Guided Local Search

Chris Voudouris and Edward Tsang
{voudcx,edward}@essex.ac.uk

Technical Report CSM-247
August 1995

Department of Computer Science,
University of Essex,
Colchester, C04 3SQ, UK

Abstract

Guided Local Search (GLS) is an intelligent search scheme for combinatorial optimization problems. A main feature of the approach is the iterative use of local search. Information is gathered from various sources and exploited to guide local search to promising parts of the search space. The application of the method to the Travelling Salesman Problem and the Quadratic Assignment Problem is examined. Results reported show that the algorithm outperforms or compares very favorably with well-known and established optimization techniques such as simulated annealing and tabu search. Given the novelty of the approach and the very encouraging results, the method could have an important contribution to the development of intelligent search techniques for combinatorial optimization.

1. Introduction

Guided Local Search is the outcome of a research project with main aim to extend the GENET neural network [29,26,5] for constraint satisfaction problems to partial constraint satisfaction [6,26] and combinatorial optimization problems. Beginning with GENET, we developed a number of intermediate algorithms such as the Tunneling Algorithm [28] to conclude with Guided Local Search (GLS) presented in this paper. In contrast to its predecessors, GLS is a general and compact optimization technique suitable for a wide range of combinatorial optimization problems.

The method takes advantage of problem and search related information to guide local search [17] 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. Whenever local search gets caught in a local minimum, the penalties are modified and local search is called again to minimize the modified cost function. Penalty modifications *regularize* the solutions generated by local search to be in accordance to prior or gathered during search information. The approach taken by GLS is similar to that of regularization methods for 'ill-posed' problems [25,10]. The idea behind regularization methods and GLS up 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 reducing so the number of candidate solutions to be considered.

Generally speaking, GLS objectives are similar to those of a wider class of combinatorial optimization algorithms known as *tabu search* [7,8,9]. In fact, GLS could be classified as a tabu search method though there are many and distinct differences with the other methods developed so far within that framework. To mention a few, GLS is a compact algorithm that can be used without any modifications for a range of problems. This contrasts with variants of tabu search which are mainly problem specific. Constraints in tabu search (also called *tabu restrictions*) refer to moves and they are usually implemented as lists (*tabu lists*). In contrast to

that, constraints in GLS refer to solutions (solution features to be precise) and take the form of penalties that augment the cost function. In tabu search, constraints are hard and can be only overridden by the complementary mechanism of *aspiration criteria*. On the other hand, GLS constraints are soft and as such can be overridden by definition. Last but not least, GLS provides the necessary mechanisms absent in other methods to capture and exploit problem and search related information

This paper is structured as follows. First, we take a quick look at local search the basis of GLS. After that, the augmentation of the cost function with penalty terms is described. Following that, the basic GLS algorithm and improvements in the basic scheme are depicted. In the applications section, we give an account of the application of the method to the Traveling Salesman and Quadratic Assignment problems. The report concludes with identifying future research directions pertinent to GLS.

2. Local Search

Local search is the basis of many heuristic methods for combinatorial optimization problems. Alone, it is a simple iterative method for finding good approximate solutions. The idea is that of trial and error. Let's consider an instance of a combinatorial optimization problem defined by the pair *(S, g)* where *S* is the set of all feasible solutions and *g* is the objective function that maps each element *s* in *S* to a real value. The goal is to find the solution *s* in *S* that minimizes the objective function *g*. The problem is stated as:

$$\min g(s),\ s \in S \tag{1-1}$$

A neighborhood *N* for the problem instance *(S, g)* is given by a mapping from S to its powerset:

$$\mathrm{N}:\ \mathrm{S} \rightarrow 2^{\mathrm{S}} \tag{1-2}$$

N(s) is called the *neighborhood* of *s* and contains all the solutions that can be reached from *s* by a single *move*. The meaning of the move here 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 neighborhood *N* iff:

$$g(x) \leq g(y), \forall y \in N(x) \tag{1-3}$$

Local search is the procedure of minimizing 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:

$$g(y) < g(x), y \in N(x) \tag{1-4}$$

In most cases, local search begins with an arbitrary solution and ends up in a local minimum. 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 neighborhood. Another example is *first improvement* local search which accepts a better solution when it is found. In the general case, the computational complexity of a local search procedure depends on the size of the neighborhood set and also the time needed to evaluate a move. The larger the neighborhood set, the longer the time needed to search it, the better the local minima.

A variety of moves and local search procedures have been used for the problems in this paper. For the purpose of describing GLS in the general case, local search is considered a general procedure of the form:

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

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

GLS makes iterative calls to such a 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 dynamically constrain solutions. This augmentation of the cost function with penalty terms is explained in the next section.

