 0.66 | 0.71 | 0.74 | 0.73 | 0.75 | 0.73 | 0.7 | 0.69 | 0.73 |
|  | G250.08 | 0.62 | 0.55 | 0.63 | 0.72 | 0.74 | 0.73 | 0.73 | 0.72 | 0.72 | 0.72 | 0.73 | 0.68 |
| QAP | tho40 | 0.96 | 0.95 | 0.92 | 0.89 | 0.77 | 0.83 | 0.81 | 0.77 | 0.71 | 0.69 | 0.62 | 0.61 |
|  | esc64a | 0.95 | 0.81 | 0.77 | 0.76 | 0.74 | 0.73 | 0.71 | 0.67 | 0.62 | 0.62 | 0.6 | 0.57 |
|  | sko72 | 0.98 | 0.97 | 0.97 | 0.97 | 0.97 | 0.94 | 0.92 | 0.94 | 0.94 | 0.9 | 0.92 | 0.91 |
|  | will 00 | 0.99 | 0.98 | 0.98 | 0.97 | 0.98 | 0.97 | 0.97 | 0.96 | 0.96 | 0.94 | 0.95 | 0.94 |
| GAP | gapA10-100 | 0.97 | 0.71 | 0.72 | 0.67 | 0.66 | 0.59 | 0.57 | 0.52 | 0.51 | 0.42 | 0.43 | 0.37 |
|  | gapA10-200 | 0.99 | 0.89 | 0.87 | 0.95 | 0.84 | 0.82 | 0.83 | 0.87 | 0.79 | 0.82 | 0.79 | 0.79 |
|  | gapA20-100 | 0.97 | 0.8 | 0.85 | 0.74 | 0.67 | 0.63 | 0.59 | 0.57 | 0.52 | 0.52 | 0.46 | 0.49 |
|  | gapA20-200 | 0.98 | 0.81 | 0.83 | 0.81 | 0.8 | 0.79 | 0.80 | 0.67 | 0.67 | 0.69 | 0.7 | 0.69 |
| BIN | bin3al | 0.99 | 0.85 | 0.82 | 0.79 | 0.77 | 0.78 | 0.74 | 0.75 | 0.73 | 0.74 | 0.72 | 0.71 |
|  | bin3a2 | 0.99 | 0.89 | 0.85 | 0.82 | 0.81 | 0.81 | 0.79 | 0.79 | 0.79 | 0.78 | 0.77 | 0.76 |
|  | bin3a3 | 0.99 | 0.85 | 0.83 | 0.8 | 0.78 | 0.78 | 0.77 | 0.75 | 0.75 | 0.77 | 0.74 | 0.73 |
|  | bin3a4 | 0.99 | 0.87 | 0.82 | 0.82 | 0.8 | 0.79 | 0.77 | 0.77 | 0.75 | 0.78 | 0.75 | 0.72 |
| TSP | st70 | 0.99 | 0.88 | 0.86 | 0.91 | 0.88 | 0.87 | 0.83 | 0.76 | 0.78 | 0.78 | 0.74 | 0.72 |
|  | kroAl00 | 0.69 | 0.66 | 0.65 | 0.61 | 0.59 | 0.56 | 0.5 | 0.53 | 0.51 | 0.49 | 0.47 | 0.48 |
|  | ch130 | 0.99 | 0.97 | 0.94 | 0.93 | 0.92 | 0.88 | 0.89 | 0.88 | 0.87 | 0.85 | 0.86 | 0.84 |
|  | a280 | 0.88 | 0.89 | 0.86 | 0.84 | 0.84 | 0.84 | 0.83 | 0.82 | 0.83 | 0.81 | 0.81 | 0.82 |

Table 25: Parallel efficiency results.


Figure 19: Parallel speedup graph for the GPP.


Figure 20: Parallel efficiency graph for the GPP.


Figure 21: Parallel speedup graph for the GAP.


Figure 22: Parallel efficiency graph for the GAP.


Figure 23: Parallel speedup graph for the QAP.


Figure 24: Parallel efficiency graph for the QAP.


Figure 25: Parallel speedup graph for the BIN.


Figure 26: Parallel efficiency graph for the BIN.


Figure 27: Parallel speedup graph for the TSP.


Figure 28: Parallel efficiency graph for the TSP.

The performance of the parallel TS is generally dependent on the complexity of the objective function and the number of neighbours that are evaluated at each iteration. This is due to the parallelisation strategy adopted by this study. Parallel communication and housekeeping activities degrade the performance of the algorithm especially where the neighbourhood is small. This is evident when one considers that as the number of processors is increased, the time required to broadcast the neighbourhood division details and to process the incoming neighbours (sequential tasks), becomes large. Hence there is a reduction in efficiency as the number of processors increases. However the large QAPs generally record efficiencies above $90 \%$. This is due to the QAP having the most complex incremental cost expression of this set of problems and a large number of neighbours to evaluate at each iteration of the TS algorithm.

As well as the above, parallel speedup and efficiency are adversely affected by the use of incremental cost expressions. This is because incremental cost expressions reduce the amount of computation required for the evaluation of the neighbourhood (i.e. the component of TS that has been parallelised). In order to demonstrate this, a problem instance that the parallel algorithm performed poorly on, gapA10-100, was run without using the incremental cost expression (Table 26). Both speedup and efficiency are improved, though the performance of TS is substantially degraded. This is verified using the Mann-Whitney $U$ test that compared efficiency with and without incremental cost expressions and gave $\mathrm{P}=0.02$.

| Processors | With Incremental Cost Expressions |  | Without Incremental Cost Expressions |  |
| :---: | :---: | :---: | :---: | :---: |
|  | Speedup | Efficiency | Speedup | Efficiency |
| 1 | 0.97 | 0.97 | 0.99 | 0.99 |
| 2 | 1.43 | 0.71 | 1.78 | 0.89 |
| 3 | 2.15 | 0.71 | 2.52 | 0.84 |
| 4 | 2.69 | 0.67 | 3.27 | 0.82 |
| 5 | 3.30 | 0.66 | 3.97 | 0.79 |
| 6 | 3.52 | 0.59 | 4.64 | 0.77 |
| 7 | 4 | 0.57 | 5.53 | 0.79 |
| 8 | 4.16 | 0.52 | 5.88 | 0.74 |
| 9 | 4.57 | 0.51 | 6.55 | 0.73 |
| 10 | 4.22 | 0.42 | 6.77 | 0.68 |
| 11 | 4.77 | 0.43 | 7.23 | 0.66 |
| 12 | 4.46 | 0.37 | 7.46 | 0.62 |

Table 26: Parallel speedup and efficiency for gapA10-100 with and without using incremental cost expressions.

### 5.10 Performance Analysis

### 5.10.1 The Effect of the Transition Operator Sets and Engine-specific Parameters

In order to determine the most effective transition operator set for a particular problem type, the Kruskal-Wallis procedure is used. "Transition operator set" was the independent (grouping) variable while objective cost was the dependent variable. For many of the problem types, a significant result was recorded which indicates that the choice of transition operator probabilities affects quality of solution.

Table 27 shows the order of performance of the transition operator sets for each problem type ${ }^{13}$. For instance, in the case of BIN problems, transition set 2 (rank 1) was the best overall while set 1 (rank 5) was the worst. In the case of UMApHMP, sets $2,3,4$ and 6 recorded equivalent objective costs and therefore "runtime" was used to distinguish between these sets.

The GS and TS engines are run with the best transition operator set (rank order 1) for each problem type.

[^12]| Problem | Transition Set Reference Number |  |  |  |  |  |  |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Significance |  |  |  |  |  |  |  |  |
| (P) |  |  |  |  |  |  |  |  |

Table 27: Rank order of the transition sets for each problem type.

The same technique that has been applied in Table 27 has been used in Table 28 and Table 29 to determine the most effective SA cooling length and TS probabilistic candidate list setting respectively. For instance, a cooling length of 10000 generally outperforms lengths of 1000 and 5000 for the CSP. In order to make an overall assessment, the average rank of the cooling length and neighbourhood probability setting is produced. From this it can be seen that:

- $S A$ - Longer cooling lengths typically improve solution quality.
- TS - Evaluating half of the neighbours at each iteration is usually more effective than evaluating all of the neighbours.

| Problem | SA Cooling Length |  |  | Significance |
| :--- | :--- | :--- | :--- | :---: |
|  | 1000 | 5000 | 10000 | (P) |
| CSP | 3 | 2 | 1 | .017 |
| BIN | 3 | 2 | 1 | .000 |
| GRAPH | 3 | 2 | 1 | .000 |
| USApHMP | 3 | 1 | 2 | .000 |
| UMApHMP | 3 | 2 | 1 | .007 |
| QAP | 2 | 1 | 3 | .670 |
| TSP | 3 | 2 | 1 | .007 |
| GAP | 3 | 2 | 1 | .082 |
| GPP | 2 | 1 | 3 | .000 |
| MKP | 3 | 2 | 1 | .000 |
| MCP | 3 | 2 | 1 | .151 |
| TTP | 3 | 2 | 1 | .000 |
| Average | $\mathbf{2 . 8 3}$ | $\mathbf{1 . 7 5}$ | $\mathbf{1 . 4 2}$ |  |
| Rank |  |  |  |  |

Table 28: Rank order of the different SA cooling lengths for each problem type.

|  | Neighbour Probability |  |  | Significance |
| :--- | :--- | :--- | :--- | :---: |
| Problem | 0.1 | 0.5 | 1 | $(\mathbf{P})$ |
| CSP | 2 | 1 | 3 | .013 |
| BIN | 1 | 2 | 3 | .002 |
| GRAPH | 2 | 1 | 3 | .047 |
| USApHMP | 3 | 1 | 2 | .000 |
| UMApHMP | 1 | 2 | 3 | .001 |
| QAP | 1 | 2 | 3 | .000 |
| TSP | 3 | 1 | 2 | .000 |
| GAP | 3 | 2 | 1 | .000 |
| GPP | 1 | 2 | 3 | .000 |
| MKP | 2 | 1 | 3 | .000 |
| MCP | 1 | 2 | 3 | .000 |
| TTP | 3 | 1 | 2 | .024 |
| Average | $\mathbf{1 . 9 2}$ | $\mathbf{1 . 5}$ | $\mathbf{2 . 5 8}$ |  |
| Rank |  |  |  |  |

Table 29: Rank order of the different TS neighbourhood probabilities for each problem type.

### 5.10.2 Overall Comparison

Table 30 to Table 41 displays the results of rank ordering the performance of the various solvers on the problem instances. The procedure for calculating the ranks is described in Section 5.8.3.2. For instance in Table 30, the SA and TS engines produce very similar results (in terms of solution quality and time to solve) for the CSP problem instance n 20 t 2 . As these results are significantly superior to the GS and SA-CSP search engines, SA and TS share a ranking of 1. Moreover, as SA-CSP produces significantly better results than GS, these techniques receive the ranks of 2 and 3 respectively. The entry in the Significance ( $\mathbf{P}$ ) column indicates that likelihood (probability) of this set of ranks occurring by chance is .000 (i.e. these ranks are reliable). The Average Rank row at the bottom of each of the tables gives an approximate indication of the overall performance of each search engine on the problem type. For instance, in Table 30 it can be seen that SA and TS generally produce low ranks (such as 1 or 2 ) compared to SA-CSP and GS. Therefore, SA and TS are the best overall search engines for the CSP.

Table 42 shows the overall rankings of the search engines for each of the problem types. These overall rankings were calculated from the ranks given in Table 30 to Table 41 using the Kruskal Wallis procedure (described in Section 5.8.3.2). This table is read in the same way as Table 30 to Table 41. For instance, it can be seen that SA and TS perform equally well on the CSP as they share a rank of 1. The performances of GS and SA-CSP are similar but worse than SA and TS, hence they both receive a ranking of 2. As Lindo and OSL could not process the problem formulations, their performance is given a rank of 3 .

The following abbreviations refer to the various solver systems:

- SA-REST is the general SA engine using the feasibility restoration technique.
- TS-REST is the general TS engine using the feasibility restoration technique.
- EUS and EUM refer to the results for the USApHMP and UMApHMP problem instances given in Table 20 and Table 21 respectively.

| Problem | Solver |  |  |  | $\begin{gathered} \text { Significance } \\ (\mathbf{P}) \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | SA engine | TS engine | GS engine | SA-CSP |  |
| n20tl | 2 | 1 | 4 | 3 | . 000 |
| n20t2 | 1 | 1 | 3 | 2 | . 000 |
| n2013 | 1 | 2 | 4 | 3 | . 000 |
| n20t4 | 1 | 2 | 4 | 3 | . 000 |
| n20t5 | 1 | 3 | 4 | 2 | . 000 |
| n40tl | 2 | 1 | 4 | 3 | . 000 |
| n40t2 | 1 | 1 | 3 | 2 | . 000 |
| n40t3 | 2 | 1 | 4 | 3 | . 000 |
| n40t4 | 1 | 2 | 4 | 3 | . 000 |
| n40t5 | 1 | 2 | 4 | 3 | . 000 |
| n60t1 | 2 | 1 | 4 | 3 | . 000 |
| n60t2 | 2 | 1 | 4 | 3 | . 000 |
| n60t3 | 1 | 2 | 3 | 4 | . 000 |
| n60t4 | 1 | 2 | 3 | 4 | . 000 |
| n60t5 | 1 | 2 | 4 | 3 | . 000 |
| n80tl | 2 | 1 | 3 | 4 | . 000 |
| n80t2 | 1 | 1 | 3 | 2 | . 000 |
| n80t3 | 1 | 2 | 3 | 4 | . 000 |
| n80t4 | 1 | 2 | 3 | 4 | . 000 |
| n80t5 | 1 | 2 | 4 | 3 | . 000 |
| Average Rank | 1.3 | 1.6 | 3.6 | 3.05 |  |

Table 30: Performance comparison on the CSP.

|  | Solver |  |  |  |
| :--- | :--- | :--- | :--- | :---: |
| Problem | SA engine | TS engine | GS engine | Sificance <br> (P) |
| binlal | 1 | 1 | 2 | .000 |
| binla2 | 1 | 1 | 2 | .000 |
| binla4 | 1 | 1 | 2 | .000 |
| binla6 | 1 | 1 | 2 | .000 |
| bin2a1 | 1 | 1 | 2 | .000 |
| bin2a2 | 1 | 1 | 2 | .000 |
| bin2a3 | 1 | 1 | 2 | .000 |
| bin2a4 | 1 | 1 | 2 | .000 |
| bin3al | 1 | 2 | 3 | .000 |
| bin3a2 | 1 | 2 | 3 | .000 |
| bin3a3 | 1 | 2 | 3 | .000 |
| bin3a4 | 1 | 2 | 3 | .000 |
| Average | $\mathbf{1}$ | $\mathbf{1 . 3 3}$ | $\mathbf{2 . 3 3}$ |  |
| Rank |  |  |  |  |

Table 31: Performance comparison on the BIN.

|  | Solver |  |  |  |
| :--- | :--- | :--- | :--- | :---: |
| Problem | SA engine | TS engine | GS engine | Sificance <br> (P) |
| C125.1 | 1 | 2 | 3 | .000 |
| C125.5 | 1 | 1 | 2 | .000 |
| C125.9 | 1 | 2 | 3 | .000 |
| C250.1 | 1 | 2 | 3 | .000 |
| C250.5 | 1 | 2 | 3 | .000 |
| C250.9 | 1 | 2 | 3 | .000 |
| Average <br> Rank | $\mathbf{1}$ | $\mathbf{1 . 8 3}$ | $\mathbf{2 . 8 3}$ |  |

Table 32: Performance comparison on GRAPH.

| Problem | Solver |  |  |  |  |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :---: |
|  | SA engine | TS engine | GS engine | QAPSIM | QAPBB $^{14}$ | Significance <br> $(\mathbf{P})$ |
|  | 1 | 1 | 2 | 2 | 1 | .001 |
| nug12 | 1 | 1 | 2 | 2 | 1 | .000 |
| nug15 | 1 | 2 | 4 | 3 | 5 | .000 |
| nug20 | 1 | 2 | 4 | 3 | 5 | .000 |
| nug30 | 1 | 2 | 4 | 3 | 5 | .000 |
| ste36a | 1 | 2 | 4 | 3 | 5 | .000 |
| tho40 | 1 | 2 | 4 | 3 | 5 | .000 |
| esc64a | 3 | 2 | 4 | 1 | 5 | .000 |
| sko72 | 1 | 2 | 4 | 3 | 5 | .000 |
| will00 | 1 | 2 | 4 | 3 | 5 | .000 |
| Average | $\mathbf{1 . 2}$ | $\mathbf{1 . 8}$ | $\mathbf{3 . 6}$ | $\mathbf{2 . 6}$ | $\mathbf{4 . 2}$ |  |
| $\quad$ Rank |  |  |  |  |  |  |

Table 33: Performance comparison on QAP.

|  | Solver |  |  |  | Significance |
| :--- | :--- | :--- | :--- | :--- | :---: |
| Problem | SA engine | TS engine | GS engine | tsp_solve | (P) |
| gr24 | 2 | 3 | 4 | 1 | .000 |
| swiss42 | 3 | 2 | 4 | 1 | .000 |
| hk48 | 2 | 3 | 4 | 1 | .000 |
| eil51 | 2 | 1 | 3 | 1 | .000 |
| brazil58 | 3 | 2 | 4 | 1 | .000 |
| st70 | 3 | 2 | 4 | 1 | .000 |
| kroA100 | 1 | 2 | 3 | 4 | .000 |
| ch130 | 1 | 2 | 3 | 4 | .000 |
| a280 | 1 | 2 | 3 | 4 | .000 |
| Average | 2 | $\mathbf{2 . 1}$ | $\mathbf{3 . 5 6}$ | 2 |  |
| $\quad$ Rank |  |  |  |  |  |

Table 34: Performance comparison on TSP.

| Problem | Solver |  |  |  |  | Significance <br> (P) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | SA engine | TS engine | GS engine | OSL | LINDO |  |
| pb6 | 4 | 3 | 5 | 1 | 2 | . 000 |
| sent01 | 3 | 4 | 5 | I | 2 | . 000 |
| sent02 | 3 | 4 | 5 | 1 | 2 | . 000 |
| weingl | 3 | 2 | 4 | 1 | 1 | . 000 |
| weing 12 | 2 | 3 | 4 | 1 | 1 | . 000 |
| weing 15 | 2 | 3 | 4 | 1 | 1 | . 000 |
| weing 25 | 3 | 4 | 5 | 1 | 2 | . 000 |
| weing 29 | 2 | 3 | 4 | 1 | 1 | . 000 |
| Average Rank | 2.75 | 3.25 | 4.5 | 1 | 1.5 |  |

Table 35: Performance comparison on MKP.

[^13]|  | Solver |  |  |  |
| :--- | :--- | :--- | :--- | :---: |
| Problem | SA engine | TS engine | GS engine | Significance <br> (P) |
| G124.02 | 1 | 1 | 2 | .000 |
| G124.04 | 1 | 1 | 2 | .000 |
| G124.08 | 1 | 2 | 3 | .000 |
| G124.16 | 1 | 2 | 3 | .000 |
| G250.01 | 1 | 2 | 3 | .000 |
| G250.02 | 1 | 2 | 3 | .000 |
| G250.04 | 1 | 2 | 3 | .000 |
| G250.08 | 1 | 2 | 3 | .000 |
| Average | $\mathbf{1}$ | $\mathbf{1 . 7 5}$ | $\mathbf{2 . 7 5}$ |  |
| Rank |  |  |  |  |

Table 36: Performance comparison on GPP.

| Problem | Solver |  |  | $\begin{gathered} \text { Significance } \\ (P) \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: |
|  | SA engine | TS engine | GS engine |  |
| ttgen 1 | 1 | 2 | 3 | . 000 |
| ttgen2 | 1 | 2 | 3 | . 000 |
| hdtt4 | 1 | 2 | 3 | . 000 |
| hdtt5 | 1 | 2 | 3 | . 000 |
| hdtt6 | 1 | 2 | 3 | . 000 |
| hdtt7 | 1 | 2 | 3 | . 000 |
| hdtt8 | 1 | 2 | 3 | . 000 |
| Average Rank | 1 | 2 | 3 |  |

Table 37: Performance comparison on TTP.

| Problem | Solver |  |  | Significance <br> (P) |
| :--- | :--- | :--- | :--- | :---: |
|  | SA engine | TS engine | GS engine |  |
| brock200_1 | 2 | 1 | 3 | .000 |
| brock200_2 | 1 | 1 | 2 | .000 |
| brock200_3 | 1 | 2 | 3 | .000 |
| c-fat200-1 | 1 | 1 | 2 | .000 |
| johnson8-2-4 | 1 | 1 | 2 | .000 |
| johnson16-2-4 | 1 | 1 | 2 | .000 |
| keller4 | 1 | 2 | 3 |  |
| Average <br> Rank | $\mathbf{1 . 1 4}$ | $\mathbf{1 . 2 9}$ | $\mathbf{2 . 4 3}$ |  |

Table 38: Performance comparison on MCP.

| Problem | Solver |  |  | Significance <br> (P) |
| :--- | :--- | :--- | :--- | :---: |
|  | SA engine | TS engine | GS engine |  |
| ap20a2 | 2 | 1 | 3 | .000 |
| ap20a3 | 2 | 1 | 3 | .000 |
| ap20a4 | 2 | 1 | 3 | .000 |
| ap20a5 | 2 | 1 | 3 | .000 |
| ap25a2 | 2 | 1 | 3 | .000 |
| ap25a3 | 2 | 1 | 3 | .000 |
| ap25a4 | 2 | 1 | 3 | .000 |
| ap25a5 | 2 | 1 | 3 | .000 |
| Average | 2 | $\mathbf{1}$ | $\mathbf{3}$ |  |
| Rank |  |  |  |  |

Table 39: Performance comparison on USApHMP.

| Problem | Solver |  |  | Significance |
| :--- | :--- | :--- | :--- | :---: |
|  | SA engine | TS engine | GS engine | .000 <br> ap10a2 |
| ap10a3 | 1 | 2 | 3 | .000 |
| ap10a4 | 1 | 2 | 3 | .000 |
| apl0a5 | 1 | 2 | 3 | .000 |
| Average <br> Rank | 1 | 2 | 3 |  |

Table 40: Performance comparison on UMApHMP.

| Problem | Solver |  |  |  |  |  |  | $\begin{gathered} \text { Significance } \\ (P) \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | SA engine | TS engine | GS engine | SA-REST | TS-REST | OSL | LINDO |  |
| gap 1-1 | 1 | 1 | 3 | 2 | 1 | 1 | 1 | . 000 |
| gap2-1 | 1 | 1 | 3 | 2 | 1 | 1 | 1 | . 000 |
| gap3-1 | I | 1 | 3 | 2 | 2 | 1 | 1 | . 000 |
| gap4-1 | 1 | 1 | 3 | 2 | 2 | 1 | 1 | . 000 |
| gap5-1 | 2 | 1 | 4 | 3 | 3 | 2 | 2 | . 000 |
| gap6-1 | 1 | 1 | 4 | 3 | 2 | 1 | 1 | . 000 |
| gap7-1 | 1 | 1 | 3 | 2 | 2 | 4 | 1 | . 000 |
| gap8-1 | 1 | 1 | 3 | 2 | 2 | 4 | 4 | . 000 |
| gapA5-100 | 3 | 2 | 4 | 5 | 5 | 1 | 1 | . 000 |
| gapA5-200 | 2 | 2 | 2 | 3 | 3 | 1 | 1 | . 000 |
| gapA10-100 | 2 | 3 | 2 | 4 | 1 | 1 | 1 | . 000 |
| gapA10-200 | 2 | 2 | 2 | 3 | 3 | 1 | 1 | . 000 |
| gapA20-100 | 1 | 3 | 4 | 2 | 3 | 1 | 1 | . 000 |
| gapA20-200 | 1 | 2 | 2 | 1 | 3 | 4 | 4 | . 000 |
| Average Rank | 1.42 | 1.57 | 3 | 2.57 | 2.36 | 1.71 | 1.28 |  |

Table 41: Performance comparison on GAP.

| Problem | General Solver |  |  |  |  | Other Software |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | SA engine | TS engine | GS engine | SA-REST | TS-REST | SA-CSP | OSL | Lindo | QAPSIM | QAPBB | RLF | dfmax | dfclique | EUS | EUM | tsp_solve |
| CSP | 1 | 1 | 2 | - | - | 2 | 3 | 3 | - | - | - | - | - | - | - | - |
| BIN | 1 | 2 | 3 | - | - | - | 4 | 4 | - | - | - | - | - | - | - | - |
| GRAPH | 1 | 2 | 4 | - | - | - | 5 | 5 | - | - | 3 | - | - | . | - | - |
| QAP | 1 | 2 | 3 | - | - | - | 5 | 5 | 3 | 4 | - | - | - | - | - | - |
| TSP | 1 | 1 | 2 | - | - | - | 3 | 3 | - | - | - | - | - | - | - | 1 |
| MKP | 2 | 2 | 3 | - | - | - | 1 | 1 | - | - | - | - | - | - | - | - |
| GPP | 1 | 2 | 3 | - | - | - | 4 | 4 | - | - | - | - | - | - | - | - |
| USAPHMP | 3 | 2 | 4 | - | - | - | - | - | - | - | - | - | - | 1 | - | - |
| UMAPHMP | 2 | 3 | 4 | - | - | - | - | - | - | - | - | - | - | - | 1 | - |
| MCP | 2 | 3 | 4 | - | - | - | 5 | 5 | - | - | - | 1 | 1 | - | - | - |
| TTP | 1 | 2 | 3 | 2 | - | - | 1 | 1 | - | - | - | - | - | - | - | - |
| GAP | 1 | 1 | 2 | 2 | 2 | - | 1 | 1 | - | - | - | - | - | - | - | - |

Table 42: Overall comparison of the performance of the various solvers.

Table 43 presents the ordering of the search engines based on the ranks given in Table 42. $a \phi b$ is used to denote that search engine $a$ had better performance than search engine $b$ while $\{a, b\}$ is used to show that search engines $a$ and $b$ exhibited similar performances.

| Problem | Ordering of Search Engines |
| :---: | :---: |
| CSP | \{SA, TS $\dagger \phi$ \{GS, SA-CSP\} $\phi$ \{OSL, Lindo\} |
| BIN | SA $\phi$ TS $\phi$ GS $\phi$ \{OSL, Lindo \} |
| GRAPH | SA $\phi$ TS $\phi$ RLF $\phi$ GS \{OSL, Lindo\} |
| QAP | SA $\phi$ TS $\phi$ (GS, QAPSIM\} $\phi$ QAPBB $\phi$ (OSL, Lindo) |
| TSP | \{SA, TS, tsp_solve \} $\phi$ GS $\phi$ \{OSL, Lindo \} |
| MKP | \{OSL, Lindo) $\phi$ [SA, TS $\dagger \phi$ GS |
| GPP | SA $\phi$ TS $\phi$ GS $\phi$ \{OSL, Lindo $\}$ |
| USApHMP | EUS $\phi$ TS $\phi$ SA $\phi$ GS |
| UMApHMP | EUM $\phi$ SA $\phi$ TS $\phi$ GS |
| MCP | \{dfmax, dfclique $\} \phi$ SA $\phi$ TS $\phi$ GS $\phi\{$ OSL, Lindo $\}$ |
| TTP | SA $\phi$ TS $\phi$ GS |
| GAP | \{SA, TS, OSL, Lindo) $\phi$ [GS, SA-REST, TS-REST\} |

Table 43: Ordering of the search engines based on their performances for each problem type.

Both SA and TS perform extremely well on the CSP, frequently finding improved solutions over the results provided by Smith et al. (1996b). The analysis shows that the SA implementation used by Smith et al. (1996b) is only as effective as the GS engine. SA and TS also perform well on the QAP as they routinely find optimal and near optimal (within a few per cent) solution costs. It is also shown that the system outperforms QAPBB and QAPSIM. There is no statistical difference between tsp_solve, SA and TS for the TSP. However, this may be due to the fact that tsp_solve could not return solutions for the three largest problem instances (kroA100, ch130 and a280). For the MCP, the heuristics, dfmax and dfclique, were always able to obtain the clique sizes reported in Battiti and Protasi (1995) and hence shared an equal first ranking.

The ILP packages are only effective on problems with dense $0-1$ structures such as the MKP and GAP. These solvers generally outperform the list-based meta-heuristics in terms of the time required to reach the optimal solution cost. However, Lindo and OSL could not deliver a solution to gap8-1 within 6400 seconds of runtime, even though the SA and TS engines typically found the optimal solution within a few seconds. The USAPHMP and the UMApHMP are also solved very efficiently using exact techniques (see Table 20 and Table 21 respectively), though the general system (SA and TS) is also capable of delivering optimal solutions in reasonable amounts of runtime.

### 5.10.3 Runtime Differences between the Various Solver Packages

This section describes the way in which each solver varies in the amount of runtime required to reach the optimal or best-known solution cost. This is done in order to give an overall idea of how the general system rates in terms of the amount of computational effort required to deliver effective solutions compared to other optimisation packages (such as OSL and Lindo). An analysis of this type is difficult as a result of the wide variability of runtimes received in the course of solving a particular
problem instance. Within the one set of 10 runs of a search engine, it is possible to record runtimes of as little as a few CPU seconds to thousands of CPU seconds. However, this may be attributed to the use of the random strategies embedded into the meta-heuristic search engines. While GS also uses random strategies, it displays a much smaller variation as it often descends quickly to the nearest local optimum and stops. As a consequence, the discussions concerning the performance of the general system's search engines are confined to best transition operator sets (as identified in Table 27) and the SA and TS search engines.

A general characteristic of both the SA and TS engines is that they find good quality solutions early in the run and spend the rest of the available runtime making smaller improvements. For instance, in the case of the QAP tho40, the SA engine finds a solution that is $2 \%$ above the best known solution cost in 21 CPU seconds and a solution that is $0.39 \%$ above the best-known in 959 seconds.

A comparison of the general system's performance to the other optimisation packages for each problem type is given below. Overall, the general system can require significantly larger runtimes to reach the same quality solutions as the heuristic and ILP programs in the worst case. However, this must be considered in context with the fact that the general system can solve a very wide range of problems and produce superior quality solutions in comparison to the other optimisation packages.

## CSP

SA-CSP solves CSP instances relatively quickly compared to the search engines of the general solver system. However, it must be noted that for $70 \%$ of the problem instances, the SA and TS engines find better solution costs than reported in Smith et al. (1996b). For the majority of the problem instances, the general system records runtimes that are approximately equal to SA-CSP in the best case. In the worst case, the general system can require in the order of 1000 times longer runtime than SA-CSP. However, a close inspection of individual runs reveals that both the SA and TS engines find the bestknown solution costs within a similar amount of runtime to SA-CSP. The general solver utilises the extra runtime to find improved solutions over those reported in Smith et al. (1996b).

## GRAPH

The RLF heuristic by Johnson et al. (1991b) terminates before 1 CPU second has elapsed for each problem instance. However, the solutions returned by this heuristic are typically of a low quality. This is in comparison to the SA and TS engines of the general system that use all of the available CPU time to produce solution costs that are closer to the best-known costs. In addition, the optimal coloring was reached for C125.1 using the SA engine.

## USApHMP and UMApHMP

The branch and bound code with the shortest path heuristic developed by Ernst and Krishnamoorthy (1997b) is a very efficient form of solving the USApHMP and the UMApHMP. However, in the best case, the general solver can produce optimal solutions in the same or less time for the majority of problem instances compared with this code. In the worst case, the SA engine is approximately 100 times slower ${ }^{15}$.

## QAP

The general system often produces solutions that are of better quality than QAPSIM and QAPBB in approximately the same amount of runtime. However, QAPSIM's runtimes are typically smaller for the larger problems. For instance, consider will 00 for which the SA engine produces solutions that are at most $0.33 \%$ above the best-known cost compared with QAPSIM that finds a solution that is $0.75 \%$ above the best-known cost. However, this comes at a price as the general system can require up to 100 times longer to produce this result. A close inspection of individual runs reveals that both the SA and TS engines find similar quality solution costs ( $0.75 \%$ above the best-known cost) to QAPSIM in approximately the same amount of runtime. In the best case, the SA search engine can produce a result that is $0.18 \%$ above the optimal solution in approximately 900 seconds.

## TSP

tsp_solve produces the optimal solution cost to each of the problem instances except the three largest (kroA100, ch130 and a280). For many of the problem instances, the general system can find the optimal solution in approximately the same amount of runtime as tsp_solve. However, for st70 (the largest problem that tsp_solve could solve), tsp_solve is at most three times faster than the general solver.

## GAP

Both OSL and Lindo are extremely efficient at finding the optimal solution to the GAP problem instances within a relatively short amount of computational time (except for gap7-1 and gap8-1). These runtimes are of approximately the same magnitude as required by the general system. However, some differences are evident, such as for gap6-1 in which OSL requires 20 times longer to obtain the optimal solution compared with the SA engine. Conversely, for gap20-200, the SA engine requires between 2 and 13 times longer to obtain the optimal solution compared with OSL.

[^14]
## MKP

OSL and Lindo typically solve MKP instances to optimality in far less runtime than either the SA and TS engines. In the best case, the SA engine produces optimal solutions for the majority of the problem instances in approximately the same amount of runtime as the Lindo package. However, in some cases (particularly for TS), the ILP codes solve the MKP problem instances 50 times faster than the general system.

## MCP

Both dfmax and dfclique are extremely efficient algorithms for the MCP problems as they solved all instances to optimality (or the best-known cost). This is contrast to the SA and TS engines that could only produce the optimal result for keller4. However, as noted in Section 5.9.3, there is some concern about the validity of some of the results produced by these programs.

### 5.11 Summary

In this chapter, a set of experiments was proposed in order to test the performance of the linked list modelling system and the general solver implementation. These experiments were designed to test the following:

- The effect of the transition operator probability sets on the performance of the search engines.
- The effect of the engine-specific parameter settings, such as SA cooling length and the neighbour selection probability, on the performance of the solver.
- The performance of the general solver compared to other optimisation packages (such as commercial LP codes and heuristic programs).
- The effectiveness of the feasibility restoration scheme for the move operator.
- The efficiency of the parallel TS engine.

The problem types and problem instances used in this study are representative of a wide range of problems in the literature. Both qualitative and quantitative descriptions of the results were presented in this chapter. The quantitative analysis relied on the use of statistical techniques in order to determine differences in performance between search engines and different parameter settings. According to Barr et al. (1995, p. 24) "statistically validated conclusions" are essential to studies of this nature but are frequently lacking in the literature.

