Guided Local Search for Combinatorial Optimisation Problems

Christos Voudouris

A thesis submitted for the degree of Ph.D

Department of Computer Science

University of Essex

1997

To my wife, Stella

Acknowledgements

I would like to express my gratitude to my supervisor Edward Tsang for his constant support and guidance throughout my years at Essex University. He introduced me to the fields of constraint satisfaction and combinatorial optimisation and he enormously helped in the evolution and refinement of the ideas presented in this thesis by his constant feedback and supervision. I would like to thank my manager at BT Laboratories Nader Azarmi, without his encouragement and support this thesis may have never been completed. Many thanks to John Ford for his comments on an earlier draft of this thesis and for supervising me during my last months of study.

I would like to thank the Department of Computer Science for the harmonious environment and the Computing Service at the University of Essex for the excellent computer facilities without which it would have been impossible to perform the computational experiments reported here. Many thanks to BT plc which supported me during my final year.

This research has been conducted in the framework of the GENET project funded by the EPSRC grant (GR/H75275).

Abstract

In this thesis, we present the heuristic technique of Guided Local Search for combinatorial optimisation problems. The technique sits on top of local search procedures and has as a main aim to guide these procedures in exploring efficiently and effectively the vast search spaces of combinatorial optimisation problems. This is achieved by exploiting prior information known about the problem in conjunction with historical information gathered during the search process. Information is converted to constraints which take the form of penalties and modify the cost function to be minimised. Local search is guided by these constraints, focusing on solutions of high quality. Guided Local Search can be combined with the neighbourhood reduction scheme of Fast Local Search which significantly speeds up the operations of the algorithm.

In this thesis, Guided Local Search is applied to the Travelling Salesman Problem, Quadratic Assignment Problem, Radio Link Frequency Assignment Problem, Workforce Scheduling and Function Optimisation. Experimental evaluation and comparisons with a variety of other heuristic methods on benchmarks instances of the these problems shows that Guided Local Search compares very favourably to famous general and specialised heuristic algorithms outperforming many of them on the benchmark instances considered.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we are going to present a technique called Guided Local Search (GLS) which is suitable for a class of difficult computational problems known as *combinatorial optimisation problems*. In this introductory chapter, we will explain the terminology used in the field, examine combinatorial optimisation problems and outline some of the most popular techniques suggested so far for tackling them.

1.1 Combinatorial Optimisation and NP-Hard Problems

Combinatorial optimisation problems appear in many areas such as resource allocation, routing, packing and scheduling. The objective is that of assigning values to a set of decision variables such that a function of these variables is minimised perhaps in the presence of some constraints. A combinatorial optimisation problem can be formulated as follows [Ree96]:

Eq. 1.1

$$\textit{minimise } f(\mathrm{x}),\ \mathrm{x} \in \mathrm{X} \subset \mathrm{R}_n$$

$$\textit{subject to } g_i(x) \geq b_i,\ i = 1,\ldots,m.$$

where x is a vector of decision variables and $f(\cdot)$ and $g_i(\cdot)$ are general functions. The condition $x \in X$ is assumed to constrain decision variables to discrete values. Here, we have presumed that the problem is that of minimisation but the modification of Eq. 1.1 for maximisation problems is straightforward.

A class of problems of particular interest in combinatorial optimisation is that of 'hard' combinatorial optimisation problems. This class includes problems famous for their difficulty such as the Travelling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP) and the Vehicle Routing Problem (VRP). Back in the late 60s, many researches recognised the difficulty of such problems and tried to identify whether 'polynomial' algorithms (i.e. algorithms which require a polynomial number of steps) can be devised to solve them. Nobody since then has been able to devise such an algorithm for any of these problems and that despite many man-centuries of research effort invested on the subject by some of the most brilliant researchers. In fact, there seems to be the case that problems such as the TSP are inherently difficult to solve, exhibiting an exponential growth in computing time with the size of the problem.

The hypothesis that no polynomial algorithm exists for solving these problems has been further supported by advances in the field of computational complexity. We briefly describe the findings. The interested reader is referred to classical texts on the subject by Papadimitriou and Steiglitz [PS82] and Garey and Johnson [GJ79] for a more formal and extensive description of these findings.

In brief, problems which have known polynomial algorithms are said to be in the class *P*. A superset of class *P* is the class *NP* where NP stands for "non-deterministic polynomial". NP consists of all problems that can be solved in polynomial time on a *non-deterministic Turing machine*. This includes all problems in P but also 'hard' problems such as the TSP, QAP, VRP, Satisfiability etc. for which all known algorithms require exponential time.

Hard problems can be transformed one to the other in polynomial time. This property has been used to define a separate sub-class in NP that of *NP-complete* problems. All famous hard combinatorial problems such as the TSP, QAP, Satisfiability, Graph Colouring, Graph Partitioning, Vehicle Routing etc. belong to this class (see [CK95] for a comprehensive list of NP-complete problems). If we were to find a polynomial algorithm for any of these problems, we would have found a polynomial algorithm for all problems in NP. No polynomial algorithm has been found so far and that despite considerable efforts and thus it is widely conjectured that no NP-complete problem is polynomially solvable.

There is a subtle difference here. The above results refer to the 'decision' version of combinatorial optimisation problems where the problem is not that of finding the optimal solution but finding an answer to a question such as 'is there a solution with cost less than C?'. It is obvious that an algorithm for the decision version can be used to solve the optimisation version by asking a series of questions to this algorithm. Concluding, optimisation problems as such are not in NP. For the optimisation versions of NP-complete problems, we will use the term *NP-hard*[1] adopted by many authors [RB93]. In addition to that and unless otherwise stated, the terms combinatorial problems and combinatorial optimisation problems will be used to refer to NP-hard optimisation problems.

[1] In general, a problem is said to be NP-hard if any problem in NP is polynomialy transformable to it even if the problem itself is not in NP. If the problem also belongs in NP then it is NP-complete. If you could reduce an NP problem to an NP-hard problem and then solve it in polynomial time, you could solve all NP problems in polynomial time.

1.2 Exact and Heuristic Search Techniques

The simplest approach to solve a NP-hard optimisation problem is to list all the feasible solutions, evaluate their objective function values and chose the best. This approach of *complete enumeration,* although widely applicable, is unusable in practice because of the vast number of possible solutions to any problem of reasonable size.

In the early days of combinatorial optimisation, most of the efforts were focused on Linear Programming (LP). The problem was reformulated by using integer variables usually taking the values 0 or 1 to produce an integer programming (IP) formulation. Such a problem can be then solved by variants of a method generally known as "Branch & Bound" (B&B). Branch and Bound is an efficient enumeration scheme which avoids complete enumeration of solutions by building a search tree of the solutions to be evaluated. This tree is pruned during search, so reducing the number of solutions that need to be evaluated before the optimal solution is found and proved to be optimal.

The worst case computational complexity of IP algorithms grows exponentially with the size of the problem. As a result of that, general IP codes usually do not scale well to large instances of problems. Furthermore, for some problems it is difficult to find an IP formulation and even if one is found it sometimes results in a large number of variables and constraints. IP is much more difficult than LP and that because the problems of concern are NP-hard optimisation problems.

Given the difficulty of NP-hard optimisation problems, many researchers have focused on another class of techniques known as *heuristic techniques* or simply *heuristics*. These techniques sacrifice the proof of optimality for solutions and instead focus on finding good near optimal solutions at a reasonable computational cost. In the early days of Operations Research, heuristics were treated with scepticism.

Nowadays, mainly due to the theoretical developments in computational complexity indicating the inherent difficulty of NP-hard problems, heuristics have gained a prominent position amongst optimisation methods. Following Reeves [Ree96], we give the following general definition for a method to qualify as a heuristic:

***Definition 1-1*:**

A heuristic technique is a method which seeks good (i.e. near optimal solutions) at a reasonable computational cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is.

Despite the rather pessimistic definition, modern heuristics can find high quality solutions for problems many times larger that those solved to optimality by exact search methods. From a historical perspective, the 'gamble' with heuristics has paid off leading to many real world systems tackling NP-hard optimisation problems in resource allocation, routing, scheduling and many other domains. In the rest of the chapter, we examine some the most famous heuristic techniques starting with *Local Search* [PS82] perhaps the oldest heuristic method.

1.3 Local Search

Local Search, also referred to as Neighbourhood Search or Hill Climbing, is the basis of many heuristic methods for combinatorial optimisation problems. In isolation, it is a simple iterative method for finding good approximate solutions. The idea is that of trial and error. For the purposes of explaining local search, we will consider a slightly different definition of a combinatorial problem to that given in Eq. 1.1.

A combinatorial optimisation problem is defined by a pair *(S, g),* where *S* is the set of all feasible solutions (i.e. solutions which satisfy the problem constraints) and *g* is the objective function that maps each element *s* in *S* to a real number. The goal is to find the solution *s* in *S* that minimises the objective function *g*. The problem is stated as:

Eq. 1.2 $$\min g(s), s \in S.$$

In the case where constraints difficult to satisfy are also present, penalty terms may be incorporated in *g*(s) to drive toward satisfying these constraints. A neighbourhood *N* for the problem instance *(S, g)* can be defined as a mapping from *S* to its powerset:

Eq. 1.3 $$N: S \rightarrow 2^S.$$

N(s) is called the *neighbourhood* of *s* and contains all the solutions that can be reached from *s* by a single *move*. Here, the meaning of a move is that of an operator which transforms one solution to another with small modifications. A solution *x* is called a *local minimum* of *g* with respect to the neighbourhood *N* iff:

Eq. 1.4 $$g(x) \leq g(y), \forall y \in N(x).$$

Local search is the procedure of minimising the cost function *g* in a number of successive steps in each of which the current solution *x* is being replaced by a solution *y* such that:

Eq. 1.5 $$g(y) < g(x), y \in N(x).$$

A basic local search algorithm begins with an arbitrary solution and ends up in a local minimum where no further improvement is possible. In between these stages, there are many different ways to conduct local search. For example, *best improvement* (*greedy)* local search replaces the current solution with the solution that improves most in cost after searching the whole neighbourhood. Another example is *first improvement* local search which accepts a better solution when it is found. The computational complexity of a local search procedure depends on the size of the neighbourhood and also the time needed to evaluate a move. In general, the larger the neighbourhood, the more the time one needs to search it and the better the local minima.

Local minima are the main problem with local search. Although these solutions may be of good quality, they are not necessarily optimal. Furthermore if local search gets caught in a local minimum, there is no obvious way to proceed any further toward solutions of better cost. Methods that build on local search to remedy this problem are sometimes referred to as meta-heuristics. One of the first methods in this class is *Repeated Local Search* where local search is restarted from a new arbitrary solution every time it reaches a local minima until a number of restarts is completed. The best local minimum found over the many runs is returned as an approximation of the global minimum. Modern meta-heuristics tend to be much more sophisticated than repeated local search pursuing a range objectives that go beyond simply escaping from local minima. In the following sections, we examine some of the most successful modern meta-heuristic techniques.

1.4 Simulated Annealing (SA)

Simulated Annealing (SA) is perhaps the most widely used meta-heuristic. Mainly because of its simplicity, SA has attracted the interest of many researchers and practitioners from a wide range of disciplines. The technique has its origins in statistical mechanics and it was inspired by the physical process of annealing used for the “cooling” of solids such that they form perfect crystals. Metropolis et al. [MRRTT53] first described an algorithm for simulating the annealing process.

Kirkpatrick et al. [KGV83] proposed the use of this simulation algorithm for searching the solutions of a combinatorial problem.

SA could be described as a randomised scheme which reduces the risk of getting trapped in local minima by allowing moves to inferior solutions. Given the neighbourhood *N*(s) of a combinatorial problem, moves are randomly selected from this set. A move from a solution s to solution s' is only accepted if:

- s' is better than s or
- s' is worse than s but $e^{\frac{-(g(s)-g(s'))}{T}} > R$,

where T is a control parameter called 'temperature' and $R\in[0,1]$ is a uniform random number. The temperature parameter T is initially set to a high value, allowing many non-improving moves to be accepted and it is gradually reduced to a value where nearly all non-improving moves are rejected. In this way, the algorithm avoids getting trapped in local minima until the final stages of search when the temperature is very low and the algorithm has already settled in a good solution.

There have been many studies on the convergence properties of SA. Research using the theory of Markov chains has proved that if the temperature is lowered slowly enough, SA will eventually converge to a global minimum. Unfortunately, the same research shows that this will, in general, require more iterations than exhaustive search. For detailed information on convergence results for SA, the reader is referred to two excellent books by van Laarhoven and Aarts [LA88] and Aarts and Korst [AK89]. Additionally, Johnson et al. [JAMS89, JAMS91] provide excellent experimental results for SA on a variety of problems which also may be very useful to the interested reader.

1.4.1 Cooling Schedules

In practice, the temperature is lowered according to a scheme referred to as the annealing (or cooling) schedule. A cooling schedule specifies [Osm95]:

- the initial starting value of the temperature parameter T,
- the cooling rate a and the temperature update rule,
- the number of iterations to be performed at each temperature,
- the termination criterion of the algorithm.

The performance of SA strongly depends on the cooling schedule. Not surprisingly, many different types of cooling schedules have been suggested. Osman [Osm95] classifies SA cooling schedules in three categories:

- *Stepwise temperature reduction*. In this case, the temperature remains constant for a number of iterations (i.e. selection of a random move followed by the acceptance test) before it is updated according to the update rule. The update rule commonly used is a geometric reduction function which reduces the temperature to $a(t) = a \cdot t$, where $a < 1$. That is why this type of cooling is often called *geometric cooling*. Best performances are reported in the literature for values of a in the range $0.8 \le a \le 0.99$ [Dow93]. The number of iterations at each temperature is related to the size of the neighbourhood but may also vary from temperature to temperature.
- *Continuous temperature reduction* [LM86]. In this type of cooling schedule, the temperature is reduced after every iteration. The reduction of the temperature is very slow and it is conducted according to the rule $a(t) = t/(1+b \cdot t)$ where b is a small value.
- *Non-monotonic temperature reduction* [Dow93, Osm93]. The temperature is reduced after each iteration though occasional increases are also allowed.

The SA algorithm terminates when the number of uphill moves accepted becomes negligible or some other type of stopping criterion is satisfied.

1.5 Tabu Search (TS)

Tabu Search (TS) has been developed by Glover [Glo86] and, independently, by Hansen [Han86]. TS is a meta-heuristic that combines a local search procedure with a number of anti-cycling rules which prevent the search from getting trapped in local minima. Over the years, the method has evolved and incorporated many new elements which further enhance its overall performance.

In this section, we will present the most important elements of TS. The interested reader may refer to one or more of the excellent survey papers available on the technique [Glo89, Glo90, GTW93, GL93, Glo94, Glo95, Glo96]. These survey papers examine in more detail the elements of TS described in this chapter and also outline some less frequently used elements not examined here.

1.5.1 The Basis for Tabu Search

The basis for tabu search is described by Glover in [Glo95] as follows. Given a function $f(\mathrm{x})$ to be optimised over a set X, TS begins in the same way as ordinary local search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each $x \in X$ has an associated neighbourhood $N(x) \subset X$ and each solution $x' \in \mathrm{N}(x)$ is reached from x by an operation called a *move*.

TS goes beyond local search by employing a strategy of modifying $N(x)$ as the search progresses, efficiently replacing it by another neighbourhood $N^*(x)$. A key aspect of tabu search is the use of special memory structures which serve to determine $N^*(x)$, and hence to organise the way in which the space is explored.

We will start our brief account of TS by examining the so-called *recency-based* memory which can be used as a stand-alone device or as the basis for more advanced TS schemes.

1.5.2 Recency-Based Memory

Recency-based memory is utilising information pertaining to the moves executed by local search to avoid reversing changes created by these moves. The information used is the "attributes" of solutions (i.e. solution properties) that change state (i.e. deleted or added) when a move is executed. These attributes are used to define the "tabu status" of moves at future iterations, that is, moves which are forbidden to be executed. For example, if a move m changes the value of a 0-1 variable x_j from 0 to 1 then the solution attribute $x_j = 0$ can be used to prevent the reversal of the changes created by the move. After move m is executed the solution attribute $x_j = 0$ becomes tabu-active rendering tabu (i.e. forbidden) all moves that reinstate this attribute in the solution. These restrictions are temporary and they last only for a small number of iterations. For that purpose, tabu-active attributes are assigned appropriate *tabu-tenures* which determine for how many iterations local search is prevented from reinstating these attributes. This mechanism is sometimes implemented using a data structure called a *tabu list* [Glo89].

A move may change the state of more than one solution attribute. In such cases, tabu restrictions on moves can be defined by rendering a move tabu only if all (or some number) of its component solution attributes are tabu-active [Glo95]. By deciding on the combinations of attributes that render a move tabu, we have the flexibility to strengthen or weaken the tabu restrictions. The choices may vary from a disjunction between the attributes (more restrictive) to a conjunction (less restrictive). Another way of controlling tabu restrictions is to assign different tabu-tenures to different types of attributes. Furthermore, tabu-tenures may vary during search leading to a dynamic and robust form of search [Tai91, GTW93].

Another part of recency-based memory are the so-called *aspiration criteria* which mainly aim at adding flexibility to compensate for the hard nature of constraints in recency-based memory. Aspiration criteria are sets of conditions which if satisfied overrule the tabu restrictions. The most commonly used aspiration criterion is to accept a move which is classified as tabu if the move generates a solution better than any previously seen. The interested reader may refer to [GL93] where a more extensive account is given on the different types of aspiration criteria.

In many applications, recency-based memory is sufficient to produce high quality solutions. However, this type of memory is of a short-term nature and therefore insufficient to support a long-term strategy necessary for a more systematic exploration of the search space. For that purpose, a set of additional tabu search elements have been developed which are known as long-term memory components. The two main goals for these components are the intensification and diversification of search.

1.5.3 Intensification Strategies

The purpose of intensification strategies is to concentrate the search on good regions of the search space or good solution features. This usually manifests itself in a solution recording mechanism which keeps a copy of high quality solutions found during the search. These solutions, often referred to as *elite solutions*, are used each time the search progresses slowly to restart it from the good regions which lie around these elite solutions. The state of the recency-based memory (when the elite solution was recorded) may also be saved and partially or fully restored when starting from this elite solution. Such approaches have been successfully used in vehicle routing [XK96] and telecommunications network design problems [XCG95].

Another form of intensification is based on identifying "consistent and strongly determined" variables. A strongly determined variable is one whose value cannot be changed except by inducing a disruptive effect on the objective function value or the values of the other variables. On the other hand, a consistent variable is one that is frequently assigned the same value in good solutions. The idea is to identify the most consistent and strongly determined variables and assign to these variables their "preferred" values by reference to a set of elite solutions. This is usually done in the framework of a multi-start approach where new starting points are generated by assigning consistent and strongly determined variables to their "preferred" values. This approach has been successfully applied to the vehicle routing problem [RT95].

1.5.4 Diversification Strategies

Diversification strategies are designed to drive the search into new regions. Often they are based on modifying choice rules to bring attributes into the solutions that are infrequently used. More of these schemes are based on type of memory called *frequency-based memory*. In short, frequency-based memory is a long-term memory which either

• records the frequency at which solution attributes occur in the solutions generated (*residence* frequencies)

• or records the frequency different moves are executed (*transition* frequencies).

Residence frequencies are used to encourage the incorporation in the solution of infrequently used attributes while transition frequencies are used to encourage the execution of less frequently performed moves.

One way to use the information recorded in residence frequencies is to periodically restart the search from solutions that incorporate the less frequently used solution attributes. More often, residence frequencies are used in a continuous fashion by being directly incorporated in the cost function multiplied by a penalty factor. Attributes with high frequencies have higher penalties than those with lower frequencies. Thus the use of the later is encouraged while the use of the former is discouraged.

Transition frequencies are used in a similar way to residence frequencies and penalties are usually introduced that discourage the execution of frequently executed moves while encouraging the execution of less frequently executed moves.

Residence or transition frequencies have been successfully used in problems such as maximum clique [SG96], bin packing [LG93], network design [XCG95], quadratic assignment [Sko90], machine scheduling [LG93b], vehicle routing [GHL94, TBGGP95, XK96] and others.

1.5.5 Candidate List Strategies

For many problems, the amount of computational effort required to search the complete neighbourhood in every iteration is prohibitive. Candidate list strategies are aiming at reducing this effort by restricting the number of solutions examined on a given iteration. The different types of candidate list strategies are the following [Glo95]:

- *Random Strategy***.** The neighbourhood is randomly sampled until enough moves are evaluated to give some assurance that some good choices were examined.
- *Subdivision Strategy*. Moves that involve more than one component are decomposed and moves which incorporate good components only are examined.
- *Aspiration Plus Strategy*. This approach establishes a threshold based on the search history for the quality of moves to be selected and examines moves until finding one which satisfies this threshold.
- *Elite Candidate Lists*. A list of elite moves is constructed after searching a large part of the neighbourhood. At subsequent iterations only solutions from this elite list are examined until the quality of moves drops below a specified threshold. At this point a new list is constructed and the process is repeated.
- *Sequential Fan Strategy*. The idea is to generate some p best alternative moves at a given step and then to create a fan of solution streams, one for each alternative. The best available moves for each stream are again examined and only the p best moves overall provide the p new streams at the next step. This technique is very much oriented towards parallel processing.

Candidate list strategies conclude our account of Tabu Search. Other elements not examined here include strategic oscillation, path re-linking, ejection chains, vocabulary building and probabilistic tabu search. The reader may refer to [GL93, Glo95, Glo96] for information on these variants. Additionally, the reader may also refer to [XCG96] on the use of statistical tests to determine the many parameter values that need to be specified when various elements of TS are integrated together to solve a combinatorial problem.

1.6 Genetic Algorithms

Genetic Algorithms (GAs) are a class of methods based on a highly abstract model of natural evolution. They were developed by Holland in the 70s [Hol75, Gol89, Dav91] and since then they have been applied to numerous domains. Only recently their potential application to combinatorial optimisation problems has been investigated. We first examine some of the terminology used in the GA literature.

A solution to a combinatorial optimisation problem is often called a *chromosome*, *string* or *vector*. Variables of the problem are called *genes* and their possible values *alleles*. The position of a variable in a chromosome is called its *locus*. Each chromosome encodes a solution to the optimisation problem and it is evaluated according to some *fitness function*. The fitness function is related to the cost function of the combinatorial optimisation problem. The *fitness value* given to a chromosome by this function represents the suitability of this chromosome (after decoding) as a solution to the combinatorial problem. For a review of Genetic Algorithm techniques in the context of combinatorial optimisation the reader may refer to [Ree93].

1.6.1 A Basic GA Algorithm

A basic GA algorithm for a combinatorial problem functions in the following way. Initially, a finite population of solutions is generated randomly or by other means. After that, an iterative process is applied to the population which at each step transforms the current population to a new population. This involves selecting pairs of parent solutions from the population according to a selection scheme which takes into account their fitness values and combining them to generate offspring solutions. The combination of the parents is performed by a special type of operator called the *crossover* or *recombination* operator. After the generation of the ‘children’ random changes are inflicted upon them by a second type of operator called the *mutation* operator. The children are finally inserted in the population by either replacing their parents (Canonical GAs) or the weakest individuals in the population (Steady State GAs[2]). This completes one iteration of the GA which transforms one generation of solutions to the next. The algorithm iterates until a termination criterion is satisfied based either on computational resources, the convergence of the population (high similarity between the solutions contained in the population) or both. In the following, we examine more closely the various elements of a GA.

1.6.1.1 Initial Population

The initial population is normally generated at random. Yet in most of the successful GAs for combinatorial optimisation, solutions in the initial population are heuristically generated (by a construction heuristic, local search, or sometimes by local search applied to a solution generated by a construction heuristic) and they are already of good quality. Particular attention must be paid that the size of this initial population is not too small to avoid premature convergence of the GA.

1.6.1.2 Genetic Operators

As mentioned above, the crossover operator is used to combine two parent solutions. There are many versions of this operator. The simplest case is that of the *1-point* crossover. A cut-point X is selected at random and each offspring consists of the pre-X section from one parent followed by the post-X section from the other. The 1-point crossover can be extended to *2-point* crossover, *3-point* crossover or even *k-point* crossover. Another useful crossover operator is the *uniform* crossover where the value of each variable in each parent is equally likely to be passed to the offspring.

[2] Steady State GAs also generate one child instead of two children as in Canonical GAs.

Many combinatorial optimisation problems require special types of operators which can combine sequences or permutations (often used to represent solutions) to produce feasible offspring solutions. An example of such an operator for the TSP is the PMX operator (Partially Mapped Crossover). Many other special operators exist for different types of combinatorial optimisation problems (see [Ree93] for some examples).

In addition to crossover and after the generation of the children, a mutation operator is employed to modify the population of solutions by introducing small random modifications to solutions randomly selected from the population. If bit vectors are used for representing the solutions, this frequently means flipping the bits of some of the solutions. In general, the probability of mutation is very low.

1.6.2 Hybrid GAs.

As Davis states in the Handbook of Genetic Algorithms [Dav91], "Traditional genetic algorithms, although robust, are generally not the most successful optimisation algorithm on any particular domain". For that reason, Davis and many others have argued that hybridising GAs with the most successful optimisation methods for particular problems gives one the best of both worlds.

The idea of including in the initial population solutions constructed by a problem-specific heuristic, mentioned above, can be viewed as a primitive form of hybridisation.

Several GA approaches which have produced very good results for famous combinatorial optimisation problems go one step further, utilising local search algorithms to optimise the solution generated by crossover or mutation operators (see [MGK88, FF94, FM96]). These GA algorithms essentially work on local minima constructed by local search trying to recombine them to produce new and hopefully better local minima. The rationale is that local minima solutions consist of good solution fragments which if properly combined by crossover type operators will lead to solutions where these fragments are combined even better and therefore be of higher quality. This leads to a type of search intensification around the areas of good solutions. Diversification of search is also important and is performed by the particular mutation operator used. From another viewpoint, Hybrid GAs can be seen as a type of local search which explores the space of good solution fragment combinations. There are similarities there with tabu search variants which also try to identify and recombine good solution fragments [RT95]. These tabu search variants are sometimes seen as part of a wider framework of techniques called *Adaptive Memory Programming* [Glo96].

1.7 GENET and Other Weighting Methods for CSPs

Guided Local Search (GLS) studied in this thesis is a meta-heuristic which guides local search in exploring the vast search spaces of combinatorial problems. The technique extends to general optimisation problems methods applied with considerable success to Constraint Satisfaction [Tsa93]. In this section, we will briefly refer to these methods and in particular to the GENET neural network [WT91, Tsa93, DTWZ94] which is a direct predecessor of GLS.

The Constraint Satisfaction Problem (CSP) is that of assigning values to a number of variables with finite domains such that a set of linear or non-linear constraints involving one or more variables are satisfied. CSP is in general NP-Hard and it is closely related to the propositional satisfiability or SAT problem [GJ79]. In contrast to most combinatorial optimisation problems, the goal in CSPs is to find one or all feasible solutions. Real world CSPs usually involve difficult non-linear constraints spanning two or more variables of the problem. Amongst other techniques, local search has been considered for solving CSPs.

A local search approach to constraint satisfaction treats a CSP as an optimisation problem. The objective function, which is to be minimised, is the number of constraints being violated. A typical local search method assigns an arbitrary value to each variable in the CSP. Then it proceeds iteratively to reduce the number of constraint violations by re-assigning values to variables, using a heuristic known as the min-conflict heuristic [MJPL92]. This iterative improvement of the number of unsatisfied constraints leads either to a solution to the CSP or to a local minimum where some constraints are still being violated but no further improvement is possible by changing the value of any of the variables. Local minima are of little use in CSPs since they violate hard problem constraints.

A successful approach to escape local minima, proposed in the context of CSPs, is to assign weights to the problem constraints (clauses for SAT) and increase these weights in a local minima for the violated constraints (unsatisfied clauses for SAT) in a effort to 'fill up' the local minimum until local search escapes from it.

Various algorithms based on this scheme have been developed in the last few years and applied either to the CSP or the SAT problem. Amongst them GENET [WT91, Tsa93, DTWZ94], Weighted GSAT [SK93, Fra96], and also the Breakout Method [Mor93]. Here, we briefly examine GENET which was the point of origin for this work.

1.7.1 The GENET Neural Network

GENET is a connectionist approach to constraint satisfaction with a basic operation that resembles the min-conflicts heuristic. Basically a CSP is represented by a network in which the nodes represent possible assignments to the variables and the edges represent constraints. One of the innovations in GENET was the use of and manipulation of weights assigned to the edges (constraints). All edges are inhibitory connections which have weights initialised to -1. GENET will continuously select assignments which receive the least inhibitory input (which roughly means violating the least number of constraints). The operation of the network is designed in such a way that will ensure its convergence to some states, which could be solutions or local minima (in terms of number of constraints violated). Each time the network converges to a local minimum, the weights associated with the violated constraints are decreased, and the network is then allowed to converge again. Since GENET always makes moves which improve the number of constraint violations, decreasing the weights allows it to escape from the local minimum to states which have lower cost. Such convergence-learning cycles continue until a solution is found or a stopping condition is satisfied.

GENET's mechanism for escaping from local minima resembles reinforcement learning [BSA83]. Basically, patterns in a local minimum are stored in the constraint weights and are discouraged to appear thereafter. For this reason, the mechanism was named "learning". GENET's learning scheme can be viewed as a method to transform the objective function (i.e. the number of constraint violations) so that a local minimum gains an artificially higher value. Consequently, local search will be able to leave the local minimum state and search other parts of the space.

In the CSP context, modifying the weights for unsatisfied constraints in local minima modifies the cost function of the problem though that does not affect the cost of an optimal solution which if exists satisfies all the constraints by definition and therefore always has zero cost.

Guided Local Search (GLS) presented in this thesis utilises a similar approach to tackle famous combinatorial problems. In these problems, modifications to the cost function, although they may affect the cost of many solutions of a combinatorial problem including the optimal can effectively guide local search in the search space. Apart from escaping local minima in a way similar to GENET and other techniques for CSP and SAT problems, GLS introduces additional functionality for distributing the search efforts over the various areas of the search space, taking into account the promise of these areas to contain the optimal solution. Furthermore, it uses sophisticated neighbourhood reduction techniques which can speed up the algorithm many times.