3. Solution Features and the Augmented Cost Function

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

Constraints on features are introduced or strengthened on the basis of information about the problem and also the course of the local search. Information pertaining to the problem is the cost of features. The cost of features represents 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:

$$I_i(s) = \begin{cases} 1, & \text{solution } s \text{ has property } i \\ 0, & \text{otherwise} \end{cases}, \quad s \in S \tag{1-5}$$

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:

$$h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s) \tag{1-6}$$

where M is the number of features defined over solutions, p_i is the penalty parameter corresponding to feature f_i and λ is the *regularization parameter*. The penalty parameter p_i gives the degree up to which the solution feature f_i is constrained. The regularization parameter λ represents the relative importance of penalties with respect to the solution cost and is of great significance because it provides the means to control the influence of the information on the search process.

GLS iterativelly uses local search and is simply modifying the *penalty vector* $\boldsymbol{p}$ given by:

$$\boldsymbol{p} = (p_1, \ldots, p_M) \tag{1-7}$$

each time local search settles down in a local minimum. Modifications are made on the basis of information. This will be explained in the next section describing the GLS method.

4. Guided Local Search

The role of constraints on features is to guide local search on the basis of information not incorporated in the cost function at the first place because either it was ambiguous or unknown at the time. Given a specific application, we often have some idea about what makes a bad or good solution. The cost function of the problem formulates that in a mathematical way. However, there are many cases where is not possible to include in the cost function all the available information about the problem. The information not included is mainly of uncertain nature.

For example, in the Traveling Salesman Problem, we know that long edges are undesirable though we can not exclude them from the beginning because they may be needed to connect remote clusters of cities in the problem. Besides, extra information becomes available during search. For instance, if a solution is visited then we can exclude this and possibly other solutions (e.g. with higher cost) from being searched in the future (branch and bound algorithm and other exact methods make use of that).

GLS paves the way to exploit such information. Information is converted into constraints on features which then are incorporated in the cost function using the modifiable penalty terms. Constraints confine local search to the promising solutions with respect to the information. Our method in its present stage exploits two pieces of information which are the cost of features and also the local minima visited by local search. To exploit this information, GLS provides a simple mechanism for introducing or strengthening constraints on solutions features. In particular, each time local search is trapped in a local minimum, GLS can increment the penalty parameter of one or more of the features defined over solutions. If the penalty parameter of a feature is incremented (the feature is said to be penalized) then the solutions that have this feature are avoided by local search. A first priority is to avoid the local minimum that trapped local search. Therefore, the feature or features penalized should be between those exhibited by the local minimum. Furthermore, features of high cost (bad features) should take precedence over low cost features (good features) in being penalized. Let's examine the scheme in detail.

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 again uses local search starting from the previously found local minimum. The modification action consists in incrementing by one the penalty parameter of one or more of the local minimum features. Information is gradually inserted in the augmented cost function by selecting which penalty parameters to increment. An overview of the approach is given in figure 1.

Sources of information are the cost of features and the local minimum itself. Let's consider 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* $\boldsymbol{c}$:

$$\boldsymbol{c} = (c_1, \ldots, c_M) \tag{1-8}$$

which contains positive or zero elements.

A particular local minimum solution exhibits a number of features. If the feature f_i is exhibited by a local minimum solution s_* the following holds:

$$I_i(s_*) = 1 \tag{1-9}$$

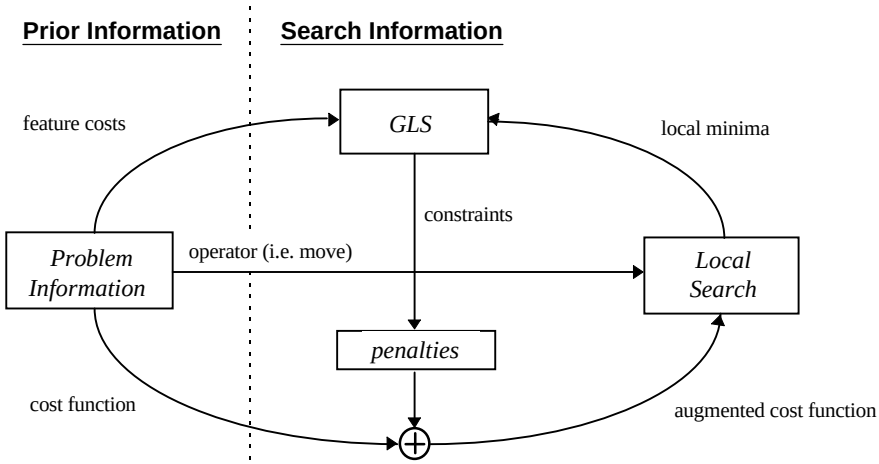

Figure 1. A schematic view of the GLS approach to combinatorial optimization problems.