The results of the various experiments indicated that:

- The general system is capable of solving a wide variety of COPs very effectively. It frequently finds the optimal and best-known solution costs in runtimes that can be as efficient as other optimisation packages. In addition, new best-known costs were recorded for some of the CSP and TTP problem instances.
- The choice of transition operators for a particular problem affects the quality of solution and runtime. Engine-specific parameter settings effect the performance of the search algorithms, but to a much lesser extent.
- The feasibility restoration scheme is computationally expensive in comparison to feasibility maintenance. The latter technique generally outperformed feasibility restoration in terms of both solution quality and runtime.
- The parallel TS engine reduces the amount of wall clock time required to solve COPs over the sequential version of the engine. This real time reduction is particularly evident for problems that exhibit the characteristics of having large neighbourhoods and complex incremental cost expressions (such as the QAP).

The next chapter presents an in-depth summary of the performance of the general system. It also describes a number of research projects that are now possible as a direct result of the work on linked list modelling systems for COPs. Finally, the achievements and new developments contained in this thesis are highlighted.

## Chapter 6: Conclusions and Further Work

### 6.1 Conclusions

The list modelling system allows COPs to be represented in an efficient and compact manner. As a result, widely used and reported local search operators can be applied directly to the solution structure. Accordingly, the list modelling system is well suited to supporting meta-heuristic search algorithms such as GS, TS, SA and GRASP. A general solver was built that accepts problems formulated in this list notation. Apart from recording good overall performance on a range of COPs compared with ILP and heuristic codes, the time to develop a problem description is minimised due to the compact algebraic language with which problems are described (see Section 4.2). A method of automatically producing incremental cost expressions was also developed. This contributed greatly to the system's performance.

The testing of the linked list modelling system was extensive and consisted of two main parts; standard and extended runs. Despite the short amount of time available to the standard runs, optimal and near optimal solutions were recorded for many problems. If the best solution for each problem is considered (disregarding the effects of both the transition operator probability sets and search engine specific parameters), the percentages of optimal solutions encountered by each of the search engines gives an encouraging picture (Table 44). For this particular implementation of the meta-heuristic search algorithms, it appears that SA typically outperforms TS. However, this is hardly a fair comparison because of two reasons: (a) different implementations of the algorithms will produce different results, and (b) TS was run using only one transition set per problem type while SA was tested under a range of conditions. Allowing SA and TS more runtime for problems in which the optimal or best-known solution could not be found resulted in further improvements (especially for TS). GS performs surprisingly well for some instances of the CSP and GAP. This can be potentially attributed to two factors: the ease of the problem instance and the contribution of the transition operator set. In the case of the CSP, there is only one effective transition operator (the swap), so it is not unreasonable to argue that these problem instances are easy to solve. For the GAP however, the combination of the move and the swap operator appears to form an efficient solving method in its own right (regardless of the search method used).

| Search Engine | Percent (\%) Optimal/Best-Known Costs |
| :--- | :--- |
| SA | 83 |
| TS | 69 |
| GS | 21 |

Table 44: The percentage of Optimal and Best-Known solution costs achieved by SA, TS and GS for the standard runs.

Feasibility restoration was implemented for the move operator and applied to the GAP. This scheme is more computationally intensive than straight feasibility maintenance as a number of individual transitions are performed at each step and incremental cost expressions are not used. If only the move operator is considered, feasibility restoration outperformed feasibility maintenance on most of the GAP problem instances (see Section 5.9.4). However, the combination of the feasibility-maintaining move with the swap operator is far more efficient than feasibility restoration by itself.

The solver allows multiple neighbourhoods to be evaluated in the course of solving a particular problem using an automatic technique that determines a set of appropriate transition operators without the need for changing the formulation of the problem. As a direct consequence of this, it is possible to gauge the effect of transition operators on solution quality and runtime. Generally, transitions that produce a small perturbation of the solution outperform those that change the solution greatly. Consider the TSP in which the application of the inversion operator requires that only four links of the tour change, whereas eight links change with the swap operator. The system, regardless of the search engine, records better performance with the inversion operator than with the swap operator. There are some problems for which a combination of operators is more suitable than one used in isolation. For instance, the move operator applied to the BIN problem produces very poor solutions, whereas the addition of the swap operator results in the optimal solution frequently being found (see Table 45, Appendix A). It is believed that this is the case because the move operator can often find it difficult to produce a feasible solution and is likely to give a large incremental cost. However, swap in isolation is also ineffective, as each bin will contain a fixed number of items throughout the search process, thus making many regions of the search space inaccessible.

Each problem type also utilised an adaptive probability setting in which the system changes the probability of each of the active transition operators according to its performance. In many cases, the adaptive probability setting performed well in comparison to other transition operator settings. However more work is needed to explore different adaptive rates and their effect on solution quality and runtime.

The engine-specific parameters also affect solution quality. In particular, the probabilistic candidate list structure alters the performance of both GS and TS. For most problems, it was found that evaluating a subsection of the neighbourhood was more effective that evaluating the entire neighbourhood (see Table 29). Further studies could be conducted to determine whether this is due to
the probabilistic sampling and/or the fact that more iterations are performed if a smaller subset of the neighbourhood is used.

The parallel implementation showed that the TS engine could be made more efficient, in terms of the actual amount of time required to solve a problem, by allowing the simultaneous evaluation of the cost functions of neighbouring solutions. However, as the use of incremental cost expressions reduces the computation overhead and as there is a large communication time associated with the parallelisation scheme, the parallel performance was somewhat limited.

### 6.2 Further Work

This research has shown that a variety of COPs can be solved efficiently by meta-heuristic search engines underpinned by the new linked list modelling system. However, while the experiments were extensive, they were not exhaustive. A greater range of problem types and parameter settings could be used to further test its performance.

An area that needs particular attention is the way in which the transition operators are applied. As noted in Section 3.4.1.2, there are a number of alternative methods that could be used. Even with the probabilistic approach demonstrated here, a greater range of probability sets could be used to determine the effectiveness of particular local search operators for different COPs.

An additional enhancement to the modelling language would be the incorporation of logical statements and operators. This includes an IF ELSE construct and logical operators such as OR, AND, XOR and NOT. It is believed that the use of logical statements and operators in conjunction with algebraic expressions would allow the modelling system to express more complex optimisation problems.

It is believed that a software product based on the linked list modelling system would have commercial application. The resulting program would greatly benefit from a graphical user interface that would allow the user to directly enter the list model using mathematical symbols. This description could be converted into a solver in much the same way as specified in this study.

In addition to the above, this project naturally leads on to the following areas of research:

- Direct hardware implementation of the list modelling system.
- The use of different dynamic data structures for the representation of COPs.
- Alternative search strategies.


### 6.2.1 Direct Hardware Implementation

The work reported here was undertaken as part of a larger project that aims to develop Application Specific Computers (ASCs) for solving COPs (Abramson 1992; Abramson, de Silva, Randall and Postula 1995 and Abramson, Logothetis, Randall and Postula 1997, 1998). The benefit of these systems is that a specific hardware design has real time advantages over conventional workstations. The architectures of these computers will be designed to support a range of meta-heuristic search techniques through system reconfiguration using the list modelling system as the general representation scheme. As such, they will be based on Field Programmable Gate Array (FPGA) technology. FPGAs allow hardware to be reconfigured without any physical modification. This offers enormous potential because it should be possible to build a reconfigurable special purpose architecture, which can be used to solve a range of optimisation problems. It is envisaged that the ASC will be connected to a conventional workstation to provide the interface (Figure 29).

The work is significant because it will allow a wide range of integer optimisation problems to be solved very rapidly using one inexpensive hardware and software platform. Thus it will be possible to solve a number of important practical problems using one system. The system will provide the advantages of generic algorithms with the speed of specific ones. The work will also lead to a conceptual advance in the role of special purpose computers.


Figure 29: Schematic of the ASC for solving COPs.

### 6.2.2 Alternative Dynamic Data Structures

Linked lists are very successful at representing a wide range of COPs efficiently. However, as discussed in Section 3.2, trees are an alternative dynamic data structure on which a general modelling system for COPs could be based. In particular, such a representation would be more appropriate than list modelling for solving problems such as the minimal spanning tree problem. The transition operator set could conceivably consist of adding, dropping and moving edges or sub-trees. However, general methods of indexing nodes and edges as well as ways of maintaining the integrity of the tree (such as ensuring that every node is connected by an edge without a circuit for the minimal spanning tree problem) would need to be developed.

### 6.2.3 Alternative Search Engines

This study has described the implementation of a general COP solver using three meta-heuristic engines, namely GS, TS and SA. However, there are other meta-heuristic algorithms that could be incorporated into the general system. One in particular is the GRASP technique (see Section 2.2.3.4). This can be implemented by modifying the initial solution procedure (described in Section 4.7) to act as the construction phase of the algorithm and using either the GS, TS or SA engines as the local search phase.

The list modelling system is particularly suited to meta-heuristics that are based on local search operators. However, it is believed that other techniques (in particular GAs and other evolutionary procedures) could be implemented in a modified version of the general environment. For GAs, the local search transition operators would be replaced by a set of recombination operators. However, as noted in Section 2.3.2.3, operators would need to be designed for specific problems, though it may be possible to use one operator for a group of similar problems. This is due to the fact that many problems have similar representations using the list notation. For instance, the QAP and TSP are very similar as they are both permutation problems that can be modeled with a fixed size sub-list and subsequently recombination could be performed using the PMX (partially match crossover) operator (Goldberg 1989) or its variants. It remains to be seen whether a rule base can be established that could match appropriate operators to problems (like the one in Table 3 that determines suitable local search operators). A GA search engine would also be particularly suited to parallel implementation as multiple solutions are generated and evaluated at every step of the algorithm.

### 6.3 Achievements and Significance

This research has presented a number of new concepts in the field of general-purpose meta-heuristic COP solvers. The most fundamental of which is a new modelling system specifically designed for

COPs. It has been shown that linked list data structures efficiently model combinatorial optimisation problems due to the natural mapping between COPs and lists.

The new modelling system is unique as it is based on a dynamic data structure, the linked list. A property of the new system is that it eliminates the need for artificial encoding constraints and variables common in ILP and CP modelling as it more directly represents the grouping characteristics of COPs. Using the linked list modelling system, a variety of local search transition operators can be directly applied to problem models (see Section 3.4.1). A method has also been developed that determines the appropriate local search operators for particular problems (see Table 3). As well as this, a probabilistic method for applying transition operators capable of adaptive behaviour was investigated.

A means of automatically generating incremental cost expressions was produced in order to allow the efficient evaluation of objective functions. This technique matches an objective function to one or more known templates so as to derive a suitable algebraic expression in list notation. Whilst this method is not entirely general, it has been applied successfully to a number of COPs. It also has the advantage that the set of templates can easily be expanded, should the need arise.

The linked list modelling system is used as the foundation of a new general-purpose solver named COSULOM. This system is capable of solving a wide range of COPs with meta-heuristics based on the use of local search transition operators. To date, GS, TS and SA engines have been implemented. The system accepts algebraic arbitrary formulations of COPs in GAMS like syntax (Brooke et al. 1997). This feature can dramatically reduce the development time for prototyping and solving COPs. Another property of the system is that it can produce feasible starting solutions to problems, which is a difficult task in itself.

In summary, it is believed that the work contained in this thesis has contributed to the knowledge of modelling systems for COPs and general-purpose meta-heuristic implementations in the following areas:

- The list modelling system is an efficient means of representing COPs as it models the fundamental "grouping, ordering and selection of discrete objects" characteristics of these problems (see Section 1.2). Encoding constraints used by other systems (in particular ILP and CP ) that can make COPs impractical to solve are eliminated in the linked list modelling system.
- The choice of transition operator(s) affects the performance of the search algorithm to a greater extent than specific meta-heuristic parameters.
- A general solver can have similar performance to special purpose solvers. The advantage of using a general solver is that the time to develop a problem description is relatively small compared with the construction of special purpose codes.
- Using a linked list modelling system for COPs, a technique of automatically generating initial feasible solutions to problems was developed. The generation of feasible solutions has been described as a difficult task in itself (Goldberg 1989).
- Using a linked list modelling system for COPs, a set of local search transition operators that preserve feasibility, can be automatically deduced using a rule base.
- Using a linked list modelling system for COPs, incremental cost expressions can be deduced from the original cost function. These expressions increase the efficiency of meta-heuristic search algorithms, as fewer operations are required to compute the change in cost than to calculate the entire cost.

As seen in Section 6.2, the work contained in this thesis has also given rise to a number of subsidiary projects.

## Bibliography

Abada, H. and El-Darzi, E. (1996) "Solving the Timetable Problem Using Tabu Search", University of Westminster, Working Paper.

Aboudi, R. and Jörnstern, K. (1994) "Tabu Search for General Zero-One Integer Programs using the Pivot and Complement Heuristic", ORSA Journal on Computing, 6, pp. 82-93.

Abramson, D. (1991) "Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms", Management Science, 37, pp. 98-113.

Abramson, D. (1992) "A Very High Speed Architecture to Support Simulated Annealing", IEEE Computer, pp. 27-34.

Abramson, D. and Dang, H. (1993) "School Timetables: A Case Study in Simulated Annealing", Applied Simulated Annealing, Lecture Notes in Economics and Mathematics Systems, Springer-Verlag, Ed: Vidal, V., Chapter 5, pp. 103-124.

Abramson, D., de Silva, A., Randall, M. and Postula, A. (1995) "Special Purpose Computer Architectures for High Speed Optimisation", Proceedings of the Second Australasian Conference on Parallel and Real Time Systems Conference, pp. 13-20.

Abramson, D., Dang, H. and Krishnamoorthy, M. (1993) "Enhanced Simulated Annealing Through Linear Programming Preprocessing", Proceedings of the 12th National Conference of the Australian Society for Operations Research, Adelaide, July 7-9, pp. 91114.

Abramson, D., Dang, H. and Krishnamoorthy, M. (1996) "A Comparison of Two Methods for Solving 0-1 Integer Programs Using a General Purpose Simulated Annealing", Annals of Operations Research, 63, pp. 129-150.

Abramson, D., Logothetis, P., Randall, M. and Postula, A. (1997) "Application Specific Computers for Combinatorial Optimisation", The Australian Computer Architecture Workshop, Sydney, Australia, February, pp. 29-43.

Abramson, D., Logothetis, P., Randall, M. and Postula, A. (1998) "A Tail Of $2^{n}$ Cities: Performing Combinatorial Optimisation Using Linked Lists On Special Purpose Computers", Keynote address, Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, February, Gipsland. pp. 17-45.

Abramson, D. and Randall, M. (1998) "A Simulated Annealing Code for General Integer Linear Programs", to appear in Annals of Operations Research.

Anbil, R., Gelman, E., Patty, B. and Tanga, R. (1991) "Recent Advances in Crew-Pairing Optimization at American Airlines", Interfaces, 21, pp. 62-74.

Bäck, T. (1992) "A User's Guide to GENEsYs 1.0", University of Dortmund, Department of Computer Science.

Badeau, P., Gendreau, M., Guerin, F., Potvin, J. and Taillard, E. (1997) "A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows", Transportation Research-C, 5, pp. 109-122.

Barendregt, H. (1984) The Lambda Calculus: Its Syntax and Semantics, North Holland, Amsterdam, 621 pages.

Barr, R. and Hickman, B. (1993) "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures and Experts' Opinions", ORSA Journal on Computing, 5, pp. 2-18.

Barr, R., Golden, B., Kelly, J., Resende, M. and Stewart, W. (1995) "Designing and Reporting on Computational Experiments with Heuristic Methods", Journal of Heuristics, 1, pp. 9-32.

Barnier, N. (1997) "Optimization by Hybridization of a Genetic Algorithm with CSP Techniques", Proceedings of the Second ESSLLI Student Session, Eds: Drewery, A., Geert-Jan, M. and Zuber, R..

Battiti, R. and Protasi, M. (1995) "Reactive Local Search for the Maximum Clique Problem", Technical Report - International Computer Science Institute, TR-95-052.

Battiti, R. and Tecchiolli, G. (1994) "The Reactive Tabu Search", ORSA Journal on Computing, 6, pp. 126-140.

Battiti, R. and Tecchiolli, G. (1995) "Local Search with Memory: Benchmarking RTS", Operations Research Spektrum, 17, pp. 67-86.

Beasley, J. (1990) "OR-library: Distributing Test Problems by Electronic Mail", Journal of the Operational Research Society, 41, pp. 1069-1072.

Beasley, J., Krishnamoorthy, M., Abramson, D. and Sharaiha, Y. (1995) "Scheduling Aircraft Landings - The Static Case", Submitted to Operations Research.

Beasley, J. and Chu, P. (1996) "A Genetic Algorithm for the Set Covering Problem", European Journal of Operational Research, 94, pp. 392-404.

Beasley, J. and Chu, P. (1997a) "A Genetic Algorithm for the Multiconstraint Knapsack Problem", Working Paper.

Beasley, J. and Chu, P. (1997b) "Constraint Handling in Genetic Algorithms: the Set Partitioning Problem", Working Paper.

Brooke, A., Kendrick, D., Meeraus, A. and Raman, R. (1997) GAMS Language Guide, GAMS Development Corporation, 305 pages.

Burkard, R., Karisch, S. and Rendl, F. (1997) "QAPLIB - A Quadratic Assignment Problem Library", Journal of Global Optimization, 10, pp. 391-403.

Campbell, J. (1994) "Integer Programming Formulations of Discrete Hub Location Problems", Annals of Operations Research, 75, pp. 387-405.

Carling, A. (1992) Introducing Neural Networks, Sigma Press, Wilmslow, 338 pages.
Caseau, Y. and Laburthe, F. (1997) "Solving Small TSPs with Constraints", Proceedings of the Joint International Conference and Symposium on Logic Programming, Ed: Naish, L., MIT Press.

Chartrand, G. (1997) Graphs as Mathematical Models, Prindle, Weber and Schmidt Inc, Boston MA, 294 pages.

Chams, M., Hertz, A. and de Werra, D. (1987) "Some Experiments with Simulated Annealing for Coloring Graphs", European Journal of Operational Research, 32, pp. 260-266.

Chanas, S. and Kobylanski, P. (1995) "A New Heuristic Algorithm Solving the Linear Ordering Problem", Technical Report - Institute of Industrial Engineering and Management, PL-50-372.

Chandy, J. and Prithviraj, B. (1996) "Parallel Simulated Annealing Strategies for VSLI Cell Placement", Proceedings of the $9^{\text {th }}$ International Conference on VSLI Design, Bangalore, India, January.

Christofides, N. and Eilon, S. (1969) "An Algorithm for the Vehicle Dispatching Problem", Operations Research Quarterly, 20, pp. 309-318.

Chu, P. and Beasley, J. (1997) "A Genetic Algorithm for the Generalised Assignment Problem", Computers and Operations Research, 24, 1997, pp. 17-23.

Clarke, G. and Wright, J. (1964) "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", Operations Research, 12, pp. 568-581.

Connolly, D. (1990) "An Improved Annealing for the QAP", European Journal of Operational Research, 46, pp. 93-100.

Connolly, D. (1992) "General Purpose Simulated Annealing", Journal of the Operational Research Society, 43, pp. 495-505.

Cook, S. (1971) "The Complexity of Theorem-Proving Procedures", Proceedings of the $3^{\text {rd }}$ Annual ACM Symposium on the Theory of Computing, pp. 151-158.

Crama, Y., Flippo, O., van de Klundert, J. and Spieksma, F. (1995) "The Assembly of Printed Circuit Boards: A Case With Multiple Machines and Multiple Board Types", Research Memorandum - Faculty of Economics and Business Administration, University of Limburg, 95/033.

Crainic, T., Toulouse, M. and Gendreau, M. (1997) "Toward a Taxonomy of Parallel Tabu Search Heuristics", INFORMS Journal on Computing, 9, pp. 61-71.

De Falco, I., Del Balio, R. and Tarantino, E. (1996) "Solving the Mapping Problem by Parallel Tabu Search", Instituto per la Ricerca sui Sistemi Informatici Paralli, Italy.

Eglesse, R. (1990) "Simulated Annealing: A Tool for Operational Research", European Journal of Operational Research, 46, pp. 271-281.

Emory, W. and Cooper, D. (1991) Business Research Methods, Irwin Inc, Boston MA, $4^{\text {th }}$ Edition, 760 pages.

Ernst, A. and Krishnamoorthy, M. (1996a) "Efficient Algorithms for the Uncapacitated Multiple Allocation p-Hub Median Problem", to appear in European Journal of Operational Research.

Ernst, A. and Krishnamoorthy, M. (1996b) "Efficient Algorithms for the Uncapacitated Single Allocation p-hub Median Problem", Locations Science (Special Issue on Hub Location), 4, pp. 139-154.

Ernst, A. and Krishnamoorthy, M. (1997a) "Solution Algorithms for the Capacitated Single Allocation Hub Location Problem", to appear in Annals of Operations Research.

Ernst, A. and Krishnamoorthy, M. (1997b) "A Novel Exact Solution Approach Based on Shortest Paths for p-hub Median Problems", Submitted to ORSA Journal on Computing.

Ernst, A., Krishnamoorthy, M. and Storer, R. (1997) "Exact and Heuristic Algorithms for Scheduling Aircraft Landings", Working Paper.

Feo, T. and Resende, M. (1989) "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem", Operations Research Letters, 8, pp. 67-71.

Feo, T. and Resende, M. (1995) "Greedy Randomised Adaptive Search Procedures", Journal of Global Optimization, 51, pp. 109-133.

Firebaugh, M. (1989) Artificial Intelligence, PWS-Kent, Boston MA, 740 pages.
Fourer, R., Gay, D. and Kernighan, B. (1993) AMPL: A Modeling Language for Mathematical Programming, Duxbury Press, California, 351 pages.

Garcia, B., Potvin J. and Rousseau, J. (1994) "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints", Computers and Operations Research, 21, pp. 1025-1033.

Gilmore, P. and Gomory, R. (1961) "A Linear Programming Approach to the Cutting Stock Problem", Operations Research, 9, pp. 849-859.

Glover, F. (1989) "Tabu Search - Part I", ORSA Journal on Computing, 1, pp. 190-206.
Glover, F. (1990) "Tabu Search - Part II", ORSA Journal on Computing, 2, pp. 4-32.
Glover, F. and Laguna, M. (1997) Tabu Search, Kluwer Academic Publishers, Boston MA, 402 pages.

Gould, R. (1988) Graph Theory, Benjamin/Cummings Publishing Company, California, 332 pages.

Goldberg, D. (1989) Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesely, Reading, 412 pages.

Grefenstette, J. (1987) "A User's Guide to GENISIS", Navy Center for Applied Research in Artificial Intelligence, Washington, D.C.

Hart, P., Nilsson, N. and Raphael, B. (1968) "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on SSC, SSC-4, pp. 100-107.

Hasan, M. and Osman, I. (1995) "Local Search Algorithms for the Maximal Planar Problem", International Transactions on Operational Research, 2, pp. 89-106.

Hertz, A., Taillard, E. and de Werra, D. (1997) "Tabu Search", Local Search in Combinatorial Optimization, Eds: Aarts, E. and Lenstra, J., Wiley, pp. 121-136.

Higgins, A., Kozan, E. and Ferreira, L. (1997) "Heuristic Techniques for Single Line Train Scheduling", Journal of Heuristics, 3, pp. 42-62.

Hopfield, J. (1982) "Neural Networks and Physical Systems with Emergent Collective Computational Responsibilities", Proceedings of the National Academy of Sciences, 79, pp. 2554-2558.

Hurwitz, C. (1994) GNU tsp_solve, Free Software Foundation, Inc., 25 pages.
IBM (1990) Optimisation Subroutine Library User Manual, IBM Corporation, 823 pages.
ILOG (1998) ILOG Optimization Suite White Paper, ILOG Corporation, 57 pages.
Ingber, L. (1993) "Simulated Annealing: Practice Versus Theory", Computer Modelling, 18, pp. 29-57.

Ingber, L. (1996) "Adaptive Simulated Annealing (ASA): Lessons Learned", Control and Cybernetics, 25, pp. 33-54.

Jaffar, J. and Maher, M. (1997) "Constraint Logic Programming: A Survey", to appear in Journal of Logic Programming.

Johnson, D. (1990) "Local Optimization and the Traveling Salesman Problem", Proceedings of the 17 ${ }^{\text {th }}$ International Colloquim on Automata, Languages and Programming, pp. 446-461.

Johnson, D., Aragon, C., McGeogh, L. and Scheveon, C. (1991a) "Optimisation by Simulated Annealing: An Experimental Evaluation Pt I, Graph Partitioning", Operations Research, 37, pp. 865-892.

Johnson, D., Aragon, C., McGeogh, L. and Scheveon, C. (1991b) "Optimisation by Simulated Annealing: An Experimental Evaluation Pt II, Graph Colouring and Number Partitioning", Operations Research, 39, pp. 378-406.

Johnson, D. and Trick, M. (1993) "Cliques, Coloring, and Satisfiability", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26.

Kampke, T. (1988) "Simulated Annealing: Use of a New Tool in Bin Packing", Annals of Operations Research, 16, pp. 327-332.

Kernighan, B. and Lin, S. (1970) "An Efficient Heuristic Procedure for Partitioning Graphs", The Bell System Technical Journal, pp. 291-307.

Kirpatrick, S., Gelatt, D. and Vecchi, M. (1983) "Optimization by Simulated Annealing", Science, 220, pp. 671-680.

Klincewicz, J. (1992) "Avoiding Local Optima in the p-hub Location Problem using Tabu Search and GRASP", Annals of Operations Research, 40, pp. 283-302.

Kohonen, T. (1982) "Self-organized Formation of Topological Correct Feature Maps", Biological Cybernetics, 43, pp. 53-69.

Kontoravdis, G. and Bard, J. (1995) "A GRASP for the Vehicle Routing Problem with Time Windows", ORSA Journal on Computing, 7, pp. 10-23.

Kouvelis, P. and Chiang, W. (1992) "A Simulated Annealing Procedure for Single Row Layout Problems in Flexible Manufacturing Systems", International Journal of Production Research, 30, pp. 717-732.

Koulamas, C., Antony, S. and Jansen, R. (1994) "A Survey of Simulated Annealing to Operations Research Problems", Omega International Journal of Management Science, 22, pp. 4156.

Krishnamoorthy, M., Ernst, A. and Beasley, J. (1997) "Formulations and Algorithms for the Personnel Task Scheduling Problem", Working Paper.

Lawler, E., Lenstra, J., Rinnoy Kan, A. and Shmoys, D. (1985) The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, Chichester, 465 pages.

Li, Y., Pardalos, P. and Resende, M. (1994) "A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem", in Quadratic Assignment and Related Problems, Ed. Pardalos, P. and Wolowicz, H., DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 16.

Lin, S. and Kernighan, B. (1973) "An Effective Heuristic Algorithm for the Traveling Salesman Problem", Operations Research, 21, pp. 498-516.

Little, J. and Darby-Dowman, K. (1995) "The Significance of Constraint Logic Programming to Operational Research", Operational Research Tutorial Papers, Eds: Lawrence, M. and Wilsdon, C., Operational Research Society, Birmingham.

Lokketangen, A., Jörnsten, K. and Storoy, S. (1994) "Tabu Search within a Pivot and Complement Framework", International Transactions in Operations Research, 1, pp. 305-316.

Luger, G. and Stubblefield, W. (1993) Artificial Intelligence, Benjamin/Cummings, California, 740 pages.

Marriot, K. and Stuckey, P. (1998) Constraint Programming: An Introduction, MIT Press, Cambridge MA, 476 pages.

Martello, S. and Toth, P. (1981) "An Algorithm for the Generalised Assignment Problem", Proceedings of the $9^{\text {th }}$ IFORS conference, Hamburg, Germany.

Mavridou, T., Pardalos, P., Pitsoulis, L. and Resende, M. (1995) "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP", Proceedings of Parallel Algorithms for Irregularly Structured Problems, and Springer-Verlag, Lecture Notes in Computer Science, 980, (1995), Eds: Ferreira, A. and Rolim, J., pp. 317-331.

Nugent, C., Vollman, T. and Ruml, J. (1968) "An Experimental Comparison of Techniques for the Assignment of Facilities to Locations", Operations Research, 16, pp. 150-173.

Or, I. (1976) "Traveling Salesman-type Combinatorial Optimization Problems and their Relation to the Logistics of Regional Blood Banking", PhD Dissertation, Northwestern University, Evanston, IL.

Osman, I. (1993) "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem", Annals of Operations Research, 41, pp. 421-451.

Osman, I. (1995) "Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches", Operations Research Spektrum, 17, pp. 211-225.

Osman, I. and Kelly, J. (1996) Metaheuristics: Theory and Applications, Kluwer Academic Publishers, Norwell MA, 704 pages.

Osman, I. and Laporte, G. (1996) "Metaheuristics: A Bibliography", Annals of Operations Research, 63, pp. 513-618.

Pardalos, P. and Wolkowicz, H. (1994) "Quadratic Assignment Problems and Related Problems", Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Volume 16, 370 pages.

Pearl, J. (1984) Heuristics: Intelligent Search strategies for Computer Problems Solving, Addison Wesely, Reading, 382 pages.

Petersen, C. (1967) "Computational Experience with Variants of the Balas Algorithm Applied to a Selection of R\&D Projects", Management Science, 13, pp. 736-750.

Peterson, C. and Södberg, B. (1989) "A New Method for Mapping Optimization Problems onto Neural Networks", International Journal of Neural Systems, 1, pp. 3-22.

Pohl, I. (1990) A Book on C, Benjamin/Cummings, California, 525 pages.
Ramanujam, J. and Sadayappan, P. (1995) "Mapping Combinatorial Optimization Problems onto Neural Networks", Information Sciences, 82, pp. 839-255.

Randall, M. (1995) "The Future and Applications of Genetic Algorithms", Proceedings of Electronic Development Towards the Year 2000 Conference, Ed: Jain, L., IEEE Computer Society Press, pp. 471-475.

Randall, M. and Abramson, D. (1998) "A General Meta-heuristic Based Solver for Combinatorial Optimisation Problems", Working Paper.

Reinelt, G. (1985) "The Linear Ordering Problem: Algorithms and Applications, Research and Exposition", Research and Exposition in Mathematics, 8, Heldermann Verlag, Berlin, 236 pages.

Reinelt, G. (1991) "TSPLIB - A Traveling Salesman Problem Library", ORSA Journal on Computing, 3, pp. 376-384.

Rosenkrantz, D., Stearns, R. and Lewis, P. (1977) "An Analysis of Several Heuristics for the Traveling Salesman Problem", SIAM Journal of Computing, 6, pp. 563-581.

Ruml, W., Ngo, J., Marks, J. and Shieber, S. (1995) "Easily Searched Encodings for Number Partitioning", to appear in Journal of Optimization Theory and Applications.

Schrage, L. (1997) Optimization Modeling with Lindo, Duxbury Press, Pacific Grove, $5^{\text {th }}$ Edition, 470 pages.

Skorin-Kapov, D. and Skorin-Kapov, J. (1994) "On Tabu Search for the Location of Interacting Hub Facilities", European Journal on Operational Research, 73, pp. 501-508.

Smith, K., Palaniswami, M. and Krishnamoorthy, M. (1996a) "A Hybrid Neural Approach to Combinatorial Optimisation", Computer and Operations Research, 23, pp. 597-610.

Smith, K., Palaniswami, M. and Krishnamoorthy, M. (1996b) "Traditional Heuristic Versus Hopfield Neural Network Approaches to a Car Sequencing Problem", European Journal of Operational Research, 93, pp. 300-316.

Smith, K., Palaniswami, M. and Krishnamoorthy, M. (1998) "Neural Techniques for Combinatorial Optimization with Applications", IEEE Transactions on Neural Networks, 9, pp. 1301-1319.

Sofianopoulou, S. (1992) "The Process Allocation Problem: a Survey of the Application of GraphTheoretic and Integer Programming Approaches", Journal of the Operational Research Society, 43, pp. 407-413.

Sosic, R. and Gu, J. (1991) "Fast Search Algorithms for the N-Queens Problem", IEEE Transactions on Systems, Man and Cybernetics, 21, pp. 1572-1576.

Sprent, P. (1993) Applied Nonparametric Statistical Methods, Chapman and Hall, London, $2^{\text {nd }}$ Edition, 342 pages.

Stuckey, P. and Tam, V. (1996) "Models for Using Stochastic Constraint Solvers in Constraint Logic Programming", International Symposium on Programming Languages, Implementation, Logics and Programs, pp. 423-437.

Taha, H. (1992) Operations Research: An introduction, Macmillan Publishing Company, New York, $5^{\text {th }}$ Edition, 822 pages.

Taillard, E. (1991) "Robust Taboo Search for the Quadratic Assignment Problem", Parallel Computing, 17, pp. 443-455.

Taillard, E. (1993) "Parallel Iterative Search Methods for Vehicle Routing Problems", Networks, 23, pp. 661-673.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F. and Potvin, J. (1997). "A Tabu Search for the Vehicle Routing Problem with Soft Time Windows", Transportation Science, 31, pp. 170-186.

Tank, D. and Hopfield, J. (1985) "'Neural' Computation of Decisions in Optimization Problems", Biological Cybernetics, 52, pp. 141-152.
van Laarhoven, P. and Aarts, E. (1987) Simulated Annealing: Theory and Applications, D Reidel Publishing Company, Dordecht, 186 pages.

Wilson, R. (1985) Introduction to Graph Theory, Longman Scientific and Technical, Hong Kong, $3^{\text {rd }}$ Edition, 166 pages.

Winston, P. (1992) Artificial Intelligence, Addison Wesley, Reading, $3^{\text {rd }}$ Edition, 737 pages.
Winston, W. (1991) Operations Research: Applications and Algorithms, Duxbury Press, California, $2^{\text {nd }}$ Edition, 1262 pages.

Woodruff, D. and Zemel, E. (1993) "Hashing Vectors for Tabu Search", Annals of Operations Research, 41, pp. 123-137.

## Glossary of Terms and Acronyms

## AI: Artificial Intelligence

ANN: Artificial Neural Network

ASA: Adaptive Simulated Annealing

ASC: Application Specific Computer

Backaus-Naur Form: See BNF.

BIN: Bin Packing Problem. A problem in which a set of weighted objects is to be partitioned between a number of bins with a certain capacity. The aim is to minimise the number of bins.

Bin Packing Problem: See BIN.

BNF: Backaus-Naur Form. A syntax for describing languages (in particular programming languages). It is often used for the production of compilers.

Car Sequencing Problem: See CSP.

Constraint: An inequality or equality relationship that limits the set of feasible solution states.

COP: Combinatorial Optimisation Problem

CP: Constraint programming.

CSP: Car Sequencing Problem. A problem in which a number of different car models are to be sequenced on an assembly line. The objective is to separate cars of the same model type as much as possible in order to evenly distribute the manufacturing workload.

CPU: Central Processor Unit

CPU time: The total time taken for the CPU to execute a program.

Field Programmable Gate Array. See FPGA.