1.8 Overview of the Thesis

In this thesis, we describe the technique of GLS and examine its application to a comprehensive set of traditional and modern real world combinatorial optimisation problems. The performance of the technique is experimentally evaluated on benchmark instances of these problems. Extensive comparisons are conducted with general and specialised heuristic algorithms including all the general heuristic methods examined in this chapter. The thesis is structured as follows. In the next chapter, we present GLS and discuss various extensions to the method. Following that, five applications of the algorithm are examined in the following order:

- Travelling Salesman Problem (chapter 3),
- Quadratic Assignment Problem (chapter 4),
- Radio Link Frequency Assignment Problem (chapter 5),
- Workforce Scheduling (chapter 6),
- Non-convex Optimisation (chapter 7).

The thesis concludes with chapter 8 where the work on GLS is summarised and future research directions are suggested.

Most of the findings in chapter 5 have appeared in the Proceedings of the 2nd International Conference on Practical Application of Constraint Technology [VT96] while the results in chapter 6 have appeared in the journal of *Operations Research Letters* [TV97]. Earlier results for GLS on the Travelling Salesman Problem, Quadratic Assignment Problem and Nonconvex Optimisation have been reported in two Essex University technical reports [VT95a, VT95b].

Chapter 2

Guided Local Search

Embarking on this research almost three years ago, the main objective was to extend GENET for Constraint Satisfaction Problems (CSPs) to a more general class of problems known as Partial Constraint Satisfaction Problems (PCSPs). Through the process of trying to apply GENET to PCSPs, we soon realised that a more general optimisation technique was hidden under GENET's neural network architecture. This technique, namely Guided Local Search, is the subject of this chapter and the core of the thesis.

2.1 History of Guided Local Search

Partial CSPs are CSPs where no solution satisfies all the constraints and one is interested in solutions which minimise the number of constraint violations and possibly other application dependent criteria (see section 5.1 for a formal definition).

The RLFAP problem described in chapter 5 was one of the first problems we tried to solve using extensions of GENET. The problem is a PCSP and requires the minimisation of constraint violations combined with domain specific optimisation criteria. Minimising constraint violations was within the capabilities of the GENET neural network but minimising the other RLFAP optimisation criteria seemed difficult and required extra complexity in the neural network architecture. Because of that, we decided at the time to convert GENET to a pure algorithm, abandoning any efforts to solve the problem by extending the model of the neural network. This resulted in the Tunnelling Algorithm [VT94] which was very successful in the RLFAP instances and moreover preserved the good performance of GENET on classic CSPs. While experimenting with the tunnelling algorithm, we had the idea to apply the method to the Travelling Salesman Problem (TSP), utilising some of the work we did on the modelling of RLFAP's optimisation criteria. To our surprise, the method worked extremely well on the TSP and some preliminary results on that were included in the paper on the tunnelling algorithm [VT94].

The success on the TSP convinced us of the great potential of the algorithm. We generalised the Tunnelling Algorithm even further, so that it could be applied to the bulk of combinatorial optimisation. The result of this generalisation was Guided Local Search. Guided Local Search exceeded all our expectation. We applied the method to seven Combinatorial Optimisation problems and obtained very good results both in terms of solution quality and running times. The method and five of its applications will be presented in this thesis. We start by introducing the principles of Guided Local Search.

2.2 Guided Local Search Principles

Guided Local Search is a general and compact optimisation technique suitable for a wide range of combinatorial optimisation problems. Guided Local Search takes advantage of problem and search-related information to guide *local search* in a search space. This is made possible by augmenting the cost function of the problem to include a set of penalty terms. Local search is confined by the penalty terms and focuses attention on promising regions of the search space. Iterative calls are made to local search. Each time local search gets caught in a local minimum, the penalties are modified and local search is called again to minimise the modified cost function.

Penalty modifications *regularise* the solutions generated by local search to be in accordance with prior information or information gathered during search. The approach taken by GLS is analogous to that of regularisation methods for 'ill-posed' problems [TAJ77, Hay94]. The idea behind regularisation methods and GLS, to an extent, is the use of prior information to help us solve an approximation problem. Prior information translates to constraints which further define our problem, so reducing the number of candidate solutions to be considered. GLS also exploits information learnt during search by imposing extra constraints on the basis of this information. GLS is essentially a *meta-heuristic* based on local search. In the following sections, we examine the various components of GLS.

2.3 Local Search

Local search is the basis of many heuristic methods for combinatorial optimisation problems. In section 1.3, we presented an overview of local search. A variety of moves and local search procedures have been used for the problems in this study. For the purpose of describing GLS in the general case, local search is considered a general procedure of the form:

$$s_2 \leftarrow \textit{procedure}\ \mathbf{LocalSearch}(s_1,g),$$

where s_1 is the initial solution, s_2 the final solution (local minimum) and g the cost function to be minimised.

In contrast to other general meta-heuristics such as SA and tabu search, GLS is not modifying the internal mechanisms of local search. Instead, it makes iterative calls to a local search procedure modifying the cost function between successive calls. Before that, the cost function of the problem is augmented to include a set of penalty terms which enable us to constrain solutions dynamically. This augmentation of the cost function with penalty terms is explained in the next section.

2.4 Solution Features

GLS employs solution features to characterise solutions. A *solution feature* can be any solution property that satisfies the simple constraint that is a non-trivial one. What it is meant by that is that not all solutions have this property. Some solutions have the property while others do not. Solution features are problem dependent and serve as the interface between the algorithm and a particular application.

Constraints on features are introduced or strengthened on the basis of information about the problem and also the course of local search. Information pertaining to the problem is the cost of features. The cost of features represents the direct or indirect impact of the corresponding solution properties on the solution cost. Feature costs may be constant or variable. Information about the search process pertains to the solutions visited by local search and in particular local minima. A feature f_i is represented by an indicator function in the following way:

Eq. 2.1 $$I_i(s) = \begin{cases} 1, & \text{solution } s \text{ has property } i \\ 0, & \text{otherwise} \end{cases}, s \in S.$$

2.5 Augmented Cost Function

Constraints on features are made possible by augmenting the cost function g of the problem to include a set of penalty terms. The new cost function formed is called the *augmented cost function* and it is defined as follows:

Eq. 2.2 $$h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s),$$

where M is the number of features defined over solutions, p_i is the penalty parameter corresponding to feature f_i and λ (lambda) is the *regularisation parameter*. The penalty parameter p_i gives the degree up to which the solution feature f_i is constrained. The regularisation parameter λ represents the relative importance of penalties with respect to the solution cost and is of great significance because it provides a means to control the influence of the information on the search process. GLS iteratively uses local search and it simply modifies the *penalty vector* ***p*** given by:

Eq. 2.3 $$\mathbf{p} = (p_1, ..., p_M)$$

each time local search settles in a local minimum. Modifications are made on the basis of information. Initially, all the penalty parameters are set to 0 (i.e. no features are constrained) and a call is made to local search to find a local minimum of the augmented cost function. After the first local minimum and every other local minimum, the algorithm takes a modification *action* on the augmented cost function and re-applies local search, starting from the previously found local minimum. The modification action is that of simply incrementing by **one** the penalty parameter of one or more of the local minimum features. Prior and historical information is gradually inserted into the augmented cost function by selecting which penalty parameters to increment.

Sources of information are the cost of features and the local minimum itself. Let us assume that each feature f_i defined over the solutions is assigned a cost c_i. This cost may be constant or variable. In order to simplify our analysis, we consider feature costs to be constant and given by the *cost vector* ***c***:

Eq. 2.4 $$\mathbf{c} = (c_1, \ldots, c_M)$$

which contains positive or zero elements. A particular local minimum solution s_* exhibits a number of features. Indicators of the features f_i exhibited take the value 1 (i.e. $I_i(s_*) = 1$).

2.6 Penalty Modifications

In a local minimum s_*, the penalty parameters are incremented by one for all features f_i that maximise the utility expression:

Eq. 2.5 $$util(s_*, f_i) = I_i(s_*) \cdot \frac{c_i}{1 + p_i} \; .$$

In other words, incrementing the penalty parameter of the feature f_i is considered an *action* with utility given by Eq. 2.5. In a local minimum, the actions with *maximum* utility are selected and then performed. The penalty parameter p_i is incorporated in Eq. 2.5 to prevent the scheme from being totally biased towards penalising features of high cost. The role of the penalty parameter in Eq. 2.5 is that of a counter which counts how many times a feature has been penalised. If a feature is penalised many times over a number of iterations then the term $\frac{c_i}{1+p_i}$ in Eq. 2.5 decreases for the feature, diversifying choices and giving the chance for other features to also be penalised. The policy implemented is that features are penalised with a frequency proportional to their cost. Due to Eq. 2.5, features of high cost are penalised more frequently than those of low cost. The search effort is distributed according to *promise* as it is expressed by the feature costs and the already visited local minima, since only the features of local minima are penalised. Incremental distribution of the search effort according to prior information, though in a probabilistic framework, can be found in a class of methods based on the *optimal search theory* of Koopman [Koo57, Sto83]. Also, counter based schemes for search diversification analogous to that of GLS are used under the name *counter-based exploration* in reinforcement learning [Thr92]. The basic GLS algorithm as described so far is depicted in Figure 2.1.

```
procedure GuidedLocalSeach(S, g, $\lambda$, [$I_1$, ...,$I_M$], [$c_1$,...,$c_M$], M)
begin
    k ← 0;
    $s_0$ ← random or heuristically generated solution in S;
    for i ←1 until M do /* set all penalties to 0 */
        $p_i$ ← 0;
    while StoppingCriterion do
    begin
        h ← g + $\lambda$ * $\sum p_i * I_i$ ;
        $s_{k+1}$ ← LocalSearch($s_k$, h);
        for i ←1 until M do
            $util_i$ ← $I_i(s_{k+1})$ * $c_i$ / (1+$p_i$);
        for each i such that $util_i$ is maximum do
            $p_i$ ← $p_i$ + 1;
        k ← k+1;
    end
    s* ← best solution found with respect to cost function g;
    return s*;
end
```

where S: search space, g: cost function, h: augmented cost function, λ: regularisation parameter, I_i: indicator function for feature i, c_i: cost for feature i, M: number of features, p_i: penalty for feature i.

Figure 2.1 Guided Local Search in pseudocode

As we will see in the following chapters, this simple algorithm can be applied with simple modifications to a variety of optimisation problems. Applying the algorithm to a problem usually involves defining the features to be used, assigning costs to the them and finally substituting the procedure *LocalSearch* in the GLS loop with a local search algorithm for the problem in hand.

2.7 Regularisation Parameter

Something that has been left out from the analysis so far is the regularisation parameter λ in the augmented cost function Eq. 2.2. This parameter determines the degree up to which constraints on features are going to affect local search. Let us examine how the regularisation parameter is going to affect the moves performed by a local search method. A move alters the solution, adding new features and removing existing features, whilst leaving other features unchanged. In the general case, the difference Δh in the value of the augmented cost function due to a move is given by the following difference equation:

Eq. 2.6

$$\Delta h = \Delta g + \lambda \cdot \sum_{i=1}^{M} p_i \Delta I_i \,.$$

As we can see in Eq. 2.6, if λ is large then the selected moves will solely remove the penalised features from the solution and the information will fully determine the course of local search. This introduces risks because information may be wrong. Conversely, if λ is 0 then local search will not be able to escape from local minima. However, if λ is small and comparable to Δg then the moves selected will aim at the combined objective of improving the solution (taking into account the cost differences) and also removing the penalised features (taking into account the information). Since the difference Δg is problem dependent, the regularisation parameter is also problem dependent. GLS can be quite tolerant to the choice of the λ, operating well for a wide range of values. In the applications, we are going to elaborate further on the role of this parameter and on how it affects GLS in specific problems.

2.8 Fast Local Search and Other Improvements

There are both minor and major optimisations that significantly improve the basic GLS method. For example, instead of calculating the utilities for all the features, we can restrict ourselves to the local minimum features since for non-local minimum features the utility as given by Eq. 2.5 takes the value 0. Also, the evaluation mechanism for moves needs to be changed to work efficiently on the augmented cost function. Usually, this mechanism is not directly evaluating the cost of the new solution generated by the move but it calculates the difference Δg caused to the cost function. This difference in cost should be combined with the difference in penalty as is shown in Eq. 2.6. This can be easily done and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only features that change state (being deleted or added). The penalty parameters of the features deleted are summed together. The same is done for the penalty parameters of features added. The change in penalty due to the move is then simply given by the difference:

Eq. 2.7
$$- \sum_{\text{over all features j added}} p_j \quad + \sum_{\text{over all features k deleted}} p_k \quad .$$

Leaving behind the minor improvements, we turn our attention to the major improvements. In fact, these improvements do not directly refer to GLS but to local search. Greedy local search selects the best solution in the whole neighbourhood. This can be very time-consuming, especially if we are dealing with large instances of problems. Next, we are going to present *Fast Local Search* (FLS), which drastically speeds up the neighbourhood search process by redefining it. The method is a generalisation of the *approximate 2-opt* method proposed in [Ben92] for the Travelling Salesman Problem. The method also relates to *Candidate List Strategies* used in tabu search (see section 1.5.5).

FLS works as follows. The current neighbourhood is broken down into a number of small sub-neighbourhoods and an *activation bit* is attached to each one of them. The idea is to scan continuously the sub-neighbourhoods in a given order, searching only those with the activation bit set to 1. These sub-neighbourhoods are called *active* sub-neighbourhoods. Sub-neighbourhoods with the bit set to 0 are called *inactive* sub-neighbourhoods and they are not being searched. The neighbourhood search process does not restart whenever we find a better solution but it continues with the next sub-neighbourhood in the given order. This order may be static or dynamic (i.e. change as a result of the moves performed).

Initially, all sub-neighbourhoods are active. If a sub-neighbourhood is examined and does not contain any improving moves then it becomes inactive. Otherwise, it remains active and the improving move found is performed. Depending on the move performed, a number of other sub-neighbourhoods are also activated. In particular, we activate all the sub-neighbourhoods where we expect other improving moves to occur as a result of the move just performed. As the solution improves the process dies out with fewer and fewer sub-neighbourhoods being active until all the sub-neighbourhood bits turn to 0. The solution formed up to that point is returned as an approximate local minimum.

The overall procedure could be many times faster than conventional local search. The bit setting scheme encourages chains of moves that improve specific parts of the overall solution. As the solution becomes locally better the process is settling down, examining fewer moves and saving enormous amounts of time which would otherwise be spent on examining predominantly bad moves.

Although fast local search procedures do not generally find very good solutions, when they are combined with GLS they become very powerful optimisation tools. Combining GLS with FLS is straightforward. The key idea is to associate solution features to sub-neighbourhoods. The associations to be made are such that for each feature we know which sub-neighbourhoods contain moves that have an immediate effect upon the state of the feature (i.e. moves that remove the feature from the solution). The combination of the GLS algorithm with a generic FLS algorithm is depicted in Figure 2.2.

The procedure *GuidedFastLocalSearch* in Figure 2.2 works as follows. Initially, all the activation bits are set to 1 and FLS is allowed to reach the first local minimum (i.e. all bits 0). Thereafter, and whenever a feature is penalised, the bits of the associated sub-neighbourhoods and only those are set to 1. In this way, after the first local minimum, fast local search calls examine only a number of sub-neighbourhoods and in particular those which associate to the features just penalised. This dramatically speeds up GLS. Moreover, local search is focusing on removing the penalised features from the solution instead of considering all possible modifications.

```
procedure GuidedFastLocalSearch(S, g, $\lambda$, [$I_1$, ...,$I_M$], [$c_1$,...,$c_M$], M, L)
begin
        k ← 0; $s_0$ ← random or heuristically generated solution in S;
        for i ←1 until M do $p_i$ ← 0; /* set all penalties to 0 */
        for i ←1 until L do $bit_i$ ← 1; /* set all sub-neighbourhoods to the active state */
        while StoppingCriterion do
        begin
                h ← g + $\lambda$ * $\sum p_i * I_i$ ;
                $s_{k+1}$ ← FastLocalSearch($s_k$, h,[$bit_1$, …,$bit_L$], L);
                for i ←1 until M do $util_i$ ← $I_i(s_{k+1})$ * $c_i$ / (1+$p_i$);
                for each i such that $util_i$ is maximum do
                begin
                        $p_i$ ← $p_i$ + 1;
                        SetBits ← SubNeighbourhoodsForFeature(i);
                        /* activate sub-neighbourhoods relating to feature i penalised */
                        for each bit b in SetBits do b ← 1;
                end
                k ← k+1;
        end
        s* ← best solution found with respect to cost function g;
        return s*;
end

procedure FastLocalSeach(s, h, [$bit_1$, …,$bit_L$], L)
begin
        while ∃bit, bit = 1 do
                for i ←1 until L do
                begin
                        if $bit_i$ = 1 then /* search sub-neighbourhood for improving moves */
                        begin
                                Moves ← set of moves in sub-neighbourhood i;
                                for each move m in Moves do
                                begin
                                        s′ ← m(s);
                                        /* s′ is the solution generated by move m when applied to s */
                                        if h(s′) < h(s) then /* for minimisation */
                                        begin
                                                $bit_i$ ← 1;
                                                SetBits ← SubNeighbourhoodsForMove(m);
                                                /* spread activation to other sub-neighbourhoods */
                                                for each bit b in SetBits do b ← 1;
                                                s ← s′;
                                                goto ImprovingMoveFound
                                        end
                                end
                                $bit_i$ ← 0; /* no improving move found */
                        end
ImprovingMoveFound:     continue;
                end;
        return s;
end
```

where S: search space, g: cost function, h: augmented cost function, λ: regularisation parameter, I_i: indicator function for feature i, c_i: cost for feature i, M: number of features, L: number of sub-neighbourhoods, p_i: penalty for feature i, bit_i: activation bit for sub-neighbourhood i, SubNeighbourhoodsForFeature(i): procedure which returns the bits of the sub-neighbourhoods corresponding to feature i, and SubNeighbourhoodsForMove(m): procedure which returns the bits of the sub-neighbourhoods to spread activation to when move m is performed.

Figure 2.2 Guided Local Search combined with Fast Local Search in pseudocode

Apart from the combination of GLS with fast local search, other variations of GLS to be presented in the applications include:

- features with variable costs where the cost of a feature is calculated during search and in the context of a particular local minimum (see chapter 4)
- penalties with limited duration (see chapter 4)
- multiple feature sets where each feature set is processed in parallel by a different penalty modification procedure (see chapter 4)
- feature set hierarchies where more important features overshadow less important feature sets in the penalty modification procedure (see chapter 5).

Before presenting the applications of GLS, we examine some of the links between GLS and other general optimisation methods also based on local search.

2.9 Connections with Other General Optimisation Techniques

2.9.1 Simulated Annealing

Non-monotonic temperature reduction schemes used in SA (see section 1.4) also referred to as *re-annealing* or *re-heating* schemes are of particular interest in relation to the work presented in this thesis. In these schemes, the temperature is decreased as well as increased in a attempt to remedy the problem that the annealing process eventually settles down failing to continuously explore good solutions. In a typical SA, good solutions are mainly visited during the mid and low parts of the cooling schedule. For resolving this problem, it has been even suggested annealing at a constant temperature high enough to escape local minima but also low enough to visit them [Con90]. It is seems extremely difficult to find such a temperature because it has to be landscape dependent (i.e. instance dependent) if not dependent of the area of the search space currently searched.

Guided Local Search presented in this thesis can be seen as addressing this problem of visiting local minima but also being able to escape from them. Instead of random up-hill moves, penalties are utilised to force local search out of local minima. The amount of penalty applied is progressively increased in units of appropriate magnitude (i.e. parameter λ) until the method escapes from the local minimum. GLS can be seen adapting to the different parts of the landscape. The algorithm is continuously visiting new solutions rather than converging to any particular solution as SA does.

Another important difference between this work and SA is that GLS is a deterministic algorithm. This is also the case for a wide number of algorithms developed under the tabu search framework.

2.9.2 Tabu Search

GLS has close links with tabu search. Both techniques can be seen as using information (historical in the case of tabu search, prior and historical information in the case of GLS) to impose constraints on local search either by modifying the neighbourhood (tabu search) or by modifying the cost function to be minimised (GLS). Let us consider the neighbourhood graph where each node is a solution to the problem and the arcs are the moves which transform one solution to another. GLS adopts a "solution or node"-centred approach to constrain local search by elevating the cost of specific nodes (i.e. solutions), rather than the "move or arc"-centred approach adopted by many tabu search variants which prevents local search from traversing specific arcs (i.e. executing moves which are tabu). The two approaches can be seen to be seeking the same goal (i.e. guide local search by using constraints) though they use different means to achieve that.

Solution attributes used in tabu search can been seen as corresponding to the solution features used in GLS. However, constraints on solution attributes by tabu search may take many forms (i.e. tabu lists, frequency-based penalties) while in GLS a single mechanism is used which utilises indicator functions to introduce constraints on solution features.

Rather than elevate selected penalties to drive the search out of a local minimum, as GLS does, the typical tabu search approach seeks a best move to escape from a local minimum based on the current evaluation function, influenced by prior memory and by candidate list strategies. Penalties in tabu search are customarily applied to selected attributes only after the move is made, as a way of preventing a return. Tabu search also typically maintains a recency-based memory to provide a mechanism to avoid reinstating selected attribute combinations found in recently generated solutions. Diversification strategies that make use of frequency-based memory are generally activated periodically, rather than continuously as in GLS.

A more detailed list of the various search elements that are present in both techniques along with the ways they are realised in each individual technique is given in Table 2.1. As we can see in this table, despite the differences between tabu and guided local search, there is common ground in many areas. This common ground may well be utilised in the future to define a more abstract class of methods which one may call *Intelligent Search* methods.

	Tabu Search	**Guided Local Search**
Local search guidance mechanism	modified neighbourhood, intelligent restarts	modified cost function
Information used	mainly the moves executed but also transition & residence frequencies and elite solution sets	feature costs, local minima visited
Constraints	• hard constraints on moves or solution attributes based on moves recently executed, aspiration criteria override the hard constraints • soft constraints on moves or solution attributes based on transition or residence frequencies	soft constraints on solution features based on search plan for distributing search effort taking into account the local gradients
Memory Utilised	• tabu lists recording attributes of moves recently executed • frequency based memory recording the frequency of moves or solution attributes during search	memory of penalty modification actions taken by GLS also used for recording penalties on features
Intervention Period	• every iteration (recency-based memory, some types of diversification strategies) • every N iterations or when local search fails to discover new better solutions (intensification strategies, diversification strategies)	at a local minimum of the augmented cost function
Search Objectives	• avoid getting trapped in local minima and reversing changes created by the moves (proactive approach). • Intensification: restart when slow progress (reactive approach) • Diversification: examine history and penalise moves frequently executed or solution attributes frequently appearing in solutions (reactive approach)	• escape from local minima (reactive approach) • plan and distribute search efforts in the short or long term according to feature costs taking into account the local gradients (proactive approach).
Intensification - Diversification balance	-	The lambda parameter of GLS controls that. • Low lambda leads to intensification (due to cost function term in the augmented cost function). • High lambda leads to diversification (due to penalty function term in the augmented cost).
Neighbourhood Reduction Mechanism	Candidate Lists Strategies	Fast Local Search fully integrated with the diversification - intensification mechanisms of GLS

Table 2.1 Links between Guided Local Search and Tabu Search methods.

2.10 GLS Applications

GLS is a generalisation of GENET and as such it can be applied with the same success as GENET in any of the applications of the latter (i.e. CSP problems). Apart from that, GLS has been successfully applied to a set of seven problems in combinatorial optimisation. This set includes the famous Travelling Salesman and Quadratic Assignment problems, the real-world problems of Radio Link Frequency Assignment, Workforce Scheduling, Bandwidth Packing and Maximum Channel Assignment, and finally a continuous Nonconvex Optimisation problem. FLS has also been applied to all these problems except for the Quadratic Assignment Problem and the NonConvex Optimisation problem. All these applications of GLS and FLS are examined in this thesis except for the Bandwidth Packing and Maximum Channel Assignment problems for which GLS and FLS have been applied in a way similar to that for the Workforce Scheduling problem examined in chapter 6. However, demonstration programs have been developed for both the Bandwidth Packing and Maximum Channel Assignment problems which can be obtained via WWW at http://cswww.essex.ac.uk/CSP/demos.

Chapter 3

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is one of the most famous problems in combinatorial optimisation. In this chapter, we are going to examine how guided local search and fast local search can be applied to the problem. The combination of GLS and FLS with TSP local search heuristics of different efficiency and effectiveness will be studied in an effort to determine the dependence of GLS on local search. Comparisons will be made with some of the best TSP heuristic algorithms and general optimisation techniques which will demonstrate the advantages of GLS over alternative heuristic approaches suggested so far for this problem.

3.1 The Problem

There are many variations of the Travelling Salesman Problem (TSP). In this work, we examine the classic symmetric TSP. The problem is defined by N cities and a symmetric distance matrix $D=[d_{ij}]$ which gives the distance between any two cities i

and j. The goal in TSP is to find a tour (i.e. closed path) which visits each city exactly once and is of minimum length. A tour can be represented as a cyclic permutation π on the N cities if we interpret $\pi(i)$ to be the city visited after city i, $i = 1,\ldots,N$. The cost of a permutation is defined as:

Eq. 3.1
$$g(\pi) = \sum_{i=1}^{N} d_{i\pi(i)}$$

and gives the cost function of the TSP [PS82].

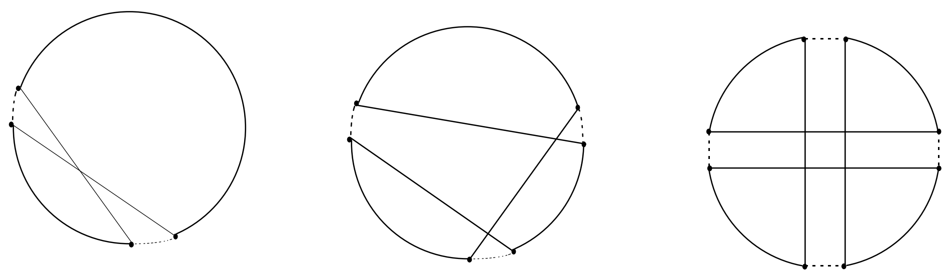
Recent and comprehensive surveys of TSP methods are those by Laporte [Lap92], Reinelt [Rei94] and Johnson & McGeoch [JM95]. The reader may also refer to [LLKS85] for a classical text on the TSP. The state of the art is that problems up to 1,000,000 cities are within the reach of specialised approximation algorithms [Ben92]. Moreover, the optimal solutions have been found and proven for non-trivial problems of size up to 7397 cities [JM95]. Nowadays, TSP plays a very important role in the development and testing of new optimisation techniques. In this context, we examine how guided local search and fast local search can be applied to this problem.

3.2 Local Search Heuristics for the TSP

Local search for the TSP is synonymous with k-Opt moves. Using k-Opt moves, neighbouring solutions can be obtained by deleting k edges from the current tour and reconnecting the resulting paths using k new edges. The k-Opt moves are the basis of the three most famous local search heuristics for the TSP, namely *2-Opt* [Cro58], *3-Opt* [Lin65] and *Lin-Kernighan (LK)* [LK73]. These heuristics define neighbourhood structures which can be searched by the different neighbourhood search schemes described in sections 1.3 and 2.8, leading to many local optimisation algorithms for the TSP. The neighbourhood structures defined by 2-Opt, 3-Opt and LK are as follows [Joh90]:

2-Opt. A neighbouring solution is obtained from the current solution by deleting two edges, reversing one of the resulting paths and reconnecting the tour (see Figure 3.1). The worst case complexity for searching the neighbourhood defined by 2-Opt is $O(n^2)$.

3-Opt. In this case, three edges are deleted. The three resulting paths are put together in a new way, possibly reversing one or more of them (see Figure 3.1). 3-Opt is much more effective than 2-Opt, though the size of the neighbourhood (possible 3-Opt moves) is larger and hence more time-consuming to search. The worst case complexity for searching the neighbourhood defined by 3-Opt is $O(n^3)$.



a) 2-Opt move **b)** 3-Opt move **c)** Non-sequential 4-Opt move

Figure 3.1 k-Opt moves for the TSP

Lin-Kernighan (LK). One would expect "4-Opt" to be the next step after 3-Opt but actually that is not the case. The reason is that 4-Opt neighbours can be remotely apart because "non-sequential" exchanges such as that shown in Figure 3.1 are possible for $k \geq 4$. To improve 3-Opt further, Lin and Kernighan developed a sophisticated edge exchange procedure where the number k of edges to be exchanged is variable [LK73]. The algorithm is mentioned in the literature as the *Lin-Kernighan* (LK) algorithm and it was considered for many years to be the "uncontested champion" of local search heuristics for the TSP. Lin-Kernighan uses a very complex neighbourhood structure which we will briefly describe here.

LK, instead of examining a particular 2-Opt or 3-Opt exchange, is building an exchange of variable size *k* by sequentially deleting and adding edges to the current tour while maintaining tour feasibility. Given node t_1 in tour *T* as a starting point: In step *m* of this sequential building of the exchange: edge (t_1, t_{2m}) is deleted, edge (t_{2m}, t_{2m+1}) is added, and then edge (t_{2m+1}, t_{2m+2}) is picked so that deleting edge (t_{2m+1}, t_{2m+2}) and joining edge (t_{2m+2}, t_1) will close up the tour giving tour T_m. The edge (t_{2m+2}, t_1) is deleted if and when step *m+1* is executed. The first three steps of this mechanism are illustrated in Figure 3.2.