Thus, the vector ***b*** of the indicator function values in the local minimum s_*:

$$b(s_*) = \left(I_1(s_*), \ldots, I_M(s_*)\right) \tag{1-10}$$

contains all the feature related information about the local minimum (i.e. gives the features of the local minimum).

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

$$util(s_*, f_i) = I_i(s_*) \cdot \frac{c_i}{1 + p_i} \tag{1-11}$$

In other words, incrementing the penalty parameter of the feature f_i is considered an *action* with utility given by (1-11). In a local minimum, the actions with *maximum* utility are firstly selected and then performed.

The utility function (1-11) makes full use of the information carrying vectors ***b****(s*$_*$*)* and ***c***. The penalty parameter p_i is incorporated in (1-11) to prevent the scheme from being totally biased towards penalising features of high cost. The role of the penalty parameter in (1-11) is that of a counter which counts how many times a feature has been penalised. If a feature is penalized many times over a number of iterations then the term $\frac{c_i}{1 + p_i}$ in (1-11) decreases for the feature diversifying choices and giving the chance to others features to be also penalized. The policy implemented is that features are penalized with a frequency proportional to their cost. Due to (1-11), features of high costs are penalized more frequently than those of low costs and the opposite. The search effort is distributed according to *promise* as it is expressed by the feature costs and the already visited local minima.

Incremental distribution of the search effort according to information though in a probabilistic framework can be found in a class of methods deriving themselves from the *optimal search theory* of Koopman [13,22]. Also, counter based schemes for search diversification like that of GLS are used under the name *counter-based exploration* in reinforcement learning [24].

Something that has been left out from the so far analysis is the regularization parameter λ in the augmented cost function (1-6). This parameter determines the degree up to which constraints on features are going to affect local search. Let's examine how the regularization

```
procedure GLS(S, g, λ, [I_1, ...,I_N], [c_1,...,c_N], M)
begin
        k ← 0;
        s_0 ← arbitrary solution in S;
        for i ←1 until M do
                p_i ← 0;
        h ← g + λ * ∑p_i*I_i;
        while StoppingCriterion do
        begin
                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
```

Figure 2. The GLS algorithm in pseudocode.

parameter is going to affect the moves performed by a local search method. A move alters the solution, adding new features, 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:

$$\Delta h = \Delta g + \lambda \cdot \sum_{i=1}^{M} p_i \Delta I_i \qquad (1\text{-}12)$$

As we can see in (1-12), if λ is large then the selected moves will solely remove the penalized 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 to improve the solution (taking into account the gradient) and also remove the penalized features (taking into account the information). Since the difference Δg is problem dependent, the regularization parameter is also problem dependent. Some ways of tuning this parameter will be discussed in the applications section.

The GLS in pseudocode as it has been described so far is given in figure 2. In this figure, the `StoppingCriterion` can be defined by a limit on the number of iterations, a time budget, an upper bound on cost or combinations of the above.

5. GLS Improvements and Fast Local Search

There are both minor and major alterations that improve the method. For example, instead of calculating the utilities (1-11) for all the features, we can restrict ourselves to the local minimum features because for not local minimum features the utility function (1-11) takes the value 0. Also, the evaluation mechanism for moves should be changed. Usually, this mechanism is not evaluating directly 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 this is shown in (1-12). This can be done easily and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only the features that change state (being removed or added). The penalty

parameters of the features removed 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 difference:

$$- \sum_{\text{over all features j added}} p_j + \sum_{\text{over all features k removed}} p_k \qquad (1\text{-}13)$$

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 searches to find the best solution in the whole neighborhood. This is very time consuming especially if we are dealing with large instances of problems. We present here *fast local search* (FLS) which drastically speeds up the neighborhood search process. The method is a generalization of the *approximate 2-opt* method proposed by Bentley [2] for the TSP.

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

Initially, all sub-neighborhoods are active. If a sub-neighborhood 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-neighborhoods are possibly activated. In particular, we activate all the sub-neighborhoods 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-neighborhoods being active until all the sub-neighborhood bits turn to 0. The solution formed up to that point is returned as a local minimum.

The overall procedure is 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 fading down examining fewer moves saving enormous amounts of time otherwise spent on examining predominantly bad moves. A concrete example of fast local search is given in the continue.

Let's consider the very common case where solutions are given by permutations. A '2-opt' type move can be used for the problem which exchanges (i.e. swaps) the contents of two positions in the permutation. The sub-neighborhoods in this case are defined by the permutation positions. Each sub-neighborhood corresponds to a permutation position and contains all the possible moves that exchange the contents of this position with any other position in the permutation. In total, there are as many sub-neighborhoods as the size of the permutation (dimension of the problem).