FPGA: Field Programmable Gate Array. An FPGA is a silicon chip that allows a hardware design to be simulated. FPGAs are useful as they allow hardware designs to be reconfigured without any physical modification.

GRASP: Greedy Randomised Adaptive Search Procedures

GA: Genetic Algorithm

GAMS: General Algebraic Modelling System

GAP: Generalised Assignment Problem. In this problem, jobs are assigned to agents subject to capacity constraints. The aim is to minimise the overall cost of this assignment.

Generalised Assignment Problem: See GAP.

GPP: Graph Partitioning Problem. A problem in which the aim is to find two equal partitions of nodes such that the number of interconnections between the partitions (characterised by the edges) is minimised.

GPSIMAN: General Purpose SIMulated ANNealing. The solver was demonstrated in Connolly (1992).

GRAPH: Graph Colouring Problem. A problem in which a minimal number of colours are to be assigned to a set of vertices such that any pair of vertices connected by an edge has two different colours.

Graph Colouring Problem: See GRAPH.

Graph Partitioning Problem. See GPP.

GS: Greedy Search

Hamiltonian Cycle: A Hamiltonian cycle is a circuit in a graph in which each vertex appears exactly once.

## ILP: Integer Linear Program(ming)

IQR: Inter Quartile Range. The IQR quantifies the spread of the distribution for non-normally distributed data. It is defined as the difference between the first and third quartile of the distribution (Emory and Cooper 1991).

Knapsack Problem: The objective of this problem is to fit a maximum number of items in the knapsack such that the profit is maximised. The sum of the weights of the items must not exceed a preset bound.

LISP: LISt Processing. A computer language designed for manipulating lists of data.

## LP: Linear Program(ming)

Markov Chain Length: The number of SA iterations for which the temperature is held constant.

## Max: Maximum

Maximal Planar Layout Problem: A facilities layout problem in which a planar graph is sought that has the qualities: (a) no edges intersect; and (b) the sum of edge weights is maximised.

Maximum Clique Problem: See MCP.

Med: Median

MCP: Maximum Clique Problem. A problem in which a clique of maximum cardinality is sought. Each clique member must be connected by an edge to every other member of the clique.

MFLOP: Millions of FLoating Point Operations per second.

MILP: $\quad$ Mixed Integer Linear Program(ming). An MILP is an LP in which some variables are continuous while others are restricted to integer values.

## Min: Minimum

Minimal Spanning Tree Problem: This problem requires that all vertices in a graph are connected by edges such that: (a) the edges do not form a circuit (i.e. a Hamiltonian cycle); (b) the sum of the weight of the edges is minimised.

MIMD: $\quad$ Multiple Instructions Multiple Data.

MKP: Multiple Knapsack Problem. This problem is an extension of the knapsack problem as it has a number of capacity constraints instead of a single constraint.

Multiple Knapsack Problem. See MKP.

Neighbourhood: The neighbourhood of a solution is the set of those solutions that can be reached from the original solution by the application of a transition operator.

NP: Nondeterministically Polynomial. Refers to a class of problems that may not be solved to proven optimality in less than exponential computational time (in the worst case).

OR: Operations Research

OSL: Optimisation Subroutine Library. A commercial LP/ILP package produced by IBM.

PMX: Partially Matched Crossover

Processor Allocation Problem: A problem in which a number of processes is allocated to a number of processors such that the total communication flow between processes on different processors is minimised.

QAP: Quadratic Assignment Problem. A problem in which a set of facilities is assigned to unique locations in order to minimise the total intercommunication cost between the facilities.

Quadratic Assignment Problem: See QAP.

SA: Simulated Annealing

School Time Tabling Problem: See TTP.

SOFM: Self-Organising Feature Maps

Transition: A transition is a perturbation of the current solution state. There are a variety of different transition types. Some of which include exchanging the position of values within the solution, adding / dropping a value to / from the solution, or changing a value within the solution to another value. Different problems have their own set of suitable transition operators.

Traveling Salesman Problem: See TSP.

## TS: Tabu Search

TSP: Traveling Salesman Problem. The objective of the TSP is to find a minimum length Hamiltonian cycle through a set of cities (nodes).

TTP: School Time Tabling Problem. A problem in which a number of tuples (consisting of teacher, room and class) are scheduled in a fixed number of time slots (periods). The aim is to minimise the number of clashes between tuples.

UMApHMP: Uncapacitated Multiple Allocation p-Hub Median Problem. This is similar to USApHMP except that each node may be connected to more than one hub.

Uncapacitated Multiple Allocation p-Hub Median Problem. See UMApHMP.

Uncapacitated Single Allocation p-Hub Median Problem. See USApHMP.

USApHMP: Uncapacitated Single Allocation p-Hub Median Problem. A problem in which a set of nodes is assigned to a set of hubs (i.e. each node is connected to a single hub) such that the total cost of the flows between every pair of nodes is minimised. The problem also involves deciding which nodes are to be the hub nodes.

Vehicle Routing Problem: See VRP.

VRP: Vehicle Routing Problem. This is a generalisation of the TSP in which a number of tours are used to visit the set of nodes. The objective is to either minimise the total length of the tour or to minimise the number of tours, subject to time and capacity constraints.

Wall clock time: The elapsed (real) time it takes to execute a program. As such, this time includes any system overhead and processor waiting.

## Appendix A: Standard Results for the SA, TS and GS engines

Table 45, Table 46 and Table 47 display the best results obtained for every problem instance with the meta-heuristic engines. The summary tables are divided into two main sections, Cost and Runtime. For each section, the minimum (denoted Min), median (denoted Med), maximum (denoted Max) and inter-quartile range (denoted $\mathbf{I Q R}$ ) are given (as each problem instance is run with 10 random seeds). The Runtime section records the amount of CPU time required to reach the best solution cost for that particular run. Note: The desired objective function cost for the BIN, GRAPH and MCP problems is 0 as a result of the reasons outlined in Section 5.1.

| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| CSP | n20t1 | 58 | 1 | 1000 | 58 | 58 | 58 | 0 | 0 | 7.5 | 27 | 6.25 |
|  |  |  | 1 | 5000 | 58 | 58 | 59 | 0 | 0 | 224 | 1617 | 95 |
|  |  |  | 1 | 10000 | 58 | 58 | 59 | 1 | 0 | 359.5 | 1339 | 932 |
|  | n20t2 | 40 | 1 | 1000 | 40 | 40 | 40 | 0 | 0 | 0.5 | 2 | 1.5 |
|  |  |  | 1 | 5000 | 40 | 40 | 40 | 0 | 0 | 3 | 8 | 7 |
|  |  |  | 1 | 10000 | 40 | 40 | 40 | 0 | 0 | 6.5 | 15 | 13.75 |
|  | n20t3 | 29 | 1 | 1000 | 29 | 29 | 29 | 0 | 0 | 0 | 4 | 0 |
|  |  |  | 1 | 5000 | 29 | 29 | 29 | 0 | 0 | 0 | 9 | 0 |
|  |  |  | 1 | 10000 | 29 | 29 | 29 | 0 | 0 | 0 | 20 | 0 |
|  | n20t4 | 10 | 1 | 1000 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 5000 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 10000 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |
|  | n20t5 | 150 | 1 | 1000 | 150 | 150 | 150 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 5000 | 150 | 150 | 150 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 10000 | 150 | 150 | 150 | 0 | 0 | 0 | 0 | 0 |
|  | n40t1 | 146 | 1 | 1000 | 142 | 144 | 145 | 0.75 | 2 | 701 | 1116 | 515 |
|  |  |  | 1 | 5000 | 143 | 144 | 145 | 1.75 | 2 | 458 | 1634 | 1330 |
|  |  |  | 1 | 10000 | 142 | 144 | 145 | 1 | 2 | 450.5 | 1277 | 517 |
|  | n40t2 | 94 | 1 | 1000 | 92 | 92 | 92 | 0 | 1 | 419.5 | 1304 | 325.75 |
|  |  |  | 1 | 5000 | 92 | 92 | 92 | 0 | 1 | 433 | 940 | 457 |
|  |  |  | 1 | 10000 | 92 | 92 | 92 | 0 | 189 | 250.5 | 917 | 222.75 |
|  | n40t3 | 66 | 1 | 1000 | 65 | 66 | 67 | 1 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 1 | 5000 | 65 | 66 | 67 | 1 | 0 | 0.5 | 5 | 0.5 |
|  |  |  | 1 | 10000 | 65 | 66 | 66 | 1 | 0 | 0.5 | 5 | 0.5 |
|  | n40t4 | 33 | 1 | 1000 | 32 | 32 | 32 | 0 | 0 | 3 | 339 | 80 |
|  |  |  | 1 | 5000 | 32 | 32 | 32 | 0 | 0 | 3.5 | 102 | 4 |
|  |  |  | 1 | 10000 | 32 | 32 | 32 | 0 | 1 | 2.5 | 166 | 41.5 |
|  | n40t5 | 352 | 1 | 1000 | 354 | 354 | 354 | 0 | 0 | 0.5 | 13 | 6 |
|  |  |  | 1 | 5000 | 354 | 354 | 354 | 0 | 0 | 0.5 | 28 | 18.25 |
|  |  |  | 1 | 10000 | 354 | 354 | 354 | 0 | 0 | 0.5 | 53 | 33.75 |
|  | n60tl | 238 | 1 | 1000 | 228 | 231 | 233 | 2 | 0 | 1 | 1265 | 11.5 |
|  |  |  | 1 | 5000 | 228 | 23.1 | 232 | 1 | 1 | 2 | 1232 | 702.25 |
|  |  |  | 1 | 10000 | 228 | 232 | 234 | 1.25 | 0 | 1 | 219 | 137.25 |
|  | n60t2 | 152 | 1 | 1000 | 144 | 146.5 | 150 | 1 | 4 | 164.5 | 1528 | 1063 |
|  |  |  | 1 | 5000 | 144 | 146 | 146 | 1 | 4 | 307.5 | 1029 | 287.5 |
|  |  |  | 1 | 10000 | 144 | 145 | . 146 | 0 | 4 | 270 | 1419 | 371.5 |


| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| $\square$ | n60t3 | 105 | 1 | 1000 | 101 | 102.5 | 103 | 1 | 2 | 3.5 | 9 | 3 |
|  |  |  | 1 | 5000 | 101 | 101 | 103 | 1 | 2 | 7 | 38 | 27.75 |
|  |  |  | 1 | 10000 | 101 | 101 | 102 | 0.75 | 3 | 15 | 35 | 26 |
|  | n60t4 | 58 | 1 | 1000 | 54 | 55 | 55 | 0.75 | 0 | 1.5 | 8 | 3 |
|  |  |  | , | 5000 | 54 | 54 | 55 | 0 | 1 | 17 | 35 | 16.5 |
|  |  |  | 1 | 10000 | 54 | 54 | 54 | 0 | 1 | 17.5 | 69 | 21.25 |
|  | n60t5 | 562 | 1 | 1000 | 558 | 558 | 558 | 0 | 1 | 9.5 | 173 | 144.5 |
|  |  |  | 1 | 5000 | 558 | 558 | 558 | 0 | 1 | 9.5 | 329 | 87.5 |
|  |  |  | 1 | 10000 | 558 | 558 | 558 | 0 | 1 | 9.5 | 390 | 231.5 |
|  | n80t1 | 330 | 1 | 1000 | 312 | 316 | 324 | 4.25 | 3 | 6.5 | 1462 |  |
|  |  |  | 1 | 5000 | 312 | 316 | 321 | 2.75 | 3 | 6.5 | 1573 | 4.75 |
|  |  |  | 1 | 10000 | 312 | 316 | 321 | 3.5 | 3 | 6.5 | 447 | 4.75 |
|  | n80t2 | 215 | 1 | 1000 | 199 | 201 | 202 | 1.5 | 18 | 924 | 1587 | 991.75 |
|  |  |  | 1 | 5000 | 196 | 200.5 | 202 | 2.75 | 25 | 249 | 918 | 392.5 |
|  |  |  | 1 | 10000 | 196 | 200 | 202 | 1.75 | 25 | 312.5 | 1468 | 683.75 |
|  | n80t3 | 146 | , | 1000 | 138 | 139 | 140 | 1 | 7 | 12 | 16 | 1.25 |
|  |  |  | 1 | 5000 | 137 | 138 | 139 | 1 | 11 | 40.5 | 67 | 41 |
|  |  |  | 1 | 10000 | 137 | 138 | 139 | 0.75 | 11 | 40.5 | 109 | 41.75 |
|  | n80t4 | 82 | 1 | 1000 | 76 | 77 | 77 | 0.75 |  | 10 | 19 | 8.5 |
|  |  |  | 1 | 5000 | 76 | 76 | 77 | 0 | 2 | 17.5 | 91 | 21 |
|  |  |  | 1 | 10000 | 76 | 76 | 76 | 0 | 6 | 26.5 | 114 | 65.5 |
|  | n80t5 | 772 | 1 | 1000 | 762 | 766 | 770 | 8 | 1 | 24 | 1320 | 69.5 |
|  |  |  | 1 | 5000 | 762 | 766 | 770 | 8 | 1 | 207 | 1216 | 1042 |
|  |  |  | 1 | 10000 | 762 | 770 | 770 | 4.5 | 1 | 10.5 | 1165 | 288.5 |
| BIN | binlal | 0 | 1 | 1000 | 42 | 53 | 63 | 12.25 | 206 | 448 | 1577 | 605.25 |
|  |  |  | 1 | 5000 | 34 | 51.5 | 53 | 3.75 | 265 | 974.5 | 1426 | 736.25 |
|  |  |  | 1 | 10000 | 41 | 52.5 | 65 | 11 | 449 | 823 | 1350 | 665.75 |
|  |  |  | 2 | 1000 | 0 | 0 | 0 | 0 | I | 1 | 2 | 0.5 |
|  |  |  | 2 | 5000 | 0 | 0 | 0 | 0 |  | 1 | 2 | 0.5 |
|  |  |  | 2 | 10000 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0.5 |
|  |  |  | 3 | 1000 | 0 | 0 | 0 | 0 | 4 | 5.5 | 9 | 2.75 |
|  |  |  | 3 | 5000 | 0 | 0 | 0 | 0 | 4 | 6 | 9 | 2.5 |
|  |  |  | 3 | 10000 | 0 | 0 | 0 | 0 | 4 | 6 | 9 | 2.5 |
|  |  |  | 4 | 1000 | 0 | 0 | 0 | 0 | 1 | 2 | 14 | 1.5 |
|  |  |  | 4 | 5000 | 0 | 0 | 0 | 0 | 1 | 2 | 14 | 1.5 |
|  |  |  | 4 | 10000 | 0 | 0 | 0 | 0 | 1 | 2 | 15 | 1.5 |
|  |  |  | 5 | 1000 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 1.25 |
|  |  |  | 5 | 5000 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 1.25 |
|  |  |  | 5 | 10000 | 0 | 0 | . | 0 | 1 | 2 | 3 | 1.25 |


| Problem | Instance | Optimal/ <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | $\begin{aligned} & \text { Annealing } \\ & \text { Length } \end{aligned}$ | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | binla 2 | 0 | 1 | 1000 | 26 | 43.5 | 45 | 8.75 | 270 | 777.5 | 1510 | 777.25 |
|  |  |  | 1 | 5000 | 26 | 40 | 50 | 7 | 265 | 470 | 1533 | 850 |
|  |  |  | 1 | 10000 | 23 | 37 | 44 | 8.25 | 423 | 862.5 | 1477 | 574.25 |
|  |  |  | 2 | 1000 | 0 | 0 | 0 | 0 | 0 | 0.5 | 1 |  |
|  |  |  | 2 | 5000 | 0 | 0 | 0 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  |  | 2 | 10000 | 0 | 0 | 0 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  |  | 3 | 1000 | 0 | 0 | 0 | 0 | 2 | 3.5 | 7 | 2.25 |
|  |  |  | 3 | 5000 | 0 | 0 | 0 | 0 | 2 | 3.5 | 7 | 2.25 |
|  |  |  | 3 | 10000 | 0 | 0 | 0 | 0 | 2 | 3.5 | 7 | 2.25 |
|  |  |  | 4 | 1000 | 0 | 0 | 0 | 0 | , | 1 | 2 | 0.25 |
|  |  |  | 4 | 5000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.25 |
|  |  |  | 4 | 10000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.25 |
|  |  |  | 5 | 1000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.25 |
|  |  |  | 5 | 5000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.25 |
|  |  |  | 5 | 10000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.25 |
|  | binla 4 | 0 | 1 | 1000 | 58 | 74.5 | 83 | 8 | 446 | 1050 | 1575 | 424.5 |
|  |  |  | 1 | 5000 | 61 | 74.5 | 88 | 2.5 | 290 | 1243.5 | 1552 | 414 |
|  |  |  | 1 | 10000 | 58 | 67.5 | 89 | 17 | 414 | 876 | 1586 | 638 |
|  |  |  | 2 | 1000 | 0 | 0 | 0 | 0 | 1 | 2.5 | 6 | 1.75 |
|  |  |  | 2 | 5000 | 0 | 0 | 0 | 0 | 1 | 2.5 | 6 | 1.75 |
|  |  |  | 2 | 10000 | 0 | 0 | 0 | 0 | 1 | 2.5 | 6 | 1.75 |
|  |  |  | 3 | 1000 | 0 | 0 | 1 | 0 | 8 | 12.5 | 24 | 6.5 |
|  |  |  | 3 | 5000 | 0 | 0 | 0 | 0 | 11 | 16.5 | 42 | 10 |
|  |  |  | 3 | 10000 | 0 | 0 | 0 | 0 | 11 | 16.5 | 42 | 10.5 |
|  |  |  | 4 | 1000 | 0 | 0 | 0 | 0 | 3 | 4 | 6 | 0.5 |
|  |  |  | 4 | 5000 |  |  |  |  |  |  |  |  |
|  |  |  | 4 | 10000 | 0 | 0 | 0 | 0 | 3 | 4 | 6 | 0.5 |
|  |  |  | 5 | 1000 | 0 | 0 | 0 | 0 | 2 | 4 | 7 | 0.75 |
|  |  |  | 5 | 5000 | 0 | 0 | 0 | 0 | 2 | 4 | 6 | 0.75 |
|  |  |  | 5 | 10000 | 0 | 0 | 0 | 0 | 2 | 4 | 6 | 0.75 |
|  | binla6 | 0 | 1 | 1000 | 55 | 64.5 | 72 | 9.5 | 444 | 847 | 1590 | 679.5 |
|  |  |  | 1 | 5000 | 52 | 64.5 | 73 | 12.75 | 237 | 1081 | 1588 | 566.5 |
|  |  |  | 1 | 10000 | 49 | 63.5 | 72 | 7.5 | 579 | 843.5 | 1121 | 245 |
|  |  |  | 2 | 1000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.5 |
|  |  |  | 2 | 5000 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0.5 |
|  |  |  | 2 | 10000 | 0 | 0 | 0 | 0 | 1 |  | 2 | 0.5 |
|  |  |  | 3 | 1000 | 0 | 0 | 2 | 0 | 6 | 12.5 | 16 | 6 |
|  |  |  | 3 3 | 5000 10000 | 0 0 | 0 0 | 0 | 0 0 | 6 | 13 13 | 30 30 | 7 7 |



\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{SA Standard Results} \& \& \& \& \multicolumn{4}{|l|}{} \& \multicolumn{4}{|l|}{} \\
\hline \multirow[t]{2}{*}{Problem} \& \multirow[t]{2}{*}{Instance} \& \multirow[t]{2}{*}{\begin{tabular}{l} 
Optimal/ \\
Best \\
Known \\
cost \\
\hline
\end{tabular}} \& \multirow[t]{2}{*}{\begin{tabular}{l}
Transition Set \\
Reference \\
Number
\end{tabular}} \& \multirow[t]{2}{*}{Annealing Length} \& \multicolumn{4}{|l|}{Cost} \& \multicolumn{4}{|l|}{Runtime (seconds)} \\
\hline \& \& \& \& \& Min \& Med \& Max \& IQR \& Min \& Med \& Max \& IQR \\
\hline \multicolumn{2}{|r|}{\multirow{39}{*}{\(\operatorname{bin} 2 \mathrm{a} 4\)

bin3al}} \& \multirow{27}{*}{0} \& 2 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 5 \& 10 \& 18 \& 4.25 <br>
\hline \& \& \& 2 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 5 \& 10 \& 18 \& 4.25 <br>
\hline \& \& \& 2 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 5 \& 10 \& 18 \& 4 <br>
\hline \& \& \& 3 \& 1000 \& 0 \& 0 \& 1 \& 0.75 \& 46 \& 64.5 \& 90 \& 18.5 <br>
\hline \& \& \& 3 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 46 \& 75.5 \& 105 \& 32.5 <br>
\hline \& \& \& 3 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 46 \& 75.5 \& 105 \& 33.25 <br>
\hline \& \& \& 4 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 19 \& 23.5 \& 31 \& 3.75 <br>
\hline \& \& \& 4 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 19 \& 23.5 \& 31 \& 3.75 <br>
\hline \& \& \& 4 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 19 \& 23.5 \& 31 \& 3.75 <br>
\hline \& \& \& 5 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 18 \& 19.5 \& 31 \& 3.25 <br>
\hline \& \& \& 5 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 18 \& 19.5 \& 31 \& 3.25 <br>
\hline \& \& \& 5 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 18 \& 20 \& 31 \& 3 <br>
\hline \& \& \& 1 \& 1000 \& 176 \& 202 \& 221 \& 13.5 \& 843 \& 1341 \& 1587 \& 362.5 <br>
\hline \& \& \& 1 \& 5000 \& 171 \& 197 \& 246 \& 20.5 \& 894 \& 1084 \& 1513 \& 467.75 <br>
\hline \& \& \& 1 \& 10000 \& 161 \& 328.5 \& 435 \& 35.75 \& 15 \& 900.5 \& 1104 \& <br>
\hline \& \& \& 2 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 5 \& 8 \& 13 \& 1.25 <br>
\hline \& \& \& 2 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 5 \& 8 \& 13 \& <br>
\hline \& \& \& 2 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 5 \& \& 13 \& 1.25 <br>
\hline \& \& \& 3 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 35 \& 49.5 \& 80 \& 15 <br>
\hline \& \& \& 3 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 35 \& 49.5 \& 80 \& 15 <br>
\hline \& \& \& 3 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 35 \& 49.5 \& 80 \& 15 <br>
\hline \& \& \& 4 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 11 \& 16 \& 25 \& 3 <br>
\hline \& \& \& 4 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 11 \& 16 \& 25 \& 3 <br>
\hline \& \& \& 4 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 11 \& 16 \& 25 \& <br>
\hline \& \& \& 5 \& 1000 \& 0 \& 0 \& 0 \& 0 \& 13 \& 17.5 \& 21 \& 3.5 <br>
\hline \& \& \& 5 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 13 \& 17.5 \& 26 \& 3.5 <br>
\hline \& \& \& 5 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 13 \& 17.5 \& 21 \& 3.5 <br>
\hline \& \& \multirow[t]{12}{*}{0} \& 1 \& 1000 \& 517 \& 581 \& 891 \& 80.5 \& 778 \& 1156 \& 1287 \& 110 <br>
\hline \& \& \& 1 \& 5000 \& 412 \& 544.5 \& 916 \& 162.25 \& 179 \& 1086.5 \& 1582 \& 600.75 <br>
\hline \& \& \& 1 \& 10000 \& 446 \& 711.5 \& 916 \& 265.5 \& 488 \& 958 \& 1426 \& 307.75 <br>
\hline \& \& \& 2 \& 1000 \& 0 \& 0 \& 3 \& 1 \& 62 \& 146 \& 186 \& 48.25 <br>
\hline \& \& \& 2 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 63 \& 171.5 \& 414 \& 87.5 <br>
\hline \& \& \& 2 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 62 \& 170 \& 413 \& <br>
\hline \& \& \& 3 \& 1000 \& 3 \& 7 \& 13 \& 2.25 \& 322 \& 360 \& 373 \& <br>
\hline \& \& \& 3 \& 5000 \& 0 \& 0 \& 2 \& 1 \& 217 \& 377.5 \& 844 \& 391.5 <br>
\hline \& \& \& 3 \& 10000 \& 0 \& 0 \& 28 \& 0.75 \& 377 \& 1036.5 \& 1557 \& 341.5 <br>
\hline \& \& \& 4 \& 1000 \& 0 \& 1 \& 3 \& 2 \& 188 \& 246 \& 276 \& 54 <br>
\hline \& \& \& 4 \& 5000 \& 0 \& 0 \& 0 \& 0 \& 737 \& 938 \& 1575 \& 375 <br>
\hline \& \& \& 4 \& 10000 \& 0 \& 0 \& 0 \& 0 \& 216 \& 528 \& 842 \& 387.75 <br>
\hline
\end{tabular}



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 3 | 1000 | 12 | 19.5 | 27 | 4 | 332 | 355 | 380 | 16 |
|  |  |  | 3 | 5000 | 1 | 3 | 7 | 2 | 785 | 1368.5 | 1532 | 190.5 |
|  |  |  | 3 | 10000 | 0 | 1 | 4 | 1.75 | 783 | 1112 | 1546 | 470 |
|  |  |  | 4 | 1000 | 3 | 6.5 | 11 | 2.5 | 221 | 265 | 287 | 31.5 |
|  |  |  | 4 | 5000 | 0 | 0 | I | 0.75 | 576 | 1296.5 | 1429 | 650 |
|  |  |  | 4 | 10000 | 0 | 0 | 1 | 0.75 | 418 | 876 | 1389 | 601.25 |
|  |  |  | 5 | 1000 | 3 | 4.5 | 7 | 1 | 205 | 250.5 | 319 | 25.75 |
|  |  |  | 5 | 5000 | 0 | 0 | 1 | 0 | 412 | 792.5 | 1215 | 419.5 |
|  |  |  | 5 | 10000 | 0 | 0 | 1 | 0 | 412 | 793 | 1215 | 420 |
| GRAPH | C125.1 | 0 | 1 | 1000 | 2 | 3 | 6 | 1.75 | 3 | 6.5 | 41 | 6.75 |
|  |  |  | 1 | 5000 | 1 | 3 | 6 | 1.75 | 3 | 7 | 90 | 6.75 |
|  |  |  | 1 | 10000 | 1 | 3 | 5 | 1.75 | 3 | 7 | 268 | 6.75 |
|  |  |  | 2 | 1000 | 1 | 3 | 5 | 2.75 | 5 | 17.5 | 36 | 15.5 |
|  |  |  | 2 | 5000 | 1 | 2 | 4 | 0.75 | 14 | 57.5 | 197 | 87 |
|  |  |  | 2 | 10000 | 0 | 2 | 2 | 0.75 | 14 | 100 | 409 | 176.25 |
|  |  |  | 3 | 1000 | 2 | 3 | 5 | 1.5 | 6 | 15.5 | 32 | 13.25 |
|  |  |  | 3 | 5000 | 1 | 2 | 5 | 1.5 | 11 | 66.5 | 198 | 68 |
|  |  |  | 3 | 10000 | 1 | 2 | 5 | 1.5 | 11 | 69.5 | 198 | 68 |
|  |  |  | 4 | 1000 | 2 | 3 | 5 | 1 | 5 | 15 | 38 | 10.75 |
|  |  |  | 4 | 5000 | 2 | 2.5 | 4 | 1.75 | 5 | 30 | 153 | 45.75 |
|  |  |  | 4 | 10000 | 0 | 2 | 4 | 1.75 | 5 | 48 | 406 | 121.75 |
|  |  |  | 5 | 1000 | 2 | 3 | 5 | 0 | 9 | 26 | 43 | $13$ |
|  |  |  | 5 | 5000 | 0 | 2 | 5 | 0.75 | 15 | 77.5 | 221 | 74.5 |
|  |  |  | 5 | 10000 | 0 | 2 | 5 | 0.75 | 15 | 77.5 | 221 | 74.75 |
|  | C125.5 | 0 | 1 | 1000 | 6 | 8 | 10 | 1 | 10 | 18.5 | 23 | 6.5 |
|  |  |  | 1 | 5000 | 3 | 6 | 8 | 2 | 23 | 44 | 113 | 68.25 |
|  |  |  | 1 | 10000 | 1 | 5.5 | 9 | 3.75 | 130 | 407 | 1481 | 295.75 |
|  |  |  | 2 | 1000 | 8 | 10 | 13 | 1.75 | 9 | 15.5 | 18 | 1.5 |
|  |  |  | 2 | 5000 | 5 | 6.5 | 9 | 1.75 | 9 | 61 | 77 |  |
|  |  |  | 2 | 10000 | 4 | 6 | 8 | 1.75 | 34 | 95 | 149 | 66 |
|  |  |  | 3 | 1000 | 8 | 9.5 | 10 | 1 | 8 | 17 | 24 | 9 |
|  |  |  | 3 | 5000 | 5 | 6.5 | 10 | 2.5 | 24 | 65.5 | 110 | 61.25 |
|  |  |  | 3 | 10000 | 3 | 6 | 10 | 1.75 | 24 | 81 | 232 | 105 |
|  |  |  | 4 | 1000 | 6 | 9 | 12 | 2.75 | 52 | 66 | 80 | 16 |
|  |  |  | 4 | 5000 | 5 | 6 | 10 | 2.25 | 14 | 40 | 101 | 23.25 |
|  |  |  | 4 | 10000 | 4 | 5 | 10 | 1.75 | 14 | 67 | 217 | 120 |
|  |  |  | 5 | 1000 | 8 | 9 | 12 | 1 | 11 | 20.5 | 23 | 4.25 |
|  |  |  | 5 | 5000 | 5 | 6 | 8 | 1.75 | 24 |  | 112 | 37.5 |
|  |  |  | 5 | 10000 | 5 | 6 | . 7 | 0.75 | 51 | 100.5 | 204 | 56.75 |



| Problem | Instance | Optimal <br> Best <br> Known <br> cost | Transition <br> Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | C250.9 | 0 | $\left\lvert\, \begin{aligned} & 4 \\ & 4 \\ & 4 \\ & 5 \\ & 5 \\ & 5 \\ & 5 \\ & 1 \\ & 1 \\ & 1 \\ & 2 \\ & 2 \\ & 2 \\ & 3 \\ & 3 \\ & 3 \\ & 4 \\ & 4 \\ & 4 \\ & 5 \\ & 5 \\ & 5 \end{aligned}\right.$ | 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 |  <br> 19 <br> 8 <br> 8 <br> 8 <br> 19 <br> 4 <br> 6 <br> 19 <br> 10 <br> 8 <br> 24 <br> 13 <br> 10 <br> 17 <br> 11 <br> 7 <br> 24 <br> 11 <br> 9 <br> 22 <br> 10 <br> 8 | $\begin{aligned} & 23 \\ & 11.5 \\ & 10 \\ & 20.5 \\ & 5.5 \\ & 11.5 \\ & 23.5 \\ & 12.5 \\ & 10 \\ & 30.5 \\ & 16.5 \\ & 12.5 \\ & 24 \\ & 120 \\ & 10 \\ & 20 \\ & 12.5 \\ & 9 \end{aligned}$ |  | 5.5 4.25 3.5 3 2.75 2.75 1.75 1.75 1.75 2. 1.75 2.5 2.75 2.5 1.5 0.75 2. 2.5 3.75 1.75 2.75 | 52 141 258 57 175 208 76 768 592 51 208 420 76 197 472 45 230 333 627 276 377 | 66.5 26.5 486 64.5 266.5 426.5 88.5 379.5 835.5 56 244 530 81.5 364.5 728 66.5 337.5 541.5 68 336.5 545 |  <br> 69 <br> 344 <br> 693 <br> 70 <br> 760 <br> 527 <br> 95 <br> 495 <br> 983 <br> 57 <br> 284 <br> 569 <br> 88 <br> 442 <br> 883 <br> 72 <br> 363 <br> 726 <br> 74 <br> 361 <br> 752 | 5.5 68 <br> 293.25 <br> 6.25 <br> 118.75 190 <br> 9 <br> 175 227.75 <br> 1.75 <br> 57 106 <br> 4.25 <br> 119.5 <br> 6.5 <br> 54.25 <br> 78.5 7 <br> 57 <br> 236.5 |
| USAPHMP | ap20a2 | 172816.7 | $\begin{aligned} & 1 \\ & 1 \\ & 1 \\ & 1 \\ & 2 \\ & 2 \\ & 2 \\ & 2 \\ & 3 \\ & 3 \\ & 3 \\ & 4 \\ & 4 \\ & 4 \\ & 5 \\ & 5 \\ & 5 \\ & 6 \\ & 6 \\ & 6 \end{aligned}$ | 1000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 | 172816.7 17286.7 178216.7 172816.7 172816.7 172816.7 172816.7 172816.7 172816.7 172816.7 172816.7 1721616.7 172816.7 1721616.7 172816.7 172816.7 1721686.7 172816.7 |  |  | $\begin{aligned} & \hline 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \\ & 0 \end{aligned}$ | 36 176 354 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 | 52.5 184.5 376.5 11 1 1 0.5 0.5 0.5 1.5 1.5 1.5 0.5 0.5 0.5 0.5 0.5 0.5 | 95 922 261 561 175 280 559 62 1210 580 59 3 542 66 295 698 29 2 2 | 26.25 21.75 64 44.25 133 8.25 0.25 0.25 0.25 1 0.75 1 44 1.5 262 0 0 0 |


| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| ap20a3 |  | 151533.1 | 1 | 1000 | 151533.1 | 151533.1 | 151533.1 | 0 | 39 | 61 | 121 | 31 |
|  |  | 1 | 5000 | 151533.1 | 151533.1 | 151533.1 | 0 | 166 | 179 | 413 | 66.25 |
|  |  | 1 | 10000 | 151533.1 | 151533.1 | 151533.1 | 0 | 321 | 339.5 | 483 | 21.75 |
|  |  | 2 | 1000 | 151533.1 | 153240.2 | 161745.95 | 5183.17 | I | 3.5 | 703 | 62 |
|  |  | 2 | 5000 | 151533.1 | 152651.78 | 161745.95 | 2015.81 | 1 | 84 | 1256 | 247.25 |
|  |  | 2 | 10000 | 151533.1 | 152093.89 | 160250.09 | 2295.48 | 1 | 488 | 1390 | 178 |
|  |  | 3 | 1000 | 151533.1 | 152651.77 | 160250.14 | 2015.85 | 0 | 2 | 218 | 45.5 |
|  |  | 3 | 5000 | 151533.1 | 151533.1 | 159170.27 | 838.93 | 0 | 208.5 | 351 | 229.5 |
|  |  | 3 | 10000 | 151533.1 | 152092.46 | 160250.14 | 2001.3 | 0 | 2 | 911 | 452.75 |
|  |  | 4 | 1000 | 151533.1 | 156439.99 | 161349.92 | 8447.17 | 1 | 5.5 | 237 | 64.25 |
|  |  | 4 | 5000 | 151533.1 | 153828.62 | 163209.45 | 8447.05 | 1 | 6 | 1438 | 183.25 |
|  |  | 4 | 10000 | 151533.1 | 151533.1 | 160250.11 | 839.01 | 1 | 253.5 | 1480 | 515.5 |
|  |  | 5 | 1000 | 151533.1 | 151533.1 | 160250.08 | 839.03 | 0 | 1 | 63 | 1.5 |
|  |  | 5 | 5000 | 151533.1 | 151533.1 | 160250.14 | 839.01 | 0 | 1 | 457 | 194.5 |
|  |  | 5 | 10000 | 151533.1 | 151533.1 | 160250.06 | 4345.4 | 0 | 1 | 503 | 0.75 |
|  |  | 6 | 1000 | 151533.1 | 152651.81 | 160250.11 | 641.99 | 1 | 1.5 | 553 | 3.75 |
|  |  | 6 | 5000 | 151533.1 | 152092.46 | 153293.8 | 1118.71 | 1 | 139.5 | 233 | 217 |
|  |  | 6 | 10000 | 151533.1 | 152092.46 | 160250.11 | 1118.71 | 1 | 315 | 461 | 435.5 |
| ap20a4 |  |  | 135624.9 | 1 | 1000 | 135624.9 | 135624.9 | 135624.9 | 0 | 32 | 97.5 |  | 123 |
|  |  | 1 |  | 5000 | 135624.9 | 135624.9 | 135624.9 | 0 | 154 | 254 | 543 | 139.5 |
|  |  | 1 |  | 10000 | 135624.9 | 135624.9 | 135624.9 | 0 | 310 | 366.5 | 621 | 181.25 |
|  |  | 2 |  | 1000 | 135624.9 | 139560.21 | 140863.03 | 2629.76 | 1 | 21 | 87 |  |
|  |  | 2 |  | 5000 | 135624.9 | 135624.9 | 141121.83 | 0 | 1 | 230 | 977 | 72.75 |
|  |  | 2 |  | 10000 | 135624.9 | 135624.9 | 141121.83 | 3614.98 | 1 | 260 | 1168 | 518.5 |
|  |  | 3 |  | 1000 | 135624.9 | 139762.7 | 144485.05 | 1692.91 | 1 | 2.5 | 195 | 44.5 |
|  |  | 3 |  | 5000 | 135624.9 | 135624.9 | 140863.06 | 2601.51 | 1 | 250 | 969 | 140.75 |
|  |  | 3 |  | 10000 | 135624.9 | 139022.43 | 140863.03 | 4852.22 | 1 | 389 | 1175 | 219.5 |
|  |  | 4 |  | 1000 | 135624.9 | 139093.58 | 144032.28 | 2061.67 | 1 | 11 | 65 |  |
|  |  | 4 |  | 5000 | 135624.9 | 135624.9 | 139093.67 | 0 | 8 | 231 | 1191 |  |
|  |  | 4 |  | 10000 | 135624.9 | 139762.27 | 142732.05 | 3400.35 | 2 | 16 | 1497 | 423.5 |
|  |  | 5 |  | 1000 | 135624.9 | 139560.25 | 143231.6 | 3056.25 | 1 | 20.5 | 76 | 62 |
|  |  | 5 |  | 5000 | 135624.9 | 137986.37 | 143599.91 | 4743.23 | 1 | 196.5 | 1181 | 246.5 |
|  |  | 5 |  | 10000 | 135624.9 | 139093.58 | 143023.16 | 3838.95 | 1 | 642.5 | 1490 | 825 |
|  |  | 6 |  | 1000 | 135624.9 | 138373.37 | 149138.55 | 2910.86 | 0 | 7.5 | 179 | 51.25 |
|  |  | 6 |  | 5000 | 135624.9 | 135624.9 | 144781.47 | 1254.13 | 1 | 200 | 664 | 155.25 |
|  |  | 6 |  | 10000 | 135624.9 | 136513.19 | 141121.83 | 2695.95 | 0 | 208 | 714 | 415.25 |
| ap20a5 |  |  | 123130.1 | 1 | 1000 | 123130.1 | 123130.1 | 123202.78 | 0 | 78 | 201 | 531 |  |
|  |  | 1 |  | 5000 | 123130.1 | 123130.1 | 123347.63 | 0 | 180 | 769 | 1080 | 592.75 |
|  |  | 1 |  | 10000 | 123130.1 | 123130.1 | . 123202.82 | 0 | 346 | 607 | 1175 | 265.75 |