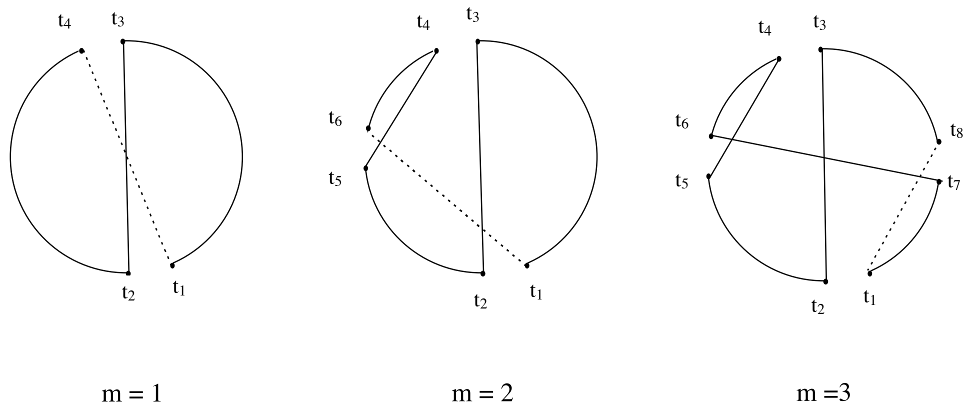


Figure 3.2 The first three steps of the Lin-Kernighan edge exchange mechanism

As we can see in this figure, the method is essentially executing a sequence of 2-Opt moves. The length of these sequences (i.e. depth of search) is controlled by the LK's *gain criterion* which limits the number of the sequences examined. In addition to that, limited backtracking is used to examine the sequences that can be generated if a number of different edges are selected for addition at steps 1 and 2 of the process.

The neighbourhood structure described so far, although it provides the depth needed, is lacking breadth, potentially missing improving 3-Opt moves. To gain breadth, LK temporarily allows tour infeasibility, examining the so-called "infeasibility" moves which consider various choices for nodes t_4 to t_8 in the sequence generation process, examining all possible 3-Opt moves and more. Figure 3.3 illustrates the infeasibility-move mechanism. The interested reader may refer to the original paper by Lin and Kernighan for a more elaborate description of this mechanism.

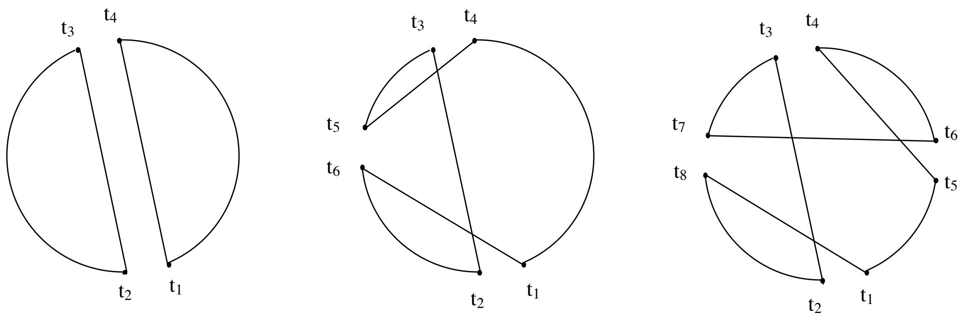


Figure 3.3 Lin-Kerhighan's infeasibility moves

LK is the standard benchmark against which all heuristic methods are tested. The worst case complexity for searching the LK neighbourhood is $O(n^5)$.

Implementations of 2-Opt, 3-Opt and LK-based local search methods may vary in performance. A very good reference for efficiently implementing local search procedures based on 2-Opt and 3-Opt is that by Bentley [Ben92]. In addition to that, Reinelt [Rei94] and also Johnson and McGeoch [JM95] describe some improvements that are commonly incorporated in local search algorithms for the TSP. We will refer to some of them later in this chapter. The best reference for the LK algorithm is the original paper by Lin and Kernighan [LK73]. In addition to that, Johnson and McGeoch [JM95] provide a good insight into the algorithm and its operations along with information on the many variants of the method. A modified LK version which avoids the complex infeasibility moves without significant impact on performance is described in [MM93].

Fast local search and guided local search can be combined with the neighbourhood structures of 2-Opt, 3-Opt and LK with minimal effort. This will become evident in the next sections where fast local search and guided local search for the TSP are presented and discussed.

3.3 Fast Local Search Applied to the TSP

A fast local search procedure for the TSP using 2-Opt has already been suggested by Bentley [Ben92]. Under the name *Don't Look Bits*, the same approach has been used in the context of 2-Opt, 3-Opt and LK by Codenotti et al. [CMMR96] to reduce the running times of these heuristics in very large TSP instances. More recently, Johnson et al. [JBMR96] also use the technique to speed up their LK variant (see [JM95]). In the following, we are going to describe how fast local search variants of 2-Opt, 3-Opt and LK can be developed on the guidelines for fast local search presented in section 2.8.

2-Opt, 3-Opt and LK-based local search procedures are seeking tour improvements by considering for exchange each individual edge in the current tour and trying to extend this exchange to include one (2-Opt), two (3-Opt) or more (LK) other edges from the tour. Usually, each city is visited in tour order and **one** or **both**[3] the edges adjacent to the city are checked if they can lead to an edge exchange which improves the solution. We can exploit the way local search works on the TSP to partition the neighbourhood in sub-neighbourhoods as required by fast local search. Each city in the problem may be seen as defining a sub-neighbourhood which contains all edge exchanges

[3] In our work, if approximations are used such as nearest neighbour lists or fast local search then both edges adjacent to a city are examined, otherwise only one of the edges adjacent to the city is examined.

originating from either one of the edges adjacent to the city. For a problem with *N* cities, the neighbourhood is partitioned into *N* sub-neighbourhoods, one for each city in the instance. Given the sub-neighbourhoods, fast local search for the TSP works in the following way (see also Figure 2.2).

Initially all sub-neighbourhoods are active. The scanning of the sub-neighbourhoods, defined by the cities, is done in an arbitrary static order (e.g. from 1st to *N*th city). Each time an active sub-neighbourhood is found, it is searched for improving moves. This involves trying either edge adjacent to the city as bases for 2-Opt, 3-Opt or LK edge exchanges, depending on the heuristic used. If a sub-neighbourhood does not contain any improving moves then it becomes inactive (i.e. bit is set to 0). Otherwise, the first improving move found is performed and the cities (corresponding sub-neighbourhoods) at the ends of the edges involved (deleted or added by the move) are activated (i.e. bits are set to 1). This causes the sub-neighbourhood where the move was found to remain active and also a number of other sub-neighbourhoods to be activated. The process always continues with the next sub-neighbourhood in the static order. If ever a full rotation around the static order is completed without making a move, the process terminates and returns the tour found. The tour is declared 2-Optimal, 3-Optimal or LK-Optimal, depending on the type of the *k*-Opt moves used.

3.3.1 Local Search Procedures for the TSP

Apart from fast local search, first improvement and best improvement local search (see section 1.3) can also be applied to the TSP. First improvement local search immediately performs improving moves while best improvement (greedy) local search performs the best move found after searching the complete neighbourhood.

Fast local search for the TSP described above can be easily converted to first improvement local search by searching all sub-neighbourhoods irrespective of their state (active or inactive). The termination criterion remains the same with fast local search: that is, to stop the search when a full rotation of the static order is completed without making a move. The LK algorithm as originally proposed by Lin and Kernighan [LK73] performs first improvement local search.

Fast local search can also be modified to perform best improvement local search. In this case, the best move is selected and performed after all the sub-neighbourhoods have been exhaustively searched. The algorithm stops when a solution is reached where no improving move can be found. The scheme is very time consuming to be combined with the 3-Opt and LK neighbourhood structures and it is mainly intended for use with 2-Opt. Considering the above options, we implemented seven local search variants for the TSP (implementation details will be given later in this chapter). These variants were derived by combining the different search schemes at the neighbourhood level (i.e. fast, first improvement, and best improvement local search) with any of the 2-Opt, 3-Opt, or LK neighbourhood structures. Table 3.1 illustrates the variants and also the names we will use to distinguish them in the rest of the chapter.

Name	**Local Search Type**	**Neighbourhood Type**
BI-2Opt	Best Improvement	2-Opt
FI-2Opt	First Improvement	2-Opt
FLS-2Opt	Fast Local Search	2-Opt
FI-3Opt	First Improvement	3-Opt
FLS-3Opt	Fast Local Search	3-Opt
FI-LK	First Improvement	LK
FLS-LK	Fast Local Search	LK

Table 3.1 Local search procedures implemented for the study of GLS on the TSP.

3.4 Guided Local Search Applied to the TSP

3.4.1 Solution Features and Augmented Cost Function

The first step in the process of applying GLS to a problem is to find a set of solution features that are accountable for part of the overall solution cost. For the TSP, a tour includes a number of edges and the solution cost (tour length) is given by the sum of the lengths of the edges in the tour (see Eq. 3.1). Edges are ideal features for the TSP. First, they can be used to define solution properties (a tour either includes an edge or not) and second, they carry a cost equal to the edge length, as this is given by the distance matrix $D=[d_{ij}]$ of the problem. A set of features can be defined by considering all possible undirected edges e_{ij} ($i = 1..N$, $j = i+1..N$, $i \neq j$) that may appear in a tour with feature costs given by the edge lengths d_{ij}. Each edge e_{ij} connecting cities i and city j is attached a penalty p_{ij} initially set to 0 which is increased by GLS during search. These edge penalties can be arranged in a symmetric penalty matrix $P=[p_{ij}]$. As mentioned in section 2.5, penalties have to be combined with the problem's cost function to form the augmented cost function which is minimised by local search. This can be done by considering the auxiliary distance matrix:

Eq. 3.2 $$D' = D + \lambda \cdot P = [d_{ij} + \lambda \cdot p_{ij}] .$$

Local search must use D' instead of D in move evaluations. GLS modifies P and (through that) D' whenever local search reaches a local minimum. The edges penalised in a local minimum are selected according to the utility function (Eq. 2.5), which for the TSP takes the form:

Eq. 3.3

$$Util\left(tour, e_{ij}\right) = I_{e_{ij}}(tour) \cdot \frac{d_{ij}}{1 + p_{ij}},$$

where

Eq. 3.4

$$I_{e_{ij}}(tour) = \begin{cases} 1, & e_{ij} \in tour \\ 0, & e_{ij} \notin tour \end{cases}.$$

3.4.2 Combining GLS with TSP Local Search Procedures

GLS as depicted in Figure 2.1 makes no assumptions about the internal mechanisms of local search and therefore can be combined with any local search algorithm for the problem, no matter how complex this algorithm is.

The TSP local searches of section 3.3.1 to be integrated with GLS need only to be implemented as procedures which, provided with a starting tour, return a locally optimal tour with respect to the neighbourhood considered. The distance matrix used by local search is the auxiliary matrix D' described in the last section. A reference to the matrix D is still needed to enable the detection of better solutions whenever moves are executed and new solutions are visited. There is no need to keep track of the value of the augmented cost function since local search heuristics make move evaluations using cost differences rather than re-computing the cost function from scratch.

Interfacing GLS with fast local searches for the TSP requires a little more effort (see also Figure 2.2). In particular, each time we penalise an edge in GLS, the sub-neighbourhoods corresponding to the cities at the ends of this edge are activated (i.e. bits set to 1). After the first local minimum, calls to fast local search start by examining only a number of sub-neighbourhoods and in particular those which associate to the edges just penalised. Activation may spread to a limited number of other sub-neighbourhoods because of the moves performed though, in general, local search quickly settles in a new local minimum. This dramatically speeds up GLS, forcing local search to focus on edge exchanges that remove penalised edges instead of evaluating all possible moves.

3.4.3 How GLS Works on the TSP

Let us now give an overview of the way GLS works on the TSP. Starting from an arbitrary solution, local search is invoked to find a local minimum. GLS penalises one or more of the edges appearing in the local minimum, using the utility function Eq. 3.3 to select them. After the penalties have been increased, local search is restarted from the last local minimum to search for a new local minimum. If we are using fast local search then the sub-neighbourhoods (i.e. cities) at the ends of the edges penalised need also to be activated. When a new local minimum is found or local search cannot escape from the current local minimum, penalties are increased again and so forth.

The GLS algorithm constantly attempts to remove edges appearing in local minima by penalising them. The effort invested by GLS to remove an edge depends on the edge length. The longer the edge, the greater the effort put in by GLS. The effect of this effort depends on the regularisation parameter λ of GLS. A high λ causes GLS decisions to be in full control of local search, overriding any local gradient information while a low λ causes GLS to escape from local minima with great difficulty, requiring many penalty cycles before a move is executed. However, there is always a range of values for λ for which the moves selected aim at the combined objective to improve the solution (taking into account the gradient) and also remove the penalised edges (taking into account the GLS decisions). If longer edges persist in appearing in solutions despite the penalties, the algorithm will diversify its choices, trying to remove shorter edges too.

As the penalties build up for both bad and good edges frequently appearing in local minima, the algorithm starts exploring new regions in the search space, incorporating edges not previously seen and therefore not penalised. The speed of this “continuous” diversification of search is controlled by the parameter λ. A low λ slows down the diversification process, allowing the algorithm to spend more time in the current area before it is forced by the penalties to explore other areas. Conversely, a high λ speeds up diversification, at the expense of intensification.

From another viewpoint, GLS realises a “selective” diversification which pursues many more choices for long edges than short edges by penalising the former many more times than the later. This selective diversification achieves the goal of distributing the search effort according to prior information as expressed by the edge lengths. Selective diversification is smoothly combined with the goal of intensifying search by setting λ to a value low enough to allow the local search gradients to influence the course of local search. Escaping from local minima comes at no expense because of the penalties but alone without the goal of distributing the search effort, as implemented by the selective penalty modification mechanism, is not enough to produce high quality solutions.

3.5 Evaluation of GLS in the TSP

To investigate the behaviour of GLS on the TSP, we conducted a series of experiments. The results presented in subsequent sections attempt to provide a comprehensive picture of the performance of GLS on the TSP. First, we examine the combination of GLS with 2-Opt, the simplest of the TSP heuristics. The benefits from using fast local search instead of best improvement local search are clearly demonstrated, along with the ability of GLS to find high quality solutions in small to medium size problems. These results for GLS are compared with results for Simulated Annealing and Tabu Search when these techniques use the 2-Opt heuristic.

From there on, we focus on efficient techniques for the TSP based on GLS. The different combinations of GLS with the local search procedures of Table 3.1 are examined and conclusions are drawn on the relation between GLS and local search. Efficient GLS variants are compared with methods based on the Lin-Kernighan algorithm (known to be the best heuristic techniques for the TSP).

3.5.1 Experimental Setting

In the experiments conducted, we used problems from the publicly available library of TSP problems, TSPLIB [Rei91]. Most of the instances included in TSPLIB have already been solved to optimality and they have been used in many papers in the TSP literature.

For each algorithm evaluated, ten runs from different **random** initial solutions were performed and the various performance measures (solution quality, running time etc.) were averaged. The solution quality was measured by the percentage *excess* above the best known solution (or optimal solution if known), as given by the formula:

Eq. 3.5
$$excess = \frac{\text{solution cost - best known solution cost}}{\text{best known solution cost}} \times 100 .$$

Unless otherwise stated, all experiments were conducted on DEC Alpha 3000/600 machines (175 MHz) with algorithms implemented in GNU C++.

3.5.2 Regularisation Parameter λ

The only parameter of GLS which requires tuning is the regularisation parameter λ. The GLS algorithm performed well for a relatively wide range of values when we tested it on problems from TSPLIB with either one of the 2-Opt, 3-Opt or LK heuristics. Experiments showed that GLS is quite tolerant to the choice of λ as long as λ is equal to a fraction of the average edge length in good solutions (e.g. local minima). These findings were expressed by the following equation for calculating λ:

Eq. 3.6
$$\lambda = a \cdot \frac{g(\text{local minimum})}{N} ,$$

where *g(local minimum)* is the cost of a local minimum tour produced by local search (e.g. first local minimum before penalties are applied) and N the number of cities in the instance. Eq. 3.6 introduces a parameter a which, although instance-dependent, results in good GLS performance for values in the more manageable range (0,1]. Experimenting with *a,* we found that it depends not only on the instance but also on the local search heuristic used. In general, there is an inverse relation between a and local search effectiveness. Not-so-effective local search heuristics such as 2-Opt require higher a values than more effective heuristics such as 3-Opt and LK. This is because the amount of penalty needed to escape from local minima decreases as the effectiveness of the heuristic increases and therefore lower values for a have to be used to allow the local gradients to affect the GLS decisions. For 2-Opt, 3-Opt and LK, the following ranges for a generated high quality solutions in the TSPLIB problems.

Heuristic	**Suggested range for *a***
2-Opt	$1/8 \leq a \leq ½$
3-Opt	$1/10 \leq a \leq ¼$
LK	$1/12 \leq a \leq 1/6$

Table 3.2 Suggested ranges for parameter a when GLS is combined with different TSP heuristics.

The lower bounds of these intervals represent typical values for a that enable GLS to escape from local minima at a *tolerable* rate. If values less than the lower bounds are used, then GLS requires too many penalty cycles to escape from local minima. In general, the lower bounds depend on the local search heuristic used and also the structure of the landscape (i.e. depth of local minima). On the other hand, the upper bounds give a good indication of the maximum values for *a* that can still produce good solutions. If values greater than the upper bounds are used then the algorithm is exhibiting excessive bias towards removing long edges and failing to reach high quality local minima. In general, the upper bounds also depend on the local search heuristic used but they are mainly affected by the quality of the information contained in the feature costs (i.e. how accurate is the assumption that long edges are preferable over short edges in the particular instance).

3.6 Guided Local Search and 2-Opt

In this section, we look into the combination of GLS with the simple 2-Opt heuristic. More specifically, we present results for GLS with best improvement 2-Opt local search (BI-2Opt) and fast 2-Opt local search (FLS-2Opt). The set of problems used in the experiments consisted of 28 small to medium size TSPs from 48 to 318 cities all from TSPLIB. The stopping criterion used was a limit on the number of iterations not to be exceeded. An iteration for GLS with BI-2Opt was considered one local search iteration (i.e. complete search of the neighbourhood) and for GLS with FLS-2Opt, a call to fast local search as in Figure 2.2. The iteration limit for both algorithms was set to 200,000 iterations. In both cases, we tried to provide the GLS variants with plenty of resources in order to reach the maximum of their performance.

The exact value of λ used in the runs was manually determined by running a number of test runs and observing the sequence of solutions generated by the algorithm. A well-tuned algorithm generates a smooth sequence of gradually improving solutions. A not so well tuned algorithm either progresses very slowly (λ is lower than it should

Problem	GLS with BI-2Opt			GLS with FLS-2Opt		
	optimal runs out of 10	Mean Excess (%)	Mean CPU Time (sec)	optimal runs out of 10	Mean Excess(%)	Mean CPU Time (sec)
att48	10	0.0	0.77	10	0.0	0.4
eil51	10	0.0	1.62	10	0.0	0.46
st70	10	0.0	7.68	10	0.0	1.2
eil76	10	0.0	3.83	10	0.0	0.97
pr76	10	0.0	15.1	10	0.0	3.01
gr96	10	0.0	16.48	10	0.0	2.26
kroA100	10	0.0	11.27	10	0.0	1.25
kroB100	10	0.0	16.36	10	0.0	2.46
kroC100	10	0.0	12.2	10	0.0	0.74
kroD100	10	0.0	12.94	10	0.0	1.78
kroE100	10	0.0	35.68	10	0.0	2.46
rd100	10	0.0	10.75	10	0.0	2.74
eil101	10	0.0	19.49	10	0.0	2.37
lin105	10	0.0	17.46	10	0.0	2.06
pr107	10	0.0	150.28	10	0.0	5.41
pr124	10	0.0	22.47	10	0.0	1.56
bier127	10	0.0	254.36	10	0.0	24.67
pr136	9	0.0009	416.78	10	0.0	32.16
gr137	10	0.0	66.54	10	0.0	7.82
pr144	10	0.0	52.84	10	0.0	6.95
kroA150	10	0.0	257.06	10	0.0	7.03
kroB150	10	0.0	289.02	10	0.0	44.85
u159	10	0.0	74.35	10	0.0	6.9
rat195	8	0.01	525.48	10	0.0	55.15
d198	0	0.08	1998.37	0	0.05	353.97
kroA200	10	0.0	614.6	10	0.0	50.16
kroB200	10	0.0	665.3	10	0.0	61.79
lin318	8	0.01	4484.4	9	0.005	346.44

Table 3.3 Performance of 2-Opt based variants of GLS on small to medium size TSP instances.

be) or very quickly finds no more than a handful of good local minima (λ is higher than it should be). The values for λ determined in this way were corresponding to values for *a* around 0.3. Ten runs from different random solutions were performed on each instance included in the set of problems and the various performance measures (excess, running time to reach the best solution etc.) were averaged. The results obtained are presented in Table 3.3.

Both GLS variants found solutions with cost equal to the optimal cost in the majority of runs. GLS with BI-2Opt failed to find the optimal solutions (as reported by Reinelt in [Rei91] and also [Rei94]) in only 15 out of the total 280 runs. From another viewpoint, the algorithm was successful in finding the optimal solution in 94.6% of the runs. Ten out of the 14 failures referred to a single instance namely *d198*. However, the solutions found for *d198* were of high quality and on average within 0.08% of optimality.

GLS with FLS-2Opt found the optimal solutions in 3 more runs than GLS with BI-2Opt, missing the optimal solution in only 11 out of the 280 runs (96.07% success rate). In particular, the algorithm missed only once the optimal solution for *lin318* but still found no optimal solution for *d198* which proved to be a relatively 'hard' problem for both variants. GLS using fast local search was on average ten times faster than GLS using best improvement local search and that without compromising on solution quality. In the worst case (*att48*), it was two times faster while in the best case (*kroA150*) it was thirty seven times faster. Remarkably, GLS with fast local search was able in most problems to find a solution with cost equal to the optimum (already known) in less than 10 seconds of CPU time on the DEC Alpha 3000/600 machines used.

The results presented in this section clearly demonstrate the ability of GLS even when combined with 2-Opt the simplest of TSP heuristics to find consistently the optimal solutions for small to medium size TSPs. The use of fast local search introduces substantial savings in running times without compromising in solution quality.

3.6.1 Comparison with General Methods for the TSP

The above performance of GLS is remarkable considering that GLS is not an exact method and that in this case it only used the short-sighted 2-Opt heuristic. Searching the related TSP literature, we could not find any other approximation methods that use only the simple 2-Opt move and consistently find optimal solutions for problems up to

318 cities. Only the Iterated Lin-Kernighan algorithm and its variants [Joh90, JM95, JBMR96] share the same consistency in reaching the optimal solutions. These algorithms will be considered later in this chapter.

A meaningful comparison that can be made is between GLS using 2-Opt and other general methods that also use the same heuristic. For that purpose, we implemented simulated annealing [KGV83] and a tabu search variant for the TSP suggested by Knox [Kno94].

3.6.2 Simulated Annealing

The Simulated Annealing (SA) algorithm implemented for the TSP was the one described by Johnson in [Joh90] and uses geometric cooling schedules (see section 1.4.1). The algorithm generates random 2-Opt moves. If a move improves the cost of the current solution then it is always accepted. Moves that do not improve the cost of the current solution are accepted with probability:

$$e^{\frac{-\Delta}{T}}$$

where Δ is the difference in cost due to the move and T is the current temperature. In the final runs, we started the algorithm from a relatively high temperature (around 50% of moves were accepted). At each temperature level the algorithm was allowed to perform a constant number of trials to reach equilibrium. After reaching equilibrium, the temperature was multiplied by the cooling rate a which was set to a high value ($a = 0.9$) . To stop the algorithm, we used the scheme with the counter described in [JAMS89].

3.6.3 Tabu Search

The tabu search variant implemented was the one proposed by Knox [Kno94] using a combination of *tabu restrictions* and *aspiration level criteria*. The method is briefly described in here.

Tabu search performs best improvement local search selecting the best move in the neighbourhood but only amongst those not characterised as *tabu*. Determining the tabu status of a move is very important in tabu search and holds the key for the development of efficient recency-based memory (see section 1.5).

In this tabu search variant for the TSP, a 2-Opt move is classified as tabu only if both added edges of the exchange are on the tabu list. If one or both of the added edges are not on the tabu list, then the candidate move is not classified as tabu. Updating the tabu list involves placing the deleted edges of the 2-Opt exchanges performed on the list. If the list is full, the oldest elements of the list are replaced by the new deleted edge information.

In order for a 2-Opt exchange to override tabu status, both added edges of the exchange must pass the aspiration test. An individual edge passes the aspiration test if the new tour resulting from the candidate exchange is better than the aspiration values associated with the edge. The aspiration values of edges are the tour cost which exists prior to making the candidate 2-Opt move. Only edges deleted by the exchanges performed have their values updated.

For the experiments reported here, the tabu list size was set to $3N$ (where N is the number of cities in the problem) as suggested by Knox [Kno94]. Tabu search was allowed to run for 200,000 iterations which is equivalent in terms of number of moves evaluated to the number of iterations GLS with BI-2Opt was given on the same instances.

Problem Name	GLS with FLS-2Opt		Simulated Annealing		Tabu Search		Repeated BI-2Opt (200,000 iterations)	
	Mean Excess (%)	Mean CPU Time (sec)	Mean Excess (%)	Mean CPU Time (sec)	Mean Excess (%)	Mean CPU Time (sec)	Mean Excess (%)	Mean CPU Time (sec)
eil51	0.0	0.46	0.73	6.34	0.0	1.14	0.23	42.4
eil76	0.0	0.97	1.21	18.0	0.0	5.24	1.85	153.45
eil101	0.0	2.37	1.76	33.29	0.0	61.41	3.97	319.15
kroA100	0.0	1.25	0.42	37.36	0.0	21.34	0.34	706.35
kroC100	0.0	0.74	0.80	36.58	0.25	4.80	0.33	1301.98
kroA150	0.0	7.03	1.86	103.32	0.03	413.06	1.41	3290.95
kroA200	0.0	50.16	1.04	229.38	0.72	776.93	1.7	731.1
lin318	0.005	346.44	1.34	829.46	1.31	2672.80	3.11	9771.28

Table 3.4 GLS, Simulated Annealing, and Tabu Search performance on TSPLIB instances.

3.6.4 Simulated Annealing and Tabu Search Compared with GLS

Simulated annealing and tabu search were tested on 8 instances from the greater set of 28 instances mentioned above. The results were averaged as with GLS. Table 3.4 illustrates the results for simulated annealing and tabu search compared with those for GLS with FLS-2Opt on the same instances. Results are also contrasted with the best solution found by repeating BI-2Opt starting from random tours until a total of 200,000 local search iterations were completed.

As we can see in Table 3.4, the superiority of GLS over the tabu search variant and simulated annealing is evident. The tabu search variant found easily the optimal solutions for small problems and it scaled well for larger problems. However, it was many times slower than GLS and moreover failed to reach the solution quality of GLS in the larger problems. Simulated annealing had a consistent behaviour finding good solutions for all problems but failed to reach the optimal solutions in all but 3 runs. All three meta-heuristics significantly improved over the performance of repeated 2-Opt.

3.7 Efficient GLS Variants for the TSP

In order to study the combinations of GLS with higher order heuristics such as 3-Opt and LK, a library of TSP local search procedures was developed in C++. The library comprises all local search procedures of Table 3.1 and allows combinations of GLS with any one of these procedures. Furthermore, a number of approximations (not used in the GLS of section 3.6) are adopted which further reduce the computation times of local search and GLS as reported in section 3.6. In the rest of the chapter, we will examine and report results for these efficient variants of GLS.

The most significant approximation introduced is the use of a pre-processing stage which finds and sorts by distance the 20 nearest neighbours of each city in the instance. 2-Opt, 3-Opt and LK were considering in exchanges only edges to these 20 nearest neighbours (see also [Rei94, JM95]). Each time the penalty was increased for an edge, the nearest neighbour lists of the cities at the ends of the edge were reordered though no new neighbours were introduced.

To reduce the computation times required by 3-Opt, 3-Opt was implemented as two locality searches each of which looks for a “short enough” edge to extend further the exchange (see [Ben92] for details). The LK implementation was exactly as proposed by Lin and Kernighan [LK73] incorporating their lookahead and backtracking suggestions (i.e. backtracking at the first two levels of the sequence generation, considering at each step only the five smallest and available candidate edges that can be added to the tour and taking into account in the selection of the edges to be added the length of the edges to be deleted by these additions).

The library is portable to most UNIX machines though experiments reported in here were solely performed on DEC Alpha workstations 3000/600 (175 MHz) using a library executable generated by the GNU C++ compiler.

The set of problems used in the evaluation of the GLS variants included 20 problems from 48 to 1002 cities all from TSPLIB. For each variant tested, 10 runs were performed and 5 minutes of CPU time were allocated to each algorithm in each run. To measure the success of the variants, we considered the percentage excess above the optimal solution as in Eq. 3.5. The normalised lambda parameter a was provided as input to the program and λ was determined after the first local minimum using Eq. 3.6. For GLS variants using 2-Opt, a was set to a = 1/6 while the GLS variants based on 3-Opt used the slightly lower value a = 1/8 and the LK variants the even lower value a = 1/10. The full set of results for the various combinations of GLS with local search can be found in Appendix A. Next, we focus on selected results from this set.