Initially all sub-neighborhoods are active. The permutation is continuously scanned from the first to last position for active sub-neighborhoods. Each time an active sub-neighborhood is found, it is examined for improving moves. If there is no improving move then the sub-neighborhood becomes inactive. Otherwise, the improving move, in our case a swap, is performed and the bits of the permutation positions involved in the swap are set to 1. This causes the currently examined sub-neighborhood to remain active and also the sub-neighborhood at the other end of the swap to become also active if it has been previously inactivated. The scanning always continues with the next position in the permutation. The process finally ends when all the bits have turned to 0 and the solution formed is returned as a local minimum.

Although, fast local search procedures like the above do not generally result to very good solutions when combined with GLS become very powerful optimization tools. Combining GLS with fast local search is straightforward. The key idea is to associate solution features to sub-neighborhoods. The associations to be made are such that for each feature we know which sub-neighborhoods contain moves that have an immediate effect upon the state of the feature (i.e. moves that remove the feature from the solution).

In the beginning of GLS, all the bits of fast local search are set to 1 and fast local search is left to reach the first local minimum (i.e. all bits 0). In the continue and whenever a feature is penalized, the bits of the associated sub-neighborhoods and only these are set to 1. In this way, after the first local minimum, fast local search calls examine only a number of sub-neighborhoods and in particular those which associate to the features just penalized. This dramatically speeds up GLS. Moreover, local search is focusing on removing the penalized features from the solution instead of considering all possible modifications.

6. Applications

In the applications section, we examine how the method of GLS has been used to tackle two well-known NP-hard problems, the Traveling Salesman Problem [14] and the Quadratic Assignment Problem [3]. All experiments reported in this paper performed on a DEC Alpha 3000/600. Unless otherwise stated, the algorithms were implemented in C++.

6.1 Traveling Salesman Problem(TSP)

6.1.1 The Problem

There are many variations to the Traveling Salesman Problem (TSP). In this report, we consider the classic Symmetric TSP. The problem is defined by N cities and a symmetric 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 it 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:

$$g(\pi) = \sum_{i=1}^{N} d_{i\pi(i)} \tag{1-14}$$

and gives the cost function of the TSP [17].

An up to date and comprehensive survey of TSP methods is that by Reinelt [20]. The reader may also refer to [14] 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 [2]. Moreover, the optimal solutions have been found and proved for problems of size more than 4000 cities [20]. Nowadays, TSP plays a very important role in the development and testing of new optimization methods. In this context, we examine how guided local search has been applied to the problem.

6.1.2 Local Search

Local search for the TSP is synonymous to the k-opt moves [15]. The simplest and less expensive of the k-opt moves is the famous 2-opt. 2-opt removes two edges from the tour to replace them with two new edges not previously included in the tour. Particular care is being taken such that the tour remains closed after the exchange. Greedy local search usually starts from a random tour. In every iteration, all possible edge exchanges are evaluated and the best is

selected and performed. Local search terminates when no 2-opt move exists that improves the tour further. The local minimum tour returned is called a *2-optimal* tour.

6.1.3 Guided Local Search

A first step in the process of applying GLS to a problem is to find a set of solution features that are responsible 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. 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 given by their length. A set of features can be defined by considering all possible edges that may appear in a tour with feature costs given by edge lengths. Once features and their costs have been defined, GLS as described in section 4 can be applied to the problem without any modifications. A fast local search procedure already exists for the TSP and it is the *approximate 2-opt* suggested by Bentley [2]. In the continue, we give a brief account of approximate 2-opt.

Each city in the instance defines a sub-neighborhood which contains all moves that exchange one of the two edges adjacent to the city. In total, there are as many sub-neighborhoods as the cities in the instance. The scanning of the cities is conducted in tour order. Each time an active sub-neighborhood is found, it is searched for improving moves. The first improving move found is performed and the algorithm then backtracks one city in the tour to continue traversing the cities in tour order. Whenever a move is performed, the cities (corresponding sub-neighborhoods) at the ends of the edges involved in the exchange are activated. The process ends when a full rotation of the tour is completed with no active sub-neighborhood being found. Interfacing GLS with the above fast local search procedure is simple. Each time we penalize an edge, the sub-neighborhoods corresponding to the cities at the ends of that edge are activated.

6.1.4 Results on Small to Medium Size TSPs

In the continue, we report results for GLS on known TSP instances from TSPLIB [19]. TSPLIB is a publicly available library of TSP problems. Most of the instances included in TSPLIB have already been solved to optimality and they have been used in many works in the past.