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 3 | 1000 | 155256.3 | 160078.63 | 165993.61 | 3521.2 | 1 | 13 | 425 | 35.25 |
|  |  |  | 3 | 5000 | 155256.3 | 156256.81 | 163228.84 | 4889.83 | 1 | 219.5 | 703 | 551.25 |
|  |  |  | 3 | 10000 | 155256.3 | 157195.11 | 165146.83 | 3953.04 | 1 | 598.5 | 1328 | 1240 |
|  |  |  | 4 | 1000 | 155256.3 | 157398.96 | 160213.63 | 4598.71 | 3 | 66.5 | 451 | 180 |
|  |  |  | 4 | 5000 | 155256.3 | 156374.84 | 160213.63 | 3317.74 | 2 | 70.5 | 711 | 85.5 |
|  |  |  | 4 | 10000 | 155256.3 | 155315.53 | 160213.58 | 3495.23 | 3 | 70.5 | 1343 | 983.75 |
|  |  |  | 5 | 1000 | 155256.3 | 160078.64 | 161743.8 | 6016.33 | 1 | 6 | 418 | 83.75 |
|  |  |  | 5 | 5000 | 155256.3 | 155256.3 | 161743.81 | 0 | 1 | 566 | 646 | 464.25 |
|  |  |  | 5 | 10000 | 155256.3 | 156929.7 | 164802.13 | 5588.05 | 1 | 437 | 1319 | 1128 |
|  |  |  | 6 | 1000 | 155256.3 | 158541.25 | 161743.8 | 4687.34 | 1 | 3.5 | 116 | 3.25 |
|  |  |  | 6 | 5000 | 155256.3 | 156197.58 | 160213.63 | 3986.15 | 1 | 109.5 | 481 | 399.5 |
|  |  |  | 6 | 10000 | 155256.3 | 155433.56 | 160213.63 | 4687.34 | 1 | 109.5 | 1302 | 832.5 |
|  | ap25a4 | 139197.2 | 1 | 1000 | 139197.25 | 139230.58 | 145696.33 | 66.66 | 122 | 248 | 735 | 288.75 |
|  |  |  | 1 | 5000 | 139197.25 | 139263.91 | 139868.34 | 66.66 | 315 | 812 | 1494 | 366 |
|  |  |  | 1 | 10000 | 139197.25 | 139263.91 | 145727.72 | 105.45 | 638 | 1065.5 | 1260 | 160.25 |
|  |  |  | 2 | 1000 | 139263.91 | 140917.82 | 150356.19 | 6473.99 | 3 | 9 | 198 | 16.25 |
|  |  |  | 2 | 5000 | 139197.25 | 139761.95 | 150356.19 | 1868.9 | 4 | 18 | 1599 | 538 |
|  |  |  | 2 | 10000 | 139655.53 | 140917.82 | 150356.19 | 3835.12 | 4 | 11.5 | 1431 | 231.75 |
|  |  |  | 3 | 1000 | 139263.91 | 140052.41 | 143895.11 | 3386.09 | 3 | 47.5 | 238 | 121.75 |
|  |  |  | 3 | 5000 | 139197.25 | 139655.47 | 143699.11 | 617.89 | 6 | 527 | 1272 | 498 |
|  |  |  | 3 | 10000 | 139263.91 | 139761.95 | 147565.45 | 3239.43 | 3 | 320.5 | 1531 | 1189.75 |
|  |  |  | 4 | 1000 | 139263.94 | 141528.89 | 146949.88 | 4313.66 | 9 | 27.5 | 1018 | 169.5 |
|  |  |  | 4 | 5000 | 139197.25 | 139459.69 | 146949.95 | 2300.07 | 8 | 453.5 | 607 | 480.25 |
|  |  |  | 4 | 10000 | 139263.91 | 140052.41 | 146949.88 | 2901.92 | 8 | 24.5 | 1256 | 907 |
|  |  |  | 5 | 1000 | 139197.25 | 139263.91 | 142693.64 | 25.25 | 1 | 64.5 | 882 | 223.75 |
|  |  |  | 5 | 5000 | 139197.25 | 139655.52 | 150627.02 | 671.11 | 3 | 586.5 | 1207 | 172 |
|  |  |  | 5 | 10000 | 139197.25 | 141528.89 | 150627.02 | 2119.43 | 1 | 6.5 | 1277 | 835 |
|  |  |  | 6 | 1000 | 139197.25 | 140733.71 | 151853.19 | 2901.86 | 1 | 92.5 | 276 | 143.75 |
|  |  |  | 6 | 5000 | 139197.25 | 139868.36 | 151853.19 | 1994.5 | 1 | 214.5 | 816 | 466.5 |
|  |  |  | 6 | 10000 | 139197.25 | 139868.36 | 150356.13 | 3495.66 | 1 | 261.5 | 930 | 639.25 |
|  | ap 25 a 5 | 123574.3 | 1 | 1000 | 123659.66 | 125298.75 | 131015.59 | 1496.57 | 172 | 428 | 1303 | 334 |
|  |  |  | 1 | 5000 | 123574.3 | 123988.34 | 125391.03 | 1259.48 | 574 | 1175 | 1518 | 270.25 |
|  |  |  | 1 | 10000 | 124272.92 | 125344.92 | 134513.42 | 1688.86 | 617 | 1068 | 1526 | 207 |
|  |  |  | 2 | 1000 | 123574.3 | 123974.75 | 134496.7 | 4565.5 | 2 | 150.5 | 1039 | 301.25 |
|  |  |  | 2 | 5000 | 123574.3 | 123694.82 | 134063.31 | 8192.3 | 21 | 658 | 1308 | 329.75 |
|  |  |  | 2 | 10000 | 123659.66 | 128832.07 | 135563.48 | 8226.61 | 4 | 946.5 | 1515 |  |
|  |  |  | 3 | 1000 | 123574.3 | 123659.66 | 140912.19 | 5216.01 | 1 | 72.5 | 232 | 102.5 |
|  |  |  | 3 | 5000 | 123574.3 | 123694.84 | 141540 | 5116.47 | 1 | 245.5 | 593 | 481.75 |
|  |  |  | 3 | 10000 | 123574.3 | 123694.84 | . 133613.92 | 2352.2 | I | 1003 | 1194 | 1137.5 |

SA Standard Results

| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 4 | 1000 | 123659.7 | 124514.34 | 136601.98 | 8414.71 | 4 | 63.5 | 358 | 153.25 |
|  |  |  | 4 | 5000 | 123574.3 | 126044.07 | 136601.98 | 6362.57 | 13 | 175 | 1243 | 454.25 |
|  |  |  | 4 | 10000 | 123574.3 | 125344.86 | 134496.7 | 8601.29 | 4 | 474.5 | 1586 | 1347.5 |
|  |  |  | 5 | 1000 | 123574.3 | 126758.79 | 137538.94 | 7931.64 | 2 | 97 | 454 | 177.5 |
|  |  |  | 5 | 5000 | 123574.3 | 127204.83 | 139347.73 | 10174.15 | 2 | 157 | 1152 | 432 |
|  |  |  | 5 | 10000 | 123574.3 | 125181.13 | 141540 | 7241.7 | 4 | 1015.5 | 1224 | 1160.75 |
|  |  |  | 6 | 1000 | 123574.3 | 123659.66 | 130046.5 | 116.78 | 1 | 112 | 1410 | 269.5 |
|  |  |  | 6 | 5000 | 123574.3 | 123659.66 | 134789.6 | 0 | 1 | 421 | 1507 | 541.5 |
|  |  |  | 6 | 10000 | 123574.3 | 129795.06 | 134900.05 | 4835.49 | 1 | 629 | 988 | 844.25 |
| UMAPHMP | apl0a2 | 163603.94 | 1 | 1000 | 163603.95 | 163603.95 | 172935.64 | 0 | 0 | 8.5 | 9 | 6.75 |
|  |  |  | 1 | 5000 | 163603.95 | 163603.95 | 186957.66 | 0 | 0 | 42.5 | 203 | 3 |
|  |  |  | 1 | 10000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 83.5 | 87 | 5.75 |
|  |  |  | 2 | 1000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 2 | 5000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 2 | 10000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 3 | 1000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 3 | 5000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 3 | 10000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 4 | 1000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 4 | 5000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 4 | 10000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0.25 |
|  |  |  | 5 | 1000 | 163603.95 | 168046.94 | 195322.39 | 18006.01 | 0 | 5 | 28 | 5.25 |
|  |  |  | 5 | 5000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 36.5 | 51 | 24.75 |
|  |  |  | 5 | 10000 | 163603.95 | 163603.95 | 168046.94 | 0 | 0 | 48.5 | 51 | 2.25 |
|  |  |  | 6 | 1000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0 |
|  |  |  | 6 | 5000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0 |
|  |  |  | 6 | 10000 | 163603.95 | 163603.95 | 163603.95 | 0 | 0 | 0 | 1 | 0 |
|  | apl0a3 | 131581.79 | 1 | 1000 | 131581.79 | 131581.79 | 133798.33 | 2216.54 | 0 | 5 | 43 | 22.25 |
|  |  |  | 1 | 5000 | 131581.79 | 131581.79 | 133798.33 | 0 | 0 | 54.5 | 317 | 107.25 |
|  |  |  | 1 | 10000 | 131581.79 | 131581.79 | 133440.58 | 0 | 0 | 215.5 | 645 | 221.5 |
|  |  |  | 2 | 1000 | 131581.79 | 131581.79 | 131581.79 | 0 | 0 | 3 | 14 |  |
|  |  |  | 2 | 5000 | 131581.79 | 131581.79 | 131581.79 | 0 | 0 | 3 | 14 | 4 |
|  |  |  | 2 | 10000 | 131581.79 | 13.1581 .79 | 131581.79 | 0 | 0 | 3 | 14 | 4 |
|  |  |  | 3 | 1000 | 131581.79 | 131581.79 | 131581.79 | 0 | 0 | 1.5 | 8 | 1.5 |
|  |  |  | 3 | 5000 | 131581.79 | 131581.79 | 131581.79 | 0 | 0 | 1.5 | 8 | 1.5 |
|  |  |  | 3 | 10000 | 131581.79 | 131581.79 | 131581.79 | 0 | 0 | 1.5 | 8 | 1.5 |
|  |  |  | 4 | 1000 | 131581.79 | 131581.79 | . 131581.79 | 0 | 0 | 0.5 | 2 | 0.25 |



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 5 | 5000 | 86028.89 | 86028.89 | 86028.89 | 0 | 0 | 128.5 | 262 | 256 |
|  |  |  | 5 | 10000 | 86028.89 | 86028.89 | 86028.89 | 0 | 0 | 260 | 524 | 518.75 |
|  |  |  | 6 | 1000 | 86028.89 | 86028.89 | 86028.89 | 0 | 0 | 8 | 40 | 11.5 |
|  |  |  | 6 | 5000 | 86028.89 | 86028.89 | 86028.89 | 0 | 0 | 8 | 40 | 11.75 |
|  |  |  | 6 | 10000 | 86028.89 | 86028.89 | 86028.89 | 0 | 0 | 8 | 40 | 11.75 |
| QAP ${ }^{16}$ | nug08 | 107 | 1 | 1000 | 107 | 107 | 107 | 0 | 0 | 0 | 0 | 0.25 |
|  |  |  | 1 | 5000 | 107 | 107 | 107 | 0 | 0 | 2 | 2 | 1.75 |
|  |  |  | 1 | 10000 | 107 | 107 | 107 | 0 | 0 | 4.5 | 5 | 3.5 |
|  |  |  | 2 | 1000 | 107 | 107 | 107 | 0 | 0 | 0.5 | 1 | 0 |
|  |  |  | 2 | 5000 | 107 | 107 | 107 | 0 | 0 | 3 | 4 | 0 |
|  |  |  | 2 | 10000 | 107 | 107 | 107 | 0 | 0 | 6.5 | 7 | 0.25 |
|  |  |  | 3 | 1000 | 107 | 107 | 107 | 0 | 1 | 1 | 2 | 0.25 |
|  |  |  | 3 | 5000 | 107 | 107 | 107 | 0 | 3 | 3.5 | 4 | 0.25 |
|  |  |  | 3 | 10000 | 107 | 107 | 107 | 0 | 7 | 7 | 7 | 0.25 |
|  |  |  | 4 | 1000 | 107 | 107 | 107 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  |  | 4 | 5000 | 107 | 107 | 107 | 0 | 0 | 3 | 3 | 3 |
|  |  |  | 4 | 10000 | 107 | 107 | 107 | 0 | 0 | 6 | 6 | 6 |
|  |  |  | 5 | 1000 | 107 | 107 | 107 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  |  | 5 | 5000 | 107 | 107 | 107 | 0 | 0 | 2.5 | 3 | 2.5 |
|  |  |  | 5 | 10000 | 107 | 107 | 107 | 0 | 0 | 5 | 5 | 5 |
|  |  |  | 6 | 1000 | 107 | 107 | 107 | 0 | 0 | 0 | 1 | 0.5 |
|  |  |  | 6 | 5000 | 107 | 107 | 107 | 0 | 0 | 0 | 3 | 3.25 |
|  |  |  | 6 | 10000 | 107 | 107 | 107 | 0 | 0 | 0 | 6 | 6.25 |
|  |  |  | 7 | 1000 | 107 | 107 | 107 | 0 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 7 | 5000 | 107 | 107 | 107 | 0 | 0 | 3.5 | 4 |  |
|  |  |  | 7 | 10000 | 107 | 107 | 107 | 0 | 0 | 6.5 | 7 | 5 |
|  |  |  | 8 | 1000 | 107 | 107 | 107 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  |  | 8 | 5000 | 107 | 107 | 107 | 0 | 0 | 3 | 3 | 3.25 |
|  |  |  | 8 | 10000 | 107 | 107 | 107 | 0 | 0 | 6.5 | 7 | 6.25 |
|  | nug 12 | 289 | 1 | 1000 | 289 | 289 | 289 |  | 1 | 2 | 3 | 1.25 |
|  |  |  | 1 | 5000 | 289 | 289 | 289 | 0 | 6 | 7 | 11 | 1 |
|  |  |  | 1 | 10000 | 289 | 289 | 289 | 0 | 11 | 11.5 | 15 | 0.5 |
|  |  |  | 2 | 1000 | 289 | 289 | 289 | 0 | 4 | 8.5 | 35 | 9.25 |
|  |  |  | 2 | 5000 | 289 | 289 | 289 | 0 | 13 | 32.5 | 83 | 19.25 |
|  |  |  | 2 | 10000 | 289 | 289 | 289 | 0 | 23 | 26 | 58 | 12 |
|  |  |  | 3 | 1000 | 289 | 289 | 289 | 0 | 2 |  | 67 |  |
|  |  |  | 3 | 5000 | 289 | 289 | 289 | 0 | 14 | 27.5 | 77 | 14 |

[^15]| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| nug 15 |  | 575 | 3 | 10000 | 289 | 289 | 289 | 0 | 25 | 36.5 | 83 | 10.5 |
|  |  | 4 | 1000 | 289 | 289 | 289 | 0 | 2 | 5 | 10 | 4.25 |
|  |  | 4 | 5000 | 289 | 289 | 289 | 0 | 10 | 15.5 | 27 | 7.75 |
|  |  | 4 | 10000 | 289 | 289 | 289 | 0 | 19 | 22.5 | 30 | 6 |
|  |  | 5 | 1000 | 289 | 289 | 289 | 0 | 0 | 2 | 6 | 1.25 |
|  |  | 5 | 5000 | 289 | 289 | 289 | 0 | 0 | 8.5 | 14 | 2.75 |
|  |  | 5 | 10000 | 289 | 289 | 289 | 0 | 0 | 15 | 19 | 0.5 |
|  |  | 6 | 1000 | 289 | 289 | 289 | 0 | 2 | 8 | 22 | 7 |
|  |  | 6 | 5000 | 289 | 289 | 289 | 0 | 11 | 13 | 33 | 2.25 |
|  |  | 6 | 10000 | 289 | 289 | 289 | 0 | 23 | 24 | 31 | 2 |
|  |  | 7 | 1000 | 289 | 289 | 289 | 0 | 3 | 11.5 | 21 | 11.25 |
|  |  | 7 | 5000 | 289 | 289 | 289 | 0 | 11 | 18 | 42 | 8 |
|  |  | 7 | 10000 | 289 | 289 | 289 | 0 | 23 | 30 | 43 | 7 |
|  |  | 8 | 1000 | 289 | 289 | 289 | 0 | 2 | 6 | 14 | 4 |
|  |  | 8 | 5000 | 289 | 289 | 289 | 0 | 10 | 13.5 | 23 | 7.75 |
|  |  | 8 | 10000 | 289 | 289 | 289 | 0 | 19 | 21 | 41 | 2.25 |
|  |  | 1 | 1000 | 575 | 575 | 575 | 0 | 0 | 5.5 | 18 | 3.75 |
|  |  | 1 | 5000 | 575 | 575 | 575 | 0 | 0 | 10 | 19 |  |
|  |  | 1 | 10000 | 575 | 575 | 575 | 0 | 0 | 27 | 37 | 11.75 |
|  |  | 2 | 1000 | 575 | 575 | 575 | 0 | 14 | 254.5 | 868 | 594.25 |
|  |  | 2 | 5000 | 575 | 575 | 576 | 0 | 61 | 181 | 779 | 383.5 |
|  |  | 2 | 10000 | 575 | 575 | 575 | 0 | 86 | 316 | 791 | 338.25 |
|  |  | 3 | 1000 | 575 | 575 | 575 | 0 | 14 | 191.5 | 906 | 326 |
|  |  | 3 | 5000 | 575 | 575 | 575 | 0 | 89 | 695 | 1517 | 521 |
|  |  | 3 | 10000 | 575 | 575 | 576 | 0.75 | 155 | 516 | 1202 | 429.75 |
|  |  | 4 | 1000 | 575 | 575 | 575 | 0 | 4 | 30 | 50 | 16.5 |
|  |  | 4 | 5000 | 575 | 575 | 575 | 0 | 20 | 46 | 121 | 33.25 |
|  |  | 4 | 10000 | 575 | 575 | 575 | 0 | 38 | 68.5 | 118 | 24.25 |
|  |  | 5 | 1000 | 575 | 575 | 575 | 0 | 3.5 | 9.5 | 24 | 6.25 |
|  |  | 5 | 5000 | 575 | 575 | 575 | 0 | 13 | 17 | 37 | 7.25 |
|  |  | 5 | 10000 | 575 | 575 | 575 | 0 | 26 | 37 | 48 | 5.5 |
|  |  | 6 | 1000 | 575 | 575 | 575 | 0 | 0 | 41 | 96 | 22.5 |
|  |  | 6 | 5000 | 575 | 575 | 575 | 0 | 0 | 76.5 | 169 | 72.75 |
|  |  | 6 | 10000 | 575 | 575 | 575 | 0 | 0 | 96.5 | 234 | 55.75 |
|  |  | 7 | 1000 | 575 | 575 | 575 | 0 | 5 | 22.5 | 78 | 40 |
|  |  | 7 | 5000 | 575 | 575 | 575 | 0 | 25 | 60.5 | 218 |  |
|  |  | 7 | 10000 | 575 | 575 | 575 | 0 | 47 | 124 | 309 | 118.5 |
|  |  | 8 | 1000 | 575 | 575 | 575 | 0 | 8 | 23 | 58 | 15.25 |
|  |  | 8 | 5000 | 575 | 575 | 575 | 0 | 25 | 33.5 | 96 | 21.5 |


| Problem | Instance | Optimal / Best Known cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | nug20 | 1285 | 8 | 10000 | 575 | 575 | 575 | 0 | 41 | 71.5 | 189 | 54 |
|  |  |  | 1 | 1000 | 1285 | 1285 | 1285 | 0 | 4 | 12 | 22 | 7 |
|  |  |  | 1 | 5000 | 1285 | 1285 | 1285 | 0 | 18 | 30 | 56 | 22.75 |
|  |  |  | 1 | 10000 | 1285 | 1285 | 1285 | 0 | 39 | 55.5 | 83 | 11.25 |
|  |  |  | 2 | 1000 | 1285 | 1293.5 | 1299 | 8.5 | 33 | 502 | 1469 | 1017.5 |
|  |  |  | 2 | 5000 | 1285 | 1291 | 1301 | 9.5 | 375 | 695 | 1277 | 145 |
|  |  |  | 2 | 10000 | 1285 | 1290.5 | 1307 | 10 | 394 | 1255.5 | 1581 | 353 |
|  |  |  | 3 | 1000 | 1285 | 1287 | 1299 | 5.25 | 382 | 1369.5 | 1514 | 524 |
|  |  |  | 3 | 5000 | 1287 | 1292.5 | 1302 | 7.75 | 615 | 1019 | 1481 | 445 |
|  |  |  | 3 | 10000 | 1287 | 1297 | 1301 | 2.5 | 388 | 1267.5 | 1448 | 267.25 |
|  |  |  | 4 | 1000 | 1285 | 1285 | 1285 | 0 | 13 | 51 | 778 | 125.75 |
|  |  |  | 4 | 5000 | 1285 | 1285 | 1285 | 0 | 67 | 131.5 | 862 | 307.75 |
|  |  |  | 4 | 10000 | 1285 | 1285 | 1285 | 0 | 140 | 312 | 797 | 133.5 |
|  |  |  | 5 | 1000 | 1285 | 1285 | 1285 | 0 | 9 | 26 | 109 | 41.75 |
|  |  |  | 5 | 5000 | 1285 | 1285 | 1285 | 0 | 21 | 47.5 | 103 | 35.5 |
|  |  |  | 5 | 10000 | 1285 | 1285 | 1285 | 0 | 60 | 88.5 | 160 | 37.5 560.5 |
|  |  |  | 6 | 1000 | 1285 | 1285 | 1287 | 0 | 47 | 466.5 | 1098 | 560.5 |
|  |  |  | 6 | 5000 10000 | 1285 1285 | 1285 1285 | 1287 1285 | 0 | 122 | 227.5 | 1087 | 361 |
|  |  |  | 6 | 10000 1000 | 1285 1285 | 1285 1285 | 1285 | 0 1.5 | 249 26 | 333 357 | 1565 848 | 202 |
|  |  |  | 7 | 1000 5000 | 1285 1285 | 1285 | 1287 | 1.5 0 | 26 126 | 357 250 | 848 1463 | 375.25 400.75 |
|  |  |  | 7 7 | 5000 10000 | 1285 1285 | 1285 1285 | 1285 1285 | 0 | 126 128 | 250 425 | 1463 1269 | 400.75 366 |
|  |  |  | 8 | 1000 | 1285 | 1285 | 1285 | O | 18 | 80.5 | 670 | 121.75 |
|  |  |  | 8 | 5000 | 1285 | 1285 | 1285 | 0 | 62 | 202.5 | 996 | 269.5 |
|  |  |  | 8 | 10000 | 1285 | 1285 | 1285 | 0 | 124 | 314.5 | 781 | 279 |
|  | nug30 | 3062 | 1 | 1000 | 3062 | 3062 | 3064 | 0 | 20 | 316.5 | 1527 | 866.5 |
|  |  |  | 1 | 5000 | 3062 | 3062 | 3062 | 0 | 104 | 315.5 | 1186 | 240.25 |
|  |  |  | 1 | 10000 | 3062 | 3062 | 3064 | 0 | 119 | 492 | 932 | 438.25 |
|  |  |  | 2 | 1000 | 3123 | 3157 | 3201 | 26.25 | 315 | 878.5 | 1586 | 685.75 |
|  |  |  | 2 | 5000 | 3117 | 3150.5 | 3182 | 39.25 | 452 | 1387.5 | 1596 | 472 |
|  |  |  | 2 | 10000 | 3140 | 3171.5 | 3242 | 41.5 | 155 | 1143 | 1495 | 384 |
|  |  |  | 3 | 1000 | 3108 | 3129.5 | 3191 | 16.25 | 382 | 1265 | 1597 | 562 |
|  |  |  | 3 | 5000 | 3101 | 3134 | 3171 | 26 | 897 | 1238 | 1525 | 372.5 |
|  |  |  | 3 4 4 | 10000 | 3117 | 3160 | 3248 | 46.5 | 669 | 1035 | 1478 | 370.25 |
|  |  |  | 4 | 1000 | 3062 | 3066 | 3074 | 7.75 | 101 | 842 | 1322 | 761.5 |
|  |  |  | 4 | 5000 | 3062 | 3064 | 3074 | 0 | 265 | 600.5 | 1091 | 347.25 |
|  |  |  | 4 5 | 10000 1000 | 3062 3062 | 3077 3063 | 3086 3075 | 10.75 7.5 | 481 83 | 885.5 228 | 1515 1579 | 416.75 1202.25 |
|  |  |  | 5 | 5000 | 3062 | 3064 | 3068 | 0 | 110 | 403 | 1451 | 359 |



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/ <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 2 | 10000 | 134328 | 137075 | 141439 | 1999.25 | 5 | 9 | 19 | 5 |
|  |  |  | 3 | 1000 | 126670 | 130890.5 | 136337 | 2849.75 | 484 | 866.5 | 1499 | 761 |
|  |  |  | 3 | 5000 | 130303 | 131914 | 134496 | 2198.75 | 861 | 927.5 | 1472 | 383.5 |
|  |  |  | 3 | 10000 | 131710 | 138038.5 | 142823 | 2703.5 | 10 | 16.5 | 640 | 22 |
|  |  |  | 4 | 1000 | 120562 | 121533 | 122183 | 358.5 | 181 | 1183.5 | 1587 | 493.75 |
|  |  |  | 4 | 5000 | 121319 | 123033 | 126319 | 2094.25 | 45 | 934 | 1310 | 476.75 |
|  |  |  | 4 | 10000 | 121051 | 125499.5 | 126451 | 1953.25 | 12 | 17.5 | 211 |  |
|  |  |  | 5 | 1000 | 120334 | 121202.5 | 122288 | 938.25 | 182 | 886.5 | 1586 | 589.5 |
|  |  |  | 5 | 5000 | 120342 | 121152 | 122008 | 415 | 487 | 1134.5 | 1450 | 420 |
|  |  |  | 5 | 10000 | 120529 | 122013.5 | 122867 | 434.25 | 735 | 895 | 1491 | 417.5 |
|  |  |  | 6 | 1000 | 121120 | 122520 | 123906 | 1355.25 | 259 | 1344.5 | 1591 | 472.25 |
|  |  |  | 6 | 5000 | 121438 | 123339 | 127714 | 1763.25 | 112 | 576 | 1510 | 982.75 |
|  |  |  | 6 | 10000 | 122313 | 124291 | 126085 | 1140.75 | 87 | 173.5 | 740 | 139.25 |
|  |  |  | 7 | 1000 | 120316 | 121876 | 124103 | 571.75 | 240 | 1364 | 1599 | 372.25 |
|  |  |  | 7 | 5000 | 121804 | 123616.5 | 125406 | 1507 | 797 | 1154 | 1575 | 690.25 |
|  |  |  | 7 | 10000 | 121918 | 123456 | 130371 | 4729.75 | 39 | 768 | 1564 | 875.75 |
|  |  |  | 8 | 1000 | 120874 | 121365.5 | 122251 |  | 438 | 1197 | 1577 | 308.5 |
|  |  |  | 8 | 5000 | 121374 | 122239 | 124386 | 1002.75 | 627 | 1151 | 1582 | 163 |
|  |  |  | 8 | 10000 | 122194 | 125082.5 | 127375 | 3361.25 | 13 | 33 | 811 | 23 |
| TSP ${ }^{17}$ | gr24 | 1272 | 1 | 1000 | 1272 | 1272 | 1272 | 0 | 5 | 8 | 66 | 3.25 |
|  |  |  | 1 | 5000 | 1272 | 1272 | 1272 | 0 | 8 | 20.5 | 38 | 12 |
|  |  |  | 1 | 10000 | 1272 | 1272 | 1272 | 0 | 16 | 30 | 63 | 13.25 |
|  |  |  | 2 | 1000 | 1272 | 1272 | 1272 | 0 | 0 | 4 | 8 | 1 |
|  |  |  | 2 | 5000 | 1272 | 1272 | 1272 | 0 | 0 | 19.5 | 21 | 0.5 |
|  |  |  | 2 | 10000 | 1272 | 1272 | 1272 | 0 | 0 | 38.5 | 43 | 1.25 |
|  |  |  | 3 | 1000 | 1272 | 1272 | 1272 | 0 | 5 | 10.5 | 32 | 11.5 |
|  |  |  | 3 | 5000 | 1272 | 1272 | 1272 | 0 | 24 | 30.5 | 49 | 17 |
|  |  |  | 3 | 10000 | 1272 | 1272 | 1272 | 0 | 48 | 50 | 85 | $18.25$ |
|  |  |  | 4 | 1000 | 1272 | 1272 | 1272 | 0 | 0 | 4.5 | 10 |  |
|  |  |  | 4 | 5000 | 1272 | 1272 | 1272 | 0 | 0 | 18 | 28 | 1.25 |
|  |  |  | 4 | 10000 | 1272 | 1272 | 1272 | 0 | 0 | 35 | 36 | 1.25 |
|  |  |  | 5 | 1000 | 1272 | 1272 | 1272 | 0 | 2 | 4 | 11 | 2 |
|  |  |  | 5 | 5000 | 1272 | 1272 | 1272 | 0 | 11 | 11 | 16 | 1.25 |
|  |  |  | 5 | 10000 | 1272 | 1272 | 1272 | 0 | 21 | 23.5 | 26 | 3.5 |
|  |  |  | 6 | 1000 | 1272 | 1272 | 1272 | 0 | 0 | 2 | 5 |  |
|  |  |  | 6 | 5000 | 1272 | 1272 | 1272 | 0 | 0 | 9.5 | 27 | 18.5 |