3.7.1 Results for GLS with First Improvement Local Search

Figure 3.4 graphically illustrates the results for the first improvement versions of 2-Opt, 3-Opt and LK when combined with GLS. In this figure, we see that the

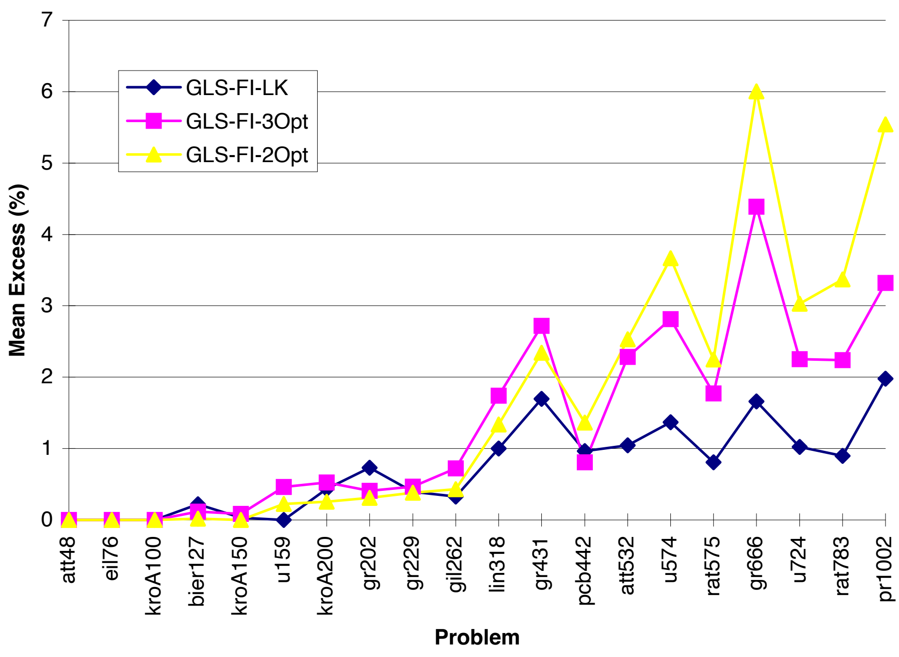


Figure 3.4 Performance of GLS variants using first improvement local search procedures

combination of GLS with FI-3Opt and FI-LK significantly improves over the performance of GLS with FI-2Opt especially when applied to large problems. FI-LK combined with GLS achieved the best performance amongst the three methods tested.

3.7.2 Results for GLS with Fast Local Search

Figure 3.5 graphically illustrates the results obtained for GLS when combined with the fast local search variants of 2-Opt, 3-Opt and LK. GLS with FI-LK (found to be best amongst the first improvement versions of GLS) is also displayed in the figure as a point of reference. In this figure, we can see that the fast local search variants of GLS are much better than the best of the first improvement local search variants (i.e. GLS-FI-LK). Another far more important observation is that for fast local search the 2-Opt variant is better than the 3-Opt variant which in turn is better than the LK variant. This is exactly the opposite order than one would have expected. One possible explanation can be derived by considering the strength of GLS. More specifically, FLS-2Opt allows GLS to perform many more penalty cycles in the time given than its

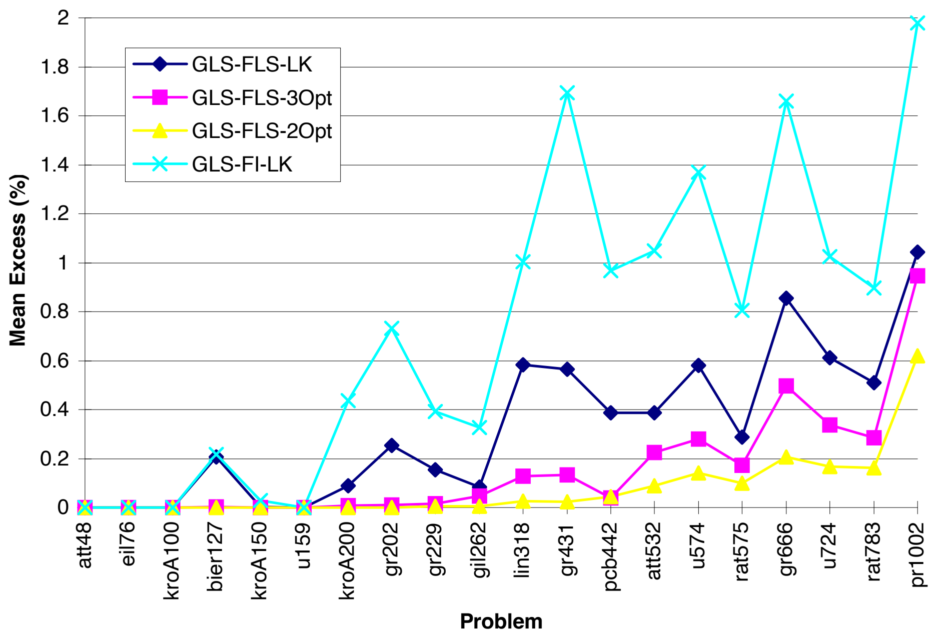


Figure 3.5 Performance of GLS variants using fast local search procedures

FLS-3Opt or FLS-LK counterparts. More GLS penalty cycles seem to increase efficiency at a level which outweighs the benefits from using a more sophisticated local search procedure such as 3-Opt or LK.

The remarkable effects of GLS on local search are further demonstrated in Figure 3.6 where GLS with FLS-2Opt is compared against Repeated FLS-2Opt and Repeated FI-LK. In Repeated FLS-2Opt and Repeated FI-LK, local search is simply restarted from a random solution after a local minimum and the best solution found over the many runs is returned. These two algorithms along with other versions of repeated local search were tested under the same settings with the GLS variants. Appendix A includes the full set of results for repeated local search. In Figure 3.6, we can see the huge improvement in the basic 2-Opt heuristic when this is combined with GLS. GLS is the only technique known to us which when applied to 2-Opt can outperform the Repeated LK algorithm (and that without requiring excessive amounts of CPU time) as illustrated in the same figure.

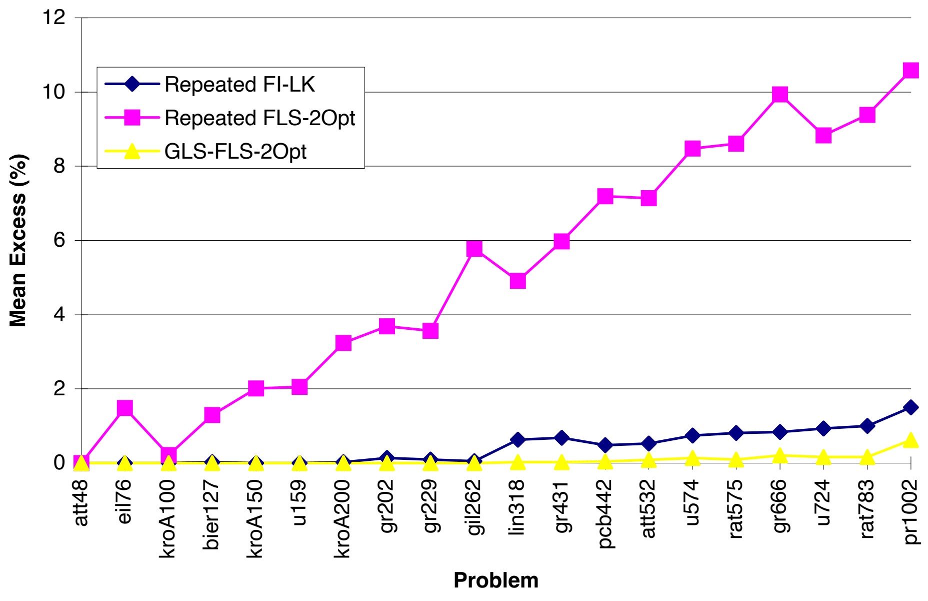


Figure 3.6 Improvements introduced by the application of GLS to the simple FLS-2Opt

3.8 Comparison with Specialised TSP algorithms

3.8.1 Iterated Lin-Kernighan

The *Iterated Lin-Kernighan* algorithm (not to be confused with Repeated LK) has been proposed by Johnson [Joh90] and it is considered to be one of the best if not the best heuristic algorithm for the TSP [JM95]. Iterated LK uses LK to obtain a first local minimum. To improve this local minimum, the algorithm examines other local minimum tours "near" the current local minimum. To generate these tours, Iterated LK first applies a random and unbiased non-sequential 4-Opt exchange (see Figure 3.1) to the current local minimum and then optimises this 4-Opt neighbour using the LK algorithm. If the tour obtained by the process (i.e. random 4-Opt followed by LK) is better than the current local minimum then Iterated LK makes this tour the current local minimum and continues from there using the same neighbour generation process. Otherwise, the current local minimum remains as it is and further random 4-Opt moves are tried. The algorithm stops when a stopping criterion based either on the number of iterations or computation time is satisfied. Figure 3.7 contains the original description of the algorithm as given in [Joh90].

```
1. Generate a random tour $T$.
2. Do the following for some prespecified number $M$ of iterations:
        2.1. Perform an (unbiased) random 4-Opt move on $T$, obtaining $T'$.
        2.2. Run Lin-Kernighan on $T'$, obtaining $T''$.
        2.3. If length($T''$) $\leq$ length ($T'$), set $T = T''$.
3. Return $T'$.
```

Figure 3.7 Iterated Lin-Kernighan as described by Johnson in [Joh90]

The random 4-Opt exchange performed by Iterated LK is mentioned in the literature as the "double-bridge" move and plays a diversification role for the search process, trying to propel the algorithm to a different area of the search space preserving at the same time large parts of the structure of the current local minimum. Martin et al. [MOF92] describe this action as a "kick" and show that can be also used with 3-Opt in the place of LK. The same authors also suggest the combination of the method with Simulated Annealing (Long Markov Chains method). Martin and Otto [MO96] further demonstrate the efficiency of this last algorithm on the TSP and also the Graph Partitioning problem though they admit that simulated annealing does not significantly improve the method for TSP problems up to 783 cities. Finally, Johnson and McGeoch [JM95] review Iterated LK and its variants and provide results for both structured and random TSP instances.

Iterated LK or Iterated 3-Opt share some of the principles of GLS in the sense that they produce a sequence of diversified local minima though this is conducted in a random rather than a systematic way. Furthermore, iterated local search accepts the new solution, produced by the 4-Opt exchange and the subsequent LK or 3-Opt optimisation, only if it improves over the current local minimum (or it is slightly worse in the case of Large Markov Chains Method which uses simulated annealing) .

Iterated LK outperforms Repeated LK previously thought to be the "champion" of TSP heuristics and also long simulated annealing runs [MO96]. More recent experiments show that even sophisticated tabu search variants of LK cannot improve over Iterated LK [ZD95] which rightly deserves the title of the "champion" of TSP meta-heuristics.

To compare Iterated LK and its other variants such as Iterated 3-Opt with GLS, we extended our C++ library mentioned above to allow the iterated local search scheme to be combined with the local search procedures of Table 3.1 included in the library. In particular, a random and unbiased Double-Bridge (DB) move was performed in a local minimum. The solution obtained was optimised by either one of the procedures of Table 3.1 before compared against the current local minimum. The new solution was accepted only if it improved over the current local minimum. To combine iterated local search with fast local search procedures, we activated the sub-neighbourhoods corresponding to the cities at the ends of the edges involved in the Double-Bridge move (see also [CMMR96]). The above extensions to the library made available a general meta-heuristic method applicable to all the local search procedures of Table 3.1. We will refer to this method as the Double-Bridge (DB) meta-heuristic.

We tested all the possible combinations of the DB meta-heuristic with the local searches of Table 3.1 (except for BI-2Opt) on the set of 20 problems used to test the GLS combinations. The same time limit (5 minutes of CPU time on DEC Alpha 3000/600 machines) was used and ten runs were performed on each instance in the set. The percentage excess was averaged in each problem for each DB variant. The best combination proved to be that of the DB heuristic with FLS-LK which outperformed DB with FI-LK (this last algorithm is roughly the same with the original method proposed by Johnson [Joh90]). The results for the various combinations of DB with local search are included in Appendix A.

Problem	Mean Excess (%) over 10 runs			
	GLS with FLS-2Opt	DB with FLS-LK	DB with FI-LK	Repeated FI-LK
att48	0	0	0	0
eil76	0	0	0	0
kroA100	0	0	0	0
bier127	0	0	0	0.0301
kroA150	0	0	0	0.00226
u159	0	0	0	0
kroA200	0	0	0	0.02452
gr202	0	0	0.00921	0.14143
gr229	0.00431	0.00475	0.01412	0.0977
gil262	0.00421	0	0.01682	0.05467
lin318	0.02641	0.24079	0.25578	0.62957
gr431	0.02392	0.22239	0.3327	0.67964
pcb442	0.04431	0.08173	0.06637	0.48525
att532	0.08994	0.08163	0.22502	0.53023
u574	0.14144	0.0924	0.11435	0.73838
rat575	0.09892	0.09745	0.13731	0.80762
gr666	0.20628	0.17587	0.41888	0.83762
u724	0.16822	0.16655	0.35696	0.93367
rat783	0.16125	0.15331	0.24075	1.00045
pr1002	0.62063	0.44633	1.04742	1.5046
Average Excess	**0.07949**	**0.08816**	**0.16178**	**0.42488**

Table 3.5 GLS with FLS-2Opt compared with variants of Iterated Lin-Kernighan.

Table 3.5 presents the results obtained for DB with FLS-LK and DB with FI-LK compared with those for GLS with FLS-2Opt found to be the best GLS variant. As a point of reference, we also provide results for FI-LK when repeated from random starting points and for the same amount of time. As we can see in Table 3.5, GLS with FLS-2Opt is better on average than both DB with FLS-LK and DB with FI-LK. The solution quality improvement over these methods although small it is very significant given that these methods are amongst the best heuristic techniques for the TSP. Note here that GLS with FLS-2Opt is by far a simpler method requiring only a fraction of the programming effort required to develop the DB variants based on LK.

To further test GLS against the DB variants of LK, we used a set of 66 TSPLIB problems from 48 to 2392 cities but this time we performed longer runs lasting 30 minutes of CPU time each. This amount of time on the DEC Alpha machines used translates to many hours of CPU time on an average PC where most of these algorithms are most likely to be utilised. Because of the large number of instances used and the long time the algorithms were allowed to run, one run was performed on each instance. The results from the experiments are presented in Table 3.6.

Even in these longer runs, GLS with FLS-2Opt still finds better solutions than the DB variants of LK. This result is of great significance since it further supports our claim that the application of GLS on FLS-2Opt successfully converted the method to a powerful algorithm. As we can see in Table 3.6, the method is able to compete and even outperform highly specialised heuristic methods for the TSP.

The relative gains from the GLS and also DB meta-heuristic are further illustrated in Figure 3.8. In this figure, we give the absolute improvement in average solution quality (i.e. excess above the optimal solution) by the GLS and DB variants over the corresponding repeated local search variants in the set of 20 problems from TSPLIB.

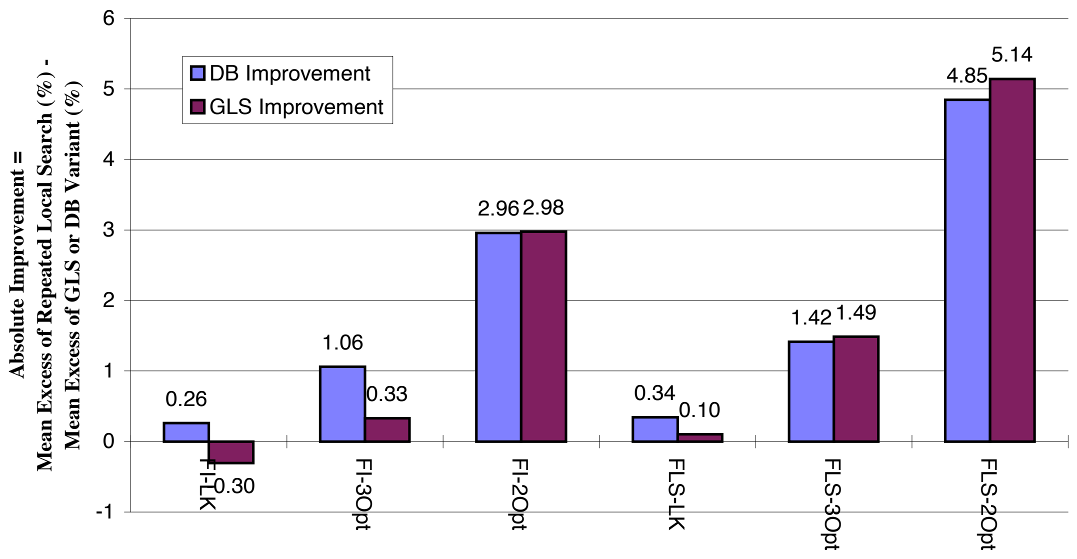


Figure 3.8 Improvements in solution quality by the GLS and DB meta-heuristics in a set of 20 TSPLIB problems

Problem	**Excess (%) in one run per instance**		
	GLS with FLS-2Opt	**DB with FLS-LK**	**DB with FI-LK**
att48	0	0	0
eil51	0	0	0
st70	0	0	0
eil76	0	0	0
pr76	0	0	0
gr96	0	0	0
rat99	0	0	0
kroA100	0	0	0
kroB100	0	0	0
kroC100	0	0	0
kroD100	0	0	0
kroE100	0	0	0
rd100	0	0	0
eil101	0	0	0
lin105	0	0	0
pr107	0	0	0
pr124	0	0	0
bier127	0	0	0
pr136	0	0	0
gr137	0	0	0
pr144	0	0	0
kroA150	0	0	0
kroB150	0	0	0
pr152	0.18458	0	0
u159	0	0	0
rat195	0	0	0
d198	0	0	0
kroA200	0	0	0
kroB200	0	0	0
gr202	0	0	0
pr226	0	0	0
gr229	0	0	0
gil262	0	0	0
pr264	0	0	0
pr299	0	0	0
lin318	0	0.27124	0
fl417	0.00843	0.00843	0.42998
gr431	0	0	0.01458
pr439	0.00653	0.04104	0
pcb442	0.01182	0	0
d493	0.02	0.00857	0.09142
att532	0.06501	0	0.04696
ali535	0.02323	0.01433	0.01433
u574	0	0.08129	0.10568
rat575	0.04429	0.08859	0.05906
p654	2.04659	2.27174	0.04619
d657	0.0184	0.0368	0.13289
gr666	0.00612	0.09988	0.20315
u724	0.05727	0.09783	0.04534
rat783	0	0.06814	0.01136
dsj1000	0.31222	0.40289	0.88742
pr1002	0.12315	0.07566	0.11658
u1060	0.05132	0.15663	0.43285
pcb1173	0.14765	0.02461	0.43767
d1291	0.22244	0.63581	1.16139
rl1304	0.20241	0	0.50366
rl1323	0.18542	0.14027	0.22909
fl1400	1.56009	2.58359	3.11025
u1432	0.05295	0.27783	0.30464
d1655	0.40722	0.27846	1.19753
vm1748	0.33219	0.32387	0.75678
u1817	0.57517	0.3916	1.02096
rl1889	0.37279	0.90953	0.52443
u2152	0.61476	0.46379	0.75327
u2319	0.00726	0.25229	0.28729
pr2392	0.35209	0.27458	0.90019
Mean	**0.12138**	**0.15575**	**0.20947**
Standard Deviation	0.33047	0.43627	0.47296

Table 3.6 GLS with FLS-2Opt compared with variants of Iterated Lin-Kernighan (long runs).

As shown in Figure 3.8, the DB meta-heuristic is more effective than GLS when combined with LK. In fact, GLS when combined with FI-LK is even worse than Repeated FI-LK. This situation dramatically changes for fast local search variants where GLS is better than DB when combined with the FLS-3Opt or FLS-2Opt local searches improving the solution quality over repeated local search up to 5.14% in the case of FLS-2Opt. The overall ranking of all the variants developed in terms of average excess in the set of 20 TSPLIB problems is given in Figure 3.9. GLS with FLS-2Opt was found to be best amongst the 18 algorithms tested.

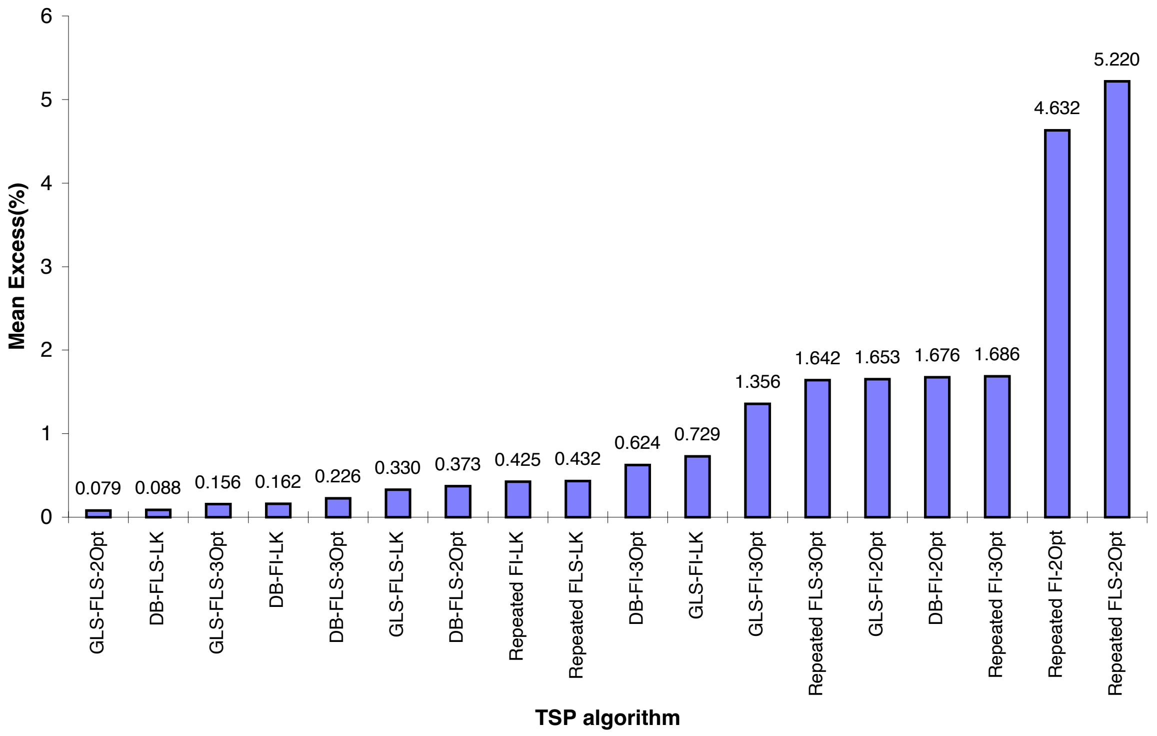


Figure 3.9 Overall ranking of the algorithms in terms of solution quality when tested on a set of 20 TSPLIB problems

3.8.2 Genetic Local Search

In an effort to further improve the LK heuristic, Genetic Algorithms recently appeared which internally use LK for improving offspring solutions generated by crossover operations. These methods, although of great complexity and therefore of limited practical use in our opinion, present theoretical interest and they will be potentially useful when parallel computers became more accessible in the future. An example of such a technique is the Genetic Local Search algorithm proposed by Freisleben and Merz [FM96]. This method, in addition to using LK for improving offspring solutions, uses a mutation operator which performs first an 4-Opt exchange on a population solution and then runs LK to convert this solution to a local minimum. Iterated LK mentioned above can be seen as a special case of this method. In [FM96], results are reported for Genetic Local Search on TSPLIB instances. The authors consider the results produced by the technique as superior to those published for any GA approaches known to them and comparable to top quality non-GA heuristic techniques. Fortunately, the experiments in [FM96] were also conducted on a DEC Alpha workstation running at 175 MHz. This permits a meaningful comparison between this GA variant and GLS. We ran GLS-FLS-2Opt on the same instances with $a = 1/6$ and for an equal number of times as the GA approach. In Table 3.7, the results from [FM96] are compared with those we obtained for GLS using FLS-2Opt.

Problem	**GLS with FLS-2Opt**		**Genetic Local Search**	
	Mean Excess	**Mean CPU time (sec)**	**Mean Excess**	**Mean CPU time (sec)**
eil51 (20 runs)	0%	1.2	0%	6
kroA100 (20 runs)	0%	1.59	0%	11
d198(20 runs)	0%	435	0%	253
att532 (10 runs)	0%	3526	0.05%	6076
rat783 (10 runs)	0%	5232	0.04%	14925

Table 3.7 GLS with FLS-2Opt compared with Genetic Local Search on five TSPLIB instances.

Except for d198 which is a hard instance for GLS (see results in section 3.6), GLS was better than the GA approach finding solutions of better quality for att532 and rat783 while running faster between 1.7 to 6.9 times. Note here that the GA is using the best heuristic for the TSP (i.e. DB followed by LK) while GLS the worst (i.e. 2-Opt). Another remarkable result which emerged from these experiments was that GLS with FLS-2Opt can consistently find the optimal solutions for problems att532 and rat783. As far as we know, optimal solutions to such large problems can be consistently found only by heuristic methods that are using LK (e.g. Iterated LK or its variant Large-Step Markov Chains method).

In fact, GLS was able to find the optimal solution in even larger problems. For example, GLS with FLS-3Opt found the optimal solution for a 2319-city problem from TSPLIB (u2319) in less than 20 minutes while GLS with FLS-2Opt found the optimal solution to a 1002-city problem from TSPLIB (pr1002) in 14 hours of CPU time despite running on Sparcstation 5 workstation which is much slower than the DEC Alpha machines used in the rest of the experiments.

3.9 Conclusions

In this chapter, the application of GLS to the TSP was examined. The combinations of GLS with commonly used TSP heuristics were described and evaluated on publicly available instances of the TSP. GLS with FLS-2Opt was found to be the best GLS variant for the TSP. The variant was compared and found to be superior to general search methods such as simulated annealing and tabu search. Furthermore, we demonstrated that GLS with FLS-2Opt is highly competitive (if not better) than some of the best specialised algorithms for the TSP such as Iterated Lin-Kernighan and Genetic Local Search.

Nonetheless, experimental results should be treated with care. Experimentation no matter how elaborate and extensive it may be, it can only give indications of which algorithms are better than others and that because of the many parameters involved in the algorithms, differences in implementation, and the limited number of instances used in experiments.

We can safely conclude that the evidence provided in this chapter is enough to place GLS amongst what somebody will characterise as efficient and effective methods for the TSP. Given the simplicity of the algorithm and the ease of tuning (i.e. single parameter), GLS with FLS-2Opt could be considered as an ideal practical method for the TSP especially when no programming effort can be devoted in implementing one of the complex specialised TSP algorithms.

Chapter 4

Quadratic Assignment Problem

The TSP, examined in the last chapter, is probably the most famous problem in combinatorial optimisation. Another problem which has also attracted the interest of researchers for many years is the Quadratic Assignment Problem (QAP). QAP could be probably listed second after the TSP in the list of the most famous combinatorial optimisation problems. The application of GLS to the QAP is examined in this chapter. Problems in GLS arising from the use of features with variable costs are identified and strategies for resolving them are proposed. Comparison with state of the art QAP algorithms demonstrates the ability of GLS to compete on equal terms with these methods and even to outperform them.

4.1 The Problem

Quadratic Assignment Problem (QAP) is one of the most difficult problems in combinatorial optimisation. The problem can model a variety of applications but it is mainly known for its use in facility location problems. For a recent QAP survey, the reader is referred to Pardalos, Rendl, and Wolkowicz [PRW93]. In the following, we describe the QAP in its simplest form.

Given a set *N = {1, 2, ..., n}* and $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{kl}]$, the QAP can be stated as follows:

Eq. 4.1
$$\min_{p \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \cdot B_{p(i)p(j)}$$

where p is a permutation of N and Π_N is the set of all possible permutations. There are several other equivalent formulations of the problem. In the facility location context, each permutation represents an assignment of n facilities to n locations. More specifically, each position i in the permutation represents a location and its contents $p(i)$ the facility assigned to that location. The matrix A is called the distance matrix and gives the distance between any two of the locations. The matrix B is called the flow matrix and gives the flow of materials between any two of the facilities. In this work, we only consider the Symmetric QAP case for which both the distance and flow matrices are symmetric.

4.2 Local Search for the QAP

QAP solutions are represented by permutations. A move commonly used for the problem is simply to exchange the contents of two permutation positions (i.e. swap the facilities assigned to a pair of locations). A best improvement local search procedure starts with a random permutation. In every iteration, all possible moves (i.e. swaps) are evaluated and the best is selected and performed. The algorithm reaches a local minimum when there is no move which improves further the cost of the current permutation.

An efficient update scheme can be used in the QAP which allows evaluation of moves in constant time. The scheme works only with best improvement local search. Move values of the first neighbourhood search are stored and updated each time a new neighbourhood search is performed to take into account changes from the move last executed (see [BT94] or [Tai95] for details). Move values do not need to be evaluated from scratch and thus the neighbourhood can be fully searched in roughly $O(n^2)$ time instead of $O(n^3)$ required otherwise[4]. To evaluate moves in constant time, we have to examine all possible moves in each iteration and have their values updated. Because of that, the scheme can not be combined with FLS which examines only a number of moves in each iteration. FLS for the QAP requires $O(n)$ operations to evaluate a move and therefore $O(n^3)$ to evaluate all moves in the neighbourhood. This prevented us from developing a efficient version of FLS for the QAP and instead we used simple GLS without neighbourhood reduction.

4.3 Guided Local Search Applied to the QAP

Applying GLS to the QAP is a simple two-stage process of identifying the solution features to be used and assigning costs to them. A set of features that can be used in the QAP is the set of all possible assignments of facilities to locations (i.e. location-facility pairs). This kind of feature is general and can be used in a variety of other assignment problems where a number of variables are assigned values from finite domains. In the QAP, there are n^2 possible location-facility combinations (features)[5].

[4] To evaluate the change in the cost function Eq. 4.1 caused by a move normally requires $O(n)$ time. Since there are $O(n^2)$ moves to be evaluated, the search of the neighbourhood without the update scheme requires $O(n^3)$ time.

[5] Features that detect assignment combinations (i.e. combinations of location-facility pairs) are also possible but the number of features in this case rises to $O(n^4)$ making practically impossible the storage of penalties for problems of size $n>30$.

After deciding on the features, the next step is to assign costs to them. Assignment of facilities to locations are tightly coupled one to the other because of the problem's cost function. For that reason, it is difficult to isolate the effect that particular assignments have on the solution cost. To deal with this problem, we used variable feature costs where the cost of a feature is evaluated in the context of the solution it appears in. In particular, feature costs are evaluated only for the features of the local minimum and their cost is given by the expression:

Eq. 4.2
$$c(i,p(i)) = \sum_{j=1}^{n} A_{ij} \cdot B_{p(i)p(j)}$$

where *i* is the location and *p(i)* is the facility assigned to that location in the local minimum solution. The above expression for the feature cost gives the cost arising from the flow of materials from facility *p(i)* to the other facilities with facility *p(i)* placed at location *i*. In a local minimum, features that maximise the utility expression Eq. 2.5 are penalised and the corresponding location-facility combinations are avoided.