For each TSPLIB instance used in our experiments, ten runs performed from different random solutions and the various performance measures (solution quality, running time etc.) were averaged. The solution quality was measured by the percentage excess over the optimal solution as it is given by the formula:

$$\frac{\text{solution cost - optimal cost}}{\text{optimal cost}} \times 100 \qquad (1\text{-}14)$$

The only parameter of GLS that required tuning was the regularization parameter λ. For the TSP instances examined, the algorithm performed well for a range of values. This range is roughly given by the following parametric equation:

$$\lambda = a \cdot \frac{g(2\text{-optimal tour})}{N}, 0 \le a < 1 \qquad (1\text{-}15)$$

where *g(2-optimal tour)* is the cost of a typical 2-optimal tour and *N* the number of cities in the instance. Best performance was recorded for values of *a* around 0.3. For the results reported here, the exact value of λ used in the final 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 be) or very quickly finding no more than a handful of good local minima (λ is higher than it should be).

The set of problems used in the first part of 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 greedy local search was considered one local search iteration and for GLS with fast local search, a call to fast local search (i.e. as in fig. 2). The iteration limit for both algorithms was set to 200K iterations. In both cases, we tried to provide the algorithms with plenty of resources in order to reach the maximum of their performance. The results obtained on this first set are presented in Table 1.

Problem Name	GLS with greedy local search			GLS with fast local search		
	optimal runs out of 10	mean per. excess (%)	mean CPU time (sec)	optimal runs out of 10	mean per. 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 1. GLS variants on small to medium size TSP instances.

Both the GLS variants found solutions with cost equal to the optimal cost in the majority of the runs. GLS with greedy local search failed to find the optimal solutions (as reported by Reinelt [19][20]) 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 the single instance namely *d198* though the solutions found for *d198* were of high quality and on average within 99.92% of optimality.

GLS with fast local search found the optimal solutions in 3 more runs than GLS with greedy local search 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 no optimal solution found 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 greedy local search 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 in less than 10 seconds of CPU time on the DEC Alpha 3000/600 machines used for experiments.

6.1.5 Other General Methods for the TSP

The above performance of GLS is remarkable considering that GLS is not an exact method and that it only uses the short-sighted 2-opt heuristic. Searching the relating to the TSP literature, we could not find any other approximation method that uses only the simple 2-opt move and consistently finds optimal solutions for problems up to 318 cities. Only the iterated Lin-Kernighan algorithm [11] shares the same consistency in reaching the optimal solutions [20] though it uses k-opt moves of higher order.

A meaningful comparison that can be made is between GLS and other general methods that also use the 2-opt move. For that reason, we implemented simulated annealing and a tabu search variant for the TSP suggested by Glover [7].

6.1.5.1 Simulated Annealing

The Simulated Annealing (SA) algorithm implemented for the TSP is the one described by Johnson [11] and uses geometric cooling schedules. 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 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 [12].

6.1.5.2 Tabu Search

The tabu search variant implemented is using a combination of *tabu restrictions* and *aspiration level criteria* that bias the algorithm towards favoring short edges and avoiding long edges. The tabu search variant is briefly described here. For more information on this particular tabu search variant the reader is referred to [7].

Tabu search performs greedy local search selecting the best move in the neighborhood but only amongst those not characterized as *tabu*. Determining the tabu status of a move is very important in tabu search and holds the key to the development of efficient search schemes. To define tabu status for 2-opt moves, the scheme implemented uses an *aspiration* function *A(e)* which for each edge *e* gives the *aspiration level* for that edge. Four edges are involved in a 2-opt exchange. To characterize a move as tabu (i.e. can not be performed), the cost of the resulting tour after the exchange is evaluated and compared to the aspiration levels of the edges in the exchange. If the cost of the resulting tour is greater than the aspiration level of at least

one of the edges involved then the move is characterized as tabu (other similar criteria are also possible).

The manipulation of edge aspiration levels is a crucial part of the process. The mechanism aims at excluding moves which lead to previously visited configurations and also biasing search to avoid long edges while favoring short edges. These objectives are similar to those set out above for GLS. Let's see how they are accomplished in the tabu search variant.