[^16]

| Problem | Optimal / <br> Best <br> Known <br> cost | Transition <br> Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| eil5 | 426 |  | 10000 | 11461 | 11652 | 11954 | 274.25 | 424 | 1318 | 1572 | 612 |
|  |  |  | 1000 | 11461 | 11461 | 11461 | 0 | 67 | 162.5 | 310 | 85.25 |
|  |  |  | 5000 | 11461 | 11461 | 11461 | 0 | 184 | 347 | 717 | 301.75 |
|  |  |  | 10000 | 11461 | 11461 | 11461 | 0 | 191 | 371.5 | 728 | 227.5 |
|  |  |  | 1000 | 11461 | 11461 | 11461 | 0 | 15 | 118.5 | 350 | 195 |
|  |  |  | 5000 | 11461 | 11461 | 11461 | 0 | 131 | 407 | 680 | 188 |
|  |  |  | 10000 | 11461 | 11461 | 11461 | 0 | 132 | 608 | 1074 | 461.5 |
|  |  |  | 1000 | 11461 | 11461 | 11470 | 0 | 53 | 215 | 521 | 126.25 |
|  |  |  | 5000 | 11461 | 11461 | 11461 | 0 | 111 | 544.5 | 1399 | 319 |
|  |  |  | 10000 | 11461 | 11461 | 11461 | 0 | 212 | 946 | 1046 | 537.5 |
|  |  |  | 1000 | 11461 | 11461 | 11508 | 0 | 77 | 273.5 | 1362 | 239 |
|  |  |  | 5000 | 11461 | 11461 | 11484 | 0 | 311 | 815.5 | 1356 | 316 |
|  |  |  | 10000 | 11461 | 11461 | 11470 | 0 | 262 | 985.5 | 1418 | 313.5 |
|  |  |  | 1000 | 11461 | 11461 | 11461 | 0 | 57 | 234 | 425 | 131.25 |
|  |  |  | 5000 | 11461 | 11461 | 11461 | 0 | 99 | 451 | 858 | 427.5 |
|  |  |  | 10000 | 11461 | 11461 | 11461 | 0 | 340 | 844.5 | 1200 | 516.75 |
|  |  |  | 1000 | 442 | 458 | 468 | 11.25 | 9 | 14.5 | 1437 | 309 |
|  |  |  | 5000 | 442 | 446.5 | 458 | 3.5 | 37 | 52.5 | 152 | 24.25 |
|  |  |  | 10000 | 432 | 440 | 458 | 5.5 | 69 | 95 | 206 | 63.5 |
|  |  |  | 1000 | 427 | 435 | 438 | 4.25 | 38 | 45.5 | 822 | 14.75 |
|  |  |  | 5000 | 426 | 431 | 436 | 3.75 | 181 | 206 | 350 | 44 |
|  |  |  | 10000 | 427 | 430 | 432 | 3.75 | 332 | 384 | 495 | 92 |
|  |  |  | 1000 | 438 | 443.5 | 459 | 8.75 | 8 | 54.5 | 1219 | 262.5 |
|  |  |  | 5000 | 427 | 433 | 446 | 3.5 | 242 | 299 | 482 |  |
|  |  |  | 10000 | 426 | 432.5 | 439 | 4.25 | 570 | 626.5 | 954 | 282.75 |
|  |  |  | 1000 | 428 | 432 | 443 | 5.5 | 3 | 11.5 | 39 | 20.25 |
|  |  |  | 5000 | 429 | 432 | 438 | 4.75 | 6 |  | 274 | 151.5 |
|  |  |  | 10000 |  | 428.5 |  |  | 6 | 348.5 | 547 | 120.25 |
|  |  |  | 1000 | 429 | 440 | 447 | 4.75 | 4 | 7.5 | 26 | 3.25 |
|  |  |  | 5000 | 429 | 433.5 | 440 | 4.75 | 6 | 43.5 | 87 | 59.5 |
|  |  |  | 10000 | 429 | 431.5 | 439 | 2.5 | 6 | 154.5 | 250 | 21. |
|  |  |  | 1000 | 431 | 436 | 441 | 3.5 | 2 | ${ }^{21.5}$ | 60 | 27.75 |
|  |  |  | 5000 | 427 | 430.5 430 | 436 | 4.25 4.75 | 46 31 | 192.5 406.5 | 326 622 | 29.5 63.75 |
|  |  |  | 10000 1000 | 426 432 | 430 | 443 | 4.75 5 | 31 | 406.5 | ${ }_{72}$ | 63.75 |
|  |  |  | 1000 | 432 | 435 | 441 | 5.75 | 3 | 7 | 72 |  |
|  |  |  | 5000 10000 | $\begin{aligned} & 430 \\ & 426 \end{aligned}$ | 432.5 429 | 439 436 | 3.25 4 | 3 3 | 129 444 | 414 519 | 263.25 70.75 |
|  |  |  | 1000 | 429 | 440.5 | 442 | 5.5 | 4 | 9.5 | 60 | 70.75 25.25 |
|  |  |  | 5000 | 427 | 430 | . 440 | 5.75 | 4 | 184 | 298 | 134 |



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | $\begin{array}{\|l} \hline 5 \\ 6 \\ 6 \\ 6 \\ 7 \\ 7 \\ 7 \\ 8 \\ 8 \\ 8 \\ \hline \end{array}$ | 10000 1000 5000 10000 1000 5000 10000 1000 5000 10000 | $\begin{array}{\|l\|} \hline 680 \\ 678 \\ 678 \\ 676 \\ 681 \\ 677 \\ 676 \\ 687 \\ 676 \\ 678 \\ \hline \end{array}$ | 683 <br> 690 <br> 681 <br> 681 <br> 695.5 <br> 682 <br> 684.5 <br> 693 <br> 684.5 <br> 682 | $\begin{aligned} & \hline 686 \\ & 692 \\ & 689 \\ & 684 \\ & 717 \\ & 702 \\ & 691 \\ & 702 \\ & 695 \\ & 683 \\ & \hline \end{aligned}$ | 2.5 <br> 4.5 <br> 5.5 <br> 1.75 <br> 18.25 <br> 7.5 <br> 4.25 <br> 4.5 <br> 3 <br> 2.25 | 395 <br> 43 <br> 402 <br> 793 <br> 24 <br> 29 <br> 36 <br> 10 <br> 24 <br> 694 | $\begin{aligned} & \hline 568 \\ & 103 \\ & 538 \\ & 1006.5 \\ & 42.5 \\ & 560.5 \\ & 1210 \\ & 76.5 \\ & 406 \\ & 1097.5 \\ & \hline \end{aligned}$ | 627 160 777 1426 173 995 1281 133 743 1448 | $\begin{aligned} & 148.25 \\ & 20.5 \\ & 185 \\ & 190 \\ & 61.75 \\ & 131.75 \\ & 131.25 \\ & 83.5 \\ & 95.5 \\ & 430.75 \\ & \hline \end{aligned}$ |
| GAP | gapl-1 | 336 | 1 |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 5000 | 271 | 297 | 334 | 44.75 | 0 | 0 | 120 | 11 |
|  |  |  | 1 | 10000 | 271 | 297 | 334 | 44.75 | 0 | 0 | 39 | 21.5 |
|  |  |  | 2 | 1000 | 336 | 336 | 336 | 0 | 1 | 1 | 1 | 0 |
|  |  |  | 2 | 5000 | 336 | 336 | 336 | 0 | 4 | 4 | 4 | 0 |
|  |  |  | 2 | 10000 | 336 | 336 | 336 | 0 | 8 | 8 | 8 | 0 |
|  |  |  | 3 | 1000 | 336 | 336 | 336 | 0 | 2 | 2.5 | 5 | 1 |
|  |  |  | 3 | 5000 | 336 | 336 | 336 | 0 | 9 | 9 | 12 |  |
|  |  |  | 3 | 10000 | 336 | 336 | 336 | 0 | 17 | 17.5 | 21 | 1.25 |
|  |  |  | 4 | 1000 | 336 | 336 | 336 | 0 | 1 | 1.5 | 3 | 0.5 |
|  |  |  | 4 | 5000 | 336 | 336 | 336 | 0 | 6 | 6.5 | 7 | 0 |
|  |  |  | 4 | 10000 | 336 | 336 | 336 | 0 | 12 | 13 | 16 | 0 |
|  |  |  | 5 | 1000 | 336 | 336 | 336 | 0 | 1 | 1.5 | 4 | 0.5 |
|  |  |  | 5 | 5000 | 336 | 336 | 336 | 0 | 6 | 7 | 8 | 0.25 |
|  |  |  | 5 | 10000 | 336 | 336 | 336 | 0 | 13 | 13 | 15 | 0.25 |
|  | gap2-1 | 434 | 1 | 1000 | 349 | 433 | 433 | 0 | 0 | 4.5 | 12 | 4.5 |
|  |  |  | 1 | 5000 | 349 | 433 | 433 | 0 | 0 | 17.5 | 353 | 3 |
|  |  |  | 1 | 10000 | 349 | 433 | 433 | 0 | 0 | 33.5 | 361 | 5 |
|  |  |  | 2 | 1000 | 434 | 434 | 434 | 0 | 0 | 1.5 | 2 | 0.25 |
|  |  |  | 2 | 5000 | 434 | 434 | 434 | 0 | 0 | 6 | 7 | 0.5 |
|  |  |  | 2 | 10000 | 434 | 434 | 434 | 0 | 0 | 12.5 | 14 | 1.25 |
|  |  |  | 3 | 1000 | 434 | 434 | 434 | 0 | 3 |  | 10 | $3$ |
|  |  |  | 3 | 5000 | 434 | 434 | 434 | 0 | 13 | 15.5 | 27 |  |
|  |  |  | 3 | 10000 | 434 | 434 | 434 | 0 | 25 | 30 | 37 | 5.25 |
|  |  |  | 4 | 1000 | 434 | 434 | 434 | 0 | 2 | 3 | 4 | 0.75 |
|  |  |  | 4 | 5000 | 434 | 434 | 434 | 0 | 9 | 9 | 10 |  |
|  |  |  | 4 | 10000 | 434 | 434 | 434 | 0 | 18 | 19 | 21 | 1.25 |
|  |  |  | 5 | 1000 | 434 | 434 | $434$ | 0 | 2 | 2.5 | 5 |  |
|  |  |  | 5 | 5000 | 434 | 434 | 434 | 0 | 9 | 11 | 13 | 1.75 |







| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 1 | 10000 | 449 | 449 | 451 | 0 | 1069 | 1259 | 1582 | 153.25 |
|  | G250.01 ${ }^{18}$ | 29 | 1 | 1000 | 30 | 32 | 32 | 1 | 908 | 1114.5 | 1447 | 258.75 |
|  | G250.02 | 114 | 1 | 1000 | 114 | 117 | 120 | 3.5 | 928 | 1041.5 | 1407 | 244 |
|  | G250.04 | 357 | 1 | 1000 | 358 | 360 | 364 | 3.75 | 217 | 1026.5 | 1238 | 170.75 |
|  | G250.08 | 828 | 1 | 1000 | 828 | 831 | 837 | 5.5 | 660 | 972 | 1514 | 446.25 |
| MKP | weing 1 | 141278 | 1 | 1000 | 120468 | 133303 | 140543 | 5980.5 | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 5000 | 127633 | 137079 | 141278 | 7391.5 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 1 | 10000 | 127633 | 137226.5 | 141278 | 7478 | 1 | 2 | 4 | 1.25 |
|  |  |  | 2 | 1000 | 89482 | 122958.5 | 140618 | 16032 | 0 | 0 | 0 | 0 |
|  |  |  | 2 | 5000 | 121442 | 139718 | 141278 | 11837 | 0 | 1 | 2 | 0.5 |
|  |  |  | 2 | 10000 | 121931 | 140778 | 141278 | 613.75 | 1 | 1.5 | 4 | 0.5 |
|  |  |  | 3 | 1000 | 112471 | 129921 | 140477 | 8772 | 0 | 0 | 1 | 0.25 |
|  |  |  | 3 | 5000 | 127633 | 133615 | 141278 | 5814 | 1 | 1 | 2 | 0.25 |
|  |  |  | 3 | 10000 | 127633 | 133615 | 141278 | 9682.5 | 1 | 2 | 3 | 1 |
|  |  |  | 4 | 1000 | 127633 | 140507 | 140543 | 6932.5 | 0 | 0 | 1 | 0.25 |
|  |  |  | 4 | 5000 | 133615 | 140605.5 | 140786 | 235 | 0 | 1 | 2 | 0.75 |
|  |  |  | 4 | 10000 | 133609 | 140782 | 141278 | 800.75 | 0 | 2 | 10 | 2.25 |
|  |  |  | 5 | 1000 | 112327 | 129700 | 140068 | 6072.25 | 0 | 0.5 | 1 | 0 |
|  |  |  | 5 | 5000 | 125968 | 133615 | 141278 | 12366.75 | 1 | 1 | 3 | 0.25 |
|  |  |  | 5 | 10000 | 127633 | 140662 | 141278 | 7667.5 | 1 | 1.5 | 5 | 1.25 |
|  |  |  | 6 | 1000 | 111354 | 132997 | 140618 | 5980.5 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 6 | 5000 | 127243 | 133615 | 141278 | 11128.5 | 1 | 1 | 2 | 0.5 |
|  |  |  | 6 | 10000 | 127633 | 133426 | 141278 | 12893.5 | 1 | 2.5 | 9 | 1 |
|  | pb6 | 776 | 1 | 1000 | 542 | 702 | 776 | 156.5 | 0 | 1 | 2 | 1 |
|  |  |  | 1 | 5000 | 694 | 776 | 776 | 11 | 0 | 3.5 | 17 | 2.25 |
|  |  |  | 1 | 10000 | 609 | 776 | 776 | 65 | 0 | 7 | 14 | 4.5 |
|  |  |  | 2 | 1000 | 473 | 543 | 678 | 135.25 | 0 | 0 | 1 | 0 |
|  |  |  | 2 | 5000 | 473 | 568.5 | 762 | 129 | 1 | 1.5 | 5 | 1.5 |
|  |  |  | 2 | 10000 | 499 | 601 | 776 | 130.25 | 2 | 3 | 6 | 1.5 |
|  |  |  | 3 | 1000 | 609 | 733 | 776 | 111.5 | 0 | 1 | 2 | 0.75 |
|  |  |  | 3 | 5000 | 650 | 776 | 776 | 14 | 0 | 3.5 | 9 | 4 |
|  |  |  | 3 | 10000 | 694 | 776 | 776 | 0 | 0 | 10.5 | 76 | 10.75 |
|  |  |  | 4 | 1000 | 745 | 770.5 | 776 | 13.25 | 0 | 1 | 5 | 1.25 |
|  |  |  | 4 | 5000 | 694 | 776 | 776 | 0 | 1 | 2.5 | 51 | 6 |
|  |  |  | 4 | 10000 | 762 | 776 | 776 | 0 | 1 | 5 | 23 | 3.25 |
|  |  |  | 5 | 1000 | 576 | 713 | 776 | 79.25 | 0 | 0.5 | 2 | 0.5 |
|  |  |  | 5 | 5000 | 590 | 770.5 | 776 | 14 | 0 | 2 | 8 | 3 |

${ }^{18}$ Run lengths of 5000 and 10000 were too long for the time available for the standard runs and are not considered here.


|  |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Problem | Instance | $\begin{array}{\|l\|} \hline \begin{array}{l} \text { Optimal/ } \\ \text { Best } \\ \text { Known } \\ \text { cost } \end{array} \\ \hline \end{array}$ | Tra <br> Reference <br> Number | Annealing Length | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | sent01 | 7772 | 6 | 10000 | 7486 | 7486 | 7486 | 0 | 4 | 14.5 | 130 | 9 |
|  |  |  | 1 | 1000 | 6740 | 7524.5 | 7690 | 259 | 4 | 6 | 40 | 4.5 |
|  |  |  | 1 | 5000 | 7317 | 7669 | 7772 | 353.25 | 14 | 29 | 196 | 10 |
|  |  |  | 1 | 10000 1000 | 7396 4305 | 7772 6513 | 7772 7328 | ${ }_{642.25}$ | ${ }^{25}$ | 94.5 1.5 | 590 5 | 371 0.5 |
|  |  |  | 2 | 5000 | 6635 | ${ }_{7328}$ | 7690 | ${ }_{727}^{642.25}$ | 1 | 1.5 | 16 | ${ }_{2.25}$ |
|  |  |  | 2 | 10000 | 6697 | 7443 | 7772 | 444 | 7 | 11.5 | 77 | 4.75 |
|  |  |  | 3 | 1000 5000 | 7006 | 7643.5 7772 | 7772 | 383.5 82 | 13 | 10 27 | 27 | 9 |
|  |  |  | 3 | 5000 10000 | 7378 7761 | 7772 7772 | 7772 | ${ }_{0}^{82}$ | 13 31 | ${ }_{80}^{27}$ | 946 | ${ }_{81.75}^{21}$ |
|  |  |  | 4 | 1000 | 7443 | 7690 | 7772 | 103 | 1 | ${ }_{5}$ | 39 | ${ }_{7} 7.75$ |
|  |  |  | 4 | 5000 | 7386 | 7772 | 7772 | 8.25 | 2 | 34 63 | 57 | 17.25 |
|  |  |  | 4 5 | 10000 1000 | 7772 | 7772 7286 | 7772 | ${ }_{601.25}$ | 2 | 63 6.5 | 445 50 | 28 8.5 |
|  |  |  | 5 | 5000 | 7396 | 7716.5 | 7772 | 140 <br> 1.25 | 13 | ${ }_{19.5}$ | 138 | 36.25 |
|  |  |  | 5 | 10000 | 7434 | 7772 | 7772 | 14 | 24 |  | 702 | 48.75 |
|  |  |  | 6 6 | $\begin{aligned} & 1000 \\ & 5000 \\ & \end{aligned}$ | $\begin{array}{\|} 6731 \\ 7317 \end{array}$ | 7381 7438.5 | 7758 7772 772 | 171.25 382.5 | 4 | 8.5 36 | ${ }_{26}^{46}$ | 7.5 3 3 |
|  |  |  | 6 | 5000 10000 | 7317 7434 | 7438.5 7772 | 7772 | 382.5 61.5 | 12 36 | ${ }_{73.5}^{36}$ | 224 743 | 33.5 64.5 |
|  | sent02 | 8722 | 1 | 1000 | 8186 | 8605 | 8710 | 50.75 | ${ }^{36}$ | 20.5 | 100 | 64.5 16.5 |
|  |  |  | 1 | ${ }^{5000}$ | 8816 | 8640 | 8721 | 67.25 | 37 | 82 | 552 | 98.5 |
|  |  |  | 2 | 10000 1000 | 8629 6927 | 8710 7738 | 8721 8610 | 73.25 733.5 | 93 2 | 183 3.5 | 1095 5 | ${ }_{1}^{662}$ |
|  |  |  | 2 | 1000 5000 | ${ }_{7829} 97$ | 7738 879.5 | 8610 8709 | ${ }^{733.5}$ | ${ }_{8}^{2}$ | 3.5 16.5 | ${ }_{90}$ | 1 |
|  |  |  | 2 | 10000 | 8363 | 8633.5 | 8708 | 137 | 17 | 55.5 | 182 | 131 |
|  |  |  |  |  |  | ${ }_{8}^{8641} \times$ | ${ }_{8713}^{8721}$ | 91.25 81 8. | ${ }_{24}^{17}$ | ${ }_{8}^{24}$ | 153 | 45.25 |
|  |  |  | 3 | 5000 10000 | 8629 | 8717.5 8721 | 8721 8722 | 81 64 | 24 101 | 88.5 165.5 | 180 1456 | ${ }_{227.5}^{65.5}$ |
|  |  |  | 4 | 1000 | 8587 | 8663.5 | 8711 | 83.75 | 1 | 16 | 35 | 12.25 |
|  |  |  | 4 | 5000 10000 | 8829 | 8721.5 87205 | 8722 | ${ }_{2}^{13}$ | 32 | 90.5 143 1 | 211 | 78.25 |
|  |  |  | 4 5 | 10000 1000 | 8711 8263 | 8720.5 8630.5 | ${ }_{8720}^{8722}$ | $\stackrel{2}{90.25}$ | 24 5 | 143 | 1163 162 | ${ }_{9}^{910.5}$ |
|  |  |  | 5 | 5000 | 8579 | 8834.5 | ${ }_{8721}$ | 97.25 | 27 |  | 1324 | ${ }_{236}$ |
|  |  |  | 5 | 10000 | 8629 | 8712 | 8722 | 61.75 | 51 | 155.5 | 1112 | 424.5 |
|  |  |  | 6 | 1000 5000 | 8354 8603 | ${ }_{8}^{8637.5}$ | 8710 8721 | 81.5 965 | ${ }_{11}$ | ${ }_{875}^{16}$ | 106 | 11.25 5875 183 |
|  |  |  | 6 | 5000 10000 | 8803 7645 | 8671 8720 | 8721 8722 | 96.5 63.5 | 11 13 | 87.5 155.5 | 354 871 | ${ }_{133}^{58.75}$ |
|  | weish25 | 9939 | 1 | 1000 5000 | ${ }_{9536}^{947}$ | ${ }_{97807}^{9605}$ | 9777 | ${ }_{5275}^{170.25}$ |  | ${ }_{335}$ | 32 | ${ }^{12.75}$ |
|  |  |  |  |  |  | 9780.5 | . 9911 | 52.75 |  | 33.5 | 277 | 39.5 |


| SA Standard Results |  | $\begin{aligned} & \hline \text { Optimal / } \\ & \text { Best } \\ & \text { Known } \\ & \text { cost } \\ & \hline \end{aligned}$ | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| weish29 |  | 9410 | 1 | 10000 | 9737 | 9770 | 9923 | 37.75 | 35 | 254 | 343 | 215.5 |
|  |  | 2 | 1000 | 7866 | 8840 | 9552 | 955.25 | 1 | 1.5 | 2 | 0.75 |
|  |  | 2 | 5000 | 9185 | 9538 | 9817 | 235.75 | 3 | 5.5 | 35 | 3 |
|  |  | 2 | 10000 | 9598 | 9723 | 9886 | 102.75 | 5 | 9 | 134 | 2.75 |
|  |  | 3 | 1000 | 9482 | 9742 | 9913 | 238.75 | 4 | 7 | 19 | 4.5 |
|  |  | 3 | 5000 | 9784 | 9850 | 9939 | 133.75 | 15 | 29 | 208 | 18.25 |
|  |  | 3 | 10000 | 8790 | 9851 | 9939 | 136 | 5 | 49.5 | 293 | 33.25 |
|  |  | 4 | 1000 | 9710 | 9841.5 | 9939 | 166.5 | 4 | 8 | 40 | 3.5 |
|  |  | 4 | 5000 | 9749 | 9892 | 9939 | 150 | 17 | 26.5 | 284 | 33.5 |
|  |  | 4 | 10000 | 9788 | 9923 | 9939 | 16 | 25 | 123 | 535 | 298 |
|  |  | 5 | 1000 | 8819 | 9650.5 | 9852 | 230.5 | 4 | 7 | 36 | 6.75 |
|  |  | 5 | 5000 | 9223 | 9551 | 9939 | 198.25 | 10 | 25.5 | 157 | 15.75 |
|  |  | 5 | 10000 | 9588 | 9792 | 9939 | 132.75 | 1 | 49.5 | 535 | 124.25 |
|  |  | 6 | 1000 | 9536 | 9739 | 9896 | 88.75 | 3 | 6.5 | 61 |  |
|  |  | 6 | 5000 | 8554 | 9756 | 9923 | 173.75 | 4 | 57 | 168 | 100.75 |
|  |  | 6 | 10000 | 9751 | 9789 | 9923 | 3.25 | 7 | 69 | 346 | 208.75 |
|  |  | 1 | 1000 | 8799 | 9063.5 | 9410 | 138.25 | 2 | 18.5 | 68 | 24 |
|  |  | 1 | 5000 | 8819 | 9060.5 | 9287 | 304.75 | 10 | 35.5 | 176 | 94 |
|  |  | 1 | 10000 | 9056 | 9271 | 9410 | 62 | 19 | 103 | 355 | 136.25 |
|  |  | 2 | 1000 | 6952 | 8188.5 | 8595 | 504.25 | 1 | 1 | 2 | 0.5 |
|  |  | 2 | 5000 | 8341 | 8736.5 | 8891 | 216.25 | 3 |  | 29 |  |
|  |  | 2 | 10000 | 8689 | 9033.5 | 9171 | 250 | 3 | 8.5 | 25 | 4.25 |
|  |  | 3 | 1000 | 8244 | 8947 | 9318 | 358.75 | 5 | 14.5 | 83 |  |
|  |  | 3 | 5000 | 8242 | 9060.5 | 9296 | 249.5 | 37 | 52 | 220 | 42.25 |
|  |  | 3 | 10000 | 9056 | 9239 | 9318 | 202.75 | 31 | 205 | 394 | 316.25 |
|  |  | 4 | 1000 | 9054 | 9251 | 9410 | 135.75 | 2 | 7.5 | 45 | 9.5 |
|  |  | 4 | 5000 | 9244 | 9304 | 9410 | 135.25 | 10 | 49.5 | 160 | 66 |
|  |  | 4 | 10000 | 9222 | 9410 | 9410 | 73.25 | 10 | 34 | 629 | $28$ |
|  |  | 5 | 1000 | 8521 | 9148 | 9521 | 277.5 | 5 | 12 | 41 | 6.75 |
|  |  | 5 | 5000 | 8852 | 9062 | 9318 | 147 | 15 | 61 | 118 | 62 |
|  |  | 5 | 10000 | 8983 | 9251 | 9410 | 252 | 32 | 100 | 227 | 121.25 |
|  |  | 6 | 1000 | 8783 | 9045.5 | 9251 | 65.25 | 3 | 8.5 | 170 | 17 |
|  |  | 6 | 5000 10000 | 9050 9054 | 9168 9239 | 9318 9318 | $\begin{aligned} & 185 \\ & 198.25 \end{aligned}$ | 7 7 | $\begin{aligned} & 31.5 \\ & 59 \end{aligned}$ | $\begin{gathered} 193 \\ 308 \end{gathered}$ | $\begin{aligned} & 125.75 \\ & 76.5 \end{aligned}$ |
| MCP | johnson8-2-4 |  | 0 | 1 | 1000 |  | 1 | 1 | 0 | 0 | 0 | , | 0 |
|  |  |  |  | 1 | 5000 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
|  |  |  |  | 1 | 10000 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 0 |
|  | johnson16-2-4 |  | 0 | 1 | 1000 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
|  |  |  |  | 1 | 5000 | 2 | 2 | . 2 | 0 | 0 | 0 | 0 | 0 |



| Problem | Instance | Optimal/ <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | hdtt4 | 0 | 3 | 10000 | 0 | 0 | 0 | 0 | 11 | 16 | 26 | 9.25 |
|  |  |  | 4 | 1000 | 0 | 0 | 0 | 0 | 11 | 18 | 24 | 6 |
|  |  |  | 4 | 5000 | 0 | 0 | 0 | 0 | 11 | 18 | 24 | 6 |
|  |  |  | 4 | 10000 | 0 | 0 | 0 | 0 | 11 | 18 | 24 | 6.25 |
|  |  |  | 5 | 1000 | 0 | 0 | 0 | 0 | 10 | 15 | 22 | 7.25 |
|  |  |  | 5 | 5000 | 0 | 0 | 0 | 0 | 10 | 15 | 22 | 7.25 |
|  |  |  | 5 | 10000 | 0 | 0 | 0 | 0 | 10 | 15 | 22 | 7.25 |
|  |  |  | 1 | 1000 | 6 | 10.5 | 17 | 4 | 2 | 4.5 | 10 | 6 |
|  |  |  | 1 | 5000 10000 | 6 | 10.5 | 17 | 4 | 2 | 4.5 | 10 | 6 |
|  |  |  | 2 | 1000 | 0 | ${ }_{0}$ | 2 | 0 | 4 | 1.5 | 20 | 7.5 |
|  |  |  | 2 | 5000 | 0 | 0 | 0 | 0 | 4 | 14.5 | 31 | 9 |
|  |  |  | 2 | 10000 | 0 | 0 | 0 | 0 | 4 | 14.5 | 31 | 9 |
|  |  |  | 3 | 1000 | 0 | 2 | 5 | 2.25 | 8 | 20 | 26 | 7 |
|  |  |  | 3 | 5000 | 0 | 0 | 2 | 0 | 21 | 42 | 89 | 33.75 |
|  |  |  | 3 | 10000 | 0 | 0 | 0 | 0 | 21 | 42 | 237 | 34 |
|  |  |  | 4 | 1000 | 0 | 0 | 2 | 1.5 | 10 | 16 | 23 | 5.75 |
|  |  |  | 4 | 5000 | 0 | 0 | 0 | 0 | 13 | 19.5 | 44 | 12.25 |
|  |  |  | 4 | 10000 | 0 | 0 | 0 | 0 | 13 | 19 | 44 | 12.25 |
|  |  |  | 5 | 1000 | 0 | 1 | 4 | 2 | 12 | 20 | 24 | 6.75 |
|  |  |  | 5 | 5000 | 0 | 0 | 0 | 0 | 20 | 27 | 45 | 15.5 |
|  |  |  | 5 | 10000 | 0 | 0 | 0 | 0 | 20 | 27 | 45 | 15.5 |
|  | hdtt5 | 0 | 1 | 1000 | 8 | 12.5 | 19 | 2.5 | 7 | 15 | 28 | 8.5 |
|  |  |  | 1 | 5000 | 8 | 12.5 | 19 | 2.5 | 7 | 15 | 28 | 8.5 |
|  |  |  | 1 | 10000 | 8 | 12.5 | 19 | 2.5 | 7 | 15 | 28 | 8.5 |
|  |  |  | 2 | 1000 | 0 | 2 | 3 | 2.75 | 16 | 25 | 30 | 5.25 |
|  |  |  | 2 | 5000 | 0 | 0 | 3 | 0 | 16 | 35 | 93 | 25.25 |
|  |  |  | 2 | 10000 | 0 | 0 | 3 | 0 | 16 | 35 | 93 | 25.5 |
|  |  |  | 3 | 1000 | 2 | 4.5 | 6 | 2.5 | 21 | 32.5 | 37 | 10 |
|  |  |  | 3 | 5000 | 0 | 0 | 2 | 0 | 44 | 82.5 | 140 |  |
|  |  |  | 3 | 10000 | 0 | 0 | 0 | 0 | 14 | 80.5 | 329 | 74.25 |
|  |  |  | 4 | 1000 | 0 | 2.5 | 5 | 2.5 | 18 | 26 | 34 | 7.75 |
|  |  |  | 4 4 | 5000 10000 | 0 | 0 | 0 | 0 | 24 | 59 | 145 | 97.5 |
|  |  |  | 5 | 1000 | 0 | 2.5 | 5 | 2 | 24 25 | 56.5 30 | 145 35 | 97.25 5.5 |
|  |  |  | 5 | 5000 | 0 | 0 | 4 | 0 | 31 | 48.5 | 111 | 44.25 |
|  |  |  | 5 | 10000 | 0 | 0 | 4 | 0 | 32 | 51.5 | 231 | 56 |
|  | hdtto | 0 | 1 | 1000 | 12 | 13.5 | 22 | 3 | 11 | 33.5 | 47 | 17 |
|  |  |  | 1 | 5000 | 9 | 13 | . 22 | 3.75 | 11 | 33.5 | 67 | 18.25 |



| SA Standard Results |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal / <br> Best <br> Known <br> cost | Transition Set <br> Reference <br> Number | Annealing Length | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  |  | 4 | 10000 | 0 | 2.5 | 4 | 1 | 184 | 476.5 | 750 | 113 |
|  |  |  | 5 | 1000 | 10 | 14 | 17 | 2.75 | 69 | 81 | 84 | 1.5 |
|  |  |  | 5 | 5000 | 4 | 4 | 6 | 1 | 198 | 255 | 378 | 110.5 |
|  |  |  | 5 | 10000 | 0 | 2 | 4 | 1.75 | 367 | 549.5 | 818 | 151.75 |

Table 45: Standard results for the SA engine.