To determine a range of values for the lambda parameter of GLS, we conducted a large number of test runs on problems from the publicly available library of QAP instances, QAPLIB [BKR91]. An equation similar to Eq. 3.6 used in the TSP was also derived for the QAP case. In particular, we found that GLS performed well for an λ given by the following parametric equation:

Eq. 4.3
$$\lambda = a \cdot \frac{g(\text{local minimum})}{n^2}, \ 1/5 \le a \le 1$$

where *g(local minimum)* is the cost of the first local minimum found during a run and *n* the size of the problem. In terms of implementation, the algorithm is given as input the parameter *a* which is used to calculate lambda after the first local minimum and before the first features are penalised.

4.4 The Issue of Features with Variable Costs

Features with variable costs are a potential problem for GLS. The problem arises because decisions to penalise features are based on feature costs. If the costs of features change during search then bad features may become good and vice versa. Penalties imposed on bad features which turn good at a latter stage may prevent these features from being used again in the solution.

For instance, let us consider a local minimum solution where facility *j* is assigned to location *i*. If location *i* is far from the locations of facilities connected with high flows to facility *j* then the assignment of facility *j* to location *i* is a bad combination. This results in a high cost for the corresponding feature. GLS will penalise the combination of location *i* with facility *j* and facility *j* will be assigned elsewhere. Although the decision is correct in this context, it may prevent local search from assigning facility *j* to location *i* at a later stage in search when the arrangement of all other facilities makes location *i* a good choice. The GLS decision based on a single local minimum solution is incorrectly generalised constraining many other potentially good solutions. The result is that diversification is triggered prematurely and GLS leaves the good areas of the search space without thoroughly searching them. To resolve this problem a number of strategies were explored. After experimentation, three strategies were identified as the most promising ones.

4.4.1 Reset Strategy

This strategy is identical to the basic GLS depicted in Figure 2.1 with the exception that all penalties are reset to 0 every *t* iterations. By resetting the penalties, GLS can revisit solutions that include features penalised earlier in the search process. This leads to an intensification of search in the "good" areas of the search space which compensates for the unnecessary diversification caused by the variable feature costs.

The drawback of the approach is that GLS looses some of its diversification ability which drives the algorithm to unexplored regions of the search space when enough effort is spent in the promising areas. In the following, we will refer to this GLS variant as *Reset-GLS*.

4.4.2 Restart Strategy

Instead of resetting the penalties, the algorithm is restarted from a "good" solution every *t* iterations. The objective is the same as with Reset-GLS, that is to intensify search in the "good" areas of the search space. The new starting points are generated by combining the *K* best solutions found during search prior to reaching the restart point, in a way that very much resembles Genetic Algorithm approaches. The approach is similar to intensification schemes used in the Vehicle Routing Problem by tabu search methods [RT95] (see section 1.5.3).

In particular, the *K* best solutions found during search prior to the restart point are organised in a list which is then sorted by the solution cost. A selection probability is assigned to each solution depending on its position in the list. In the version of the procedure implemented, the ten best solutions were used and the probabilities assigned from best to worst solution were 0.36, 0.18, 0.12, 0.09, 0.07, 0.06, 0.05, 0.04, 0.02, and 0.01 respectively. New solutions were generated using the following procedure.

Starting from an empty permutation and scanning the locations from left to right, each location is assigned the same facility as in a solution pseudo-randomly selected from the list of the best solutions according to the above probabilities. After all locations have been assigned facilities, the permutation is again scanned from left to right and facilities which appear more than once are randomly replaced by the unassigned facilities. GLS is restarted from the solution generated without resetting the penalties.

To recapitulate, the restart strategy tries to achieve search intensification in the “good” areas of the search space by restarting the algorithm from a solution which is formed by combining the best local optima visited up to the restart point. Although variable feature costs may mislead the algorithm into unpromising areas, the restart strategy tries to bring the method back to the areas of the good solutions. Moreover, different search trajectories are tried in these areas after each restart because of the memory of the algorithm (i.e. penalties) which is not cleared. In the following, we will refer to this GLS variant as *Restart-GLS*.

4.4.3 Multiple Feature Sets Strategy

In the QAP, GLS decides which features to penalise using the costs of features as measured in the context of a particular local minimum. As the algorithm leaves this local minimum and swaps are performed, feature costs gradually change up to the point where they have totally different values from those calculated in the local minimum. In other words, the information used in GLS decisions gradually becomes invalid after the point these decisions are made. A sensible thing to do is to remove the effects of decisions as soon as the information they were based on becomes invalid.

In a more global perspective, information which is valid only for a certain period of time should lead to restrictions of equal duration on local search. When information becomes invalid or out of context, the restrictions imposed on the basis of this information should be retracted. Tabu search as originally presented by Glover [Glo89, Glo90] makes extensive use of this principle. This same principle can be also used to explain why dynamic tabu lists are preferable over their static counterparts in many problems [Tai91, LG93]. The former, by varying the duration of restrictions, match better than the latter the duration for which search history information is valid.

We put to use the above ideas and developed a strategy which overcomes the problem of variable feature costs in GLS. The strategy uses a tabu list [Glo89] to retract the effects of decisions made earlier in the search process. More specifically, penalties increased are decreased after a certain number of penalty increases is performed. The scheme uses an array of size *t* where the *t* most recent features penalised are recorded. The array is treated as a circular list, adding elements in sequence in positions 1 through *t* and then starting over at position 1. Each time the penalty of a feature is increased (by one unit), the feature is inserted in the array and the penalty of the feature previously stored in the same position is decreased (by one unit).

One problem with this approach is that GLS totally loses its long term memory and therefore is unable to diversify search. This is the opposite problem from that with the Reset-GLS and Restart-GLS variants which either reset long-term memory after a relatively large number of iterations (Reset-GLS) or do not reset it at all (Restart-GLS). A simple way to work around the problem is to introduce a second set of features identical to the first feature set. This feature set is to undertake the task of long term diversification by exploiting search history information that is the local minima visited.

Penalties for this second set are neither reset nor decreased but only increased as in the basic GLS providing the long-term memory needed to drive search to new regions. Moreover, features costs are considered constant and equal to 1.0 such that the search effort is uniformly distributed amongst the features in the set.

GLS works on the two feature sets independently and in parallel. This merely means that in a local minimum both sets are examined and the features with the highest utility value in each set are penalised. Additionally, two different regularisation parameters λ_1 and λ_2 are used, one for each feature set to allow appropriate balancing of short-term and long-term penalties. In implementation terms, two parameters a_1 and a_2 are fed as inputs to the algorithm and the calculation of λ_1 and λ_2 takes place after the first local minimum using Eq. 4.3.

In the penalty incrementation procedure of GLS for the second set (i.e. long-term penalties), ties amongst features are frequent especially at the beginning of search because of the equal feature costs. In order to avoid penalising too many features, ties are broken deterministically and the first feature found to maximise the utility function is penalised. Experimentation with random tie-breaking strategies showed no improvement in performance.

Summarising, the multiple feature sets strategy uses two identical feature sets but with different feature costs and with penalties of different duration to accomplish the objectives of intensification and diversification of search. The first set with variable feature costs is utilised to impose short-term penalties for the purposes of intensification. The second set with constant feature costs is utilised to impose long-term penalties for the purposes of diversification. Two independent GLS processes working on these sets are used which, when combined, achieve the overall goal of the distribution of search effort according to promise. The separation of intensification and diversification became necessary in this case because the information used to achieve each of these two sub-goals is valid for different periods of time. In the following, we will refer to this GLS variant as *Multiple-GLS*.

4.5 Experimental Evaluation of Basic GLS and its Variants

We conducted many experiments in order to develop the basic GLS and the various strategies for resolving the problem with the variable feature costs. Problems included in QAPLIB [BKR91] were used in the experiments. A typical value for a which worked well for most problems tested and all variants was $a = 0.5$ ($a_1 = 0.5$ in the case of Multiple-GLS). In addition to that, we found that the a_2 parameter used only in Multiple-GLS for the second feature set needed to be smaller than the a_1 used for the first feature set. A value $a_2 = 0.25$ combined very well with the value $a_1 = 0.5$.

For the t parameter required by all three GLS variants, multiples of the problem size n were tried. For Reset-GLS and Restart-GLS large values performed better. In particular, a value $t = 200n$ performed well for Reset-GLS while the value $t = 100n$ was a good choice for Restart-GLS. Multiple-GLS required much lower values for t. This is because the parameter serves a different purpose in this case (i.e. sets the duration of the short-term penalties). A range of values for t which resulted in good performance for Multiple-GLS was $n \leq t \leq 10n$. The value $t = 4n$ was used to generate all the results reported in this chapter.

The results presented in this section refer to a set of ten QAP instances of sizes from 15 to 40, all from QAPLIB. The set is a mixture of problems of different nature and size intended to test the basic GLS and its variants on different types of flow and distance matrices. For each algorithm, ten runs were performed on each instance, starting from random solutions. The algorithms were allowed to run for 100,000 iterations (i.e. full neighbourhood searches) or until a solution with cost equal or less than the best known solution[6] was found. Repeated local search was also implemented to give a point of reference for measuring the success of algorithms. This last algorithm was simply restarting local search after a local minimum.

A run was characterised as successful if it resulted in the best known solution. The solution quality was measured in per cent excess above the best known solution (see Eq. 3.5). Table 4.1 illustrates the results obtained.

Problem Name	best known solution	Basic GLS $a = 0.5$	Reset-GLS $a = 0.5$, $t = 200n$	Restart-GLS $a = 0.5$, $t = 100n$	Multiple-GLS $a_1 = 0.5$, $a_2 = 0.25$, $t = 4n$.	Repeated Local Search
		successful runs (Mean Excess)	successful runs (Mean Excess)	successful runs (Mean Excess)	successful runs (Mean Excess)	successful runs (Mean Excess)
nug15	1150	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
nug20	2570	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
rou20	725522	5 (0.022)	8 (0.002)	7 (0.015)	10 (0)	4 (0.055)
nug30	6124	10 (0)	10 (0)	9 (0.007)	10 (0)	2 (0.31)
tho30	149936	9 (0.004)	10 (0)	8 (0.046)	10 (0)	1 (0.355)
kra30a	88900	10 (0)	10 (0)	10 (0)	10 (0)	3 (0.966)
kra30b	91420	4 (0.056)	7 (0.023)	5 (0.049)	8 (0.015)	0 (0.163)
ste36a	9526	7 (0.069)	6 (0.086)	5 (0.206)	9 (0.01)	0 (1.148)
ste36b	15852	1 (1.156)	4 (2.324)	8 (0.343)	10 (0)	3 (0.574)
tho40	240516	0 (0.169)	0 (0.076)	0 (0.142)	0 (0.051)	0 (0.849)
Total successes		66/100	75/100	72/100	87/100	33/100
Mean solution quality		0.1476%	0.2511%	0.0808%	0.0076%	0.442%

Table 4.1 Comparison of GLS variants for the QAP.

The results clearly demonstrate that basic GLS is better than repeated local search. The algorithm finds the best known solution in 66% of the runs, twice the success rate of local search without GLS. The strategies for resolving the problem of variable feature costs had a varied success. Reset-GLS, although improved over basic GLS in terms of successful runs, had a worse mean solution quality. This can be attributed to

[6] Exact methods generally find it difficult to solve QAP problems of size greater than 20. QAPLIB includes many instances with size greater than 20 and therefore out of range for exact methods. These problems have been

the inferior diversification strategy because of the penalties being reset. On the other hand, Restart-GLS had fewer successful runs than Reset-GLS, though significantly improved over basic GLS's mean solution quality.

The sophisticated Multiple-GLS strategy paid off finding the best known solution in 87% of the runs. Moreover, the Multiple-GLS strategy achieved a remarkable mean excess of 0.0076% unmatched by any of the other algorithms tested. Much of this success can be attributed to the second feature set of Multiple-GLS responsible for diversification. In fact, we performed experiments with no short-term penalties (i.e. $a_1 = 0$). For $a_2 = 0.5$, the algorithm was still able to show a very good performance, finding the optimal solution in 82% of the runs with a mean excess of 0.0306%. Lower and higher values for a_2 resulted in slightly worse performance. This suggests another strategy for overcoming the problem of features with variable feature costs that is to set all feature costs to the same value (i.e. use only the second feature set of Multiple-GLS). However, this strategy could be improved further by using short-term penalties based on variable costs to play the crucial refinement role needed in order for the algorithm to reach a performance such as that presented in Table 4.1.

4.6 Efficient Heuristic Methods for the QAP

Efficient heuristic methods for the QAP are based on tabu search. Two very successful tabu search methods for the QAP are Robust Taboo Search (Ro-TS) due to Taillard [Tai91] and Reactive Tabu Search (RTS) due to Battiti and Tecchiolli [BT94]. Other works applying tabu search to the QAP not examined here include Skorin-Kapon [Sko90] and Chakrapani and Skorin-Kapov [CS93] to name but two. Moreover, the

---

tackled in the past by many approximation methods and very good solutions are already known for them. Whether these solutions are also optimal is an open question.

Genetic Hybrids (GH) method due to Fleurent and Ferland [FF94] which found the best known solutions for many of the large problems in QAPLIB is based on Ro-TS. In this case, Ro-TS is used as the mutation operator which improves solutions produced by GH's crossover operator.

We compared GLS with Ro-TS and RTS and also GH. Before proceeding to examine these results. We briefly describe Ro-TS and RTS. For a description of GH the reader can refer to the original paper by Fleurent and Ferland [FF94] or to Taillard's excellent review and comparison of Ro-TS, RTS and GH on both symmetric and asymmetric QAPs [Tai95].

4.6.1 Robust Taboo Search

Robust Taboo Search uses the same local search procedure as GLS (see section 4.2). Additionally, tabu restrictions are imposed which exclude specific moves from being selected. A move is non admissible (i.e. tabu) if at least one of the following conditions is satisfied (u and t are the parameters of the algorithm) [Tai91, GTW93, Tai95]:

- if during the last u iterations, a solution had facility i placed at location r and facility j placed at location s then a move which places both i at location r and j at location s again is forbidden (unless this move results in a new best solution).
- if the number of iterations performed is greater than t and facility i has never been at location r during the last t iterations then a move which does not place facility i at location r is forbidden.

The parameter u changes during search taking random values in the range $0.9n < u < 1.1n$. This leads to a dynamic tabu list strategy [GTW93, GL93]. A good range of values for parameter t is $2n^2 \leq t \leq 5n^2$ [Tai95].

The short-term tabu restrictions based on parameter u prevent the reversal of moves previously executed, enabling the algorithm to escape from local minima and at the same time intensify search in the "good" areas of the search space. On the other hand, tabu restrictions using parameter t aim to diversify search in the long term forcing it to enter new regions of the search space. This is achieved by incorporating in the solution, location-facility combinations not visited in the near past. The two objectives of the algorithm are the same as the objectives of Multiple-GLS, though different means are used to accomplish them.

For our experiments, we implemented Ro-TS in C++. The parameter u was dynamically changing as described above while the parameter t was set to $3.5n^2$ which is in the middle of the range suggested by the author.

4.6.2 Reactive Tabu Search

Reactive Tabu Search uses the same short-term memory as Ro-TS though the choice of parameter u is different. The parameter u is dynamically controlled using a simple feedback mechanism. In particular, if the search returns to a solution already visited then the value of u is increased to force local search out of the domain of attraction of the current local minimum. On the other hand, if u is not changed for a number of iterations then it is decreased.

On the diversification front, if solutions are often visited then a number of random exchanges is made to force local search to explore new regions. All random exchanges executed are made tabu to prevent a return. For our experiments, we obtained and used the original source code in C of Battiti and Tecchiolli [BT94]. The default parameters provided by the authors were used in our experiments.

4.7 Comparison of GLS with Efficient QAP Heuristic Methods

In this section, we compare Multiple-GLS, found to be the best GLS variant, with Ro-TS, RTS and also GH on problems of different size and nature. We first compare GLS with Ro-TS and RTS on the set of small to medium used for comparing the GLS variants. This problem set represents a good mixture of real-world and randomly generated problems. Following that, we report results for GLS on a set of random large QAP instances with sizes up to 100 generated by Skorin-Kapov [Sko90] and compare our results with those reported by Talliard [Tai95] for Ro-TS, RTS and GH on the same set of problems.

Before proceeding with the comparisons, we would like to clarify some issues relating to the computation times required by Multiple-GLS. In particular, Multiple-GLS, Ro-TS and RTS need around the same time to complete an iteration (i.e. complete search of the neighbourhood). The dominant computation is the evaluation of the $O(n^2)$ moves in the neighbourhood. This computation is conducted in almost exactly the same way for all three methods. Actually, GLS is performing fewer moves than the other two methods if allowed to run for the same the number of iterations. This is because to escape from a local minimum GLS may perform more than one iteration (i.e. neighbourhood searches) without executing a move. In between these iterations, each penalty modification cycle requires $O(n^2)$ time to compute the feature costs and utilities for the first feature set and $O(n)$ time for the second feature set. Although, one may think that GLS requires more time than tabu searches to complete the same number of iterations because of the intervening penalty modification cycles that is not the case. The reason is that the iteration following a penalty modification cycle requires less time for GLS than an iteration for tabu search since no move value updates are made during this iteration. In addition, evaluating tabu restrictions on moves requires in general more time than the corresponding calculation of penalty differences in GLS. In fact, our implementation of Multiple-GLS proved to need less time to complete the same number of iterations than our corresponding implementation of Ro-TS[7] for all but very large problems (e.g. n =100) and even in that case, Ro-TS was less than half second per minute faster than Multiple-GLS. In general, Multiple-GLS, Ro-TS and RTS can be considered to require roughly the same amount of time to complete the same number of iterations. This is very important since it allows us to make a fair comparison of these techniques based on the number of iterations they perform.

4.7.1 Small To Medium Size QAPs

We compared Multiple-GLS with Ro-TS and RTS on the set of small to medium size QAP instances used for the comparison of the GLS variants in section 4.5. Ro-TS and RTS were allowed to run for 100,000 iterations on each problem and the results from 10 runs were averaged. The performance of Ro-TS and RTS was measured in terms of the number of successful runs (i.e. runs that resulted in the best known solution) and also solution quality (i.e. per cent excess above the best known solution). Given that Ro-TS and RTS required roughly the same time to complete 100,000 iterations as Multiple-GLS, results for Ro-TS and RTS can be directly compared with each other

[7] The implementations of Multiple-GLS and Ro-TS were both in C++ and they were sharing large parts of the code. We tried to optimise as much as possible the non-shared parts of both methods.

and with those for Multiple-GLS reported in Table 4.1. This comparison is made in Table 4.2.

Problem Name	best known solution	Multiple-GLS $a_1 = 0.5$, $a_2 = 0.25$, $t = 4n$.		Robust Tabu Search (Ro-TS)		Reactive Tabu Search (Re-TS)	
		successful runs	solution quality	successful runs	solution quality	successful runs	solution quality
nug15	1150	10	0	10	0	10	0
nug20	2570	10	0	10	0	2	0.506
rou20	725522	10	0	10	0	10	0
nug30	6124	10	0	10	0	1	0.441
tho30	149936	10	0	10	0	10	0
kra30a	88900	10	0	10	0	9	0.134
kra30b	91420	8	0.015	10	0	7	0.039
ste36a	9526	9	0.01	7	0.019	0	1.094
ste36b	15852	10	0	10	0	9	0.025
tho40	240516	0	0.051	1	0.041	3	0.024
Total Successes		87/100	0.0076%	88/100	0.006%	61/100	0.2263%

Table 4.2 Comparison of Multiple-GLS with Robust Tabu Search and Reactive Tabu Search.

In this table, we see that GLS is highly competitive with Ro-TS and both methods are much better than Re-TS. Ro-TS had just one more successful run than GLS while in terms of solution quality, Ro-TS was better than GLS by just 0.0016%. This result is so close that neither of these techniques can be said to be better than the other on this set of problems.

RTS lagged behind both Re-TS and Multiple-GLS. This can be partly attributed to the fact that the default parameters were used for Re-TS and partly to the case that the method may not be suitable for these types of problems.

4.7.2 Large QAPs

Multiple-GLS uses the long-term penalties to distribute the search effort over the whole of the search space. Long-term penalties are supported by the short-term penalties which intensify search as the algorithm progresses into new regions. One would expect, that for larger problems this may be an advantageous strategy to follow, because of the systematic exploration strategy introduced by the long-term penalties.

To investigate the benefits of using Multiple-GLS on large problems, we tested Multiple-GLS on a set of 12 large QAP instances from QAPLIB with sizes from 49 to 100 which have been randomly generated by Skorin-Kapov (see [Sko90] for details). Talliard [Tai95] reports results for these instances for Ro-TS, RTS, and also GH. In the competitive tests which Taillard performed on these problems, he allocates 1000*n* iterations for the tabu searches and a roughly equivalent amount of time to the GH method. We allowed Multiple-GLS to run for the same number of iterations. The results from ten runs were averaged in each instance.

In Table 4.3, we compare the solution quality (i.e. mean excess) of Multiple-GLS with those reported by Taillard for Ro-TS, RTS, and GH. The results are averaged when several problems of the same size and type are solved.

Problem	**Multiple-GLS**	**Ro-TS**	**Re-TS**	**GH**	**Best known Solution**
Sko49	0.068	0.096	0.068	0.120	23386
Sko56	0.104	0.090	0.145	0.181	34458
Sko64	0.098	0.063	0.125	0.174	48498
Sko72	0.147	0.181	0.110	0.200	66256
Sko81	0.117	0.088	0.110	0.250	90998
Sko90	0.158	0.179	0.164	0.314	115534
Sko100a-f	0.118	0.162	0.141	0.264	150252.7
Mean Solution Quality	**0.117**	**0.139**	**0.131**	**0.235**	

Table 4.3 Comparison of Multiple-GLS with Ro-TS, Re-TS and GH on large QAPs.

In this table, we see that Multiple-GLS achieves the best solution quality with RTS second, Ro-TS third and GH the worst method amongst the four. As Taillard points out, the GH needs long computation times to be competitive on these problems. In general, GH performs better on structured rather than random problems. However, the comparison clearly indicates that GLS is competitive with all these state of the art QAP methods and able to outperform them at least on the these particular problems with the particular limit on the number of iterations. One possible explanation for this is that using the long-term penalties, GLS more systematically diversifies search in large search spaces than the other methods while the intensification strategy adopted by Multiple-GLS enables the algorithm to produce good solutions is short time.

4.8 Conclusions

In this chapter, we clearly demonstrated the applicability of the GLS algorithm to the famous Quadratic Assignment Problem. The structure of the problem provided an ideal candidate for examining the problem of variable feature costs and allowed us to propose various strategies to resolve it. Retracting the effects of GLS decisions, when the information they were based on becomes invalid, proved to be the best strategy for resolving the problem. The use of parallel GLS processes aimed separately at the intensification and diversification of search was also proposed in this context. The final GLS variant adopting these modifications was compared to state of the art techniques for the QAP. GLS proved to be highly competitive with these methods in the experiments carried out, even outperforming them in large QAPs when time resources are limited.

Chapter 5

Radio Link Frequency Assignment Problem

In the last two chapters, we focused on two challenging but nonetheless simple problems in terms of objectives and constraints. Modern applications frequently require solving more complex problems than the TSP and QAP. Some of these problems are not pure optimisation problems but also involve some aspects of constraint satisfaction. In such cases, we sometimes seek solutions which violate the minimum number of constraints. In more realistic settings, constraint violations incur different costs and solutions are sought that minimise the total cost from constraint violations and possibly other criteria. In this chapter, we examine how Guided Local Search and Fast Local Search can be applied to such problems often referred to as Partial Constraint Satisfaction Problems (PCSPs) or constrained optimisation problems. The Radio Link Frequency Assignment Problem (RLFAP) is examined as a representative problem in this class. RLFAP stems from real-world situations in military telecommunications. The effectiveness and efficiency of the GLS technique is demonstrated on publicly available instances of the problem. Comparison with other search techniques demonstrates the advantages of the GLS method over alternative approaches to PCSPs.

5.1 Partial Constraint Satisfaction Problem

The Partial Constraint Satisfaction Problem can model a variety of constraint satisfaction problems with various forms of optimisation. In the classic CSP, one is trying to assign values to finite domain variables such that a set of linear and/or non-linear constraints on these variables are satisfied. In PCSP, the satisfaction of constraints becomes the subject of optimisation and solutions that minimise the number of constraint violations or more complex optimisation criteria are sought. Before formally defining the PCSP, we introduce some terminology used in the CSP related literature.

The assignment of a value to a variable is called a *label*. The label which involves the assignment of a value v to the variable x (where v is in the domain of x) is denoted by the pair $<x,v>$. A simultaneous assignment of values to a set of variables is called a *compound label* and is represented as a set of labels, denoted by $(<x_1,v_1>,<x_2,v_2>,...,<x_k,v_k>)$. A *complete compound label* is a compound label which assigns a value to every variable in the CSP. The goal in CSP is to find one or all complete compound labels that satisfy the constraints.

A *Partial Constraint Satisfaction Problem* (PCSP) is a Constraint Satisfaction Problem in which one is prepared to settle for partial solutions — solutions which may violate some constraints or assignments of values to some, but not all variables — when solutions do not exist (or, in some cases, cannot be found) [FW92, Tsa93].

This kind of situation often occurs in applications like industrial scheduling where the available resources are not enough to cover the requirements. Under these circumstances, partial solutions are acceptable and a problem solver has to find the one that minimises an objective function.

The objective function is domain-dependent and may take various forms. In one of its simplest forms, the optimisation criterion may be the number of the constraint violations. For more realistic settings, some constraints may be characterised as "hard constraints" and they must be satisfied whilst others, which are referred to as "soft constraints", may be violated if necessary. Moreover, constraints may be assigned violation costs which reflect their relative importance. Partly following Tsang [Tsa93], we define the Partial Constraint Satisfaction Problem formally as follows:

Definition 5.1:

A partial constraint satisfaction problem (PCSP) is a quadruple:

$$(Z, D, C, g)$$

where
• $Z = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables,
• $D = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$ is a set of finite domains for the variables in Z,
• $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of constraints on an arbitrary subset of variables in Z,
• g is the objective function which maps every compound label to a numerical value.

The goal in a PCSP is to find a compound label (partial or complete) which optimises (minimises or maximises) the objective function g. Given the above definition, standard CSPs and *Constraint Satisfaction Optimisation Problems* (CSOPs) (where optimal solutions are required in CSPs, see [Tsa93]) can both be cast as PCSPs. Under the Partial CSP formulation, all compound labels (partial or complete) are *candidate solutions* since constraint violations are part of the cost function.

Versions of branch and bound and other complete methods have been suggested for tackling PCSPs [FW92, WF93, JFM96]. But complete algorithms are inevitably limited by the combinatorial explosion problem. A heuristic method for the related MAX-SAT problem has also been recently proposed by Jiang, Kautz, and Selman [JKS95]. The method is a direct descendant of GSAT [SLM92] and uses random walk for escaping local minima. To use the method for PCSPs, the PCSP problem has to be converted to MAX-SAT. This conversion is not always straightforward and normally result in a MAX-SAT problem with an even bigger search space than the original PCSP[8]. Also, Wallace and Freuder [WF96] have tested restart, random walk and tabu search variants of the min-conflicts heuristic [MJPL92] on random PCSPs of sizes up to 100 variables minimising the number of constraint violations.

General heuristic methods such as Genetic Algorithms, Tabu Search and Simulated Annealing have also been tried on PCSPs and in particular on the RLFAP problem. The performance of these techniques is going to be examined later in this chapter.

5.2 The Radio Link Frequency Assignment Problem

The *Radio Link Frequency Assignment Problem* was abstracted from the real life application of assigning frequencies (values) to radio links (variables). Eleven instances of the problem, which involve various optimisation criteria, were made publicly available by the French Centre d'Electronique l'Armament [RLFAP94]. The

[8] In fact, a PCSP with n variables each with domain size m will have a search space m^n. The equivalent MAX-SAT problem will have 2^{mn} which in normally bigger than m^n (because $m < 2^m$ when $m \geq 1$).

problem is NP-Hard and it is a variant of the T-graph colouring problem as introduced by Hale [Hal80]. Two different types of binary constraints are involved in the RLFAP:

- The absolute difference between two frequencies must be greater than a given number k (i.e. for two frequencies X and Y, $|X - Y| > k$);
- The absolute difference between two frequencies must be exactly equal to a given number k (i.e. for two frequencies X and Y, $|X - Y| = k$).

The above constraints are either hard or soft constraints. A problem specifies the variables which are subject to these constraints and the constraint graph is not complete (i.e. not every variable is constrained by every other variable). If all the constraints can be satisfied then either:

- (C1) the solution which assigns the fewest number of different values to the variables,
- (C2) or the solution where the largest assigned value is minimal

is preferred. For insoluble problems, *violation costs* are defined for the constraints. Furthermore, for some insoluble problems, default values are defined for some of the variables. If any of the default values is not used in the solution returned, then a predetermined *mobility* cost applies. Table 5.1 depicts the characteristics of the RLFAP instances.

RLFAP Instance	No. Variables	No. Constraints	Soluble	Minimise
Scen01	916	5,548	Yes	number of different values used (C1)
Scen02	200	1,235	Yes	number of different values used (C1)
Scen03	400	2,760	Yes	number of different values used (C1)
Scen04	680	3,968	Yes	number of different values used (C1)
Scen05	400	2,598	Yes	number of different values used (C1)
Scen06	200	1,322	No	maximum value used (C2)
Scen07	400	2,865	No	weighted constraint violations
Scen08	916	2,744	No	weighted constraint violations
Scen09	680	4,103	No	weighted constraint violations + mobility costs
Scen10	680	4,103	No	weighted constraint violations + mobility costs
Scen11	680	4,103	Yes	number of different values used (C1)

Table 5.1 Characteristics of RLFAP instances. The domains of variables consist of 6-44 integer values.