Initially, all aspiration levels for edges are set to a large ("infinite") value. Whenever a 2-opt exchange is performed the aspiration levels of the edges in the exchange are set to $A(e)=min(A(e),g(s),g(s'))$ where $g(s)$ is the tour length before the exchange and $g(s')$ the tour length after the exchange. The result of reducing the edge aspiration is that moves which exchange these edges become tabu. To bias search, short edges removed and long edges added have their aspiration levels reset (to the "infinite" value) quicker than short edges added and long edges removed. Four tabu lists are set up one for each edge category (short-removed, long-removed, short-added, long-added). Short edges removed and long edges added are inserted to a short tenure tabu list while long edges removed and short edges added are inserted to a longer tenure tabu list. In this way, the algorithm is biased to favour short edges and avoid long edges by simply differentiating on the reset periods for edge aspiration levels. For the experiments reported here, the tabu list size was set to $N/2$ for the short tenure tabu lists (short-removed and long-added edges) and to $3/4N$ for the long tenure tabu lists (short-added and long-removed edges) where N the number of cities in the instance. Tabu search was left to run for 200K iterations which is equivalent in terms of number of moves evaluated to the number of iterations GLS with greedy local search was given on the same instances.

6.1.5.3 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 2 illustrates the results for simulated annealing and tabu search compared with those from GLS with fast local search on the same instances. Results are also contrasted with the best solution found by repeating 2-opt until a total of 200K local search iterations were completed.

Problem Name	GLS - fast local search		Simulated Annealing		Tabu Search		Repeated 2-opt (200K iterations)	
	mean per. excess (%)	mean CPU time (sec)	mean per. excess (%)	mean CPU time (sec)	mean per. excess (%)	mean CPU time (sec)	per. excess (%)	CPU time (sec)
eil51	0.0	0.46	0.73	6.34	0.00	21.70	0.23	42.4
eil76	0.0	0.97	1.21	18.0	0.00	31.84	1.85	153.45
eil101	0.0	2.37	1.76	33.29	0.06	80.02	3.97	319.15
kroA100	0.0	1.25	0.42	37.36	0.50	103.24	0.34	706.35
kroC100	0.0	0.74	0.80	36.58	1.22	138.38	0.33	1301.98
kroA150	0.0	7.03	1.86	103.32	3.45	466.61	1.41	3290.95
kroA200	0.0	50.16	1.04	229.38	6.01	312.53	1.7	731.1
lin318	0.005	346.44	1.34	829.46	6.05	978.85	3.11	9771.28

Table 2. GLS, Simulated Annealing, Tabu Search on TSP instances.

The superiority of GLS is evident. The tabu search variant although found easily the optimal solutions for small problems, it scaled badly for problems of size greater than 100 cities. Simulated annealing had a more consistent behaviour finding good solutions for all problems in the range but failed to reach the optimal solutions in all but 3 runs.

6.1.6 Results on Large TSP Instances

Encouraged by the remarkable performance of GLS on small to medium size problems, we tested GLS with fast local search on a second set of larger instances from 532 to 2392 cities again from TSPLIB for which the optimal solutions are also known. For problems of that size what it is usually required is high quality solutions in reasonable time. We considered as a satisfactory solution to be one within 99% of optimality (percentage excess ≤ 1%) and left GLS with fast local search to run on the instances until that solution quality was reached. Five runs performed on each of the five problems from random tours. Remarkably, GLS managed to reach the desired solution quality in all 25 runs. The running times required for that varied from around a minute for the smallest of the problems to less than an hour for the largest. These running times are presented in Table 3.

Problem Name	Mean CPU time (min:sec)
att532	1:15
gr666	1:53
rat783	2:47
u1432	7:56
pr2392	52:21

Table 3. GLS running times to reach a solution within 99% of optimality.

One of the problems in this second set is the famous 532-city problem (*att532*) due to Padberg and Rinaldi [16] which is considered a very difficult problem for both exact and approximate methods [11]. A further 5 longer runs (400.000 iterations) were performed on this particular instance using GLS with fast local search. The result was that GLS found the optimal solution in one out of the five runs. The time needed for that was 53 minutes. The average solution quality on *att532* for the long runs was at 0.03%.

6.1.7 Evaluation of GLS Performance on the TSP

The results reported thus far for GLS provide strong evidence that GLS is able to find high quality solutions in short time even for large TSP instances. GLS was compared and proved superior to simulated annealing and a tabu search variant. Despite the fact that our technique uses the simple 2-opt, solutions with cost equal to optimal found for problems of size up to half thousand cities including the famous and difficult 532-city problem of Padberg and Rinaldi. Fast local search effectively speeded up the algorithm without any impact on solution quality. GLS with fast local search was capable to find solutions with cost equal to optimal in a few CPU seconds for many of the small to medium size instances while high quality solutions within 99% of optimality were reached in reasonable time for problems of size up to 2392 cities. Because of its simplicity and effectiveness, GLS could be considered as an ideal method for the TSP especially when no programming effort can be devoted in implementing one of the complex specialized TSP algorithms.

6.2 Quadratic Assignment Problem(QAP)

6.2.1 The Problem

Quadratic Assignment Problem (QAP) is one of the most difficult problems in combinatorial optimization. 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 [18]. In the following, we describe the QAP in its simple 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:

$$\min_{p \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \cdot B_{p(i)p(j)} \tag{1-16}$$

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. 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. Both matrices are considered symmetric.