For the TS and GS standard runs, the most promising transition operator set (as determined form the SA trials) is used. The statistical analysis used to determine this is outlined in Section 5.8.3.1.

| $\begin{array}{\|l\|} \hline \text { TS Standard Results } \\ \hline \text { Problem } \\ \hline \end{array}$ | Instance | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
| CSP | n20t1 | 58 | 0.1 | 58 | 58 | 58 | 0 | 6 | 26 | 128 | 77 |
|  |  |  | 0.5 | 58 | 58 | 58 | 0 | 0 | 1 | 6 | 1.25 |
|  |  |  | 1.0 | 58 | 58 | 58 | 0 | 1 | 6 | 14 | 2.25 |
|  | n20t2 | 40 | 0.1 | 40 | 40 | 40 | 0 | 1 | 8 | 18 | 5.75 |
|  |  |  | 0.5 | 40 | 40 | 40 | 0 | 0 | 0.5 | 3 | 0.75 |
|  |  |  | 1.0 | 40 | 40 | 40 | 0 | 0 | 2 | 8 | 3.5 |
|  | n20t3 | 29 | 0.1 | 29 | 29 | 29 | 0 | 7 | 86 | 613 | 164 |
|  |  |  | 0.5 | 29 | 29 | 29 | 0 | 1 | 10.5 | 46 | 18.25 |
|  |  |  | 1.0 | 29 | 29 | 29 | 0 | 3 | 10.5 | 84 | 14.5 |
|  | n20t4 | 10 | 0.1 | 10 | 10 | 10 | 0 | 1 | 39 | 78 | 18.5 |
|  |  |  | 0.5 | 10 | 10 | 10 | 0 | 1 | 1.5 | 6 | 1.5 |
|  |  |  | 1.0 | 10 | 10 | 10 | 0 | 0 | 5 | 15 |  |
|  | n20t5 | 150 | 0.1 | 150 | 150 | 150 | 0 | 0 | 0.5 | 4 | 1 |
|  |  |  | 0.5 | 150 | 150 | 150 | 0 | 0 | 0.5 | 2 | 0.75 |
|  |  |  | 1.0 | 150 | 150 | 150 | 0 | 0 | 0 | 1 | 0.25 |
|  | n40t1 | 146 | 0.1 | 142 | 143 | 145 | 1.75 | 66 | 615.5 | 1464 | 600 |
|  |  |  | 0.5 | 142 | 142 | 143 | 1 | 7 | 679.5 | 1202 | 788.75 |
|  |  |  | 1.0 | 142 | 143 | 145 | 1.75 | 13 | 88.5 | 1232 | 441.25 |
|  | n40t2 | 94 | 0.1 | 92 | 92. | 92 | 0 | 13 | 116.5 | 1232 | 415.75 |


| TS Standard Results  <br> Problem  | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Min | Med | Max | 1QR | Min | Med | Max | IQR |
|  |  | 0.5 | 92 | 92 | 92 | 0 | 2 | 187 | 1411 | 185 |
|  |  | 1.0 | 92 | 92 | 97 | 1.5 | 8 | 400 | 1548 | 886 |
| n40t3 | 66 | 0.1 | 66 | 67 | 67 | 0 | 52 | 354 | 1134 | 283.5 |
|  |  | 0.5 | 65 | 65 | 66 | 0 | 25 | 277 | 1584 | 577.75 |
|  |  | 1.0 | 65 | 66 | 68 | 1 | 2 | 177.5 | 705 | 408.5 |
| n40t4 | 33 | 0.1 | 32 | 32 | 32 | 0 | 3 | 86.5 | 298 | 77.75 |
|  |  | 0.5 | 32 | 32 | 32 | 0 | 2 | 9 | 58 | 34 |
|  |  | 1.0 | 32 | 32 | 33 | 0 | 4 | 26 | 321 | 38.5 |
| n40t5 | 352 | 0.1 | 354 | 354 | 354 | 0 | 0 | 46 | 350 | 101.75 |
|  |  | 0.5 | 354 | 362 | 368 | 4.5 | 0 | 9 | 124 | 73.5 |
|  |  | 1.0 | 354 | 354 | 354 | 0 | 1 | 2.5 | 1075 | 26.25 |
| n60t1 | 238 | 0.1 | 227 | 228 | 229 | 0.75 | 71 | 680.5 | 1597 | 1050.25 |
|  |  | 0.5 | 228 | 228 | 229 | 0 | 48 | 294 | 1474 | 325 |
|  |  | 1.0 | 227 | 228 | 231 | 0.75 | 71 | 680.5 | 1597 | 1049.75 |
| n60t2 | 152 | 0.1 | 144 | 144.5 | 146 | 1 | 263 | 453 | 1134 | 242.5 |
|  |  | 0.5 | 144 | 147 | 150 | 3 | 57 | 791 | 1494 | 661.25 |
|  |  | 1.0 | 147 | 150 | 164 | 12 | 44 | 827.5 | 1521 | 788 |
| n60t3 | 105 | 0.1 | 104 | 105 | 105 | 0 | 34 | 211 | 1566 | 600.5 |
|  |  | 0.5 | 102 | 103 | 104 | 0 | 132 | 433.5 | 741 | 331 |
|  |  | 1.0 | 104 | 105 | 107 | 1 | 33 | 275.5 | 881 | 394 |
| n60t4 | 58 | 0.1 | 54 | 54 | 55 | 0.75 | 2 | 340.5 | 1150 | 408.75 |
|  |  | 0.5 | 54 | 54 | 54 | 0 | 5 | 452.5 | 1393 | 559.5 |
|  |  | 1.0 | 54 | 54 | 55 | 0.75 | 19 | 412 | 1320 | 711 |
| n60t5 | 562 | 0.1 | 558 | 566 | 574 | 0 | 1 | 6.5 | 1053 |  |
|  |  | 0.5 | 558 | 566 | 566 | 0 | 9 | 72.5 | 992 | 225.25 |
|  |  | 1.0 | 558 | 566 | 566 | 0 | 28 | 652.5 | 1549 | 902 |
| n80t1 | 330 | 0.1 | 311 | 313 | 314 | 1 | 425 | 1174 | 1566 | 477 |
|  |  | 0.5 | 312 | 314 | 316 | 1.75 | 127 | 779 | 1211 | 584.25 |
|  |  | 1.0 | 312 | 315 | 326 | 5 | 148 | 877.5 | 1561 | 740 |
| n80t2 | 215 | 0.1 | 196 | 199 | 200 | 1.5 | 124 | 452 | 826 | 368.75 |
|  |  | 0.5 | 198 | 200 | 218 | 5.75 | 180 | 731 | 1499 | 989 |
|  |  | 1.0 | 196 | 218 | 223 | 14.25 | 116 | 567.5 | 1530 | 453 |
| n80t3 | 146 | 0.1 | 143 | 143.5 | 144 | 1 | 9 | 302 | 1268 | 587.25 |
|  |  | 0.5 | 140 | 142 | 143 | 0.75 | 66 | 967 | 1572 | 552.75 |
|  |  | 1.0 | 142 | 143 | 144 | 1 | 34 | 751 | 1516 | 577 |
| n80t4 | 82 | 0.1 | 76 | 77 | 77 | 0 | 26 | 92.5 | 233 | 109.25 |
|  |  | 0.5 | 76 | 76 | 77 | 1 | 14 | 477.5 | 1437 | 695.5 |
|  |  | 1.0 | 76 | 77 | 78 | 0 | 15 | 244.5 | 1457 | 562.25 |
| n80t5 | 772 | 0.1 | 762 | 778 | 778 | 8 | 10 | 44 | 126 | 67.5 |
|  |  | 0.5 | 762 | 778 | 786 | 6 | 19 | 136.5 | 1396 | 685.5 |
|  |  | 1.0 | 770 | 780 | 796 | 8 | 85 | 260 | 1193 | 247.5 |


| Problem | Instance | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  | IQR | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med | Max |  | Min | Med | Max | IQR |
| BIN | binlal | 0 | 0.1 | 0 | 0 | 0 | 0 | 2 | 4.5 | 10 | 2 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 7 | 9.5 | 31 | 14 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 14 | 16.5 | 69 | 17.75 |
|  | binla2 | 0 | 0.1 | 0 | 0 | 0 | 0 | 2 | 3 | 6 |  |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 6 | 8 | 15 | 1.75 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 12 | 14.5 | 27 | 3 |
|  | binla 4 | 0 | 0.1 | 0 | 0 | 0 | 0 | 11 | 15 | 36 | 5 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 11 | 19 | 55 | 12 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 22 | 66.5 | 457 | 32 |
|  | binlab | 0 | 0.1 | 0 | 0 | 0 | 0 | 4 | 8.5 | 13 | 6.5 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 8 | 11.5 | 26 | 4 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 15 | 17.5 | 84 | 26.25 |
|  | bin2al | 0 | 0.1 | 0 | 0 | 0 | 0 | 68 | 135 | 381 | 63 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 301 | 675.5 | 1189 | 422.5 |
|  |  |  | 1.0 | 0 | 0 | 2 | 0.75 | 326 | 968 | 1470 | 511.75 |
|  | $\operatorname{bin} 2 \mathrm{a} 2$ | 0 | 0.1 | 0 | 0 | 0 | 0 | 33 | 40 | 55 | 9 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 113 | 133 | 181 | 16.75 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 227 | 255.5 | 439 | 33.5 |
|  | $\operatorname{bin} 2 \mathrm{a} 3$ | 0 | 0.1 | 0 | 0 | 0 | 0 | 50 | 101 | 160 | 44 |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 118 | 208 | 473 | 99.5 |
|  |  |  | 1.0 | 0 | 0 | 1 | 0 | 283 | 738.5 | 1509 | 757.25 |
|  | bin 244 | 0 | 0.1 | 0 | 0 | 0 | 0 | 42 | 56.5 | 84 |  |
|  |  |  | 0.5 | 0 | 0 | 0 | 0 | 127 | 154 | 329 | 36.25 |
|  |  |  | 1.0 | 0 | 0 | 0 | 0 | 233 | 282.5 | 580 | 139.75 |
|  | bin 3 al | 0 | 0.1 | 0 | 4.5 | 14 | 2.75 | 883 | 1300.5 | 1477 | 202.5 |
|  |  |  | 0.5 | 30 | 46 | 60 | 7.25 | 1540 | 1592.5 | 1599 | 34.5 |
|  |  |  | 1.0 | 197 | 227.5 | 265 | 36.75 | 1564 | 1590.5 | 1600 | 10.75 |
|  | bin 3 a 2 | 0 | 0.1 | 4 | 6 | 9 | 3.75 | 914 | 1256.5 | 1552 | 270 |
|  |  |  | 0.5 | 31 | 60 | 77 | 21.75 | 1506 | 1581.5 | 1600 | 43.5 |
|  |  |  | 1.0 | 171 | 237.5 | 287 | 30.25 | 1587 | 1594 | 1599 | 9.75 |
|  | $\operatorname{bin} 3 \mathrm{a} 3$ | 0 | 0.1 | 0 |  | 2 | 0.75 | 636 | 1212 | 1480 | 149.5 |
|  |  |  | 0.5 | 23 | 38.5 | 48 | 11.75 | 1547 | 1581 | 1600 | 30.5 |
|  |  |  | 1.0 | 175 | 198 | 314 | 48 | 1588 | 1592.5 | 1597 | 5.5 |
|  | bin 3 a 4 | 0 | 0.1 | 7 | 12.5 | 18 | 3.5 | 1082 | 1440 | 1599 | 292.25 |
|  |  |  | 0.5 | 55 | 67.5 | 96 | 8.5 | 1525 | 1593 | 1599 | 8.75 |
|  |  |  | 1.0 | 229 | 271.5 | 330 | 12.75 | 1574 | 1590 | 1596 | 4 |
| MKP | weing1 | 141278 | 0.1 | 141278 | 141278 | 141278 | 0 | 0 | 7 | 15 | 5.5 |
|  |  |  | 0.5 | 141278 | 141278 | 141278 | 0 | 9 | 39 | 83 | 54.25 |
|  |  |  | 1.0 | 135763 | 135763 | 135763 | 0 | 0 | 0.5 | 2 | 0.25 |
|  | pb6 | 776 | 0.1 | 776 | 776 | 776 | 0 | 0 | 34 | 115 | 61 |
|  |  |  | 0.5 | 776 | 776 | 776 | 0 | 1 | 25.5 | 108 | 43.5 |


| Problem | Instance | Optimal/ Best Known cost | Neighbourhood probability | Cost |  | Max |  | Runtime (seconds) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Min | Med |  | IQR | Min | Med | Max | IQR |
|  |  |  | 1.0 | 704 | 704 | 704 | 0 | 7 | 11.5 | 81 | 8.25 |
|  | weish12 | 6339 | 0.1 | 6323 | 6339 | 6339 | 0 | 138 | 699 | 1508 | 196 |
|  |  |  | 0.5 | 6292 | 6339 | 6339 | 0 | 21 | 419.5 | 1464 | 602 |
|  |  |  | 1.0 | 6292 | 6292 | 6292 | 0 | 72 | 352 | 918 | 608 |
|  | weish 15 | 7486 | 0.1 | 7417 | 7442.5 | 7486 | 51 | 74 | 597.5 | 1515 | 613 |
|  |  |  | 0.5 | 7418 | 7486 | 7486 | 24 | 111 | 718.5 | 1497 | 415.75 |
|  |  |  | 1.0 | 7416 | 7416 | 7416 | 0 | 54 | 146 | 380 | 189.75 |
|  | sent01 | 7772 | 0.1 | 7739 | 7740 | 7772 | 16 | 186 | 666 | 1595 | 1080.75 |
|  |  |  | 0.5 | 7739 | 7772 | 7772 | 27.5 | 216 | 525.5 | 1441 | 642 |
|  |  |  | 1.0 | 7682 | 7682 | 7682 | 0 | 76 | 218 | 770 | 130.75 |
|  | sent02 | 8722 | 0.1 | 8661 | 8692.5 | 8704 | 10.5 | 83 | 681 | 1575 | 878 |
|  |  |  | 0.5 | 8587 | 8721 | 8722 | 0 | 315 | 882.5 | 1398 | 511.25 |
|  |  |  | 1.0 | 8477 | 8563 | 8693 | 113.25 | 266 | 965.5 | 1583 | 572.25 |
|  | weish25 | 9939 | 0.1 | 9761 | 9828.5 | 9857 | 56.5 | 121 | 961.5 | 1473 | 678 |
|  |  |  | 0.5 | 9776 | 9832 | 9832 | 15 | 190 | 479.5 | 738 | 345.25 |
|  |  |  | 1.0 | 9832 | 9832 | 9832 | 0 | 165 | 326 | 1128 | 314 |
|  | weish29 | 9410 | 0.1 | 9053 | 9154.5 | 9345 | 150.75 | 164 | 904 | 1194 | 618.75 |
|  |  |  | 0.5 | 9084 | 9130 | 9316 | 37 | 171 | 533.5 | 1590 | 571.75 |
|  |  |  | 1.0 | 9023 | 9023 | 9023 | 0 | 75 | 267 | 630 | 230.25 |
| GRAPH | C125.1 | 0 | 0.1 | 7 | 8.5 | 9 | 1 | 225 | 664 | 1569 | 372.75 |
|  |  |  | 0.5 | 2 | 5 | 6 | 1 | 73 | 928 | 1553 | 867.25 |
|  |  |  | 1.0 | 3 | 4.5 | 7 | 3 | 46 | 754.5 | 1529 | 584.25 |
|  | C125.5 | 0 | 0.1 | 9 | 11 | 13 | 1.5 | 219 | 688 | 1227 | 550 |
|  |  |  | 0.5 | 4 | 6.5 | 9 | 1.75 | 252 | 1015 | 1557 | 690.25 |
|  |  |  | 1.0 | 6 | 8 | 9 | 1 | 462 | 784.5 | 1362 | 296 |
|  | Cl25.9 | 0 | 0.1 | 7 | 7 | 8 | 1 | 140 | 946.5 | 1535 | 936.25 |
|  |  |  | 0.5 | 2 | 3.5 | 5 | 1.75 | 585 | 845.5 | 1283 | 459.25 |
|  |  |  | 1.0 | 2 | 4 | 7 | 1.75 | 258 | 1148.5 | 1459 | 398 |
|  | C250.1 | 0 | 0.1 | 17 | 24.5 | 26 | 5.25 | 574 | 797 | 1454 | 500.25 |
|  |  |  | 0.5 | 16 | 22 | 24 | 3.5 | 570 | 1186 | 1530 | 494 |
|  |  |  | 1.0 | 21 | 30.5 | 33 | 6 | 798 | 1341 | 1587 | 511.25 |
|  | C250.5 | 0 | 0.1 | 14 | 24.5 | 28 | 5.5 | 284 | 1167 | 1564 | 317 |
|  |  |  | 0.5 | 21 | 24.5 | 28 | 1.75 | 838 | 1082 | 1460 | 286 |
|  |  |  | 1.0 | 28 | 32 | 41 | 4.75 | 882 | 1422.5 | 1538 | 459.75 |
|  | C250.9 | 0 | 0.1 | 17 | 22 | 24 | 3.5 | 740 | 1307 | 1543 | 344.5 |
|  |  |  | 0.5 | 22 | 24 | 28 | 1 | 884 | 1299.5 | 1523 | 289 |
|  |  |  | 1.0 | 21 | 28.5 | 38 | 3.25 | 924 | 1304.5 | 1557 | 287.5 |
| USAPHMP | ap20a2 | 172846.7 |  |  | 172816.7 | 175008.09 | 935.09 | 255 |  | 1570 | 771.75 |
|  |  |  | 0.5 | 172816.7 | 172816.7 | 172816.7 | 0 | 3 | 17.5 | 83 | 22.5 |
|  |  |  | 1.0 | 172816.7 | 179884.91 | 201353.31 | 3532.63 | 0 | 1.5 | 6 | 2.75 |
|  | ap20a3 | 151533.1 | 0.1 | 151547.63 | 153312.63 | 153828.77 | 779.66 | 68 | 619.5 | 1319 | 629.75 |




| TS Standard Results |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  |  | Med | Max | IQR | Min | Med | Max | IQR |
|  | ch130 | 6110 | 1.0 | 21717 | 22949.5 | 23767 | 429.5 | 1322 | 1444 | 1596 | 189.25 |
|  |  |  | 0.1 | 6306 | 6396.5 | 6499 | 99.5 | 288 | 735.5 | 1582 | 555.5 |
|  |  |  | 0.5 | 6287 | 6519 | 6814 | 79.75 | 886 | 1143.5 | 1343 | 311 |
|  |  | 2579 | 1.0 | 6414 | 6605 | 6988 | 194 | 1579 | 1595.5 | 1599 | 5 |
|  | a280 |  | 0.1 | 14229 | 15045.5 | 15264 | 287.25 | 1568 | 1579 | 1592 | 11.75 |
|  |  |  | 0.5 | 26511 | 27799 | 29001 | 341.25 | 1503 | 1527 | 1536 | 18 |
|  |  |  | 1.0 | 28866 | 30155 | 31376 | 482.5 | 1508 | 1530 | 1554 | 19 |
| GAP | gapl-1 | 336 | 0.1 | 336 | 336 | 336 | 0 | 1 | 3.5 | 9 | 3.25 |
|  |  |  | 0.5 | 336 | 336 | 336 | 0 | 0 | 0.5 | 1 | 0.5 |
|  |  | 434 | 1.0 | 336 | 336 | 336 | 0 | 0 | 0.5 | 7 | 0.25 |
|  | gap2-1 |  | 0.1 | 434 | 434 | 434 | 0 | 2 | 12.5 | 33 | 19.5 |
|  |  |  | 0.5 | 434 | 434 | 434 | 0 | 0 | 0 | 1 | 0.25 |
|  |  | 580 | 1.0 | 434 | 434 | 434 | 0 | 0 | 0 | 1 | 0.25 |
|  | gap3-1 |  | 0.1 | 579 | 580 | 580 | 0 | 60 | 457.5 | 1102 | 333.75 |
|  |  |  | 0.5 | 580 | 580 | 580 | 0 | 2 | 6.5 | 24 | 13.25 |
|  |  | 656 | 1.0 | 580 | 580 | 580 | 0 | 0 | 1.5 | 10 | 1.5 |
|  | gap4-1 |  | 0.1 | 656 | 656 | 656 | 0 | 13 | 39.5 | 158 | 111.25 |
|  |  |  | 0.5 | 656 | 656 | 656 | 0 | 0 | 2 | 18 | 4.25 |
|  | gap5-1 | 563 | 1.0 | 656 | 656 | 656 | 0 | 0 | 1.5 | 17 | 2.75 |
|  |  |  | 0.1 | 561 | 562.5 | 563 | 1 | 10 | 346.5 | 1329 | 564 |
|  |  |  | 0.5 | 563 | 563 | 563 | 0 | 1 | 3.5 | 15 | 5 |
|  | gap6-1 | 761 | 1.0 | 563 | 563 | 563 | 0 | 0 | 1 | 4 | 1.25 |
|  |  |  | 0.1 | 759 | 759.5 | 760 | 1 | 39 | 245.5 | 1415 | 555 |
|  |  |  | 0.5 | 761 | 761 | 761 | 0 | 4 | 29 | 178 | 56 |
|  | gap7-1 | 942 | 1.0 | 761 | 761 | 761 | 0 | 4 | 17 | 144 | 16.75 |
|  |  |  | 0.1 | 939 | 940 | 942 | 1.5 | 203 | 683 | 1423 | 507.75 |
|  |  |  | 0.5 | 942 | 942 | 942 | 0 | 8 | 74.5 | 222 | 74.5 |
|  | gap8-I | 1133 | 1.0 | 942 | 942 | 942 | 0 | 3 | 35.5 | 102 | 59 |
|  |  |  | 0.1 | 1126 | 1127 | 1129 | 1 | 10 | 356 | 1132 | 674 |
|  |  |  | 0.5 | 1131 | 1132 | 1133 | 1 | 67 | 451 | 1180 | 644 |
|  | gapA5-100 | 1698 | 1.0 | 1132 | 1132 | 1133 | 1 | 172 | 257.5 | 854 | 131.25 |
|  |  |  | 0.1 | 1699 | 1700.5 | 1701 | 1.75 | 131 | 747.5 | 1194 | 253.25 |
|  |  |  | 0.5 | 1698 | 1698 | 1699 | 1 | 36 | 328.5 | 829 | 312.5 |
|  | gapA5-200 | 3235 | 1.0 | 1698 | 1698 | 1699 | 0 | 23 | 170 | 937 | 346.5 |
|  |  |  | 0.1 | 3237 | 3241 | 3243 | 1.75 | 46 | 159 | 1416 | 575.75 |
|  |  |  | 0.5 | 3236 | 3237 | 3240 | 1.75 | 232 | 350.5 | 1035 | 205.5 |
|  | gapA10-100 | 1360 | 1.0 | 3235 | 3235 | 3237 | 0.75 | 399 | 629 | 1011 | 424 |
|  |  |  | 0.1 | 1361 | 1366 | 1367 | 0.75 | 15 | 400.5 | 895 | 431.5 |
|  |  |  | 0.5 | 1361 | 1361 | 1362 | 0 | 24 | 223 | 1134 | 194 |
|  |  |  | 1.0 | 1360 | 1361 | 1361 | 1 | 43 | 281 | 1306 | 208 |
|  | gapA10-200 | 2623 | 0.1 | 2630 | 2633 | 2636 | 2.5 | 70 | 244 | 1168 | 628.25 |


| TS Standard Results | Instance | Optimal/ Best Known cost |  |  |  | Max | IQR |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem |  |  | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  | Min | Med |  |  | Min | Med | Max | 1QR |
|  |  |  | 0.5 | 2625 | 2626 | 2629 | 2 | 236 | 324.5 | 1158 | 158.25 |
|  |  |  | 1.0 | 2624 | 2625 | 2629 | 1.75 | 458 | 893 | 1423 | 296 |
|  | gapA20-100 | 1158 | 0.1 | 1165 | 1167 | 1168 | 1 | 105 | 366 | 1324 | 803.5 |
|  |  |  | 0.5 | 1158 | 1159 | 1160 | 1 | 46 | 325 | 1166 | 309.5 |
|  |  |  | 1.0 | 1158 | 1159 | 1161 | 0.75 | 130 | 754 | 1281 | 765.75 |
|  | gapA20-200 | 2339 | 0.1 | 2349 | 2351 | 2353 | 1.5 | 74 | 636.5 | 1451 | 602 |
|  |  |  | 0.5 | 2342 | 2344.5 | 2347 | 1.75 | 284 | 750.5 | 1444 | 647 |
|  |  |  | 1.0 | 2342 | 2343 | 2346 | 1.75 | 503 | 839.5 | 1437 | 297.75 |
| GPP | G124.02 | 13 | 0.1 | 13 | 13 | 15 | 0.75 | 256 | 583.5 | 1153 | 447 |
|  |  |  | 0.5 | 14 | 16.5 | 22 | 2.5 | 254 | 634.5 | 1361 | 371.75 |
|  |  |  | 1.0 | 17 | 19 | 25 | 4.75 | 198 | 647 | 858 | 192.5 |
|  | G124.04 | 63 | 0.1 | 63 | 64 | 64 | 0 | 27 | 229.5 | 1531 | 396.75 |
|  |  |  | 0.5 | 64 | 67 | 70 | 2.75 | 51 | 682 | 1571 | 952.75 |
|  |  |  | 1.0 | 64 | 68 | 78 | 6 | 120 | 651.5 | 1482 | 803 |
|  | G124.08 | 178 | 0.1 | 178 | 178 | 179 | 0 | 180 | 874 | 1308 | 831.25 |
|  |  |  | 0.5 | 179 | 181.5 | 187 | 3.5 | 112 | 342 | 1449 | 637 |
|  |  |  | 1.0 | 181 | 186.5 | 196 | 8 | 206 | 785 | 1492 | 442.25 |
|  | G124.16 | 449 | 0.1 | 449 | 449 | 457 | 6 | 104 | 288.5 | 533 | 192.25 |
|  |  |  | 0.5 | 449 | 449 | 461 | 1.5 | 158 | 867.5 | 1421 | 776 |
|  |  |  | 1.0 | 449 | 459.5 | 476 | 15 | 113 | 624.5 | 1379 | 577.5 |
|  | G250.01 | 29 | 0.1 | 39 |  | 53 | 3.5 | 933 | 1313 | 1563 | 301.5 |
|  |  |  | 0.5 | 55 | 59.5 | 65 | 5.5 | 771 | 1283.5 | 1555 | 364 |
|  |  |  | 1.0 | 55 | 62.5 | 69 | 6 | 1427 | 1598.5 | 1599 | 42.5 |
|  | G250.02 | 114 | 0.1 | 117 | 125 | 130 | 2.5 | 613 | 1467 | 1549 | 743.5 |
|  |  |  | 0.5 | 129 | 144 | 158 | 11.5 | 1002 | 1474.5 | 1587 | 123.25 |
|  |  |  | 1.0 | 148 | 162.5 | 184 | 15.25 | 1598 | 1598.5 | 1599 | 0 |
|  | G250.04 | 357 | 0.1 | 365 | 376 | 388 | 8 | 648 | 1366.5 | 1598 | 422 |
|  |  |  | 0.5 | 377 | 393 | 398 | 9.25 | 1316 | 1513 | 1599 | 121 |
|  |  |  | 1.0 | 427 | 430 | 457 | 7.25 | 1598 | 1598.5 | 1599 | 0.25 |
|  | G250.08 | 828 | 0.1 | 834 | 847 | 868 | 16.25 | 1316 | 1513 | 1599 | 121 |
|  |  |  | 0.5 | 855 | 878.5 | 888 | 15.25 | 1460 | 1515 | 1600 | 56.5 |
|  |  |  | 1.0 | 928 | 939 | 964 | 7.5 | 1598 | 1598.5 | 1599 | 0 |
| MCP | johnson8-2-4 | 0 | 0.1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  | johnson16-2-4 | 0 | 0.1 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
|  | jommon16-2-4 |  | 0.5 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
|  | keller4 | 0 | 0.1 | 0 | 0 | 0 | 0 | 0 | 6 | 38 | 18.75 |
|  |  |  | 0.5 | 0 | 1 | 3 | 1 | 0 | 3.5 | 570 | 371.5 |
|  |  |  | 1.0 | 1 | 2 | 3 | 2 | 0 | 0 | 0 | 0 |



Table 46: Standard results for the TS engine.


$\qquad$
 $\qquad$ Nin




¢ ลे

| GS Standard | Resunstance |
| :--- | :--- |
| Problem |  |


믈
シั
ت
气
$\stackrel{\bar{U}}{\dot{G}}$
$\stackrel{Y}{\square}$
$\stackrel{0}{\stackrel{0}{\square}}$
$Z$
$\underset{G}{\square}$
n40t5
n60t1
n6012
을

| GS Standard Results |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  | n6014 | 58 | 1.0 | 106 | 108.5 | 113 | 3.75 | 2 | 3 | 4 | 0.5 |
|  |  |  | 0.1 | 67 | 75 | 84 | 7 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 59 | 62.5 | 67 | 3.5 | 1 | 1.5 | 2 | 0.25 |
|  | n60t5 | 562 | 1.0 | 55 | 59 | 63 | 3.5 | 2 | 3 | 4 | 0.75 |
|  |  |  | 0.1 | 576 | 604 | 620 | 18 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 572 | 589 | 624 | 16 | 1 | 1.5 | 2 | 0.25 |
|  | n804 | 330 | 1.0 | 572 | 593 | 610 | 14.5 | 2 | 3 | 4 | 0.5 |
|  |  |  | 0.1 | 338 | 356.5 | 367 | 13.25 | 1 | 1.5 | 2 | 0.5 |
|  |  |  | 0.5 | 325 | 340.5 | 348 | 10 | 5 | 6.5 | 7 | 0.75 |
|  | n80t2 | 215 | 1.0 | 323 | 336.5 | 340 | 5.5 | 9 | 13.5 | 16 | 3.75 |
|  |  |  | 0.1 | 230 | 267.5 | 286 | 32.25 | 1 | 1 | 2 | 0.25 |
|  |  |  | 0.5 | 215 | 248.5 | 262 | 23.75 | 4 | 6 | 8 | 1.75 |
|  | n80t3 | 146 | 1.0 | 210 | 238 | 251 | 12 | 9 | 14 | 16 | 3.5 |
|  |  |  | 0.1 | 149 | 153 | 169 | 6.25 | 1 |  | 1 | 0.25 |
|  |  |  | 0.5 | 146 | 149 | 153 | 2.5 | 4 | 5.5 | 7 | 1.5 |
|  | n80t4 | 82 | 1.0 | 146 | 148 | 154 | 4.25 | 7 | 11 | 15 | 2 |
|  |  |  | 0.1 | 89 | 98.5 | 105 | 5.25 | 1 | 1 | 1 | 0.25 |
|  |  |  | 0.5 | 82 | 85 | 97 | 4.75 | 5 | 5.5 | 7 | 0.75 |
|  | n80t5 | 772 | 1.0 | 78 | 84 | 88 | 4.25 | 10 | 11.5 | 15 | 1.75 |
|  |  |  | 0.1 | 810 | 837 | 860 | 12.5 | 1 | 1 | 2 | 0 |
|  |  |  | 0.5 | 780 | 803 | 830 | 20.5 | 5 | 6 | 8 | 1.25 |
|  |  |  | 1.0 | 784 | 810 | 838 | 18 | 8 | 11.5 | 14 | 2 |
| BIN | binlal | 0 | 0.1 | 3 | 10 | 45 | 16.75 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 0.5 | 0 | 6 | 25 | 15 | 5 | 6.5 | 7 | 1 |
|  | binla2 | 0 | 1.0 | 0 | 4 | 14 | 5 | 10 | 11 | 15 | 1.5 |
|  |  |  | 0.1 | 3 | 13.5 | 41 | 18.25 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 0.5 | 0 | 2.5 | 18 | 2.75 | 5 | 6 | 7 | 0.75 |
|  | binla4 | 0 | 1.0 | 0 | 1 | 8 | 2.5 | 9 | 10.5 | 14 | 1 |
|  |  |  | 0.1 | 20 | 54 | 69 |  | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 0.5 | 1 | 9.5 | 28 | 18.5 | 5 | 7.5 | 9 |  |
|  | binla 6 | 0 | 1.0 | 1 | 21.5 | 35 | 15.25 | 10 | 11 | 17 | 1.25 |
|  |  |  | 0.1 | 12 | 45.5 | 60 | 18.75 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 0.5 | 0 | 11 | 30 | 17.5 | 5 | 6.5 | 9 | 2 |
|  | bin 2 al | 0 | 1.0 | 0 | 17.5 | 38 | 20 | 9 | 11 | 14 | 2.25 |
|  |  |  | 0.1 | 44 | 91 | 136 | 36.25 | 15 | 18 | 22 | 2.25 |
|  |  |  | 0.5 | 24 | 58.5 | 95 | 18.75 | 58 | 77 | 102 | 12.5 |
|  | bin2a2 | 0 | 1.0 | 13 | 52.5 | 82 | 20.75 | 118 | 151.5 | 204 | 23.5 |
|  |  |  | 0.1 | 12 | 51 | 103 | 50.75 | 15 |  | 24 | 6.25 |
|  |  |  | 0.5 | 0 | 20 | 66 | 25.5 | 61 | 89 | 115 | 26.25 |
|  |  |  | 1.0 | 4 | 19 | 63 | 31.25 | 129 | 167.5 | 212 | 47.25 |
|  | bin 2 a 3 | 0 | 0.1 | 64 | 105 | 154 | 43 | 15 | 18 | 21 | 2.75 |



| GS Standard Results |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/ Best <br> Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | 1QR |
| GRAPH | C125.1 | 0 | 0.1 | 18 | 25 | 30 | 5 | 2 | 3.5 | 5 | 0.25 |
|  |  |  | 0.5 | 14 | 19.5 | 24 | 4.75 | 10 | 13 | 16 | 4.25 |
|  |  |  | 1.0 | 13 | 17 | 21 | 3.5 | 19 | 26 | 35 | 5.25 |
|  | C125.5 | 0 | 0.1 | 43 | 48 | 56 | 3.5 | 1 | 1.5 | 2 | 0.5 |
|  |  |  | 0.5 | 25 | 30 | 39 | 5.25 | 7 | 7.5 | 8 | 1 |
|  |  |  | 1.0 | 24 | 29.5 | 38 | 9 | 11 | 14.5 | 17 | 1.75 |
|  | C125.9 | 0 | 0.1 | 27 | 31 | 35 | 4.25 | 2 | 2.5 | 3 | 0.25 |
|  |  |  | 0.5 | 17 | 22 | 29 | 3.5 | 9 | 10 | 11 | 0.5 |
|  |  |  | 1.0 | 17 | 21 | 25 | 3.75 | 18 | 19 | 22 | 1 |
|  | C250.1 | 0 | 0.1 | 42 | 57 | 64 | 13.25 | 29 | 40.5 | 59 | 12.75 |
|  |  |  | 0.5 | 37 | 47 | 62 | 4.25 | 137 | 166.5 | 234 | 28 |
|  |  |  | 1.0 | 32 | 42 | 58 | 9.5 | 324 | 379 | 569 | 79.75 |
|  | C250.5 | 0 | 0.1 | 68 | 86 | 93 | 14.5 | 16 | 20 | 23 | 3 |
|  |  |  | 0.5 | 57 | 70 | 77 | 7.75 | 70 | 85.5 | 102 | 8.5 |
|  |  |  | 1.0 | 55 | 66.5 | 69 | 4.75 | 158 | 171 | 184 | 11.75 |
|  | C250.9 | 0 | 0.1 | 63 | 78 | 91 | 15.25 | 21 | 25.5 | 30 | 2 |
|  |  |  | 0.5 | 56 | 60.5 | 69 | 10 | 96 | 109.5 | 118 | 10 |
|  |  |  | 1.0 | 50 | 55 | 61 | 6.75 | 195 | 212 | 237 | 12.5 |
| USApHMP | ap20a2 | 172846.7 | 0.1 | 233530.88 | 304352.21 | 503992.56 | 157399.63 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 175638.32 | 229751.74 | 282306.84 | 48856.14 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 179884.84 | 219489.93 | 282306.84 | 33237.34 | 0 | 0 | 0 | 0 |
|  | ap20a3 | 151533.1 | 0.1 | 206197.44 | 269925.77 | 401995.03 | 78571.9 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 162662.77 | 207271.01 | 218225.59 | 29053.31 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 159116.36 | 195730.8 | 214503.03 | 30715.42 | 1 | 1 | 1 | 0 |
|  | ap20a4 | 135624.9 | 0.1 | 199164.77 | 239693.38 | 312220.66 | 32353.52 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 150274.91 | 164348.25 | 193745.73 | 11627.36 | 0 | 0 | 1 | 0.25 |
|  |  |  | 1.0 | 143641.97 | 160768.04 | 192892.05 | 9724.58 | 1 | 1 | 2 | 0.25 |
|  | ap20a5 | 123130.1 | 0.1 | 158279.31 | 193951.73 | 390419.72 | 34389.45 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 130424.72 | 139049.02 | 140874.13 | 5474.04 | 1 | 1 | 1 | 0 |
|  |  |  | 1.0 | 126366.75 | 135365.77 | 147606.81 | 4973.03 | 1 | 1.5 | 2 | 0.25 |
|  | ap25a2 | 175542.0 | 0.1 | 253240.83 | 294952.57 | 504773.94 | 67331.26 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 188214.92 | 215822.65 | 309291.47 | 27667.06 | 0 | 0.5 | 1 | 0.25 |
|  |  |  | 1.0 | 187950.11 | 211359.6 | 347292.39 | 21474.61 | 1 | 1 | 2 | 0.25 |
|  | ap25a3 | 155256.3 | 0.1 | 187802.8 | 253153.69 | 402041.78 | 105081.53 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 164168.44 | 174893.87 | 187440.48 | 12642.07 | 1 | 1 | 2 | 0.25 |
|  |  |  | 1.0 | 163228.89 | 178133.7 | 188583.02 | 13567.05 | 1 | 2.5 | 3 | 1 |
|  | ap25a4 | 139197.2 | 0.1 | 195984.84 | 228987.92 | 290637.06 | 51264.94 | 0 |  |  | 0 |
|  |  |  | 0.5 | 149223.63 | 156970.94 | 184919.14 | 17878.67 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 1.0 | 148521.67 | 160001.87 | 184919.19 | 14936.3 | 2 | 3.5 | 4 | 0.25 |
|  | ap25a5 | 123574.3 | 0.1 | 156083.73 | 191566.92 | 343711.84 | 45218.95 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 131935.94 | 151471.9 | 155406.77 | 8800.01 | 2 | 2 | 3 | 0.5 |