The eleven RLFAPs are ideal for testing the effectiveness of GLS in PCSPs because they contain both soluble and insoluble problems and non-trivial optimisation criteria are defined for both soluble and insoluble problems. Besides, results from other research exists, which could be used to measure the success of GLS. In RLFAP, complete compound labels are sought. For PCSPs where partial compound labels are sought, the reader can refer to chapter 6 where GLS is used to tackle a real world workforce scheduling problem in this last category.

5.3 Local Search for Partial PCSPs

A local search procedure for Partial CSPs can be based on the min-conflicts heuristic of Minton et al. [MJPL92] and the computational model of the GENET network [WT91, Tsa93, DTWZ94]. An 1-optimal type move can be used which changes the value of one variable at a time. Starting from a random and complete assignment, variables are examined in an arbitrary static order. Each time a variable is examined, the current value of the variable changes to the value which yields the minimum value for the cost function. Ties are randomly resolved allowing moves which transit to solutions with equal cost. These moves, often called sideways moves [SLM92], enable local search to examine plateau of states occurring in the landscapes of many CSPs and Partial CSPs. One problem with sideways moves is that of detecting local minima. This problem can be overcome using the limited sideways scheme described in [VT94] and also [Dav97]. In particular, we characterise a solution as a local minimum when all variables have been examined and no change occurred in the value of the cost function. Although we allow sideways moves to occur locally, if these moves do not result in a better solution after all variables have been examined then a local minimum is concluded.

```
procedure LocalSearch(Z, D, g, $S_i$)
begin
    S ← $S_i$;
    repeat
        $g_{before}$ ← g(S);
        for each variable x in Z do
        begin
            S ← S - {<x,$v_i$>};
            for each value v in Dx do
                $g_v$ ← g(S + {<x,v>});
            BestSet ← set of values with minimum $g_v$;
            $v_{i+1}$ ← random value in BestSet; (* sideways moves *)
            S ← S + {<x,$v_{i+1}$>};
        end
        $g_{after}$ ← g(S);
    until ($g_{after}$ = $g_{before}$) (* local minimum is concluded *)
    $S_{i+1}$ ← S;
    return $S_{i+1}$;
end
```

Figure 5.1 Local Search for PCSPs in pseudocode

The pseudocode in Figure 5.1 depicts a basic local search procedure for PCSPs. The procedure starts with a solution S_i (which is a compound label as described in section 5.1) and returns a local minimum solution S_{i+1}.

5.4 Guided Local Search for Partial CSPs

Applying guided local search to a problem simply requires the existence of a local search procedure, preferably a version of fast local search, and also a set of features which will be used to bias local search. Both prerequisites are domain dependent allowing the GLS algorithm to adapt to particular combinatorial optimisation problems. A local search procedure for PCSPs has been described in the last section. Fast local search for PCSPs will be explained later in this chapter. For the moment, we focus our attention on the features to be used in PCSPs. In particular, we examine the features used in the RLFAP instances. The same or similar features can be used in many other problems in the PCSP class.

5.4.1 Constraints

The main cost factor in PCSPs is constraint violation costs (sometimes described as relaxation costs). In a simple setting, all the problem's constraints have violation costs defined (high for hard constraints) which denote their relative importance. The cost of a solution is given by the sum of violation costs for the constraints violated by the solution. To define a basic cost function for the problem, each constraint c_i in the problem is represented by an indicator function I_{c_i} which takes the value 1 (if the constraint is violated) or the value 0 (if the constraint is satisfied). This indicator function has the following form:

Eq. 5.1
$$I_{c_i}(S) = \begin{cases} 1, & \text{if S violates constraint } \mathrm{c_i} \\ 0, & \text{if S satisfies constraint } \mathrm{c_i} \end{cases}$$

where *S* is a compound label as described in section 5.1.

A cost function accounting only for constraint violations can be defined as follows:

Eq. 5.2
$$g(S) = \sum_{i=1}^{m} I_{c_i}(S) \cdot ViolationCost(c_i)$$

where *ViolationCost* is a function which maps each constraint to its violation cost.

A basic set of features can be defined for this cost function by considering the representation of constraints as indicator functions. Each constraint in the problem is interpreted as a feature with an indicator function as given by Eq. 5.1 and a feature cost as given by the violation cost of the constraint. The augmented cost function for Eq. 5.2 has the following form:

Eq. 5.3
$$h(S) = \sum_{i=1}^{m} I_{c_i}(S) \cdot ViolationCost(c_i) + \lambda \cdot \sum_{i=1}^{m} I_{c_i}(S) \cdot p_{c_i}.$$

Essentially, the above augmented cost function introduces an extra penalty parameter p_{c_i} for each constraint c_i in the problem. The role of these extra penalty parameters is to enable GLS to guide local search towards the satisfaction of all or particular constraints. Note here, that feature costs although equal to the violation costs are not incorporated for a second time in the augmented cost function. They are solely used to determine which features (i.e. constraints) are to be further penalised in a local minimum. In the case of PCSPs, the utility function of GLS (see Eq. 2.5) takes the following form:

Eq. 5.4
$$Util(S, c_i) = I_{c_i}(S) \cdot \frac{ViolationCost(c_i)}{1 + p_{c_i}}.$$

GENET's learning scheme is essentially a version of the above penalty modification mechanism where $Util(S, c_i) = I_{c_i}(S)$ and thus all violated constraints are penalised.

Let us consider now the RLFAP. In the RLFAP, a set of constraints is given for each instance. Apart from relaxing each constraint and including its violation cost in the cost function using an indicator function, each constraint defines a feature which is used to guide local search. Feature costs are set equal to the corresponding violation costs and the cost function is augmented with a set of modifiable penalty parameters one for each constraint (see Eq. 5.3). Initially, the penalty parameters are set to 0 and each constraint (if violated) accounts only for its violation cost. Each time local search settles in a local minimum, the penalties for some of the constraints violated (the corresponding features are exhibited) are increased according to the general scheme described in section 2.6 using the utility function Eq. 5.4. Constraints with high violation costs are penalised more frequently than those with low costs because of Eq. 5.4. In the short term, local search escapes from the local minimum while in the long term, it is biased to spend more time on solutions that satisfy high cost constraints rather than low cost constraints.

5.4.2 Assignment Costs.

Some of the insoluble RLFAP instances (Scen09 and Scen10) involve assignment costs. In particular, a cost is incurred when a variable is assigned a value which is different from a default value provided. These costs are called *mobility costs* and apply to only some of the variables. RLFAP mobility costs are comparable to constraint violation costs and are linearly combined with constraint violation costs to form the objective function.

The local search of Figure 5.1 remains unchanged for these problems. If GLS were also to remain unchanged then the distribution of the search effort would only be determined by the constraint violation costs ignoring the extra mobility costs to be minimised. This will not result to the best possible performance. Extra information pertaining to mobility costs may be exploited to affect the distribution of the search effort. The set of features based on constraints is augmented with extra features that detect assignments of particular values to variables which incur mobility costs. The costs of these new features are set equal to the corresponding mobility costs. GLS operates on the combined set of features which now contains both constraints and assignments.

5.4.3 Minimise the Number of Different Values Used

In resource allocation problems, the main concern is the efficient utilisation of resources. In many cases, this translates into satisfying all requests using the minimum number of resources possible. Frequencies are the resources in RLFAP. As mentioned in section 5.2. some of the RLFAP instances are soluble (Scen01-05 and Scen11). For these instances, solutions are sought that satisfy all constraints and also use as few frequencies as possible. In other words, the goal is to find a solution which satisfies all constraints and also minimises the number of different values used. The problem is similar to finding the minimum number of colours (i.e. chromatic number) needed to colour a graph.

One possibility is to include this criterion in the cost function as it is described for graph colouring by Johnson et al. [JAMS91]. The alternative approach examined here is not to include this criterion in the objective function but instead to bias local search using penalties such that this criterion is minimised. In particular, a feature is defined for each value in the union of the domains. This feature is exhibited only when the corresponding value is assigned to at least one of the variables. By penalising the feature, we can discourage the associated value from being assigned to any of the variables. The costs of these features should be such that we prefer to penalise values that are assigned to only a few of the variables. The motivation is that values that are assigned to only a few of the variables could be swapped for values that are assigned to many of the variables, so decreasing the total number of values used. The fewer the number of variables that are assigned a value the higher should be the cost of the related feature. For a value v in the union of domains the cost of the associated feature f_v is given by:

Eq. 5.5
$$c(s_*, f_v) = \frac{\text{total number of variables}}{(\text{number of variables assigned value } v \text{ in } s_*) + 1}$$

where s_* is the local minimum solution in the context of which the feature cost is evaluated. The above feature costs are not constant like those in sections 5.4.1 and 5.4.2. This is because we cannot be sure which value can be avoided unless a solution has been found that satisfies all the constraints. If the solution violates some of the constraints, these constraints are penalised first, taking precedence over the value features in the penalty modification scheme. This leads to a feature set *hierarchy* where feature sets at the lower levels of the hierarchy are only penalised if no features of higher levels are exhibited.

5.4.4 Minimise Maximum Value Used

This last criterion is involved in only one of the RLFAP instances (Scen05). The approach taken for this criterion was to penalise constraints first and if these were satisfied to penalise the maximum value used without considering the utility function (Eq. 2.5).

5.5 Fast Local Search for Partial CSPs

A greedy local search for PCSPs evaluates all possible 1-optimal moves over all variables before selecting and performing the best move. The local search procedure described in section 5.3 is already a faster alternative to greedy local search since the neighbourhood is confined to the values of each variable. In spite of that, further improvements may be introduced in the algorithm of Figure 5.1 using the activation bits technique of Fast Local Search described in section 2.8.

In the case of PCSPs, a bit is attached to each problem variable. If the bit of a variable is 1 then the variable is called *active* and it is examined for improving moves otherwise it is called *inactive* and it is ignored by local search. Whenever a variable is examined and a move is performed the activation bit of the variable remains set to 1 otherwise it turns to 0 and the variable is not examined in future iterations. Additionally, if a move is performed, activation spreads to other variables which have their bits set to 1. In particular, we set to 1 the bit of variables where improving moves may occur as a result of the move just performed. In general, such variables are those that are connected via a constraint to the variable where the current move was performed. Three main schemes for the spreading of activation may be used. The schemes determine which variables are to be activated when the value of a variable changes and they are the following:

S1. Activate all variables connected via a constraint to the variable which changed value.
S2. Activate only variables that are connected via a constraint which is violated.
S3. Activate only variables that are connected via a constraint that changed state (i.e. violated → satisfied or satisfied → violated) as a result of the move.

S2 and S3 are the more approximate schemes among the three, activating fewer variables than S1.

The overall procedure starts with all the bits set to 1. The variables are continuously scanned from first to last. Only variables with the bit set to 1 are being searched. Each time a variable is searched and its value is changed, the variable remains active and also activation spreads to other related variables according to one of the activation schemes (S1, S2, or S3). On the other hand, if the value of the variable is not changed the variable becomes inactive (i.e. the bit is set to 0). The process stops under the same conditions that apply to local search without activation bits depicted in section 5.3.

Each time local search settles in a local minimum, GLS penalises some of the features. A limited number of variables are activated and a fresh fast local search cycle starts. Depending on the features penalised, we activate variables relating to these features such that moves examined aim at removing the penalised features from the solution. Table 5.2 gives the relation between features penalised and variables activated.

Feature penalised	Activate
Constraint	Variables associated with the constraint
Assignment	Variable the assignment refers to
Value	Variables assigned the value

Table 5.2 Associations between features penalised and variables activated.

Next, we give results indicative of the performance of GLS on the RLFAP instances. Some of these results were up to recently the best known solutions for these instances.

5.6 Performance of Guided Local Search on the RLFAP Instances

To evaluate the performance of GLS, we apply it to the eleven instances of RLFAPs in the public domain [RLFAP94]. The objective is to evaluate the above mentioned different activation schemes for GLS, find out whether GLS could possible find solutions in all soluble problems, and find good quality solutions in all the problems.

Experiments performed on the RLFAP using each of the three activation schemes showed that all schemes perform equally well in terms of solution quality with S3 having a slight advantage in run times over scheme S2 and being much faster than scheme S1. The results reported here give the average performance of the algorithm using the activation scheme S3.

Ten runs were performed on each instance starting from random initial solutions. In each run, the algorithm was allowed to complete 100,000 penalty cycles (i.e. GLS iterations as in Figure 2.2) before being stopped. Hard constraints in all instances were assigned a high violation cost of 1,000,000. The regularisation parameter λ was also set to this value though values of λ in the range $[2\times10^5, 2\times10^6]$ also performed well. Table 5.3 presents the results obtained. Experiments were performed on a DEC Alpha 3000/600 (175 MHz) with GLS implemented in C++.

RLFAP Instance	Best Solution	Average Cost (Std. Dev.)	Worst Solution	Average Iterations	Average Time (CPU sec.)
Scen01	16	18.6 (2.3)	22	1,895	8.77
Scen02	14	14 (0.0)	14	233	0.59
Scen03	14	15.4 (1.3)	18	1,626	5.62
Scen04	46	46 (0.0)	46	60	0.46
Scen05	792	792 (0.0)	792	1,584	8.50
Scen06	3,628	4,333.8 (766.0)	6,042	34,365	120.87
Scen07	427,054	530,641.1 (79,666.7)	700,685	20,412	78.79
Scen08	294	335.7 (34.7)	377	50,626	232.88
Scen09	15,805	15,999.7 (194.7)	16,340	31,150	129.4
Scen10	31,533	31,686.6 (146.1)	31,942	64,258	297.29
Scen11	28	not applicable	Not solved	21,577	93.97

Table 5.3 Average performance of GLS on the RLFAP instances.

RLFAP Instance	Best solutions found by GLS
Scen01	16
Scen02	14
Scen03	14
Scen04	46
Scen05	792
Scen06	3,570
Scen07	374,705
Scen08	282
Scen09	15,680
Scen10	31,517
Scen11	28

Table 5.4 Best solutions for RLFAP found by GLS.

The tunnelling algorithm, a predecessor of GLS, significantly improved the best known solutions on the RLFAP that accompanied the initial release of the instances (see [VT94]). GLS with fast local search found even better solutions, improving over the tunnelling algorithm in many instances. Table 5.4 summarises the best solutions found by GLS for the RLFAP instances. Note here that solutions for Scen02-Scen05 have been proven optimal by complete search techniques [THL95].

5.7 Comparison with Extended GENET and a Tabu Search Variant.

Independently from this work another method also based on the GENET neural network has been developed for the RLFAP by G. vom Scheidt [Sch95]. The method is like GENET a neural network architecture and is described in the paper by Boyce et al. [BDST95] where it is compared with a tabu search variant. For convenience, we shall call this method *extended GENET*. Extended GENET in pure algorithm terms (after removing the neural network element) has many similarities as well as differences with GENET [WT91, DTWZ95] and GLS. Although it uses an augmented cost function (minimised by the NN), it penalises all constraint violations by increasing penalties proportionally to the constraint violation costs. No scheme is used for distributing the search effort (no memory of past actions) though a similar effect is attempted by varying penalty increments amongst constraints. Minimisation of the number of different values used is achieved by incorporating an additional cost term to the cost function weighted by an appropriate coefficient. The algorithm has not been applied to instances involving mobility costs (Scen09 and Scen10). Extended GENET makes use of a fast local search procedure using an activation scheme similar to S1 but does not consider sideways moves.

Table 5.5 contrasts the results reported in Boyce et. al [BDST95] for tabu search and extended GENET with those reported for GLS in Table 5.3. Experiments in [BDST95] were also performed on DEC Alpha machines with the algorithms implemented in C++ and therefore a relatively fair comparison in running times can be made. As one can see in Table 5.5, GLS outperforms both extended GENET and the tabu search variant. For problems (Scen01-Scen05), GLS succeeded more times in finding the optimum than either tabu search or extended GENET. Moreover, GLS found better solutions than these two methods in all the insoluble instances

(Scen06-Scen10). For problem Scen11, the other methods tried to find an assignment that satisfied the constraints and therefore no comparison can be made with GLS which went further trying to minimise the number of different values used. In terms of run times GLS was between 6 and 56 times faster than extended GENET while tabu search required an enormous amount of time in comparison with either extended GENET or GLS probably because of inefficient implementation.

RLFAP Instance	best solution found			found optimum			average time		
	GLS	Ext. GENET	Tabu Search	GLS	Ext. GENET	Tabu Search	GLS	Ext. GENET	Tabu Search
Scen01	16	16	18	30%	20%	n.a.	8.77sec	75sec	3hrs
Scen02	14	14	14	100%	100%	70%	0.59sec	9sec	4min
Scen03	14	14	14	40%	10%	20%	5.62sec	32sec	34min
Scen04	46	46	n.a.	100%	100%	n.a.	0.46sec	12sec	n.a.
Scen05	792	792	n.a.	100%	30%	n.a.	8.50sec	8min	n.a.
Scen06	3,628	3,852	9,180	-	-	-	2min	10min	14min
Scen07	427,054	435,132	6,541,695	-	-	-	1.3min	18min	46min
Scen08	294	366	1,745	-	-	-	3.9min	32min	6hrs
Scen09	15,805	n.a.	16,873	-	-	-	2.2min	n.a.	18min
Scen10	31,533	n.a	31,943	-	-	-	5min	n.a	2hrs
Scen11[9]	28 values	0 viol.	0 viol.	80%	60%	60%	1.6min	25sec	54min

Table 5.5 Comparison of GLS with tabu search and extended GENET. Results for tabu search and extended GENET are from Boyce et al. [BDST95].

Table 5.6 provides further evidence on the superiority of GLS over extended GENET. The solution quality of GLS is compared with that of extended GENET on the insoluble problems (Scen06-Scen09). Results for extended GENET are from [Sch95].

RLFAP	Average Solution Cost (Average CPU Time)		Percentage excess of Ext. GENET solutions over GLS solutions (Times faster than GENET)
Instance	GLS	Extended GENET	
Scen06	4,333.8 (2 min)	5,076 (10.2 min)	17% (5 times)
Scen07	530,641.1 (1.3 min)	727,458 (18.3 min)	37% (14 times)
Scen08	335.7 (3.9 min)	451 (31.7 min)	34% (8 times)

Table 5.6 GLS and extended GENET on insoluble instances. Results for extended GENET are from [Sch95].

[9] For Scen11, GLS minimizes the number of different values used while tabu search and extended GENET simply try to find a assignment that satisfied the constraints.

5.8 Comparison with the CALMA Project Algorithms

The RLFAP instances were made publicly available in the framework of the European collaborative project CALMA (Combinatorial Algorithms for Military Applications). Six research groups from three countries participated in the project. Summary results have been reported recently by Tiourine et al. [THL95] on a set of algorithms, including extended GENET and tabu search mentioned in the last section, developed by the six CALMA project research groups. In Table 5.7, we compare these summary results (from Tiourine et al. [THL95]) with the results for GLS.

As it can be seen in Table 5.7, GLS achieves a very good performance compared with the other algorithms and taking into account the values of the best known solutions. In summary, it applies to all problems finding solutions of high quality while it is many times faster than the other algorithms. Algorithms which produce marginally better solutions than GLS (e.g. Genetic Algorithms-LU) were applied to only a subset of the problems and require substantially more time, fine tuning and probably implementation effort. On the other hand, although algorithms such as SA-EUT, extended GENET-KCL and Variable Depth Search-EUT, are applied to most problems and find solutions of good quality, they are between 5 to 100 times slower than GLS (especially on the insoluble instances). This cannot be attributed just to the different machines used in experiments. Besides, although the GA by UEA produces good results for Scenarios 6 and 11, it performs badly in Scenarios 7 and 8; compared to it, GLS is not only much faster, but also more consistent in its performance. Bessiere et al. [BFR95] also applied arc-consistency algorithms to Scenarios 3, 5, 8 and 11. Since only the satisfiability issue (not optimisation) was addressed their results are not comparable with the rest in this section. To conclude, GLS is a highly competitive, if not the best, method amongst the algorithms developed so far for the problem that are known to us. It is the fastest algorithm which consistently provides quality solutions (never much worse than the best found so far, sometimes the best). Significantly, this is achieved almost without any tuning required.

5.9 Discussion

We are well aware of the danger of over-generalising results obtained in competitive tests, especially when running time is compared, as Hooker pointed out [Hoo95]. In the experiments, we have shown that GLS is capable of solving RLFAPs where solutions exist, and finding solutions with top quality in insoluble RLFAPs, compared with, and in many cases, better than, other state-of-the-art algorithms designed for RLFAPs.

The running time that we present in Table 5.7 is meant for reference only. The timing should not be compared seriously, especially when different machines have been used and we know nothing about the software platforms used in other research projects. However, there is some value in reporting the running time: it gives an idea for evaluating algorithms.

5.10 Conclusions

In this chapter, the application of the method to Partial CSPs was studied in the context of a real world PCSP, namely the Radio Link Frequency Assignment Problem (RLFAP). Results reported on RLFAP demonstrated the effectiveness and efficacy of the method. The technique finds high quality solutions in very short running times, outperforming alternative schemes suggested for the problem. Given the generality and effectiveness of the approach, GLS can be considered a promising optimisation technique for real world constrained optimisation problems.

Scenario / Method over optimality	1	2	3	4	5	11	Time	6	7	8	9	10	Time	Machine
Simulated Annealing (EUT)	2	0	2	0	0	2	1min	6%	65%	5%	0%	0%	310min	SUN SPARC 4
Taboo Search (EUT)	2	0	2	0	-	0	5min	-	-	-	-	-	-	SUN SPARC 4
Variable Depth Search (EUT)	2	0	2	0	-	10	6min	3%	0%	14%	0%	0%	85min	SUN SPARC 4
Simulated annealing (CERT)	4	0	0	0	-	10	41min	42%	1299%	70%	2%	0%	42min	SUN SPARC 10
Tabu Search (KCL) (see Section 9)	2	0	0	0	0	2	40min	167%	1804%	566%	8%	1%	111min	DEC Alpha
Extended GENET (KCL) (see Section 9)	0	0	0	0	0	2	2min	12%	27%	40%	-	-	20min	DEC Alpha
Genetic Algorithms (UEA)	6	0	2	0	-	10	24min	0%	386%	134%	3%	0%	120min	DEC Alpha
Genetic Algorithms (LU)	-	-	-	-	-	-	-	0%	0%	0%	0%	0%	hours	DEC Alpha
Partial Constraint Satisfaction (CERT)	4	0	6	0	0	-	28min	83%	2563%	246%	47%	12%	6min	SUN SPARC 10
Potential Reduction (DUT)	0	0	2	0	0	-	3min $^{+}$	27%	-	-	4%	1%	10min +	HP9000/720
Branch and Cut (DUT, EUT)	0*	0*	0*	0*	0*	0*	<10min $^{+}$	-	-	-	-	-	-	-
Constraint Satisfaction (LU)	0*	0*	0*	0*	0*	0*	hours	-	-	-	-	-	-	PC
Guided Local Search	**0**	**0**	**0**	**0**	**0**	**6**	**20sec**	**4%**	**9%**	**7%**	**0.7%**	**0.003%**	**2.88min**	**DEC Alpha**
Best Known Solution	16*	14*	14*	46*	792*	22*		3437	343594	262	15571	31516		

Table 5.7 Comparison of GLS with the CALMA project algorithms. Results for the CALMA project algorithms are from Tiourine et al. [THL95].

CERT - Centre d'Etudes et de Recherces de Toulouse, France;
DUT - Delft University of Technology, The Netherlands;
EUT - Eidhoven University of Technology, The Netherlands;
KCL - King's College London, United Kingdom;
LU - Limburg University, Maastricht, The Netherlands;
UEA - University of East Anglia, Norwich, United Kingdom.

soluble instances (Scen1-5, Scen11): 4 number of frequencies above optimal solution for best solution found by algorithm
insoluble instances (Scen6-10): 42% deviation from the best known solution for best solution found by algorithm
- method not applicable or no solution is reported
* optimality of solution is proved
$^{+}$ pre-processing time not included
Time columns give, for reference, average running times for the classes of soluble and insoluble instances.

GLS was implemented in C++ and run on DEC Alpha 3000/600 (175 MHz).

Chapter 6

Workforce Scheduling

In the last chapter, we presented the application of GLS and FLS to a constrained optimisation problem in which the main objective was the minimisation of constraint violations. Constrained optimisation problems are not always of this type. In many domains, partial solutions are sought which assign values only to a subset of the variables such that all the problem's constraints are satisfied. Such problems are very useful in modelling overloaded resource allocation systems. In these systems, hard resource constraints are satisfied only if a subset of activities is allocated resources or in PCSP terms if a subset of the variables is assigned values. A penalty (or utility) is defined for each activity when this activity is not allocated (or allocated) resources. If penalties are used instead of utilities then the optimal solution is that which minimises the sum of penalties for the unallocated activities. NP-hard problems such as the Maximum Knapsack [MT90], Maximum Channel Assignment [Sim90] and

Bandwidth Packing [LG93, AFPR93] are of this type. In this chapter, we are going to examine BT's Workforce Scheduling which apart from the above characteristics also incorporates elements from the well-known Vehicle Routing Problem with Time Windows (VRPTW) [Sol87]. The problem examined in here is representative of the situations arising in the Work Manager job allocation system of British Telecommunications plc. Work Manager is probably the largest automated job allocation system in the world providing work for almost 20,000 field engineers.

6.1 BT's Workforce Scheduling Problem

The problem is to schedule a number of engineers to a set of jobs, minimising total cost according to a function which is to be explained below. Each job is described by a triple:

(Loc, Dur, Type)

where *Loc* is the location of the job (depicted by its *x* and *y* co-ordinates), *Dur* is the standard duration of the job and *Type* indicates whether this job must be done in the morning, in the afternoon, as the first job of the day, as the last job of the day, or "don't care".

Each engineer is described by a 5-tuple:

(Base, ST, ET, OT_limit, Skill)

where *Base* is the x and y co-ordinates at which the engineer locates, *ST* and *ET* are this engineer's starting and ending time, *OT_limit* is his/her overtime limit, and *Skill* is a skill factor between 0 and 1 which indicates the fraction of the standard duration that this engineer needs to accomplish a job. In other words, the smaller this *Skill* factor, the less time this engineer needs to do a job. If an engineer with skill factor 0.9 is to serve a job with a standard duration (*Dur*) of 20 then this engineer would actually take 18 minutes to finish the job.

The cost function which is to be minimised is defined as follows:

Eq. 6.1
$$TotalCost = \sum_{i=1}^{NoT} TC_i + \sum_{i=1}^{NoT} OT_i^2 + \sum_{j=1}^{NoJ} \left(Dur_j + Penalty\right) \times UF_j$$

where:

NoT = number of engineers,
NoJ = number of jobs,
TC_i = Travelling Cost of engineer i,
OT_i = Overtime of engineer i,
Dur_j = Standard duration of job j,
UF_j = 1 if job j is not served; 0 otherwise,
Penalty = constant (which is set to 60 in the tests).

The travelling cost between (x_1, y_1) to (x_2, y_2) is defined as follows:

Eq. 6.2
$$TC\left((x_1, y_1), (x_2, y_2)\right) = \begin{cases} \dfrac{\frac{\Delta_x}{2} + \Delta_y}{8} & , \Delta_x > \Delta_y \\ \dfrac{\frac{\Delta_y}{2} + \Delta_x}{8} & , \Delta_y \geq \Delta_x \end{cases}$$

Here Δ_x is the absolute difference between x_1 and x_2, and Δ_y is the absolute difference between y_1 and y_2. The greater of the x and y differences is halved before summing. Engineers are required to start from and return to their bases everyday. An engineer may be assigned more jobs than he/she can finish.

6.2 Local Search for Workforce Scheduling

To tackle BT's workforce scheduling problem, we represent a candidate solution (i.e. a possible schedule) by a permutation of the jobs. Each permutation is mapped into a schedule using the deterministic algorithm depicted in Figure 6.1:

```
procedure **Evaluation** (input: one particular permutation of jobs)
    1.  For each job, order the qualified engineers in ascending order of the distances
        between their bases and the job (such orderings only need to be computed once and
        recorded for evaluating other permutations).
    2.  Process one job at a time, following their ordering in the input permutation. For each
        job x, try to allocate it to an engineer according to the ordered list of qualified
        engineers:
        2.1.  to check if engineer g can do job x, make x the first job of g; if that fails to
              satisfy any of the constraints, make it the second job of g, and so on;
        2.2.  if job x can be fitted into engineer g's current tour, then try to improve g's
              new tour (now with x in it): the improvement is done by a simple 2-opting
              algorithm (see section 3.2), modified in the way that only better tours which
              satisfy the relevant constraints will be accepted;
        2.3.  if job x cannot be fitted into engineer g's current tour, then consider the next
              engineer in the ordered list of qualified engineers for x; the job is
              unallocated if it cannot fit into any engineer's current tour.
    3.  The cost of the input permutation, which is the cost of the schedule thus created, is
        returned.
```

Figure 6.1 Algorithm for mapping job permutations into complete schedules

Given a permutation, local search is performed in a simple way: a pair of jobs is examined at a time. Two jobs are swapped to generate a new permutation if the new permutation is evaluated (using the Evaluation procedure above) to a lower cost than the original permutation.

The starting point of local search is generated heuristically and deterministically: the jobs are ordered by the number of qualified engineers for them. Jobs which can be served by the fewest number of qualified engineers are placed earlier in the permutation.

6.3 Fast Local Search for Workforce Scheduling

So far we have defined an ordinary first improvement local search algorithm. Each solutions has $O(n^2)$ neighbours, where n is the number of jobs in the workforce scheduling problem.

To apply the fast local search to workforce scheduling, each job permutation position has associated with it an activation bit, which takes binary values (0 and 1). These bits are manipulated according to the general FLS algorithm of section 2.8. In particular,

1. all the activation bits are set to 1 (or "on") when local search starts;
2. the bit for job permutation position x will be switched to 0 (or "off") if every possible swap between the job at position x and the other jobs under the current permutation has been considered, but no better permutation has been found;
3. the bit for job permutation position x will be switched to 1 whenever x is involved in a swap which has been accepted.