6.2.2 Local Search

QAP solutions are represented by permutations. A move commonly used for the problem is simply to exchange (i.e. swap) the contents of two permutation positions. A best improvement local search procedure starts with a random permutation. In every iteration, all swaps are evaluated and the best is selected and performed. The algorithm reaches a local minimum when there is no swap which improves further the cost of the current permutation. A fast local search method for the QAP has been already described in section 5 and it is the general one for permutation-based representations.

Evaluating a swap can be made in constant time for best improvement local search (see [1]). Unfortunately, the scheme requires all possible swaps to be evaluated before a move is performed and therefore can not be used in the case of fast local search which for the QAP requires $O(n)$ operations to evaluate a move. As a result of that, the benefits from fast local search are limited in this problem.

6.2.3 Guided Local Search

Applying GLS to the QAP is again a simple process of identifying the solution features to be used and assigning costs to them. The features used for the QAP were all the possible location-facility pairs (other features are also possible). This kind of feature is general and can be used for 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).

After deciding on the features, the next step is to assign costs to them. Assignments 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 particular assignments have on the solution cost. To deal with this problem, we used variable feature costs where the cost of a feature was evaluated in the context of the solution it appeared in. In particular, feature costs were evaluated only for the features of the local minimum and their cost was given by the expression:

$$c(i, p(i)) = \sum_{j=1}^{n} A_{ij} \cdot B_{p(i)p(j)} \tag{1-17}$$

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 good from facility $p(i)$ to the other facilities with facility $p(i)$ placed in location i. Features that maximized the utility expression (1-11) were penalized and the corresponding location-facility combinations were avoided. A modification of the GLS algorithm that appeared to improve performance in this case was to reset the penalty parameters after a relatively long period of time. In particular, the penalty parameters were reset (set to 0) every 5000 iterations.

6.2.4 Results

Before reporting the results, we first examine the scheme for determining the regularization parameter of GLS for the QAP. We conducted a large number of test runs on problems from the publicly available library of QAP instances, QAPLIB [4]. A relation similar to (1-11) 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:

$$\lambda = a \cdot \frac{g(\text{local minimum})}{n^2}, 0 < a \leq 1 \qquad (1\text{-}18)$$

where *g(local minimum)* is the cost of the first local minimum found during a run and *n* the size of the problem. A value of *a* equal to 0.5 gave very good results for all instances tested. The final program incorporated these findings and the regularization parameter was automatically calculated after the first local minimum by (1-18) with *a=0.5*.

Along with GLS, we implemented Taillard's *robust taboo search* [23][9] a very effective method for the QAP and also obtained the original code in C for reactive tabu search another tabu search variant successfully applied to the problem [1]. Repeated local search was also implemented. This last algorithm was simply restarting local search after a local minimum. The methods were tested on 12 QAP of sizes from 15 to 50 locations all from QAPLIB. For each algorithm, ten runs performed on each instance starting from random solutions. The algorithms left to run for 100K iterations[1] or until a solution with cost equal or less than the best known solution[2] was found. A run was characterized as successful if it resulted to the best known solution. Table 4 illustrates the results obtained.

Problem Name	best known solution [4]	GLS		Robust TS		Reactive TS		Repeated 2-opt	
		mean cost	successful runs	mean cost	successful runs	mean cost	successful runs	mean cost	successful runs
nug15	1150	1150	10	1150	10	1150	10	1150	10
nug20	2570	2570	10	2570	10	2583	2	2570	10
rou20	725522	725540	7	725522	10	725522	10	725919	4
nug30	6124	6124	10	6124	10	6151	1	6143	2
tho30	149936	149936	10	149936	10	149936	10	150469	1
kra30a	88900	88900	10	90089	4	89019	9	89759	3
kra30b	91420	91441	7	91452	8	91456	7	91569	0
ste36a	9526	9530.4	7	9547.8	7	9630.2	0	9635.4	0
ste36b	15852	16185.6	4	16291.8	5	15856	9	15943	3
tho40	240516	240751.6	0	240588.8	0	240572.6	3	242557.4	0
sko42	15812	15816	6	15816.2	5	15899.4	0	15943	0
wil50	48816	48843.4	0	48827.2	0	48934.6	0	48958.8	0
Total successes		81 out of 120 runs		79 out of 120 runs		61 out of 120 runs		33 out of 120 runs	

Table 4. GLS, Robust TS, Reactive TS, and Repeated 2-opt on small to medium size QAP instances.