| GS Standard Results |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Optimal/ Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  | Min | Med | Max | IQR | Min | Med | Max | IQR |
|  |  | 1.0 | 131935.94 | 147875.14 | 157888.53 | 9831.49 | 3 | 3.5 | 5 | 1 |
| UMApHMP | 163603.94 | 0.1 | 185318.72 | 242074.61 | 361375.28 | 63470.97 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 166119.73 | 200881.02 | 211700.2 | 20185.28 | 0 | 0 | 0 | 0 |
|  | 131581.79 | 1.0 | 164247.89 | 200461.82 | 203588.11 | 8433.7 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 182709.98 | 206646.53 | 268276.88 | 21443.61 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 132696.41 | 163504.21 | 207662.33 | 45733.24 | 0 | 0 | 0 | 0 |
|  | 107354.73 | 1.0 | 134671.78 | 171344.55 | 206426.98 | 23168.67 | 0 | 0 | 1 | 0 |
|  |  | 0.1 | 154327.73 | 179371.49 | 389808.22 | 56900.24 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 133080.23 | 171462.32 | 197763.72 | 12053.78 | 0 | 0 | 0 | 0 |
|  | 86028.88 | 1.0 | 133142.03 | 153865.2 | 185588.33 | 12123.51 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 120292.92 | 154365.82 | 327885 | 55508.74 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 107701.38 | 128705.51 | 197709.23 | 23074.68 | 0 | 0 | 0 | 0 |
|  |  | 1.0 | 110898.82 | 135786.1 | 171808.41 | 17128.22 | 0 | 0 | 0 | 0 |
| QAP | 107 | 0.1 | 113 | 142.5 | 172 | 22 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 107 | 117.5 | 121 | 7.5 | 0 | 0 | 0 | 0 |
|  | 289 | 1.0 | 107 | 111 | 119 | 5.5 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 307 | 348.5 | 415 | 24.25 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 297 | 307 | 326 | 14.5 | 0 | 0 | 0 | 0 |
|  | 575 | 1.0 | 293 | 298.5 | 319 | 11.25 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 641 | 676 | 710 | 26.75 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 593 | 621 | 656 | 32 | 0 | 0 | 0 | 0 |
|  | 1285 | 1.0 | 580 | 593 | 630 | 12.75 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 1383 | 1421.5 | 1532 | 65.5 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 1304 | 1348 | 1407 | 21 | 0 | 0 | 0 | 0 |
|  | 3062 | 1.0 | 1303 | 1341 | 1351 | 20 | 0 | 0 | 0 | 0 |
|  |  | 0.1 | 3302 | 3379 | 3462 | 57.5 | 0 | 0 | 0 | 0 |
|  |  | 0.5 | 3160 | 3196.5 | 3240 | 41.75 | 1 | 1.5 | 2 | 0.25 |
|  | 4763 | 1.0 | 3142 | 3193.5 | 3278 | 45.25 | 2 | 3 | 4 | 0.5 |
|  |  | 0.1 | 5508 | 6055.5 | 6864 | 377.5 | 0 | 0.5 | 1 | 0.25 |
|  |  | 0.5 | 5179 | 5383.5 | 5693 | 277.25 | 3 | 4 | 5 | 0.25 |
|  | 120258 | 1.0 | 5088 | 5535 | 5967 | 347 | 6 | 7.5 | 11 | 1.75 |
|  |  | 0.1 | 129426 | 134078 | 137393 | 3930.75 | 1 | 1 | 1 | 0.25 |
|  |  | 0.5 | 123593 | 127623.5 | 129126 | 1881.25 | 4 | 5 | 7 | 1.25 |
|  | 58 | 1.0 | 122832 | 125942 | 127565 | 1383.75 | 8 | 11.5 | 17 | 2.5 |
|  |  | 0.1 | 59 | 60.5 | 68 | 1.75 | 2 | 2.5 | 4 | 0.5 |
|  |  | 0.5 | 58 | 60 | 61 | 1.75 | 8 | 11 | 14 | 2.5 |
|  | 33158 | 1.0 | 58 | 58.5 | 61 | 2 | 17 | 21.5 | 27 | 5 |
|  |  | 0.1 | 34297 | 34412.5 | 34890 | 148.75 | 14 | 18.5 | 24 | 6 |
|  |  | 0.5 | 33629 | 34009 | 34333 | 146.25 | 97 | 108.5 | 154 | 24.25 |
|  |  | 1.0 | 33752 | 33991 | 34665 | 199 | 203 | 228 | 288 | 17.25 |
|  | 136522 | 0.1 | 138072 | 138502 | . 138941 | 394.75 | 92 | 112 | 148 | 27.75 |


| GS Standard Results |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/Best Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | 1QR |
|  |  |  | 0.5 | 137447 | 137952.5 | 138524 | 605.75 | 596 | 686 | 712 | 61.75 |
|  |  |  | 1.0 | 137852 | 138065.5 | 138642 | 263 | 1272 | 1435.5 | 1553 | 149 |
| TSP | gr24 | 1272 | 0.1 | 1440 | 1741.5 | 2034 | 254.5 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 1320 | 1367 | 1454 | 74.5 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 1272 | 1308 | 1415 | 78.25 | 0 | 0 | 0 | 0 |
|  | swiss 42 | 1273 | 0.1 | 1429 | 1639 | 1917 | 176 | 0 | 0.5 | 1 | 0 |
|  |  |  | 0.5 | 1335 | 1373.5 | 1508 | 53 | 2 | 3 | 3 | 0.25 |
|  |  |  | 1.0 | 1294 | 1376 | 1407 | 41.5 | 5 | 6 | 6 | 0.75 |
|  | hk48 | 11461 | 0.1 | 13090 | 14097 | 16478 | 869.5 | 1 | 1 | 1 | 0.25 |
|  |  |  | 0.5 | 11908 | 12009 | 12302 | 223 | 5 | 5.5 | 6 | 0.5 |
|  |  |  | 1.0 | 11760 | 12127.5 | 13023 | 243 | 10 | 10.5 | 11 | 0.75 |
|  | eil5 | 426 | 0.1 | 494 | 545.5 | 593 | 45.25 | 1 | 1.5 | 2 | 0.25 |
|  |  |  | 0.5 | 446 | 456.5 | 475 | 14.75 | 6 | 7 | 8 | 0.25 |
|  |  |  | 1.0 | 432 | 455.5 | 465 | 11.5 | 13 | 14 | 15 | 1 |
|  | brazil58 | 25395 | 0.1 | 27231 | 28831 | 31563 | 1917 | 3 | 3 | 4 | 0.5 |
|  |  |  | 0.5 | 25923 | 26636.5 | 27457 | 985.5 | 12 | 15 | 16 | 1.5 |
|  |  |  | 1.0 | 25400 | 26601.5 | 27148 | 1422.75 | 27 | 30 | 33 | 3 |
|  | st70 | 675 | 0.1 | 744 | 842.5 | 926 | 46 | 6 | 7 | 8 | 0.5 |
|  |  |  | 0.5 | 686 | 715 | 737 | 27 | 33 | 35 | 43 | 3.25 |
|  |  |  | 1.0 | 687 | 727.5 | 777 | 26.25 | 65 | 68.5 | 74 | 3 |
|  | kroA100 | 21282 | 0.1 | 23578 | 25822 | 26942 | 1699 | 123 | 128 | 150 | 7.25 |
|  |  |  | 0.5 | 21947 | 22824 | 23885 | 824.25 | 587 | 645.5 | 678 | 39.5 |
|  |  |  | 1.0 | 21741 | 23339 | 23983 | 1027.5 | 1201 | 1279.5 | 1387 | 102.25 |
|  | ch 130 | 6110 | 0.1 | 6668 | 7079.5 | 7469 | 448.5 | 156 | 171 | 186 |  |
|  |  |  | 0.5 | 6355 | 6723.5 | 6996 | 213.25 | 801 | 840.5 | 893 | 34.75 |
|  |  |  | 1.0 | 6414 | 6605 | 6988 | 213.5 | 1466 | 1593 | 1597 | 9.75 |
|  | a280 | 2579 | 0.1 | 14229 | 14844 | 15264 | 235.5 | 1576 | 1589 | 1598 | 9.5 |
|  |  |  | 0.5 | 26511 | 27799 | 29001 | 341.25 | 1502 | 1526.5 | 1535 | 18 |
|  |  |  | 1.0 | 28866 | 30155 | 31376 | 482.5 | 1508 | 1529.5 | 1553 | 19 |
| GAP | gap 1-1 | 336 | 0.1 | 271 | 299.5 | 311 | 13.25 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 288 | 307 | 321 | 12 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 294 | 320 | 330 | 12.75 | 0 | 0 | 0 | 0 |
|  | gap2-1 | 434 | 0.1 | 345 | 374 | 406 | 45.25 | 0 |  | 0 | 0 |
|  |  |  | 0.5 | 385 | 418 | 427 | 6 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 403 | 418.5 | 429 | 9.75 | 0 | 0 | 0 | 0 |
|  | gap3-1 | 580 | 0.1 | 500 | 525.5 | 553 | 10.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 539 | 562 | 577 | 6 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 554 | 571.5 | 578 | 3 | 0 | 0 | 0 | 0 |
|  | gap4-1 | 656 | 0.1 | 567 | 598.5 | 626 | 29.5 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 620 | 644 | 656 | 10.75 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 637 | 644.5 | 654 | 5.75 | 0 | 0 | 0 | 0 |



| GS Standard Results |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance | Optimal/ Best <br> Known cost | Neighbourhood probability | Cost |  |  |  | Runtime (seconds) |  |  |  |
|  |  |  |  | Min | Med | Max | IQR | Min | Med | Max | 1QR |
|  | G250.01 | 29 | 1.0 | 457 | 471.5 | 496 | 18.75 | 60 | 92 | 119 | 39.25 |
|  |  |  | 0.1 | 50 | 57.5 | 67 | 4.5 | 150 | 177.5 | 189 | 9.25 |
|  |  |  | 0.5 | 55 | 61 | 68 | 4.25 | 742 | 795.5 | 968 | 50.25 |
|  | G250.02 | 114 | 1.0 | 55 | 62.5 | 69 | 6 | 1421 | 1592 | 1593 | 42.25 |
|  |  |  | 0.1 | 135 | 147.5 | 158 | 8.5 | 206 | 224 | 259 | 23 |
|  |  |  | 0.5 | 130 | 146.5 | 164 | 15.25 | 824 | 1083 | 1481 | 161 |
|  | G250.04 | 357 | 1.0 | 148 | 162.5 | 184 | 15.25 | 1592 | 1592 | 1592 | 0 |
|  |  |  | 0.1 | 378 | 400.5 | 416 | 14.75 | 202 | 247 | 333 | 51.5 |
|  |  |  | 0.5 | 377 | 394 | 400 | 6 | 1110 | 1281 | 1507 | 199.5 |
|  | G250.08 | 828 | 1.0 | 427 | 430 | 457 | 7.25 | 1592 | 1592 | 1592 | 0 |
|  |  |  | 0.1 | 868 | 884 | 976 | 25.25 | 200 | 275.5 | 298 | 32.25 |
|  |  |  | 0.5 | 855 | 879 | 898 | 19.75 | 1166 | 1452.5 | 1595 | 250 |
|  |  |  | 1.0 | 928 | 939 | 964 | 7.5 | 1592 | 1592 | 1593 | 0.5 |
| MCP | johnson8-2-4 | 0 | 0.1 | 1 | 1.5 | 3 | 1 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 1 | 1 | 3 | 1 | 0 | 0 | 0 | 0 |
|  | johnson16-2-4 | 0 | 1.0 | 1 | 1.5 | 3 | 1 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 5 | 16.5 | 24 | 5.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 5 | 16.5 | 24 | 5.75 | 0 | 0 | 0 | 0 |
|  | keller4 | 0 | 1.0 | 5 | 16.5 | 24 | 5.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 21 | 29.5 | 37 | 6.5 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 21 | 29 | 37 | 6.5 | 0 | 0 | 0 | 0 |
|  | c-fat200-1 | 0 | 1.0 | 21 | 27.5 | 37 | 7 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 29 | 38.5 | 49 | 12.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 29 | 36 | 52 | 12.75 | 0 | 0 | 0 | 0 |
|  | brock200_2 | 0 | 1.0 | 29 | 34 | 52 | 15.5 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 39 | 46 | 57 |  | 0 |  | 0 | 0 |
|  |  |  | 0.5 | 33 | 46 | 57 | 11.75 | 0 | 0 | 0 | 0 |
|  | brock200_1 | 0 | 1.0 | 35 | 46 | 57 | 10 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 96 | 132.5 | 160 | 32.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 96 | 132.5 | 160 | 32.75 | 0 | 0 | 0 | 0 |
|  | brock200_3 | 0 | 1.0 | 89 | 132.5 | 160 | 32.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.1 | 52 | 68 | 97 | 11.75 | 0 | 0 | 0 | 0 |
|  |  |  | 0.5 | 52 | 68 | 89 | 11.75 | 0 | 0 | 0 | 0 |
|  |  |  | 1.0 | 52 | 65.5 | 97 | 18 | 0 | 0 | 0 | 0 |
| TTP | tgen | 0 | 0.1 | 3 | 6 | 8 | 1.5 | 61 | 68.5 | 79 | 5.5 |
|  |  |  | 0.5 | 0 | 2 | 3 | 2 | 275 | 320.5 | 346 | 27 |
|  |  | 0 | 1.0 | 1 | 1.5 | 3 | 1 | 562 | 609.5 | 698 | 67.75 |
|  | ttgen2 |  | 0.1 | 1 | 7 | 15 | 4 | 93 | 104.5 | 114 |  |
|  |  |  | 0.5 | 1 | 3.5 | 5 | 2 | 425 | 485 | 543 | 13.5 |
|  |  |  | 1.0 | 0 | 1 | 3 | 1.5 | 855 | 933.5 | 1039 | 53.75 |
|  | hdtt4 | 0 | 0.1 | 16 | 21.5 | 32 | 5.75 | 3 | 3 | 4 | 0.5 |


| GS Standard Results |  | Optimal/ Best Known cost |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Problem | Instance |  | Neighbourhood probability | Cost |  |  |  | Runt <br> Min | Med | Max | 1QR |
|  |  |  | 0.5 | 8 | 13 | 16 | 5.5 | 15 | 16 | 18 | 1.25 |
|  |  |  | 1.0 | 5 | 8.5 | 12 | 3 | 30 | 33.5 | 41 | 6 |
|  | hdtt5 | 0 | 0.1 | 18 | 30.5 | 39 | 4.5 | 6 | 8 | 9 | 1 |
|  |  |  | 0.5 | 15 | 16.5 | 23 | 3.5 | 36 | 40.5 | 45 | 3.25 |
|  |  |  | 1.0 | 11 | 16.5 | 22 | 4 | 70 | 74 | 85 | 4 |
|  | hdtt6 | 0 | 0.1 | 36 | 39.5 | 44 | 3.75 | 14 | 15.5 | 18 | 1 |
|  |  |  | 0.5 | 20 | 23.5 | 32 | 3.5 | 68 | 80 | 87 | 7.5 |
|  |  |  | 1.0 | 19 | 21 | 27 | 2.5 | 150 | 162 | 168 | 12 |
|  | hdtt 7 | 0 | 0.1 | 37 | 46.5 | 57 | 8 | 21 | 28 | 33 | 5.25 |
|  |  |  | 0.5 | 26 | 32 | 41 | 5.75 | 131 | 142 | 158 | 6.5 |
|  |  |  | 1.0 | 26 | 31 | 37 | 6.75 | 269 | 293.5 | 383 | 16.25 |
|  | hdtt8 | 0 | 0.1 | 47 | 51.5 | 65 | 4.25 | 38 | 49.5 | 58 | 1.75 |
|  |  |  | 0.5 | 32 | 38 | 46 | 7.25 | 209 | 229.5 | 253 | 22.25 |
|  |  |  | 1.0 | 29 | 35.5 | 43 | 3.5 | 459 | 475.5 | 547 | 29 |

Table 47: Standard results for the GS engine.

## Appendix B: List Formulations of Common COPs

This appendix contains the list formulations of the problems in Table 1 excluding those presented in Section 3.3.2. The alternative formulations for BIN, GRAPH and MCP are also included here.

## BIN

Minimise $|x|$
s.t.

| $\sum_{j=1}^{\mid x(i)!} w(x(i, j)) \leq W_{\max }$ | $\forall i$ | $1 \leq i \leq\|x\|$ |
| :--- | :--- | :--- |
| $1 \leq x(i, j) \leq N$ | $\forall i$ | $1 \leq i \leq\|x\|$ |
|  | $\forall j$ | $1 \leq j \leq\|x(i)\|$ |

min_count $(x)=1$
max_count $(x)=1$
Where:
$N$ is the number of items.
$x(i, j)$ is the $j$ 'th item assigned to bin $i$.
$W_{\max }$ is the maximum bin weight.
$w(i)$ is the weight of item $i$.

## GRAPH

## Minimise $|x|$

s.t.
$\sum_{i=1}^{|x(k)|-1} \sum_{j=i+1}^{|x(k)|} \operatorname{edge}(x(k, i), x(k, j))=0 \quad \forall k \quad 1 \leq k \leq|x|$

| $\|x(i)\| \geq 1$ | $\forall i$ | $1 \leq i \leq\|x\|$ |
| :--- | :--- | :--- |
| $1 \leq x(i, j) \leq N$ | $\forall i$ | $1 \leq i \leq\|x\|$ |
|  | $\forall j$ | $1 \leq j \leq\|x(i)\|$ |

min_count $(x)=1$
max_count $(x)=1$
Where:
$x(i, j)$ is the $j$ 'th item with colour $i$.
edge( $i, j)$ is 1 if there exists and edge between vertex $i$ and $j$, else it is 0 .
$N$ is the number of vertices.

## Capacitated Single Allocation p-Hub Median Problem

## Minimise

$\sum_{l=1}^{|x|} \sum_{k=1}^{|x|} \sum_{j=1}^{|x(l)| x(k) \mid} \sum_{i=1} W(x(k, i), x(l, j)) \cdot(\chi d(x(k, i), x(k, 1))+\tau d(x(k, 1), x(l, i))+\delta d(x(l, 1), x(l, j)))+\sum_{i=1}^{|x|} f(x(i, 1))$
s.t.

$$
\begin{array}{lll}
\sum_{i=1}^{|x|} \sum_{j=1}^{|x(i)!| x(l) \mid} \sum_{k=1} W(x(i, j), x(l, k)) \leq \Gamma(l) & \forall l & 1 \leq l \leq|x| \\
|x|=P & & \\
|x(i)| \geq 1 & \forall i & 1 \leq i \leq P \\
1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq P \\
\min \_c o u n t(x)=1 & \forall j & 1 \leq j \leq|x(i)| \\
\max \_\operatorname{count}(x)=1 & &
\end{array}
$$

Where:
$x(i, j)$ is the $j$ 'th node on the $i$ 'th hub (Note: $x(i, 1)$ is a hub).
$P$ is the number of hubs.
$N$ is the number of nodes.
$W(i, j)$ is the flow from node $i$ to node $j$.
$d(i, j)$ is the distance from node $i$ to node $j$.
$f(i)$ is the fixed cost of establishing $i$ as a hub.
$\Gamma(l)$ is the capacity of hub $l$.
$\chi$ is the collection cost coefficient.
$\delta$ is the distribution cost coefficient.
$\tau$ is the transfer cost coefficient.

## VRP

Minimise $\sum_{k=1}^{V}\left(\sum_{i=1}^{|x(k)|} d(x(k,(i-1)), x(k, i))+d(x(k,|x(k)|), 0)\right)$
s.t.

$$
\sum_{i=1}^{|x(k)|} c(x(k, i)) \leq T(k) \quad \forall k \quad 1 \leq k \leq V
$$

$$
\begin{gathered}
\sum_{i=1}^{|x(k)|}(d(x(k,(i-1)), x(k, i))+f(x(k, i)))+d(x(k,|x(k)|), 0) \leq g(k) \\
\forall k \quad 1 \leq k \leq V
\end{gathered}
$$

$|x|=V$

| $\|x(i)\| \geq 1$ | $\forall i$ | $1 \leq i \leq V$ |
| :--- | :--- | :--- |
| $1 \leq x(i, j) \leq N$ | $\forall i$ | $1 \leq i \leq V$ |
|  | $\forall j$ | $1 \leq j \leq\|x(i)\|$ |

min_count $(x)=1$
$\max$ _count $(x)=1$
Where:
$x(i, j)$ is the number of the customer who is serviced by vehicle $i$ on its $j$ 'th stop.
$V$ is the number of vehicles in the fleet.
$N$ is the number of customers.
$T(i)$ is the capacity of vehicle $i$.
$c(i)$ is the demand of customer $i$.
$f(i)$ is the drop time required for customer $i$.
$g(i)$ is the total allowable time/distance for vehicle $i$.
$d(i, j)$ is the distance from customer $i$ to $j$ (Note: customer 0 is the depot).

## Knapsack Problem

Maximise $\sum_{i=1}^{|x|} c(x(i))$
s.t.

$$
\sum_{i=1}^{|x|} w(x(i)) \leq W_{\max }
$$

$0 \leq|x| \leq N$
$1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq|x|$
max_count $(x)=1$
Where:
$x(i)$ is the $i$ 'th item in the knapsack.
$c(i)$ is the profit of including an item $i$ in the knapsack.
$w(i)$ is the weight of item $i$.
$W_{\text {mur }}$ is the maximum weight that the knapsack can support.
$N$ is the number of items.

## Set Partitioning Problem

Minimise $\sum_{i=1}^{|x|} c(x(i))$
s.t.

$$
\begin{array}{lll}
\sum_{i=1}^{|x|} a(j, x(i))=1 & \forall j & 1 \leq j \leq M \\
0 \leq|x| \leq N \\
1 \leq x(i) \leq N & & \\
\text { max_count }(x)=1 & \forall i & 1 \leq i \leq|x|
\end{array}
$$

Where:
$x(i)$ is the $i$ 'th column number in the solution.
$c(i)$ is the cost of including column $i$.
$a$ is the covering matrix.
$M$ is the number of columns.
$N$ is the number of rows.

## Set Covering Problem

Minimise $\sum_{i=1}^{|x|} c(x(i))$
s.t.

$$
\begin{array}{lll}
\sum_{i=1}^{|x|} a(j, x(i)) \geq 1 & \forall j & 1 \leq j \leq M \\
1 \leq|x| \leq N & & \\
1 \leq x(i) \leq N & \forall i & 1 \leq i \leq|x| \\
\text { max_count }(x)=1 & &
\end{array}
$$

Where:
$x(i)$ is the $i$ 'th column number in the solution.
$c(i)$ is the cost of including column $i$.
$a$ is the covering matrix.
$M$ is the number of columns.
$N$ is the number of rows.

## Aircraft Landing Problem

Minimise $\sum_{i=1}^{N} c(i) \cdot A B S\left(t(x(i))-t_{p}(i)\right)$
s.t.

| $t(x(i)) \geq t_{t}(i)$ | $\forall i$ | $1 \leq i \leq N$ |
| :--- | :--- | :--- |
| $t(x(i)) \leq t_{l}(i)$ | $\forall i$ | $1 \leq i \leq N$ |
| $A B S(t(x(i))-t(x(j))) \geq I(i, j)$ | $\forall i$ | $1 \leq i \leq N$ |
|  | $\forall j$ | $1 \leq j \leq N$ |
|  |  |  |
| $\|x\|=N$ | $\forall i$ | $1 \leq i \leq N$ |
| $1 \leq x(i) \leq M$ |  |  |

Where:
$c(i)$ is the cost per unit time for a delay or early landing.
$x(i)$ is the time index of when aircraft $i$ will land.
$t(i)$ is the $i^{\prime}$ th possible landing time.
$t_{p}(i)$ is the desired landing time of aircraft $i$.
$t_{t}(i)$ is the earliest time to land for aircraft $i$.
$t_{l}(i)$ is the latest time to land for aircraft $i$.
$I(i, j)$ is the minimum number of time periods that aircraft $i$ can land after aircraft $j$.
$M$ is the total number of landing times.
$N$ is the number of aircrafts.

## Processor Allocation Problem

Minimise $\sum_{i=1}^{M} \sum_{j=1}^{|x(i)|} \sum_{k=i+1}^{M} \sum_{l=1}^{|x(k)|} c(x(i, j), x(k, l))$
s.t.

$$
\begin{array}{lll}
\sum_{j=1}^{|x(i)|} r(x(i, j)) \leq R & \forall i & 1 \leq i \leq M \\
& & \\
|x|=M & & \\
|x(i)| \geq 1 & \forall i & 1 \leq i \leq M \\
1 \leq x(i, j) \leq N & \forall i & 1 \leq i \leq M \\
& \forall j & 1 \leq j \leq|x(i)|
\end{array}
$$

min_count $(x)=1$
max_count $(x)=1$
Where:
$x(i, j)$ is the $j$ 'th process on processor $i$.
$N$ is the number of processes.
$M$ is the number of processors.
$c(i, j)$ is the communication cost between process $i$ and $j$.
$R$ is the resource available on each processor.
$r(i)$ is the resource required by process $i$.
$p(i)=\left\{\begin{array}{l}0, \text { if }(i=0) \\ 1, \text { otherwise }\end{array}\right.$

## Single Layout Problems in FMS (Flexible Manufacturing Systems)

Minimise $\sum_{i=1}^{N} \sum_{j=1}^{N} w(i, j) \cdot d(x(i), x(j)) \cdot a(x(i), x(j))$
s.t.
$|x|=N$
$1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq N$
min_count $(x)=1$
max_count $(x)=1$
Where:
$x(i)$ is the location of station $i$.
$a(i, j)=\left\{\begin{array}{c}\text { if }(i>j), 1 \\ 0, \text { otherwise }\end{array}\right.$
$N$ is the number of stations / locations.
$d(i, j)$ is the distance from location $i$ to $j$.
$w(i, j)$ gives the number of parts that are routed from station $i$ to $j$.

## Personal Time Scheduling Problem

Minimise $\sum_{i=1}^{N} \sum_{j=1}^{|x(i)|} C(x(i, j), j)$
s.t.

$$
\sum_{i=1}^{|x(j)|-1} \sum_{k=i+1}^{|x(j)|} a(x(j, i), x(j, k))=0 \quad \forall j \quad 1 \leq j \leq N
$$

$$
|x|=N
$$

| $\|x(i)\| \geq 1$ | $\forall i$ | $1 \leq i \leq N$ |
| :--- | :--- | :--- |
| $1 \leq x(i, j) \leq M$ | $\forall i$ | $1 \leq i \leq N$ |
|  | $\forall j$ | $1 \leq j \leq \leq x(i) \mid$ |

min_count $(x)=1$
max_count $(x)=1$
Where:
$C(i, j)$ is the cost of agent $i$ performing overtime on job $j$.
$a(i, j)$ is 1 if job $i$ and $j$ clash, 0 otherwise.
$N$ is the number of agents.
$M$ is the number of jobs.

## Number Partitioning Problem

```
Minimise \(A B S\left(\sum_{i=1}^{|x(1)|} a(x(1, i))-\sum_{i=1}^{|x(2)|} a(x(2, i))\right)\)
s.t.
    \(|x|=2\)
    \(|x(i)|>\)
    \(1 \leq x(i, j) \leq N\)
        \(\forall i \quad 1 \leq i \leq 2\)
        \(\forall i \quad 1 \leq i \leq 2\)
        \(\forall j \quad 1 \leq j \leq|x(i)|\)
    min_count \((x)=1\)
    \(\max \_c o u n t(x)=1\)
```

Where:
$x(i, j)$ is the $j$ 'th element of the set of real numbers in the $i$ 'th partition.
$a(i)$ is the $i^{\prime}$ th real number in a sequence between 0 and 1 .
$N$ is the number of terms in the sequence.

## Linear Ordering Problem

Maximise $\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} C(x(i), x(j))$
s.t.
$|x|=N$
$1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq N$
min_count $(x)=1$
max_count $(x)=1$
Where:
$x(i)$ is the $i$ th ordered item.
$N$ is the number of items.
$C(i, j)$ is the cost of ordering item $i$ before $j$.

## FPGA Placement Problem

## Minimise

$$
\begin{aligned}
& \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w(i, j) \cdot(A B S(p(x(i), 1)-p(x(j), 1))+A B S(p(x(i), 2)-p(x(j), 2))) \\
& \text { s.t. } \quad \begin{array}{l}
|x|=N \\
\quad \begin{array}{l}
1 \leq x(i) \leq n \\
\text { max_count }(x)=1
\end{array} \quad \forall i \quad 1 \leq i \leq N
\end{array}
\end{aligned}
$$

Where:
$x(i)$ is the placement of block $i$.
$N$ is the number of blocks.
$n$ is the number of available co-ordinates on the grid.
$w(i, j)$ is the weight between block $i$ and $j$.
$p(i, j)$ represents the $j$ 'th component of the $i$ 'th co-ordinate. The set of co-ordinates on a two dimensional grid is represented as a vector. $j=1$ represents the horizontal component and $j=2$ represents the vertical component. This is similar to the tuples array in the time tabling problem (TTP).

## MCP

Maximise $|x|$
s.t.

$$
\begin{aligned}
& \sum_{i=1}^{|x|-1} \sum_{j=i+1}^{|x|} e d g e(x(i), x(j))=\frac{|x|^{2}-|x|}{2} \\
& 1 \leq|x| \leq N \\
& 1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq|x| \\
& \max _{-} \operatorname{count}(x)=1
\end{aligned}
$$

Where:
$x(i)$ is the $i$ 'th node assigned to the clique.
$N$ is the number of nodes.
$e d g e(i, j)$ is 1 if there is an edge between nodes $i$ and $j$.

## Machine Scheduling / Job Sequencing Problem

Minimise $\sum_{i=1}^{N} w(x(i)) \cdot \max \left(0, \sum_{j=1}^{i} p(x(j))-d(x(i))\right)$
s.t.
$|x|=N$
$1 \leq x(i) \leq N \quad \forall i \quad 1 \leq i \leq N$
min_count $(x)=1$
$\max \_\operatorname{count}(x)=1$
Where:
$N$ is the number of jobs.
$w(i)$ is the tardiness penalty of job $i$.
$p(i)$ is the processing time of job $i$.
$d(i)$ is the due date of job $i$.

## N Queens Problem

Minimise

$$
\sum_{i=1}^{n-1} \sum_{j=i+1}^{n}\left(\sum_{k=1}^{2} s(p(x(i), k), p(x(j), k))+s(A B S(p(x(i), 1)-p(x(j), 1)), A B S(p(x(i), 2)-p(x(j), 2)))\right)
$$

s.t.

$$
\begin{aligned}
& |x|=n \\
& 1 \leq x(i) \leq N \\
& \max \text { count }(x)=1
\end{aligned} \quad \forall i \quad 1 \leq i \leq n
$$

Where:
$n$ is the number of queens.
$N$ is the number of available co-ordinates on the chessboard.
$p(i, j)$ represents the $j$ 'th component of the $i$ 'th co-ordinate. The set of co-ordinates on the chessboard is represented as a vector. $j=1$ represents the horizontal component and $j=2$ represents the vertical component.
$s(i, j)=\left\{\begin{array}{c}1, \text { if }(i=j) \\ 0, \text { otherwise }\end{array}\right.$

## Cutting Stock Problem

Minimise $\sum_{i=1}^{M} w(x(i))$
s.t.

| $o c c(x, x(i)) \cdot P(k, x(i)) \geq R(k)$ | $\forall k$ | $1 \leq k \leq L$ |
| :--- | :--- | :--- |
|  | $\forall i$ | $1 \leq i \leq M$ |
| $\|x\|=M$ |  |  |
| $1 \leq x(i) \leq N$ | $\forall i$ | $1 \leq i \leq M$ |

Where:
$M$ is the number of boards.
$N$ is the number of cutting configurations.
$L$ is the number of shelf sizes.
$P(i, j)$ is the number of shelves of size $i$ in configuration $j$.
$R(i)$ is the required number of shelves of size $i$.

## Appendix C: 0-1 ILP formulations of the Test Problems

This appendix contains the 0-1 ILP formulations of the problems in Table 5. Note:

- (a) refers to the formulation used to test the problems in this study while (b) is the alternative formulation.
- the various formulations for USApHMP and UMApHMP are contained in Ernst and Krishnamoorthy (1996a, 1996b, 1997b).
- a 0-1 ILP model could not be formulated for TTP or BIN (a).


## CSP

Minimise $\sum_{i=1}^{M} \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} y_{i j k} P_{i, k-j}$
s.t.

$$
\begin{aligned}
& \sum_{j=1}^{N} x_{i j}=D_{i} \quad \forall i \quad 1 \leq i \leq M \\
& \sum_{i=1}^{M} x_{i j}=1 \quad \forall j \quad 1 \leq j \leq N \\
& x_{i j}+x_{i k} \leq y_{i j k}+1 \quad \forall i \quad 1 \leq i \leq M \\
& 1 \leq j \leq N-1 \\
& j+1 \leq k \leq N \\
& x_{i j} \in\{0,1\} \quad \forall i \forall j \quad 1 \leq i \leq M \quad 1 \leq j \leq N \\
& \begin{array}{lllll}
y_{i j k} \in\{0,1\} & \forall i & 1 \leq i \leq M & 1 \leq j \leq N-1 & j+1 \leq k \leq N
\end{array}
\end{aligned}
$$

Where:
$x_{i j}$ is 1 if the $j^{\prime}$ th car is model $i, 0$ otherwise.
$P_{i j}$ is the separation penalty for the $j$ 'th model separated by $i$ places in the sequence.
$N$ is the number of cars.
$M$ is the number of models.
$D_{i}$ is the number of cars of model $i$ in the sequence.