During local search, only those job permutation positions whose activation bits are 1 will be examined for swapping. In other words, positions which have been examined for swapping but failed to produce a better permutation will be heuristically ignored. Positions which are involved in a successful swap recently will be examined further. The overall effect is that the size of neighbourhood is greatly reduced and resources are invested in examining swaps which are more likely to produce better permutations.

6.4 Guided Local Search for Workforce Scheduling

To apply GLS to workforce scheduling, we need to implement a local search algorithm for workforce scheduling, identify a set of features to be used and assign costs to them. In the previous section, we have described a fast local search algorithm for BT's workforce scheduling problem.

Our next task is to define the solution features to be used and assign costs to them. In the workforce scheduling problem, the inability to serve jobs incurs a cost, which plays an important part in the objective function which is to be minimised. Therefore, we intend to bias local search to serve jobs of high importance. To do so, we define a feature for each job in the problem:

Eq. 6.3
$$I_{job_j}(schedule) = \begin{cases} 1, & job_j \text{ is unallocated in } schedule \\ 0, & job_j \text{ is allocated in } schedule \end{cases}.$$

The cost of this feature is given by *(Dur*$_j$ *+ Penalty)* which is equal to the cost incurred in the cost function (Eq. 6.1) when a job is unallocated. The jobs penalised in a local minimum are selected according to the utility function (Eq. 2.5) which for workforce scheduling takes the form:

Eq. 6.4
$$Util\left(schedule, job_j\right) = I_{job_j}(schedule) \cdot \frac{\left(Dur_j + Penalty\right)}{1 + p_j}.$$

The travelling cost is taken care of by the ordering of engineers by their distance to the jobs in the local search described in the Evaluation procedure above as well as 2-Opting. (If the travelling cost in this problem is found to play a role as important as unallocated jobs, we could associate a penalty to each possible edge as we did for the TSP in chapter 3 to further minimise this cost factor). Integrated into GLS, FLS will switch on (i.e. switching from 0 to 1) the activation bits associated with the positions where the penalised jobs currently lie.

It may be worth noting that since the starting permutation is generated heuristically, and local search is performed deterministically, the application of FLS and GLS presented here does not involve any randomness.

6.5 Experimental Results and Comparison with GAs, SA and CLP.

The best results published so far on the workforce scheduling problem is in Azarmi & Abdul-Hameed [AA95]. Azarmi & Abdul-Hameed have looked at simulated annealing, constraint logic programming [Hen89, LWR95] and genetic algorithms [Hol75, Gol89, Dav91, WT94, ERR94]. The results are based on a benchmark test problem with 118 engineers and 250 jobs. Each job can be served by 28 engineers on average, which means the search space is roughly 28^{250}, or 10^{360}, in size. This suggests that a complete search is very unlikely to succeed in finding the optimal solution.

Azarmi & Abdul-Hameed [AA95] reported results obtained by a particular genetic algorithm (GA), two constraint logic programming (CLP) implementations, ElipSys and CHIP, and a simulated annealing (SA) approach. Azarmi & Abdul-Hameed cited Muller et. al. [MMS93] for the GA approach and Baker [Bak93] for the SA approach. Results obtained by GA and CLP were "repaired" (i.e. amended by local search). All the tests reported there relax the constraints in the problem by:

(a) taking first jobs as AM jobs, and last jobs as PM jobs; and

(b) allowing no overtime.

The best result (total cost) so far was 21,025, which was obtained by the SA approach. No timing was reported on the tests. These results are shown in Table 6.1 (Group I).

To allow comparison between our results and the published ones, we have made the same relaxation to the problem. The results are reported in Group II of Table 6.1. FLS obtained a result of 20,732, which is better than all the reported results. This result is further improved by GLS. The best result obtained in this group is 20,433, when λ is set to 100 in GLS. Such results are remarkable as the best results published were obtained by nontrivial amount of work by prominent research groups in UK. (Note that a saving of 1% could be translated to tens of thousands of pounds per day!)

In the objective function, the overtime term is squared. This discourages overtime in schedules, but it does not mean that a good schedule cannot have overtime. We tried to restate this constraint, but gave each engineer a limit in overtime. The best result, which were found by limiting overtime to 10 minutes per engineer, is shown in Group III of Table 6.1. FLS in this group obtained a result of 20,224, which was better than all the results in Group II. The best result in Group III, which is 19,997, was found by GLS when λ was set to 20.

The λ parameter is the only parameter that needs to be set in GLS (there are relatively more parameters to set in both GA and SA). The above test results show that the total cost is not terribly sensitive to the setting of λ.

Algorithms		Total cost	CPU time (sec)	Travel cost	Cost (number) of unallocated jobs	over-time cost
Group I: Best results reported in the literature (no overtime allowed):						
GA		23,790	N.A.	N.A.	N.A. (67)	disallow
GA + repair		22,570	N.A.	N.A.	N.A. (54)	disallow
CLP - ElipSys + repair		21,292	N.A.	4,902	16,390 (53)	disallow
CLP - CHIP + repair		22,241	N.A.	5,269	16,972 (48)	disallow
SA		21,025	N.A.	4,390	16,660 (56)	disallow
Group II: Best results on FLS and GLS with overtime disallowed:						
Fast Local Search (FLS)		20,732	1,242	4,608	16,124 (49)	disallow
	λ = 10	20,556	5,335	4,558	15,998 (48)	disallow
	λ = 20	20,497	7,182	4,533	15,864 (49)	disallow
Fast GLS	λ = 30	20,486	6,756	4,676	15,810 (50)	disallow
	λ = 40	20,490	5,987	4,743	15,747 (48)	disallow
	λ = 50	20,450	3,098	4,535	15,915 (49)	disallow
	λ = 100	20,433	9,183	4,707	15,726 (48)	disallow
Group III: Best results on FLS and GLS, with a maximum of 10 minutes overtime allowed:						
Fast Local Search (FLS)		20,224	1,244	4,651	15,448 (51)	125
	λ = 10	20,124	4,402	4,663	15,329 (50)	132
	λ = 20	19,997	4,102	4,648	15,209 (49)	140
Fast GLS	λ = 30	20,000	2,788	4,690	15,155 (48)	155
	λ = 40	20,070	4,834	4,727	15,194 (48)	149
	λ = 50	20,055	2,634	4,690	15,197 (49)	168
	λ = 100	20,132	2,962	4,779	15,152 (48)	201
1. GA, CLP and SA results from Azarmi & Abdul-Hameed [AA95], Muller et. al. [MMS93] and Baker [Bak93]; 2. FLS and GLS are implemented in C++, all results obtained from a DEC Alpha 3000/600 175MHz machine. 3. The benchmark problem, which has 118 engineers and 250 jobs, was obtained from British Telecom Research Laboratories, UK.						

Table 6.1 Results obtained in BT's benchmark workforce scheduling problem.

6.6 The Role of FLS in BT's Workforce Scheduling Problem

To evaluate the role of the activation bits in the efficiency of FLS, we compared FLS with a best improvement local search algorithm which used the same moves as FLS, but without using activation bits to reduce its neighbourhood (we refer to this algorithm as LS). The results are shown in Table 6.2.

When no overtime is allowed, FLS runs 16 times faster than LS, which converged to a slightly worse local minimum. When a maximum of 10 minutes is allowed for overtime, FLS runs 20 times faster than LS, though LS produced a slightly better result. Our conclusion is that the activation bits help to speed up FLS significantly and there is no convincing evidence that quality of results has been sacrificed in the workforce scheduling problem.

Algorithms		Total cost	CPU time (sec)	speedup by FLS in cpu time	Travel cost	Cost (number) of unallocated jobs	over-time cost
No overtime allowed	FLS	20,732	1,242	16 times	4,608	16,124 (49)	disallow
	LS	20,788	20,056		4,604	16,184 (50)	disallow
Max. 10 min. OT allowed	FLS	20,224	1,244	20 times	4,651	15,448 (51)	125
	LS	20,124	25,195		4,595	15,358 (48)	171
Notes: Local Search (LS) use the same hill climbing strategy as FLS, but no activation bits are used; Both algorithms implemented in C++, all results obtained from a DEC Alpha 3000/600 175MHz machine.							

Table 6.2 Evaluation of the efficiency of FLS.

6.7 Remarks

We have also experimented with random starting permutations and a starting permutation with the jobs ordered by the ratio between their duration and the number of qualified engineers. Their results are shown in Table 6.3.

Heuristics used in generating starting permutation	Initial Cost	After FLS		After Fast GLS	
		cost	cpu sec	cost	cpu sec
Random ordering	25,886	21,204	767	20,287	7,639
Job duration / # of qualified eng.	23,828	20,286	903	20,187	2,468
# of qualified engineers	22,846	20,224	1,218	20,132	2,962
Notes: a maximum of 10 minutes is allowed in overtime; a maximum of 500 penalty cycles is allowed in GLS, which uses $\lambda = 100$; all programs implemented in C++; all results obtained from a DEC Alpha 3000/600 175MHz machine.					

Table 6.3 Ordering heuristics used in starting permutation.

In Table 6.3, an (almost) arbitrary λ value of 100 has been chosen to give the reader more information about the sensitivity of GLS over this parameter (though this was not the parameter under which the best result were generated when overtime was allowed). Results in Table 6.3 show that the result of FLS can be affected by the initial ordering of the jobs, though even the worst result is comparable with those reported in the literature. However, Fast GLS is relatively insensitive to it - all the results of GLS are better than the best result reported in the literature.

6.8 Conclusions

Real world problems are often characterised by complex objective functions, side constraints and hierarchical structure. To deal effectively with them, it is sometimes necessary to develop tailor-made techniques which combine together a number of heuristics. These heuristics may operate at different stages of the optimisation process or at different levels of the problem. Using BT's workforce scheduling, we demonstrated how GLS and FLS can provide the foundation for such tailor-made techniques.

GLS and FLS easily integrate with each other and with the complex move operators and heuristics often required. Moreover, they provide the tools to identify the most important cost factors in the problem and minimise them effectively. Tuning is relatively simple reducing the demands from the users of the scheduler. Finally, solutions obtained by the GLS-FLS combination are of high quality and in the case of BT's workforce scheduling problem better than the best results reported in the literature. Last but not least, this chapter viewed in conjunction with the chapter on the RLFAP problem provides a complete guide for applying GLS and FLS to constrained optimisation problems.

Chapter 7

Nonconvex Optimisation

In the preceding chapters, we examined the application of Guided Local Search to a number of hard combinatorial optimisation problems from the well-known TSP and QAP to real world problems such as the RLFAP and BT's Workforce Scheduling problem. In this chapter, we are going to demonstrate that the potential applications of GLS are not limited to optimisation problems of discrete nature but also to difficult continuous optimisation problems.

7.1 Nonconvex Optimisation and Global Optimisation Methods

Continuous optimisation problems arise in many engineering disciplines (such as electrical and mechanical engineering) in the context of analysis, design or simulation tasks. Particularly difficult problems are those with non-linear multi-extremal cost functions (that is functions with many local minima). These problems, also known as nonconvex optimisation problems [HL95], are difficult to solve using deterministic gradient-based algorithms used extensively elsewhere in continuous optimisation. Gradient algorithms can be easily trapped in the many local minima of the cost function, so failing to reach the global minimum.

Global Optimisation (GO) methods which seek the global minimum are utilised to solve such problems. The most simple global optimisation algorithm is to run a gradient algorithm many times and from different starting points in the hope that the global minimum will be amongst the local minima obtained over the many runs. Example of such algorithm is the variation of the Sequential Unconstrained Minimisation Technique suggested in [HL95]. Many other GO algorithms exist which make use of gradient techniques or derive directly from general search methods such as Genetic Algorithms [Hol75], Simulated Annealing [KGV83, Ing89], Function Smoothing [ST90], Orthogonal Arrays with the GRG algorithm [KC93] to name but a few.

7.2 Local Search for Continuous Optimisation Problems

Recently and mainly driven by the use of Genetic Algorithms [Hol75, Gol98, Dav91] in combinatorial optimisation, GO methods have been developed which deal with nonconvex optimisation as a combinatorial optimisation task. The idea is to convert the continuous problem to a discrete one by encoding the real variables of the cost function as binary strings.

In the case of binary encoding, a binary string value is interpreted to represent an integer in base-2 notation. The mapping of the binary string to a real variable works as follows. The binary string value is first converted to the corresponding integer. This integer is then scaled by the appropriate coefficient to give a real value in the desired range (i.e. domain of variable) [Dav91]. One binary string is used for each problem variable and combinatorial search is utilised to find these binary string configurations which after decoding result in the optimal value for the real-valued cost function. Increasing the number of bits used for representing each variable increases the accuracy of the solution but also results in an increase of the combinatorial search space.

Although binary encoding schemes were principally developed for Genetic Algorithms, they have also been used in the context of local search [WZ93, BT94]. To explain how local search operates in this case, let us consider the problem with two variables $x \in A \subset \Re$ and $y \in B \subset \Re$ and a function f(x, y) to be minimised in $A \times B \subset \Re^2$. A local search move flips the value of a bit in the binary string representing the solution (comprises the binary strings of the function's variables). In the x-y plane, bit flips translate to "jumps" in either the *x* or *y* direction. The more significant the bit changed, the larger the step of the "jump" performed. Local search starting from a random binary string examines all possible bit flips and performs that which results in the maximum reduction in cost (minimisation case). The new solution if better replaces the old solution and the procedure continues from there on until a solution is reached for which no further improvement is possible. As before, GLS can be used to help local search escape from local minima moreover distribute search efforts in the search space.

7.3 The Sine Envelope Sine Wave (F6) Function

As mentioned in section 7.1, nonconvex optimisation refers to non-linear multi-extremal cost functions. An example of such a function, mentioned many times in the literature, is the sine envelope sine wave function also known as F6 [Dav91, WZ93, BT94]:

Eq. 7.1 $$F6(x,y) = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{\left[1.0 + 0.001\cdot\left(x^2+y^2\right)\right]^2}$$

minimised in the domain [-100,100]×[-100,100]. F6 has been suggested as a benchmark for Genetic Algorithms [SCED89].

A cross section of the function is shown in Figure 7.1. The global minimum of F6 is located at (0,0) where the function takes the value 0. The basin of the global minimum is very narrow and therefore difficult to reach unless a lucky start is made from within the domain of attraction of the global minimum. The many local minima of the function are arranged in concentric cycles around the global minimum forming an ideal trap for hill-climbing based techniques. In F6, local gradients provide limited (if any) information on the location of the global minimum. GLS may be exploited to help local search to escape from local minima and moreover distribute search effort in the search space.

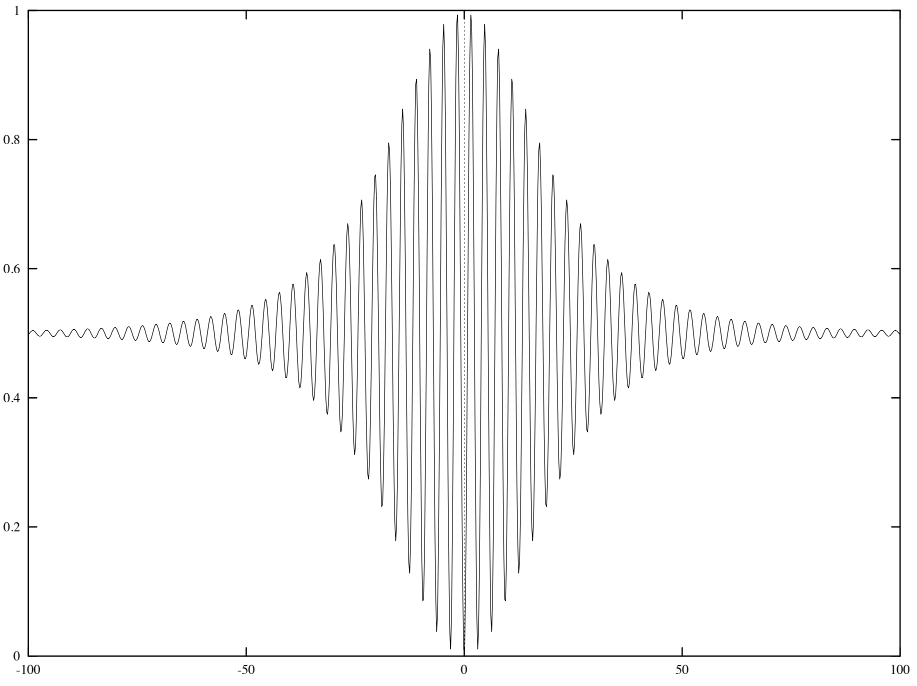


Figure 7.1 Cross section of F6 function

7.4 Guided Local Search for Global Optimisation

GLS is iteratively posting constraints which modify the landscape and guide local search out of local minima and towards promising areas in the search space. Constraint posting in this problem could be based on information gathered during the search process. For example, if local search reaches a local minimum then an assumption can be made that the global minimum is unlikely to reside in the surrounding area. Constraints could then be introduced that exclude this area from being searched in future iterations. These constraints are essentially soft because we cannot be sufficiently confident that local search thoroughly searches the space around a solution when this solution is visited.

A set of features is defined that allow us to constrain solutions. A feature can be any solution property represented by an indicator function (see section 2.4). A simple setting for global optimisation is to divide the domains of variables into a number of non-overlapping and equally-sized intervals. Let us consider the variable $x \in (a,b]$. A set of features f_i, i=1, ...,n, can be defined by the intervals $(a_0=a,a_1]$, $(a_1, a_2]$, ..., $(a_{n-1}, a_n=b]$ as follows:

Eq. 7.2

$$I_i(x) = \begin{cases} 1, & x \in (a_{i-1}, a_i] \\ 0, & otherwise \end{cases}.$$

Each feature f_i is attached to a penalty parameter p_i to allow GLS to penalise solutions that are characterised by the feature such that they can be avoided. The cost function is augmented with penalty terms to form the augmented cost function. This function replaces the original function and it is minimised instead. The augmented version of F6 is defined as follows:

Eq. 7.3

$$H(x,y) = F6(x,y) + \lambda \cdot \left(\sum_{i=1}^{n} I_{xi}(x) \cdot p_{xi} + \sum_{j=1}^{m} I_{yj}(y) \cdot p_{yj} \right),$$

where *n* the number of features defined over the domain of *x*, *m* is the number of features defined over the domain of *y,* and λ is the parameter that controls the relative importance of constraints with respect to the primary cost term (i.e. function to be minimised). Initially, all penalty parameters of features are set to 0 ($p_{xi} = 0, p_{yj} = 0, i = 1, ..., n, j = 1, ..., m$). Each time local search settles in a local minimum, we simply increment by one the penalty parameters of the features exhibited by the local minimum (only two at a time). This increases by 2*λ the cost of all solutions that lie in the intersection of the zones corresponding to the penalised features and by λ the cost of all solutions that lie in either one of these zones (see Figure 7.2). As a result, local search will primarily avoid the rectangular area with centre the local minimum and also to a lesser degree the two zones that run parallel to the co-ordinate axis as shown in Figure 7.2. This simple technique can be used to minimise arbitrary functions. In fact, there is nothing that binds the method to F6 which may not be used for other functions with two or more variables. In the following, we examine the results obtained for F6.

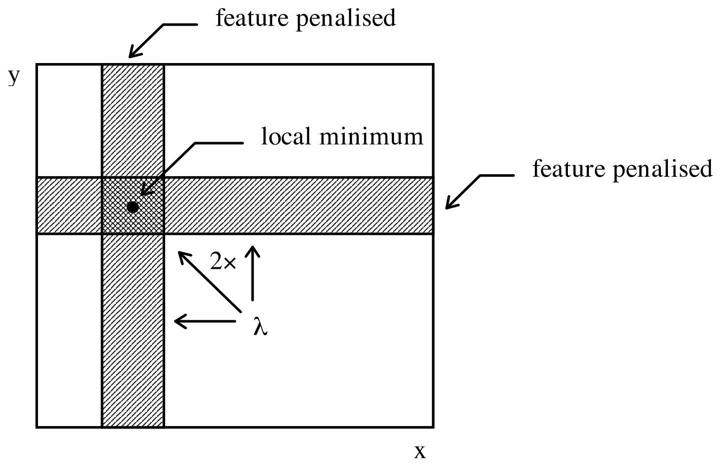


Figure 7.2 Changes in cost due to penalising the features exhibited by a local minimum

7.5 Experimentation with the F6 Function

Following Davis [Dav91], we used 22-bits for representing each variable. An equal number of features was used to cover the domain of each variable (*n*=*m*). The algorithm was relatively insensitive to the parameter λ and performed well for values of λ greater than 0.2. The value 0.25 for λ was used in the tests. Experiments were performed for varying *n* (i.e. number of features per variable) to determine how this parameter affects GLS. The values tried for *n* were 5, 10, 15, 20, 50, and 100. Fifty runs from random solutions (random binary strings) were performed for each value of *n* considered with the iteration limit set to 10,000 local search improvement cycles. Table 7.1 illustrates the results obtained. The best setting proved to be *n*=*m*=5. Under this setting, the algorithm succeeded in finding the exact optimal solution (0,0) in 100% of 50 runs. Under all settings, the algorithms found the exact optimum many times.

No. of features	n=m=5	n=m=10	n=m=15	n=m=20	n=m=50	n=m=100
Mean Cost	0.00E+00	4.55E-11	3.19E-10	2.73E-10	1.97E-04	3.21E-04
Best Solution	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Worst Solution	0.00E+00	2.28E-09	2.28E-09	2.28E-09	9.72E-03	9.72E-03
Mean Iterations	2287.32	2566.22	2954.08	3526.9	4132.66	3738.48
Mean Time	2.823333	3.150668	3.634334	4.382333	5.188333	4.654
Mean Funct. Eval.	104958.6	117778.8	135588	161878.4	189675.6	171578.5
Optimal Runs	50	49	43	44	31	22
Total runs	50	50	50	50	50	50

Table 7.1 GLS performance on F6 (Time in CPU seconds on a DEC Alpha 3000/600 175MHz).

This performance further improves if more time is given to the algorithm. For example, in the case (*n=m=100*) where most failures occurred (28 out of 50 runs), we performed the same experiment but this time allowed the algorithm to complete 100,000 local search iterations. The performance of GLS significantly improved and the algorithm found the exact optimum in 50 out of 50 runs (no failures).

The main observation made was that GLS performance degraded as the number of features used increased. More features meant more effort to leave a particular area but also more careful exploration. For this particular function, diversification of search to sample the whole search space proved important to find the global minimum quickly. The distribution of points visited for *n*=*m*=10 during 10,000 iterations of local search is shown in Figure 7.3. During the particular run that generated Figure 7.3, the optimal solution was found early and after 1965 iterations. Despite that, the algorithm was allowed to continue until 10,000 iterations were completed to get a better picture of the solutions visited by the algorithm. As one can see in Figure 7.3, the algorithm distributed its efforts over the whole of the search space but visited mainly local minima. That is why points in are arranged in concentric cycles around the point (0,0).

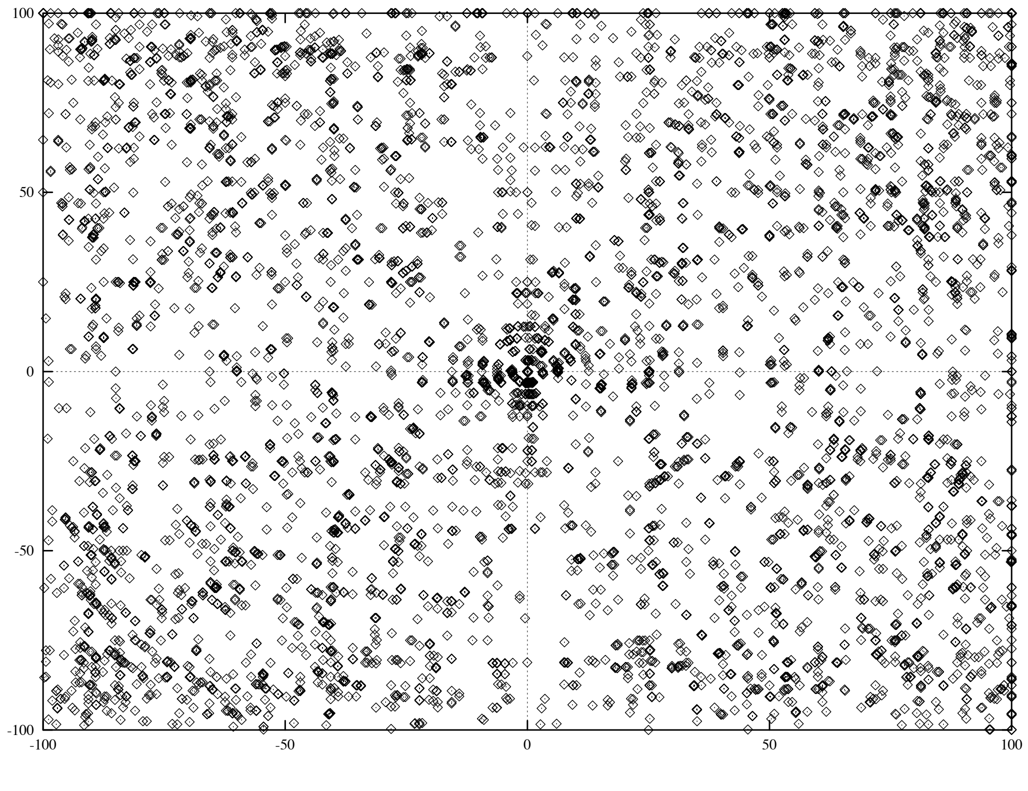


Figure 7.3 All the points visited during the first 10,000 iterations of local search

This is more clearly demonstrated in Figure 7.4 where a 3-D view of the visited points is shown. The shape formed is exactly the bottom part of F6 which suggests that the points are actually local minima in the great majority. Note here, that GLS is exploring

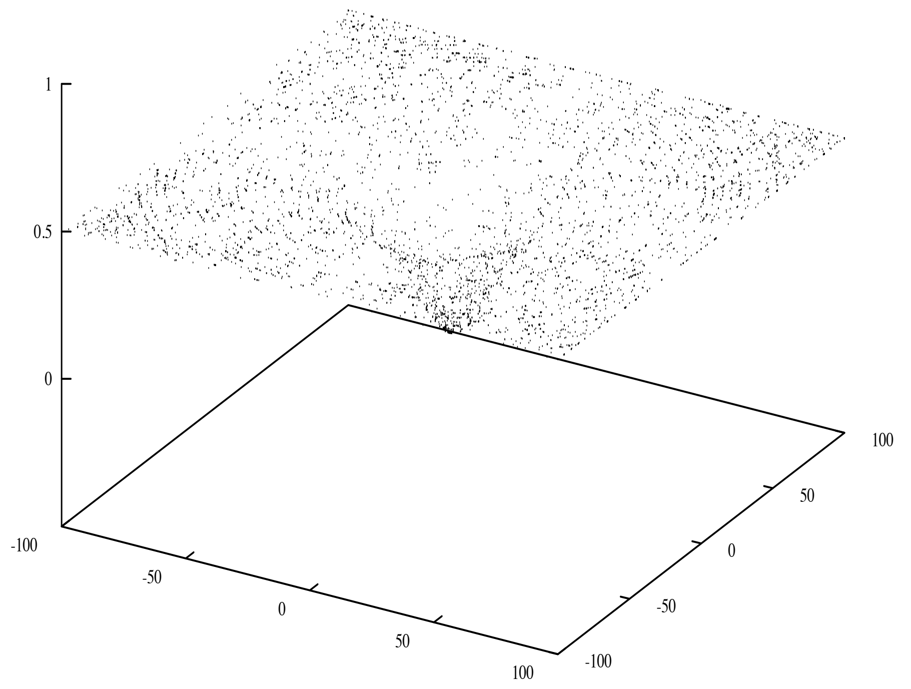


Figure 7.4 3-D View of Figure 7.3

binary space and not numeric space. In general, local minima and their attraction basins in the binary space are different from the local minima and their attraction basins appearing in the numeric space. Because of the symmetrical landscape, the binary encoding used and the structure of the GLS features, the majority of the solutions visited by GLS in the case of F6 have the property of being numeric local minima as illustrated in Figures 1.3 and 1.4. This is not necessarily the case for functions with non-symmetrical landscapes. In these cases, grey encodings (see [BT94] for example) and/or features of different structure may yield better performance than the encoding scheme and features used in this chapter.

7.6 Conclusions

In this chapter, we have shown that GLS has the potential to be utilised in the optimisation of real-valued functions with numerous local minima, which are considered to be difficult for gradient-based methods. The application of GLS to optimise the F6 function, a benchmark for Genetic Algorithms, has been examined. GLS repeatedly located the exact global optimum of the function. The chapter also serves in demonstrating how artificial solution features can be created when no features can be deduced from the structure of the objective function, which adds support to our claim that GLS has wide applications.

Chapter 8

Summary and Conclusions

This study demonstrated the effectiveness and efficiency of the GLS approach to combinatorial optimisation alone or when combined with FLS. We demonstrated that the use of information significantly improves simple local search heuristics transforming them to powerful optimisation algorithms able to compete or even outperform state of the art specialised methods. Furthermore, we demonstrated that the proposed approach is general enough to be applicable to a diversity of problem from the famous TSP and QAP to RFLAP and Workforce Scheduling and even to continuous optimisation problems. In this last chapter, we summarise the research conducted, conclude on GLS and FLS and also discuss the prospects of future research on the subject.

8.1 Summary of the Research Conducted

Guided local search is a novel approach which facilitates the engineering of intelligent search schemes which exploit problem and history information to guide a local search algorithm in a search space. Constraints on solution features are introduced and dynamically manipulated. The objectives of search intensification and diversification are unified in the single objective of distributing the search effort according to information. Various search distribution policies can be implemented. In this study, we examined the case of distributing the search effort according to feature costs either predetermined or evaluated during search.

We demonstrated the effectiveness of the proposed technique in two of the most prominent problems in combinatorial optimisation, the TSP and the QAP. Comparisons conducted with a total of fifteen methods for the TSP and four methods for the QAP showed that the GLS algorithm is better than or at least very competitive to many state of the art algorithms for the problems. Optimal or high quality solutions were consistently found in a variety of instances from the problem libraries TSPLIB and QAPLIB proving the robustness of GLS across these two landmark problems in combinatorial optimisation.