QAP
Minimise $\quad \sum_{i=1}^{N} \sum_{a=1}^{N} \sum_{j=1}^{N} \sum_{b=1}^{N} C_{i a j b} y_{i a j b}$
s.t.

$$
\sum_{i=1}^{N} x_{i a}=1 \quad 1 \leq a \leq N
$$

$$
\begin{array}{ll}
\sum_{a=1}^{N} x_{i a}=1 & 1 \leq i \leq N \\
x_{i a}+x_{j b} \leq y_{i a j b}+1 & \forall i<j, \forall a \neq b, C_{i a j b} \neq 0 \\
x_{i j} \in\{0,1\} & \forall i \forall j \quad 1 \leq i, j \leq N \\
y_{i a j b} \in\{0,1\} & \forall i<j, \forall a \neq b
\end{array}
$$

Where:
$x_{i j}$ is 1 if facility $i$ is placed at location $j, 0$ otherwise.
$C_{i a j b}$ is the communication cost of assigning facility $i$ to location $a$ and facility $j$ to location $b$.
$N$ is the number of facilities/locations.

## TSP

$\operatorname{Minimise} \sum_{i=2}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} d_{j k} y_{i j(i-1) k}+\sum_{j=1}^{N} \sum_{k=1}^{N} d_{j k} y_{1 j N k}$
s.t.
$x_{i k}+x_{j l} \leq y_{i k j l}+1$ IF $i=N$ THEN $j=1$ ELSE $j=i+1 \quad \forall k \neq l \quad 1 \leq i, j, k, l \leq N$
$\sum_{k=1}^{N} x_{i k}=1 \quad 1 \leq i \leq N$
$\sum_{i=1}^{N} x_{i j}=1 \quad 1 \leq j \leq N$
$x_{i j} \in\{0,1\} \quad 1 \leq i \leq N \quad 1 \leq j \leq N$
$y_{i k j l} \in\{0,1\} \quad$ IF $i=N$ THEN $j=1$ ELSE $j=i+1 \quad \forall k \neq l \quad 1 \leq i, j, k, l \leq N$
Where:
$x_{i j}$ is 1 if city $i$ is the $j$ 'th city visited, 0 otherwise.
$d_{i j}$ is the distance between city $i$ and city $j$.
$N$ is the number of cities.

## GRAPH

(a)

Minimise $\sum_{i=1}^{M} \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} y_{i j k} e d g e_{j k}$

$$
\begin{aligned}
& \sum_{i=1}^{M} x_{i j}=1 \quad \forall j \quad 1 \leq j \leq N \\
& x_{i j}+x_{i k} \leq y_{i j k}+1 \quad \forall i \quad 1 \leq i \leq M \\
& 1 \leq j \leq N-1 \\
& j+1 \leq k \leq N \\
& x_{i j} \in\{0,1\} \quad \forall i \forall j \quad 1 \leq i \leq M \quad 1 \leq j \leq N \\
& y_{i j k} \in\{0,1\} \quad \forall i \quad 1 \leq i \leq M \quad 1 \leq j \leq N-1 \quad j+1 \leq k \leq N
\end{aligned}
$$

Where:
$x_{i j}$ is 1 if vertex $j$ is assigned to colour $i, 0$ otherwise.
$e d g e_{i j}$ is 1 if there is an edge between vertex $i$ and vertex $j$.
$N$ is the number of vertices in the graph. $M$ is the number of colours.
(b)

Minimise $\sum_{k=1}^{K} C_{k}$
s.t.

$$
\begin{array}{ll}
\sum_{k=1}^{K} x_{i k}=1 & 1 \leq i \leq N \\
x_{i k}+x_{j k} \leq 1 & \forall k \quad \forall i, j \text { s.t. an edge } i \rightarrow j \text { in } G \\
\sum_{i=1}^{N} x_{i k} \leq N \cdot C_{k} & 1 \leq k \leq K \\
C_{k} x_{i k} \in\{0,1\} & \forall i \forall k
\end{array}
$$

Where:
$K$ is the maximum number of colours allowed.

## BIN

(b)

Minimise $\sum_{j=1}^{M} c_{j}$
s.t.

$$
\begin{array}{ll}
\sum_{j=1}^{M} x_{i j}=1 & 1 \leq i \leq N \\
\sum_{i=1}^{N} x_{i j} w_{i}<W_{\max } & 1 \leq j \leq M \\
\sum_{i=1}^{N} x_{i j}<N \cdot C_{j} & 1 \leq j \leq M \\
x_{i j} \in\{0,1\} & 1 \leq i \leq N 1 \leq j \leq M
\end{array}
$$

Where:
$x_{i j}$ is 1 if item $i$ is assigned to bin $j, 0$ otherwise. $M$ is the maximum number of bins allowed.

GAP
Minimise $\sum_{i=1}^{N} \sum_{j=1}^{M} c_{i j} x_{i j}$
s.t.

$$
\sum_{i=1}^{N} a_{i j} x_{i j} \leq b_{j} \quad 1 \leq j \leq M
$$

$$
\begin{array}{lc}
\sum_{j=1}^{M} x_{i j}=1 & 1 \leq i \leq N \\
x_{i j} \in\{0,1\} & 1 \leq i \leq N \quad 1 \leq j \leq M
\end{array}
$$

Where:
$x_{i j}$ is 1 if job $i$ is assigned to agent $j, 0$ otherwise.
$c_{i j}$ is the cost of assigning job $i$ to agent $j$.
$a_{i j}$ is resource required by agent $j$ to perform job $i$.
$M$ is the number of agents.
$N$ is the number of jobs.

## MKP

Maximise $\sum_{i=1}^{N} P_{i} x_{i}$
s.t.
$\sum_{j=1}^{N} w_{i j} x_{j} \leq b_{i}$
$x_{i} \in\{0,1\} \quad \forall i \quad 1 \leq i \leq N$
Where:
$x_{i}$ is 1 if project $i$ is included in the project mix, 0 otherwise.
$P_{i}$ is the profit of including project $i$ in the project mix.
$N$ is the total number of projects.
$w_{i j}$ is the number of units of resource $i$ required by resource $j$.
$b_{i}$ is the maximum number of resource units available from resource $i$.

## MCP

(a) Minimise $\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} y_{i j}$
s.t.

$$
\begin{aligned}
& x_{i}+x_{j} \leq y_{i j}+1 \\
& 1 \leq i \leq N-1 \\
& i+1 \leq j \leq N \\
& \text { if }(i, j) \notin e d g e \\
& \sum_{i=1}^{N} x_{i}=M \\
& x_{i} \in\{0,1\} \quad \forall i \quad 1 \leq i \leq N \\
& y_{i j} \in\{0,1\} \quad \forall i, j \quad 1 \leq i \leq N-1 \quad i+1 \leq j \leq N \\
& \text { if }(i, j) \notin e d g e
\end{aligned}
$$

Where:
$x_{i}$ is 1 if node $i$ is included in the clique, 0 otherwise.
$N$ is the total number of nodes.
$M$ is the number of nodes in the clique.
(b) Maximise $\sum_{i=1}^{N} x_{i}$
s.t.

$$
x_{i}+x_{j} \leq 1
$$

$$
1 \leq i \leq N-1
$$

$$
i+1 \leq j \leq N
$$

$$
\text { if }(i, j) \notin e d g e
$$

$$
x_{i} \in\{0,1\} \quad \forall i \quad 1 \leq i \leq N
$$

## GPP

Minimise $\sum_{i=1}^{N} \sum_{j=1}^{N} e d g e_{i j} y_{i j}$
s.t.

$$
\begin{array}{lll}
x_{1 i}+x_{2 j} \leq y_{i j}+1 & \forall i & 1 \leq i \leq N \\
& \forall j & 1 \leq j \leq N \\
\sum_{i=1}^{N} x_{i j}=\frac{N}{2} & \forall j & 1 \leq j \leq 2 \\
\sum_{i=1}^{2} x_{i j}=1 & \forall j & 1 \leq j \leq N \\
x_{i j}, y_{i j} \in\{0,1\} & \forall i & 1 \leq i \leq N
\end{array}
$$

Where:
$x_{i j}$ is 1 if vertex $i$ is assigned to partition $j, 0$ otherwise.
edge $e_{i j}$ is 1 if there is an edge between vertex $i$ and vertex $j, 0$ otherwise.
$N$ is the number of vertices in the graph.

# Appendix D: Algebraic Language User Manual 

User Manual:

## A Text Based Language for the List Modelling System

## D1.0 Introduction

This manual describes how to compose list-based models of combinatorial optimisation problems for the COSULOM solver package. The system converts this text-based description to a code description that is compatible with COSULOM* ${ }^{*}$. The syntax of the language is similar to that used in GAMS (Brooke, Kendrick, Meeraus and Raman 1997). The main differences occur in relation to the specific list modelling features of the language.

This manual concentrates on the syntax of the sections required to construct a valid problem model (see Section D2.0). As well as this, a technique for separating problem specification and problem data into different files is described.

## D2.0 Writing A Problem Description

A description of the list model of a combinatorial optimisation problem is divided into a number of distinct sections. These sections describe the problem data and the problem specifications.

Throughout this manual, a model of a small multiple knapsack problem (Beasley and Chu 1997) is developed. The list model is:

$$
\left.\begin{array}{lll}
\text { Maximise } & \sum_{i=1}^{|x|} c(x(i)) & \\
\text { s.t. } & \sum_{i=1}^{|x|} a(x(i), j) \leq b(j) & \forall j \\
& 1 \leq j \leq M \\
& 0 \leq|x| \leq N \\
1 \leq x(i) \leq N \\
\text { max_count }(x)=1
\end{array}\right) \forall i \quad 1 \leq i \leq x \mid
$$

[^17]Where:
$x(i)$ is the $i$ 'th project in the project mix.
$c(i)$ is the profit of including project $i$ in the project mix.
$a(i, j)$ is the number of units of resource $j$ required by project $i$.
$b(j)$ is the maximum number of resource units available from resource $j$.
$N$ is the number of projects.
$M$ is the number of different resources.

For the following sections, the conventions that are used to explain syntax are as follows:
bold type: indicates reserved words, such as SCALAR, SETS and ;
italic type: $\quad$ indicates user-defined names and data, such as $N, A(N)$ and 34

Examples of each of the sections are shown in courier font.

## D2.1 Description Sections

## D2.1.1 SCALAR

The SCALAR section defines the constants that are used in the problem model. Scalars can contain either integer or floating point values. Typically the entries in the SCALAR section are related to the size of the problem. The syntax of the SCALAR section is:

## SCALAR

scalar_name_1/number/
scalar_name_n/number/;

An example for the multiple knapsack problem is:

```
SCALAR
```

    \(\mathrm{N} / 10 /\)
    M / 4 /;
    
## D2.1.2 SETS

The SETS section is used to define range variables. Range variables are a useful means for defining the lower and upper bounds on a summation operation or to define a group of similar constraints (see Section D2.16). The syntax of the SETS section is:

## SETS

```
set_name_1/number - number /
set_name_n/number - number /;
```

In the above, number, may be either from the scalar list or an integer value. An example for the multiple knapsack problem is:

SETS

```
J / 1 - M /;
```


## D2.1.3 PARAMETER

The PARAMETER section allows vectors of values to be defined. Like SCALAR, values may be either integer or floating point. Integer and floating point values can also be mixed within the one vector. For each vector definition, a name and the dimension of the vector are required. The syntax of the PARAMETER section is:

## PARAMETER

parameter_name_I(N)/item(1) ... item $(N) /$
parameter_name_n $(N) / \operatorname{item}(1) \ldots \operatorname{item}(N) /$;

An example for the multiple knapsack problem is:

## PARAMETER

```
C(N) / 12 10 20 24 9 17 16 8 15 7 /
B(M) / 17 19 25 22 /;
```


## D2.1.4 TABLE

The TABLE section allows matrices of values to be specified, in much the same way as in the PARAMETER section. The syntax of the TABLE section is as follows:

## TABLE

table_name_l $(M, N) / \operatorname{item}(1,1) \ldots \operatorname{item}(1, N) \ldots \operatorname{item}(M, N) /$
table_name_n $(M, N) / \operatorname{item}(1,1) \ldots$ item $(1, N) \ldots$ item $(M, N) /$;

An example for the multiple knapsack problem is:

TABLE

$\begin{array}{llllllllll}2 & 3 & 1 & 8 & 4 & 7 & 2 & 3 & 5 & 4\end{array}$
$\begin{array}{llllllllll}1 & 5 & 6 & 3 & 2 & 7 & 5 & 4 & 10 & 1\end{array}$
$\begin{array}{llllllllll}2 & 3 & 3 & 4 & 2 & 7 & 5 & 6 & 1 & 5\end{array} /$

## D2.1.5 SOLUTION

The SOLUTION section is used to define the list constraints of the problem model. These include the Size, Count and Value Range constraints (see Section 3.3.1 of the thesis). The syntax of the SOLUTION section is:

## SOLUTION

SIZE $(x)$ relational_operator number
MIN_COUNT $(x)=$ number
MAX_COUNT $(x)=$ number
lowerbound relational_operator x relational_operator upperbound;

The SIZE statement is used to set the bounds of the list (i.e. the number of sub-lists and the number of elements on those sub-lists). For instance, if the model requires $M$ sub-lists, $\operatorname{SIZE}(X)=M$ is used. The relational_operator is from the set $\{<=,<,=,>\rangle=$,$\} . To denote the size of an individual sub-list, a$ statement like $\operatorname{SIZE}(X(1))<=N$ is used. This indicates that sub-list 1 on list $X$ can contain no more than $N$ elements. Should SIZE be undefined, there are no limits on the number of sub-lists or the number of elements that can be contained on the sub-lists.

The last line of the syntax description defines the range of legal values that can occur on the list. This statement cannot be omitted.

An example for the multiple knapsack problem is:

```
SOLUTION
    SIZE(X)=1
    MAX_COUNT (X)=1
    1<=X<=N;
```


## D2.1.6 EQUATIONS

The EQUATIONS section is used to describe the objective cost, incremental cost expressions and the constraints of the problem. The syntax of the EQUATIONS section is:

## EQUATIONS

COST.. expression
ICOST(transition_operator): ic_expression
constraint_name_1: expression relational_operator expression
constraint_name_n: expression relational_operator expression;

COST.. indicates the objective cost. The objective cost is an arbitrary algebraic expression that can consist of the following functions:

Function name and arguments
Description

| Function name and arguments | Description |
| :---: | :---: |
| SIZEOF (i) | The size of list i. Returns lil. |
| SUM (expression, $i=j, k$ ) | The sum of an expression between a lower and an upper bound. Returns $\sum_{i=j}^{k}$ expression. |
| ABS (i) | Returns the absolute value of $i$. |
| minimum (expression_1, expression_2) | Returns the value of smallest expression of expression_1 and expression_2. |
| maximum(expression_1,expression_2) | Returns the value of largest expression of expression_1 and expression_2. |
| $\min ($ expression, $i, j, k)$ | Returns the expression of least value between a lower and upper bound. i.e. $\min (\underset{i=j}{\operatorname{expression}})$. |
| max(expression, $i, j, k$ ) | Returns the expression of greatest value between a lower and upper bound. i.e. <br> $\max \left({ }^{k}{ }^{k}\right.$ ression). <br> $i=j$ |
| occ ( $\mathrm{x}, \mathrm{i}$ ) | Returns the number of times that value $i$ occurs on list x . |
| list(x, i, j) | Returns the sub-list number of the $j$ 'th occurrence of the value $i$ on list $x$. |
| pred(i,j,k,l) | Returns $i-1$ unless $i-1<j$ at which $k$ is returned. |
| $\operatorname{succ}(i, j, k, 1)$ | Returns $i+1$ unless $i+1>j$ at which $k$ is returned. |

The system can generate its own incremental cost expressions from the objective function. However the user can provide their own for particular transition operators using ICOST statements. transition_operator is from the set \{move, swap, invert, reposition, add, drop, change\}. ic_expression contains special incremental cost constants as well as all of the functions used in expression (explained in Section 4.5 of the thesis). These are:

| Constant name and arguments | Description |
| :--- | :--- |
| COST_OLD | Returns the cost of the objective function before the transition was |
| E1 | made |
| E2 | Element 1 |
| L1 | Element 2 |
| L2 | Sub-list 1 |
| P1 | Sub-list 2 |
| P2 | Position 1 |

The last part of the EQUATIONS section concerns the constraints of the problem. Each constraint is given an individual name. If a range variable (from the SETS section) is used, it is possible to create a number of constraints. This is often useful when the same constraint is associated with some or all of the sub-lists. An example of an EQUATIONS section for the multiple knapsack problem is:

ICOST (CHANGE) :
CAPACITY:

```
COST_OLD - C(E1) + C(E2)
```

```
COST_OLD - C(E1) + C(E2)
```

$\operatorname{SUM}(\bar{A}(J, X(I))<=B(J) ;$

## D2.1.7 SOLVE

The SOLVE statement is used to specify which search engine is to be used. The syntax is:

## SOLVE USING engine

engine can be any of the following:

SA a simulated annealing search engine
TS a tabu search engine
GS a greedy search engine

## D2.2 Complete Example File

The complete description file for the multiple knapsack problem (as used above) is:

```
SCALAR
    N / 10 /
    M / 4 /;
SETS
    J / 1 - M /;
PARAMETER
    C(N) / 12 10 20 24 9 17 16 8 15 7/
    B(M) / 17 19 25 22 /;
TABLE
    A(M,N) / 3 9 % 7 5 4 4 4 8 6 5 2 
        2
        1 5}506\mp@code{3
        2 3 3 4 2 7 5 6 1 5 /;
SOLUTION
    SIZE (X)=1
    MAX_COUNT (X)=1
    1<=X<=N;
EQUATIONS
    COST.. SUM(C(X(I)),I=1,SIZEOF(X(1))
    ICOST(ADD): COST_OLD + C(E1)
    ICOST(DROP): COST_OLD - C(E1)
    ICOST(CHANGE): COST_OLD - C(E1) + C(E2)
    CAPACITY: SUM(A(J,X(I))<=B(J);
SOLVE USING TS
```


## D2.2 File Construction

The preprocessing directive \#include (as commonly found in C applications) can be used in the language. The greatest benefit in this context is to separate data from the problem specification. Using
this approach, one specification file can be used for multiple data files (i.e. many instances of the same problem). The syntax of the \#include directive is:

## \#include "filename"

filename is the name of the file that is to be included. An example using the multiple knapsack problem is given below. The first file is called "mkp_problem_data" and contains the problem data. The second is "mult_knapsack" and contains the specification of the list model, objective cost, incremental cost expressions and constraints.

The file "mkp_problem_data":

```
SCALAR
    N / 10 /
    M / 4 /;
SETS
    J/1 - M/;
PARAMETER
    C(N) / 12 10 20 24 9 17 16 8 15 7/
    B(M) / 17 19 25 22 /;
TABLE
    A(M,N) / 3 9 7 54448 6 5 2
        2
        1
        2 3 3 4 2 7 5 6 1 5 /;
```

The file "mult_knapsack":

```
#include "mkp__problem_data"
SOLUTION
    SIZE(X)=1
    MAX_COUNT (X)=1
    1<=X<=N;
EQUATIONS
    COST.. SUM(C(X(I)),I=1,SIZEOF(X(1))
    ICOST (ADD) :
    ICOST(DROP):
    ICOST (CHANGE) :
    CAPACITY: SUM(\overline{A}(J,X(I))<=B(J);
SOLVE USING TS
    COST_OLD + C(E1)
    COSI(CHANGE): COST_OLD - C(E1) + C(E2)
```


## References

Beasley, J. and Chu, P. (1997) "A Genetic Algorithm for the Multiconstraint Knapsack Problem", Working Paper.

Brooke, A., Kendrick, D., Meeraus, A. and Raman R. (1997) GAMS Language Guide, GAMS Development Corporation, 305 pages.

## Appendix E: Overview of the $C$ Problem Description Files

COSULOM uses two C files in order to represent a list problem model. These are named "user.c" and "user.h". The first of these files represents the list constraints, objective function, incremental cost expressions and constraints. "user.h" contains the data for a particular problem instance. Both of the files are described in Section E1.0 and E2.0 respectively. By way of example, the multiple knapsack problem instance as given in Appendix D, is used to illustrate these two files (Section E3.0).

## E1.0 "user.c"

The files "misc.h", "list.h" and "user.h" must be included within "user.c". "list.h" and "misc.h" contain the definition of list structures and some necessary function prototypes. "user.c" consists of four functions:

- user_define_solution
- user_evaluate_cost
- user_evaluate_incremental_cost
- user_evaluate_constraints

These are described below.

## E1.1 user_define_solution

This function is used by the system to create an appropriate list structure for the solution of a particular problem. As such, the list constraints are specified in this function. The list constraints themselves are implemented by calling the system functions described below.

The function prototype of user_define_solution is:

```
void user_define_solution(solution_list_type)
```

solution_list_type is the $C$ type that describes the list structure of the solution.

## E1.1.1 set_min_count and set_max_count

These functions are used to specify the Count constraints. The function prototypes are:

```
void set_min_count(solution_list_type)
void set_max_count(solution_list_type)
```

The syntax for each function is:

```
set_min_count (solution)
set_max_count (solution)
```

solution is a variable name used by the system to store the current solution state. For this implementation, min_count and max_count are set to 1 if the appropriate function is called.

## E1.1.2 set_size_bounds

This function is used to specify the shape and size of the solution list structure. The function prototype is:

```
void set_size_bounds(solution_list_type, int, int, int)
```

The syntax is:

```
set_size_bounds(solution,list_identifier, size_lowerbound, size_upperbound)
```

list_identifier is used to denote the list while size_lowerbound and size_upperbound set the limits of how small and how large a list can become respectively. To specify the list of lists (i.e. the list that supports the sub-list structures), 0 is used. For instance, to specify the structure of an M agent GAP (i.e. there are $M$ sub-lists), the following would be used:

```
set_size_bounds(solution, 0,M,M)
```

As many problems such as the travelling salesman problem, quadratic assignment problem and multiple knapsack problem have only one sub-list, set_size_bounds (solution, $0,1,1$ ) is used. Sub-lists are numbered from 1 onwards.

## E1.1.3 set_value_bounds

This function specifies the range of values for the list elements. The function prototype is:

```
void set_value_bounds(solution_list_type, int, int)
```

The syntax is:

```
set_value_bounds(solution, value_lowerbound, value_upperbound)
```

value_lowerbound and value_upperbound set the limits of the value range. For the multiple knapsack problem, this becomes:

```
set_value_bounds(solution,1,N)
```


## E1.1.4 A Complete user_define_solution Function

The complete function for the multiple knapsack problem becomes:

```
void user_define_solution(solution_list_type solution)
{
    set_max_count(solution);
    set_size_bounds(solution,0,1,1);
    set_value_bounds(solution,0,1,N) ;
}
```


## E1.2 user_evaluate_cost

This function specifies the objective function of the combinatorial optimisation problem. The function prototype is:

```
float user_evaluate_cost(solution_list_type)
```

Within this function, arbitrary C statements can be used in order to calculate the value of the objective function. This usually involves the data (constants and arrays) declared in "user.h" (see Section E2.0). The following system functions are available to user_evaluate_cost (as well other C functions).

| Function Prototype | Function Syntax | Description |
| :---: | :---: | :---: |
| ```int find_value_at( solution_list_type, int, int) int get_sizeof_list( solution_list_type, int) float fmin(float, float) float fmax(float, float) int occ( solution_list_type, int) int list( solution_list_type, int, int) int pred(int, int, int, int) int succ(int, int, int, int)``` | ```find_value_at(solution , sub_list_number. position_number) get_sizeof_list( solution, list_number) fmin(i,j) fmax(i,j) occ(x,i) list (x,i,j) pred (i,j,k,l) succ}(i,j,k,l``` | Returns the value of the element on the sub_list_number sub-list at the position_number position. <br> Returns the length of list list_number. <br> Returns the minimum of $i$ and $j$. <br> Returns the maximum of $i$ and $j$. <br> Returns the number of times that value $i$ occurs on list $x$. Returns the sub-list number of the $j$ 'th occurrence of the value $i$ on list $x$. <br> Returns $i-l$ unless $i-l<j$ at which $k$ is returned. <br> Returns $i+l$ unless $i+l>j$ at which $k$ is returned. |

The complete function for the multiple knapsack problem becomes:

```
float user_evaluate_cost(solution_list_type solution)
{
    float sum=0.0;
    int i;
    for(i=1;i<=get_sizeof_list(solution,1);i++)
    {
            sum+=c[find_value_at(solution,1,i)];
    }
    return sum;
}
```


## E1.3 user_evaluate_incremental_cost

This function is used to calculate the incremental cost resulting from the local search transition made by the system. The function prototype is:

```
float user_evaluate_incremental_cost(solution_list_type,
last_transition_type, float)
```

The syntax is:
user_evaluate_incremental_cost (solution, last_transition, cost_old)
last_transition_type is a C struct in which the information about the last transition is recorded:

```
struct last_transition_struct
{
    transit_type transition;
    int element1;
    int list1;
    int position1;
    int element2;
    int list2;
    int position2;
};
typedef struct last_transition_struct last_transition_type;
transit_type is given by:
enum tran{move_maint, move_rest, swap, invert, reposition, change,
add, drop};
typedef enum tran transit_type;
```

The use of the members of last_transition_type is similar to that specified in Section 4.5. A switch statement is the most efficient means of selecting an appropriate incremental cost expression to evaluate. The complete function for the multiple knapsack problem becomes:

```
float user_evaluate_incremental_cost(solution_list_type solution,
last_transition_type last__transition, float cost_old)
{
    float cost_new;
    switch(last_transition.transition)
    {
        case add:
            cost_new=cost_old+c[last_transition.element1];
            break;
        case drop:
            cost_new=cost_old-c[last_transition.element1];
            break;
        case change:
            cost_new=cost_old-c[last_transition.element1]
            +c[last_transition.element2];
            break;
    }
    return cost_new;
}
```

If incremental cost expressions are not available for a particular objective function, user_evaluate_cost can be called from this function. For example:

```
float user_evaluate_incremental_cost(solution_list_type solution,
last_transition_type last_transition, float cost_old)
{
    return user_evaluate_cost(solution);
}
```


## E1.4 user_evaluate_constraints

This function is used to specify the constraints of the problem model. The function prototype is:

```
float user_evaluate_constraints(solution_list_type, int)
```

The syntax is:

```
user_evaluate_constraints(solution, constraint_number)
```

The function returns the amount of constraint violation for the constraint given by constraint_number. Constraint violation is calculated according to Equations (44-49). The complete function for the multiple knapsack problem becomes:

```
float user_evaluate_constraints(solution_list__type solution, int
constraint)
{
    float sum=0.0;
    int i;
    for(i=1;i<=get_sizeof_list(solution,1);i++)
    {
        sum+=a[find_value_at(solution,1,i)][constraint];
    }
    return fmax(0.0,sum-b[constraint]);
}
```

If the problem model contains no problem constraints, the following should be used:

```
float user_evaluate_constraints(solution_list_type solution, int
constraint)
{
    return 0.0;
}
```


## E2.0 "User.h"

This file stores the constants and arrays necessary to describe a problem instance. To declare the constants, the standard \#define directive is used. One constant is required to be present. This is num_problem_constraints that tells the system how many problem constraints the problem model has. For the multiple knapsack problem, it can be declared as:

```
#define num_problem_constraints M
```

provided that $M$ has been previously declared. Arrays are declared and initialised in the customary $C$ manner.

## E3.0 Complete Example Files for the Multiple Knapsack Problem Instance

The file "user.c" is:

```
/****************************************************
* user.c for the MKP
* Written by Marcus Randall
**************************************************/
#include <stdio.h>
#include <stdlib.h>
#include "misc.h"
#include "list.h"
#include "user.h"
void user_define_solution(solution_list_type solution)
{
    set_max_count(solution);
    set_size_bounds(solution,0,1,1);
    set_value_bounds(solution,0,1,N);
}
float user_evaluate_cost(solution_list_type solution)
{
    float sum=0.0;
    int i;
    for(i=1;i<=get_sizeof_list(solution,1);i++)
    {
        sum+=c[find_value_at(solution,1,i)];
    }
    return sum;
```

```
}
float user_evaluate_incremental_cost(solution_list_type solution,
last_transition_type last_transition,float cost_old)
    float cost_new;
    switch(last_transition.transition)
    {
        case add:
        cost_new=cost_old+c[last_transition.element1];
        break;
        case drop:
        cost_new=cost_old-c[last_transition.element1];
        break;
        case change:
        cost_new=cost_old-c[last_transition.element1]
        +c[last_transition.element2];
        break;
    }
    return cost_new;
}
float user_evaluate_constraints(solution_list_type solution,int
constraint)
{
    float sum=0.0;
    int i;
    for(i=1;i<=get_sizeof_list(solution,1);i++) Continued...
    {
        sum+=a[find_value_at(solution,1,i)][constraint];
    }
    return fmax(0.0,sum-b[constraint]);
}
```

The file "user.h" is:

```
/***************************************************
* user.h for an MKP instance
* Written by Marcus Randall
*************************************************/
#define N 10
#define M 4
#define num_problem_constraints M
int c[N]=
{
    12, 10, 20, 24, 9, 17, 16, 8, 15, 7
},
b}[M]
{
    17, 19, 25, 22
},
a[M][N]=
{
    3, 9, 7, 5, 4, 4, 8, 6, 5, 2,
    2, 3, 1, 8, 4, 7, 2, 3, 5, 4,
    1, 5, 6, 3, 2, 7, 5, 4, 10, 1,
    2, 3, 3, 4, 2, 7, 5, 6, 1, 5
};
```


# Appendix F: "A Simulated Annealing Code for General Integer Linear Programs" 

The following is a pre-print of the paper "A Simulated Annealing Code for General Integer Linear Programs" by D. Abramson and M. Randall. This paper is to appear in the journal The Annals of Operations Research.
[image removed]

## Appendix G: Papers Arising from this Study

1. Abramson, D., de Silva, A., Randall, M. and Postula, A. (1995) 'Special Purpose Computer Architectures for High Speed Optimisation", Proceedings of the Second Australasian Conference on Parallel and Real Time Systems Conference, pp. 13-20.
2. Abramson, D., Logothetis, P., Postula, A. and Randall, M. (1997) "Application Specific Computers for Combinatorial Optimisation" Proceedings of the Australian Computer Architecture Workshop, February, Sydney.
3. Abramson, D. and Randall, M. (1998) "A Simulated Annealing Code for General Integer Linear Programs" to appear in Annals of Operations Research.
4. Abramson, D., Logothetis, P., Postula, A. and Randall, M. (1998) "FPGA Based Custom Computing Machines for Irregular Problems", Proceedings of the Fourth International Symposium on High-Performance Computer Architecture, (HPCA98), February 1-4, Las Vegas, Nevada.
5. Randall, M. and Abramson, D. (1998) "An Empirical Study of State Encoding in Tabu Search", in preparation (for journal publication).
6. Randall, M. and Abramson, D. (1998) "A General Meta-heuristic Based Solver for Combinatorial Optimisation Problems", in preparation (for journal publication).

7 Randall, M. and Abramson, D. (1998) "A General Parallel Tabu Search Algorithm for Combinatorial Optimisation Problems", in preparation (for journal publication).
8. Abramson, D., Logothetis, P., Randall, M. and Postula, A. (1998) "A Tail Of $2^{n}$ Cities: Performing Combinatorial Optimisation Using Linked Lists On Special Purpose Computers", keynote address, Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, February, Gipsland, pp. 17-45.


[^0]:    ${ }^{1}$ See Appendix $F$ for a preprint of this paper.

[^1]:    2 A subset of $N(x), N_{s}(x) \quad\left(N_{s}(x) \subseteq N(x)\right)$ can be used to represent $N(x)$. This is done if the problem is large and subsequently $N(x)$ has many elements (Glover 1989 and Hertz, Taillard and de Werra 1997). There is a variety of neighbourhood sampling methods and these are referred to as candidate list structures (Glover and Laguna 1997).

[^2]:    ${ }^{3}$ For the sake of brevity, problem models that use $\min \_c o u n t(x)=0$ and max_count $(x)=1$ are expressed using only max_count $(x)=1$. Similarly, problems that require the constraints min_count $(x)=1$ and max_count $(x)=$ unbounded are expressed using only min_count $(x)=1$.

[^3]:    ${ }^{4}$ There are also other possible models for determining which transition operator is to be used at a particular transition. One possibility is to use a round-robin approach in which each operator is used in turn, though other models are possible. Alternatively, the neighbourhood of each possible operator could be examined to determine which operator yields the most effective solution. Glover and Laguna (1997) discusses this concept in some detail.

[^4]:    ${ }^{5}$ According to Connolly (1990, p. 96) Q8-7 derives its name "from the fact that it was the seventh modification of the eigth annealing scheme tested on the QAP by the author".

[^5]:    "The tabu tenure is varied between 1 and the tabu list size throughout the search process. This is done so as to introduce some element of intensification and diversification into the search (Glover and Laguna 1997. p.48).

[^6]:    ${ }^{7}$ The sequential SA, TS and GS runs use only a single RS6000 processor.

[^7]:    ${ }^{8}$ Problems gap 1-1-gap8-1 are maximised while gapA5-100-gapA20-200 are minimised. This is to keep the results consistent with Chu and Beasley (1997).

[^8]:    ${ }^{9}$ Setting the size of the tabu list to an indicator of problem size has been used with some success by Taillard (1991, 1997). For instance: the tabu list size for gapl-1 ( 15 jobs) becomes 15 , and hk48 ( 48 cities) is 48 .

[^9]:    ${ }^{10}$ Runtimes are rounded to the nearest second due to the small variability in the SP2 timimg mechanism. For instance, a runtime of 0.35 seconds is rounded down to 0 .

[^10]:    ${ }^{11}$ This solver runs 20 sequential SA searches. See Smith et al. (1996b).

[^11]:    ${ }^{12}$ These programs record an optimal of 4 for johnson8-2-4. However the general solver, LINDO and OSL all record a value of 3 for the same problem.

[^12]:    ${ }^{13}$ Post-hoc analysis was used to determine whether there were any significant performance differences between the transition operator sets for each problem type. However, as it is difficult to construct overall ranks from this information, the Kruskal Wallis median ranks were also taken into consideration.

[^13]:    ${ }^{14}$ The rank of 5 indicates that QAPBB could not solve the problem instance within 6400 seconds.

[^14]:    ${ }^{15}$ This statement neglects differences in the speed of the DEC $3000 / 700$ (used by Ernst et al. (1997b)) and the RS6000 processor (as used in this study). However, they are comparable processors.

[^15]:    ${ }^{16}$ All run lengths for esc64a, sko72 and will00 were too long for the time available for the standard runs. These problem instances appear in the Extended Runs instead.

[^16]:    ${ }^{17}$ All run lengths for kroA100, ch130 and a280 were too long for the time available for the standard runs. These problem instances appear in the Extended Runs instead.

[^17]:    * Note: Parameters such as SA cooling length, tabu list size and transition operator probabilities are set at runtime rather than in the problem description itself.