The application of the method to real world problems with various objectives and constraints was also studied in the context of the constrained optimisation problems of Radio Link Frequency Assignment and Workforce Scheduling. GLS was compared with twelve methods for the Radio Link Frequency Assignment Problem and five methods for the Workforce Scheduling problem. These comparisons clearly demonstrated the advantages of using GLS both in terms of solution quality and running times. Solutions found in the benchmark instances of RLFAP and Workforce Scheduling are amongst the best found so far for these problems. The applicability of

GLS to NonConvex optimisation problems was also demonstrated laying the foundations for the development of new methods based on GLS for this very important class of problems.

The technique of FLS was also presented and the benefits from combining it with GLS were studied in the TSP, RLFAP and Workforce Scheduling. The GLS-FLS combination leads to highly efficient variants of GLS which are many times faster than basic GLS without sacrificing solution quality.

Summarising the contents of the thesis, GLS was presented along with FLS. The method was applied to five combinatorial optimisation problems and compared with 35 algorithms including some of the best heuristic methods for these problems. Variants of almost all the general optimisation methods mentioned in the introduction were compared with GLS in at least one of the problems examined. In particular, GLS was compared with:

- Simulated Annealing on the TSP, RLFAP, and Workforce Scheduling,
- Tabu Search on the TSP, QAP, and RLFAP,
- Genetic Algorithms on the TSP, QAP, Workforce Scheduling, and RLFAP,
- Iterated Local Search on the TSP,
- Repeated Local Search on the TSP and QAP,
- Neural Networks on the RLFAP.

We believe that this is one of the most extensive studies for a newly presented combinatorial optimisation method.

8.2 Concluding Remarks on GLS and FLS

For many years, general heuristics for combinatorial optimisation problems with most prominent examples the methods of Simulated Annealing and Genetic Algorithms heavily relied on randomness to generate good approximate solutions to difficult NP-Hard problems. The introduction and acceptance of Tabu Search by the Operations Research community mainly due to the efforts of Glover, Laguna, Taillard, de Werra, Hertz, Battiti, Tecchioli and others initiated an important new era for heuristic methods where deterministic algorithms exploiting history information started appearing and being used in real world applications.

8.2.1 Guided Local Search

Guided local search proposed in this thesis follows this trend. GLS heavily exploits information (not only the search history) to distribute the search effort in the various regions of the search space. Important structural properties of solutions are captured by solution features. Solutions features are assigned costs and local search is biased to spend its efforts according to these costs. Penalties on features are utilised for that purpose.

When local search settles in a local minimum, the penalties are increased for selected features of the local minimum. By penalising features appearing in local minima, GLS avoids the local minima visited (exploiting historical information) but also diversifies choices for the various structural properties of solutions captured by the solution features. Features of high cost are penalised more times than features of low cost: the diversification process is directed and deterministic rather than undirected and random.

Feature costs contain uncertain information making sometimes speculative assumptions about the desirability of particular structural properties of solutions. Some of these properties could be essential parts of good solutions despite the high cost they may incur on the solution cost. GLS is flexible in such cases by combining search intensification with the continuous diversification process caused by the penalties on feature costs.

8.2.2 The Role of Parameter λ

The task of combining diversification with intensification is accomplished by the regularisation parameter λ which controls the influence of the information on the search process. The local gradients are directing the search process to good solutions undertaking the task of intensification. The parameter λ linearly combines the local gradients with the penalties of GLS blending the two functions of intensification and diversification. If λ is low then GLS is intensifying search slowing down the diversification process. Conversely, if λ is high then the feature costs fully determine the course of local search. For values of λ in the middle of these two extreme cases, an optimal blending of intensification and diversification is achieved. Intensification of search can also be achieved by using penalties of limited duration (see section 4.4.3) or incentives implemented as negative penalties that encourage the use of specific features rather than discourage them as with the penalties in the basic GLS. This last case of incentives has not been explored in our work and it may lead to more advanced schemes for guiding local search.

8.2.3 Fast Local Search

The selective diversification scheme of GLS where particular features are penalised and alternative solutions structures are sought that avoid these features is ideally combined with FLS which limits neighbourhood search to particular parts of the overall solution.

To allow the blending of local gradients with penalties, GLS increases the penalties for features and subsequently invokes local search to remove the penalised features from the solution. Because of λ, local gradients can affect this decision by allowing or not allowing a move to be executed which removes the penalised features. This is an essential part of the operation of GLS and enables the blending of intensification (expressed by the local gradients) and diversification (expressed by the penalties). FLS speeds up this blending allowing a quick test of the local gradients after a penalty increase. The moves which remove the penalised features are checked and if no improving move is found, control immediately returns to GLS which penalises alternative features or the same features depending on the effort already invested in these features as given by the penalties already applied to them.

In general, many penalty cycles may be required before a move is executed out of the local minimum. This should not be viewed as an undesirable situation. It is caused by the uncertainty in the information as captured by the feature costs which makes necessary the testing of the GLS decisions against the local gradients. FLS significantly reduces the computation times required to measure the local gradients in a local minimum allowing far more many penalty modification cycles to be performed by GLS for the same amount of running time.

8.3 Future Research

This thesis offers a first study of GLS and FLS. The method is still in its infancy and future research is required to further develop the method and adapt it to other problems. The use of incentives implemented as negative penalties which encourage the use of specific solution features is one promising direction to be explored. Other potentially interesting research directions include automated tuning of the regularisation parameter, definition of effective termination criteria, and different utility functions for selecting the features penalised.

GLS could also be used to distribute the search effort in other techniques such as Genetic Algorithms. In particular, GLS could be invoked at specific intervals to detect the presence of particular features in a GA population and subsequently diversify or intensify genetic search by applying penalties or incentives on particular features which are considered “bad” or “good” respectively. The GA could be guided to avoid or favour specific features spending its search efforts according to the information which again can be captured in the form of feature costs. The same utility function (Eq. 2.5) could be used by simply replacing the indicator function in Eq. 2.5 with a measure taking values in the interval [0,1] that will reflect how frequently a feature is appearing in the solutions of the population.

Finally, we found it very easy to adapt GLS and FLS to the different problems examined in this thesis something which suggests that it may be possible to built a generic software platform for combinatorial optimisation based on GLS. Although local search is problem dependent, most of the other structures of GLS and also FLS are problem independent. Furthermore, a step by step procedure is usually followed when GLS is applied to a new problem (i.e. identify features, assign costs, etc.) something which makes easier the use of the technique by non-specialists (e.g. software engineers).

References

[AA95] N. Azarmi and W. Abdul-Hameed. Workforce scheduling with constraint logic programming. *BT Technology Journal*, Vol. 13, No. 1, 81-94, 1995.

[AFPR93] C. A. Anderson, K. Fraughnaugh, M. Parker, and J. Ryan. Path assignment for call routing: An application of tabu search. *Annals of Operations Research*, 41, 301-312, 1993.

[AK89] E. H. L. Aarts and J. H. M. Korst. *Simulated Annealing and Boltzmann machines*. Wiley, Chichester, 1989.

[Bak93] S. Baker. Applying simulated annealing to the workforce management problem. ISR Technical Report, BT Laboratories, Martlesham Heath, Ipswich, 1993.

[BDST95] J. F. Boyce, C. H. D. Dimitropoulos, G. vom Scheidt, J. G. Taylor. GENET and Tabu Search for combinatorial optimisation problems. *Proceedings of World Congress on Neural Networks*, Washington D.C., INNS Press, 1995.

[Ben92] J. L. Bentley. Fast Algorithms for Geometric Travelling Salesman Problems. *ORSA Journal on Computing*, Vol. 4, 387-411, 1992.

[BFR95] C. Bessiere, E. C. Freuder, and J. Regin. Using Inference to Reduce Arc Consistency Computation. *Proceedings of IJCAI-95*, 592-598, 1995.

[BKR91] R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB - A Quadratic Assignment Library. *European Journal of Operational Research*, 55, 115-119, 1991. (Problems and updated version of paper are available by Web at http://fmatbhp1.tu-graz.ac.at/~karisch/qaplib/)

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neurolike adaptive elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13, 834-846, 1983.

[BT94] R. Battiti and G. Tecchioli. The Reactive Tabu Search. *ORSA Journal on Computing*, Vol. 6, 126-140, 1994.

[Bur84] R. E. Burkard. Quadratic Assignment Problem. *European Journal of Operational Research*, 15, 283-289, 1984.

[CK95] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza", 1995. (also available via Web at : http://www.nada.kth.se/~viggo/problemlist/compendium.html)

[CMMR96] B. Codenotti, G. Manzini, L. Margara and G. Resta. Pertrubation: An Efficient Technique for the Solution of Very Large Instances of the Euclidean TSP. *ORSA Journal on Computing*, Vol. 8, No. 2, 125-133, 1996.

[Con90] D. T. Connoly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46, 93-100, 1990.

[Cro58] A. Croes. A Method for Solving Travelling-Salesman Problems. *Operations Research*, 5, 791-812, 1958.

[CS93] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41, 327-341, 1993.

[Dav91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[Dav97] A. Davenport. GENET: a connectionist architecture for constraint satisfaction. PhD Thesis, Department of Computer Science, University of Essex, 1997.

[Dow93] K. A. Dowsland. Simulated Annealing. In Reeves, C. R. (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 20-69, 1993.

[DTWZ94] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. *Proceedings of AAAI-94*, 325-330, 1994.

[ERR94] A. E. Eiben, P-E. Raua, and Zs. Ruttkay. Solving constraint satisfaction problems using genetic algorithms. *Proceedings of 1st IEEE Conference on Evolutionary Computing*, 543-547, 1994.

[FF94] C. Fleurent and J. A. Ferland. Genetic Hybrids for the Quadratic Assignment Problem. *DIMACS Series in Mathematics and Theoritical Computer Science*, 16, 173-187, 1994.

[FM96] B. Freisleben and P. Merz. A Genetic Local Search Algorithm for Solving the Symmetric and Asymmetric TSP. *Proceedings of IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 616-621, 1996.

[Fra96] J. Frank. Weighting for Gobot: Learning Heuristics for GSAT. *Proceedings of AAAI-96*, Vol. 1, 338-343,1996.

[FW92] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58, 21-70, 1992.

[GHL94] M. Gendreau, A. Hertz, and G. Laporte. A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, Vol 40, No. 10, 1276-1290, 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[GL93] F. Glover and M. Laguna. Tabu Search. In Reeves, C. R. (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 71-141, 1993.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers Ops Res.*, 5, 533-549, 1986.

[Glo89] F. Glover. Tabu search Part I. *ORSA Journal on Computing*, Vol. 1, 190-206, 1989.

[Glo90] F. Glover. Tabu search Part II. *ORSA Journal on Computing*, Vol. 2, 4-32, 1990.

[Glo94] F. Glover. Tabu Search: improved solution alternatives for real world problems. In Birge & Murty (ed.), *Mathematical Programming: State of the Art*, 64-92, 1994.

[Glo95] F. Glover. Tabu Search Fundamentals and Uses. Graduate School of Business, University of Colorado, Boulder, 1995.

[Glo96] F. Glover. Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. To appear in: *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington eds., Kluwer Academic Publishers, 1996.

[Gol89] D. E. Goldberg. *Genetic algorithms in search, optimisation, and machine learning*. Addison-Wesley, 1989.

[GTW93] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41, 3-28, 1993.

[Hal80] W. K. Hale. Frequency Assignement: Theory and Applications. *Proceedings of the IEEE*, 68, 1497-1514, 1980.

[Han86] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. *Congress on Numerical Methods in Combinatorial Optimisation*, Capri, Italy, 1986.

[Hay94] S. Haykin. Neural Networks. Macmillan, 1994.

[Hen89] P. van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press, 1989.

[HL95] F. S. Hiller and G. Lieberman. *Introduction to Operations Research.* Sixth edition, McGraw-Hill, New York, 1995.

[Hol75] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan press, Ann Arbor, MI, 1975.

[Hoo95] J. N. Hooker. Testing Heuristics: we have it all wrong. *Journal of Heuristics*, Vol. 1, No. 1, 33-42, 1995.

[Ing89] A. L. Ingber. Very fast simulated re-annealing. Journal of Mathematical Computer Modelling, Vol. 12 No. 8, 967-973, 1989.

[JAMS89] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimisation by simulated annealing: an experimental evaluation, part I, graph partitioning. *Operations Research*, 37, 865-892, 1989.

[JAMS91] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimisation by simulated annealing: an experimental evaluation, part II, graph coloring and number partitioning. *Operations Research*, 39, 378-406, 1991.

[JBMR96] D. Johnson, J. Bentley, L. McGeoch, and E. Rothberg. Near-optimal solutions to very large traveling salesman problems. In preperation. (for experimental results from this work see also [JM95])

[JFM96] M. Jampel and E. Freuder and M. Maher (ed.). Over-Constrained Systems. *Lecture Notes in Computer Science*, 1106, Springer-Verlag, 1996.

[JKS95] Y. Jiang, H. Kautz and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat. *First International Joint Workshop on Artificial Intelligence and Operations Reasearch,* Timberline Lodge, Timberline, Oregon, USA, June, 1995.

[JM95] D. Johnson and L. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimisation. Manuscript, November, 1995 (appear in *Local Search in Optimization*, Ed. E. H. L. Aarts and J. K. Lenstra, Wiley, Chichester, forthcoming).

[Joh90] D. Johnson. Local Optimisation and the Travelling Salesman Problem. *Proceedings of the 17th Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science, 443, 446-461, Springer-Verlag, 1990.

[KC93] S. Kota and S. Chiou. Use of Orthogonal Arrays in Mechanism Synthesis. *Mechanical Machine Theory*, Vol. 28, 777-794, 1993.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation by Simulated Annealing. *Science*, Vol. 220, 671-680, 1983.

[Kno94] J. Knox. Tabu Search Performance on the Symmetric Travelling Salesman Problem. *Computers Ops Res.*, Vol. 21, No. 8, pp. 867-876, 1994.

[Koo57] B. O. Koopman. The Theory of Search, Part III. The Optimum Distribution of Searching Effort. *Operations Research*, 5, 613-626, 1957.

[LA88] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer, Dordrecht, 1988.

[Lap92] G. Laporte. The Travelling Salesman Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 231-247, 1992.

[LG93] M. Laguna and F. Glover. Bandwidth Packing: A Tabu Search Approach. *Management Science*, Vol. 39, No. 4, 492-500, 1993.

[LG93b] M. Laguna and F. Glover. Integrating Target Analysis and Tabu Search for Improved Scheduling Systems. *Expert Systems with Applications*, Vol. 6, 287-297, 1993.

[Lin65] S. Lin. Computer Solutions of the Travelling-Salesman Problem. *Bell Systems Technical Journal,* 44, 2245-2269, 1965.

[LK73] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21, 498-516, 1973.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (ed.). *The Travelling Salesman Problem: A guided tour in combinatorial optimisation*. John Wiley & Sons, 1985.

[LM86] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34, 111-124, 1986.

[LWR95] J. Lever, M. Wallace, and B. Richards. Constraint logic programming for scheduling and planning. *BT Technology Journal*, Vol.13, No.1., 73-80, 1995.

[MGK88] H. Muhlenbein, M. Gorges-Schleuter, and O. Kramer. Evolution Algorithms in combinatorial optimization. *Parallel Computing*, 7, 65-85, 1988.

[MJPL92] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58, 161-205, 1992.

[MM93] K. Mak and A. J. Morton. A modified Lin-Kernighan traveling-salesman heuristic. *Operations Research Letters*, 13, 127-132, 1993.

[MMS93] C. Muller, E. H. Magill, and D.G. Smith. Distributed genetic algorithms for resource allocation. Technical Report, Strathclyde University, Glasgow, 1993.

[MO96] O. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. In G. Laporte and I. H. Osman (eds.), *Metaheuristics in Combinatorial Optimisation*, *Annals of Operations Research*, Vol. 63, 1996.

[MOF92] O. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11, 219-224, 1992.

[Mor93] P. Morris. The breakout method for escaping from local minima. *Proceedings of AAAI-93*, 40-45, 1993.

[MRRTT53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chem. Phys.*, 21, 1087-1091, 1953.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[Osm93] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41, 421-451, 1993.

[Osm95] I. H. Osman. An Introduction to Meta-Heuristics. M. Lawrence and C. Wilson (eds.), In: *Operational Research Tutorial Papers*, Operational Research Society Press, Birmingham, UK, 92-122, 1995.

[PRW93] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The Quadratic Assignment Problem: a survey and recent developments. *DIMACS Series in Discrete Mathematics and Theoritical Computer Science*, Vol. 16, 1-42, 1993.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimisation: Algorithms and Complexity*. Prentice-Hall, 1982.

[RB93] C. R. Reeves and J. E. Beasley. Chapter 1: Introduction. In Reeves, C. (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 1-19, 1993.

[Ree93] C. R. Reeves. Genetic Algorithms. In Reeves, C. (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 151-196, 1993.

[Ree96] C. R. Reeves. Modern Heuristic Techniques. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves and G. D. Smith (ed.), John Wiley & Sons, *Modern Heuristic Search Methods*, 1-25, 1996.

[Rei91] G. Reinelt. A Travelling Salesman Problem Library. *ORSA Journal on Computing*, 3, 376-384, 1991.

[Rei94] G. Reinelt. The Travelling Salesman: Computational Solutions for TSP Applications. *Lecture Notes in Computer Science*, 840, Springer-Verlag, 1994.

[RLFAP94] The RLFAP instances are available from listserver@saturne.cert.fr. To get more information and the instances send the command "get csp-list CELAR.blurb" to the above list server. The problems are also available via ftp from the Constraints Archive maintained by Michael Jampel (ftp.cs.city.ac.uk:/pub/constraints/csp-benchmarks/celar/).

[RT95] Y. Rochat and E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1, 147-167, 1995.

[SCED89] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimisation. *Proceedings of 3rd Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, 51-60, 1989.

[Sch95] G. vom Scheidt. Extension of GENET to Partial Constraint Satisfaction Problems and Constraint Satisfaction Optimisation Problems. Master's Thesis, Dept. of Mathematics, King's College London, UK, 1995.

[SG96] P. Soriano and M. Gendreau. Diversification Strategies in Tabu Search algorithms for the Maximum Clique Problem. *Annals of Operations Research*, 63, 189-207, 1996.

[Sim89] H. U. Simon. Approximation algorithms for channel assignment in cellular radio networks. *Proc. Fundamentals of Computation Theory*, Lecture Notes in Computer Science 380, Springer-Verlag, 405-416, 1989.

[SK93] B. Selman and H. Kautz. Domain independent versions of GSAT solving large structured satisfiability problems. Proceedings of IJCAI-93, 290-295, 1993.

[Sko90] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, Vol. 2, 33-45, 1990.

[SLM92] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. *Proceedings of AAAI-92*, 440-446, 1992.

[Sol87] M. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35, 254 - 265, 1987.

[ST90] M. A. Styblinski and T. S. Tang. Experiments in Nonconvex Optimisation: Stochastic Approximation with Function Smoothing and Simulated Annealing. *Neural Networks*, Vol. 3, 467-483, 1990.

[Sto83] L. D. Stone. The Process of Search Planning: Current Approaches and Continuing Problems. *Operations Research*, 31, 207-233, 1983.

[Tai91] E. Taillard. Robust taboo search for the QAP. *Parallel Computing*, 17, 443-455, 1991.

[Tai96] E. Taillard. Comparison of Iterative Searches for the Quadratic Assignment Problem. *Location Science*, 3, 87-105, 1995.

[TAJ77] A. N. Tikhonov, V. Y. Arsenin, and F. Jonh. *Solutions of Ill-Posed Problems*. John Wiley & Sons, 1977.

[TBGGP95] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. Publication CRT-95-66, Centre de recherche sur les transports, Université de Montréal, 1995.

[THL95] S. Tiourine, C. Hurkens, and J. K. Lenstra. An overview of algorithmic approaches to frequency assignment problems. CALMA Symposium, Scheveningen, November, 1995. (available via ftp at ftp://ftp.win.tue.nl/pub/techreports/CALMA/index.html)

[Thr92] S. B. Thrun. Efficient exploration in reinforcement learning. Technical report CMU-CS-92-102, School of Computer Science, Carnegie-Mellon University, 1992.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[TV97] E. Tsang, and C. Voudouris. Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. *Operations Research Letters*, Vol. 20, No. 3, 119-127, 1997.

[VT94] C. Voudouris and E. Tsang. Tunnelling Algorithm for Partial CSPs and Combinatorial Optimisation Problems. Technical Report CSM-213, Department of Computer Science, University of Essex, Colchester, UK, 1994.

[VT95a] C. Voudouris and E. Tsang. Guided Local Search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK, August, 1995.

[VT95b] C. Voudouris and E. Tsang. Function Optimization using Guided Local Search. Technical Report CSM-249, Department of Computer Science, University of Essex, Colchester, UK, September, 1995.

[VT96] C. Voudouris and E. Tsang. Partial Constraint Satisfaction Problems and Guided Local Search. *Proceedings of 2nd Int. Conf. on Practical Application of Constraint Technology* (PACT'96), London, April, 337-356, 1996.

[WF93] R. J. Wallace and E. C. Freuder. Conjunctive Width Heuristics for Maximal Constraint Satisfaction. *Proceedings of AAAI-93*, 762-768, 1993.

[WF96] R. Wallace and E. Freuder. Heuristic Methods for Over-Constrained Constraint Satisfaction Problems. *Lecture Notes in Computer Science*, 1106, Springer-Verlag, 207-216, 1996.

[WT91] C. J. Wang and E. Tsang. Solving constraint satisfaction problems using neural-networks. *Proceedings of IEE Second International Conference on Artificial Neural Networks*, 295-299, 1991.

[WT94] T. Warwick and E. Tsang. Using a genetic algorithm to tackle the processors configuration problem. *Symposium on Applied Computing*, 217-221, 1994.

[WZ93] D. L. Woodruff and E. Zemel. Hashing vectors for tabu search. *Annals of Operations Research*, 41, 123-137, 1993.

[XCG95] J. Xu, S. Y. Chiu, and F. Glover. A Probabilistic Tabu Search Heuristic for the Telecommunications Network Design. Graduate School of Business, University of Colorado, Boulder (to appear in Journal of Combinatorial Optimization), 1995.

[XCG96] J. Xu, S. Y. Chiu, and F. Glover. Fine-Tuning a Tabu Search Algorithm with Statistical Tests. Graduate School of Business, University of Colorado, Boulder, 1996.

[XK96] J. Xu and J. Kelly. A new network flow-based tabu search heuristic for the vehicle routing problem. Graduate School of Business, University of Colorado, Boulder (to appear in Transportation Science), 1996.

[ZD95] M. Zachariasen and M. Dam. Tabu Search on the Geometric Travelling Salesman Problem. In I. H. Osman and J. P. Kelly (ed.), *Meta-heuristics: Theory and Applications*, Kluwer, Boston, 571-587, 1996.

Appendix A

Results on the Travelling Salesman Problem

The set of problems used in the evaluation of the Repeated Local Search, Guided Local Search and Iterated Local Search (using the Double Bridge move) variants on the TSP included 20 problems from 48 to 1002 cities all from TSPLIB (see Chapter 3 for details on these techniques). For each variant tested, 10 runs were performed from random solutions and 5 minutes of CPU time were allocated to each algorithm in each run on a DEC Alpha 3000/600 (175MHz) machine. To measure the success of the variants, we considered the percentage excess above the optimal solution as in Eq. 3.5. For GLS variants, the normalised lambda parameter a was provided as input and λ was determined after the first local minimum using Eq. 3.6. For GLS variants using 2-Opt, a was set to $a = 1/6$ while the GLS variants based on 3-Opt used the slightly lower value $a = 1/8$ and the LK variants the even lower value $a = 1/10$. Results for GLS are shown in Table A.1.

Iterated Local Search was using the Double Bridge move. No simulated annealing was used which is roughly equivalent to the Large-Markov Chains Methods with temperature T set to 0. Results for Iterated Local Search are shown in Table A.2.

Finally, Repeated Local Search was restarting from a random solution whenever local search was reaching a local minimum. Results for Repeated Local Search are shown in Table A.3. The names of the variants were formed according to the following convention:

<meta-heuristic>-<local search type>-<neighbourhood type>.

Problem	No.Cities	Mean Excess (%) over 10 runs					
		GLS-FI-LK	GLS-FI-3Opt	GLS-FI-2Opt	GLS-FLS-LK	GLS-FLS-3Opt	GLS-FLS-2Opt
att48	48	0	0	0	0	0	0
eil76	76	0	0	0	0	0	0
kroA100	100	0	0	0	0	0	0
bier127	127	0.218207	0.116586	0.019699	0.206625	0.002198	0
kroA150	150	0.029784	0.084075	0.000754	0.001508	0.001131	0
u159	159	0	0.460551	0.225285	0	0	0
kroA200	200	0.436189	0.526083	0.257083	0.088872	0.00681	0
gr202	202	0.732321	0.406375	0.309512	0.252988	0.011703	0
gr229	229	0.392788	0.468195	0.381644	0.152969	0.015007	0.004309
gil262	262	0.328007	0.723297	0.428932	0.084104	0.046257	0.004205
lin318	318	1.00264	1.74284	1.33884	0.583407	0.129197	0.02641
gr431	431	1.69438	2.71862	2.34071	0.563665	0.134003	0.023919
pcb442	442	0.966363	0.80783	1.36634	0.38816	0.038403	0.044311
att532	532	1.04746	2.28599	2.52871	0.386116	0.224662	0.089937
u574	574	1.36892	2.81263	3.66807	0.580951	0.278824	0.141444
rat575	575	0.806142	1.77174	2.25011	0.287908	0.171268	0.098922
gr666	666	1.66056	4.38707	6.00476	0.855251	0.497863	0.206279
u724	724	1.02505	2.25101	3.03054	0.61298	0.336674	0.168218
rat783	783	0.897116	2.24052	3.36929	0.511015	0.285033	0.161254
pr1002	1002	1.97877	3.31969	5.54336	1.04229	0.945357	0.620626
Average Excess		**0.729235**	**1.356155**	**1.653182**	**0.32994**	**0.15622**	**0.079492**

Table A.1 Results for GLS on the TSP.

Problem	No.Cities	Mean Excess (%) over 10 runs					
		DB-FI-LK	DB-FI-3Opt	DB-FI-2Opt	DB-FLS-LK	DB-FLS-3Opt	DB-FLS-2Opt
att48	48	0	0	0	0	0	0
eil76	76	0	0	0	0	0	0
kroA100	100	0	0	0	0	0	0
bier127	127	0	0	0	0	0	0
kroA150	150	0	0.001508	0.003393	0	0	0
u159	159	0	0	0	0	0	0
kroA200	200	0	0.077295	0.10113	0	0.004767	0.075252
gr202	202	0.009213	0.088396	0.457171	0	0.155129	0.257719
gr229	229	0.014116	0.157576	0.382387	0.004755	0.064115	0.124515
gil262	262	0.016821	0.20185	0.626577	0	0.075694	0.475189
lin318	318	0.255776	0.719027	1.14588	0.240786	0.279093	0.3519
gr431	431	0.332703	0.94403	2.13495	0.222386	0.394192	0.615294
pcb442	442	0.066367	0.368861	1.8961	0.081728	0.309977	0.684745
att532	532	0.225023	1.03554	2.64971	0.08163	0.270534	0.422957
u574	574	0.114348	1.20038	2.94269	0.092399	0.404823	0.553042
rat575	575	0.13731	1.15016	3.75904	0.097446	0.445888	0.649638
gr666	666	0.418878	1.25178	3.27054	0.175874	0.359528	0.816489
u724	724	0.356955	1.43617	3.94106	0.166547	0.367693	0.627535
rat783	783	0.240745	1.79764	5.00454	0.153305	0.516693	0.744947
pr1002	1002	1.04742	2.05625	5.19902	0.446332	0.872049	1.05727
Average Excess		**0.161784**	**0.624323**	**1.675709**	**0.088159**	**0.226009**	**0.372825**

Table A.2 Results for Iterated Local Search on the TSP.

Problem	No.Cities	Mean Excess (%) over 10 runs					
		REP-FI-LK	REP-FI-3Opt	REP-FI-2Opt	REP-FLS-LK	REP-FLS-3Opt	REP-FLS-2Opt
att48	48	0	0	0	0	0	0
eil76	76	0	0	1.35688	0	0	1.48699
kroA100	100	0	0.39564	0.222254	0	0.225543	0.215205
bier127	127	0.030098	0.403696	1.19629	0.027899	0.370386	1.29513
kroA150	150	0.002262	0.8317	2.00912	0.002262	0.8038	2.01553
u159	159	0	0.30038	1.62619	0	0.265447	2.05894
kroA200	200	0.024517	1.00688	3.30768	0.004767	0.922092	3.23583
gr202	202	0.141434	1.22958	3.58591	0.129731	1.19995	3.68352
gr229	229	0.097695	1.36774	3.40129	0.094427	1.27301	3.56443
gil262	262	0.054668	1.3709	5.12195	0.054668	1.2868	5.77796
lin318	318	0.629565	2.17992	4.37936	0.636703	2.022676	4.9128
gr431	431	0.679641	2.07801	5.33877	0.665232	2.20915	5.97495
pcb442	442	0.48525	1.77636	6.65012	0.516956	1.72417	7.19544
att532	532	0.530232	2.29033	6.28368	0.579354	2.29141	7.13899
u574	574	0.738382	2.91397	7.46674	0.703157	2.6934	8.4788
rat575	575	0.807618	2.69895	7.69231	0.887347	2.70781	8.61066
gr666	666	0.837619	3.18259	8.14712	0.847811	2.97203	9.94096
u724	724	0.933667	2.90551	7.76903	1.0241	2.87473	8.83202
rat783	783	1.00045	3.2864	8.46468	1.06518	3.39882	9.38792
pr1002	1002	1.5046	3.50511	8.62028	1.39138	3.59138	10.5847
Average Excess		**0.424885**	**1.686183**	**4.631983**	**0.431549**	**1.64163**	**5.219539**

Table A.3 Results for Repeated Local Search on the TSP.