GLS and robust taboo search performed equally well finding the best known solution in around 66% of the runs. Reactive tabu search came third at 50% with repeated 2-opt fourth at 25%. In fact, GLS found the best known solution in two more runs than any other method with

[1] Note here, that the three algorithms needed around the same time to complete an iteration. The dominant computation was the evaluation of the $N(n^2)$ moves to search the neighborhood. This computation was conducted in almost exactly the same way for all four methods.

[2] 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 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.

close second the robust taboo search. For five of the problems, both GLS and robust taboo found the best known solution in ten out of ten runs something very encouraging considering that both are approximation algorithms. Overall, the results proved that GLS can perform equally well to robust tabu which admittedly generates very good solutions for the QAP. Reactive tabu search outperformed repeated 2-opt and overall had a very good performance though found fewer best solutions than GLS or robust taboo search did.

A second set of runs was performed on large QAP instances. Thirteen instances from QAPLIB were used with sizes from 42 to 100 which have been randomly generated (see [21] for more). A limited amount of time equal to 5 minutes on a DEC Alpha 3000/600 was given to each algorithm. Ten runs performed on each instance from random initial solutions. The objective of this second experiment was to determine which of the algorithms will return the best solution given a limited amount of time. The best solutions found by the methods are reported in Table 5.

Problem	best known solution [4]	GLS	Robust TS	Reactive TS	best solution found by:
sko42	15812	15812	15812	15834	GLS, Robust TS
sko49	23386	23386	23386	23486	GLS, Robust TS
sko56	34458	34462	34462	34594	GLS, Robust TS
sko64	48498	48500	48498	48572	Robust TS
sko72	66256	66280	66298	66562	GLS
sko81	90998	91086	91008	91240	Robust TS
sko90	115534	115632	115624	116002	Robust TS
sko100a	152002	152144	152138	152520	Robust TS
sko100b	153890	153978	154038	154296	GLS
sko100c	147862	147992	147914	148300	Robust TS
sko100d	149576	149758	149762	150058	GLS
sko100e	149150	149282	149500	149726	GLS
sko100f	149036	149214	149258	149526	GLS

Table 5. GLS, Robust TS, and Reactive TS on large QAP instances.

The last column in this table gives the algorithm that found the best solution for each problem over all algorithms and runs. GLS and robust taboo search had exactly the same number of best solutions (i.e. 8 out of 13). Reactive tabu search did not find a best solution though the solutions found by the method were very close to the best found either by robust taboo search or GLS but in none of the cases were better. GLS and robust taboo search solutions were very close and sometimes equal (for *sko42* and *sko49*) to the best known solutions.

6.2.5 Evaluation of GLS Performance on the QAP

The results presented on QAP instances provide evidence that GLS is an effective method for the QAP. The algorithm matched the performance of robust taboo search which is already known to perform very well on QAPs while outperformed reactive tabu search a successful tabu search variant and also repeated 2-opt. The algorithm consistently found the best known solutions for small to medium size problems. Furthermore, the solutions found in large problems where very close to the best known. Given the alternative settings that can be pursued in the QAP (different features or other more effective expressions for the feature costs), the performance of GLS could be further improved.

7. Conclusions and Future Research Directions

Guided local search is a novel approach that enables intelligent search schemes to be built that exploit information to guide a local search algorithm in a search space. Constraints on solution

features are introduced and dynamically manipulated to distribute the search effort over the regions of a search space. Various search distribution policies can be implemented. In this report, 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 method in two of the most prominent problems in combinatorial optimization, the TSP and the QAP. Comparisons conducted with a total of six other methods over many QAP and TSP instances. Results showed that the GLS algorithm is from many times better to at least very competitive to these other methods. Optimal or high quality solutions consistently found in a variety of problems from the problem libraries TSPLIB and QAPLIB proving the robustness of GLS across combinatorial optimization problems, problem instances and instance sizes. The reader may also refer to [27] where further evidence is provided on the effectiveness of both GLS and fast local search this time in the context of a real-world and difficult scheduling problem.

Future research on GLS will investigate the possiblity of learning feature costs and/or the search policy. Alternative feature-cost settings will also be investigated for existing applications of the method. The algorithm has only one parameter and from this proint of view, tuning is a relatively easy task. Nevertheless, domain indepentant self-tuning mechanishms is worthwhile to investigate.

Acknowledgements

We would like to thank the Computing Service of the University of Essex for givinig us access to the DEC Alpha machines to carry out the experiments reported in this work. This project and Chris Voudouris are funded by the EPSRC grant (GR/H75275).

References

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

[2] J. J. Bentley, “Fast Algorithms for Geometric Traveling Salesman Problems”, *ORSA Journal on Computing*, Vol. 4, 387-411 (1992).

[3] R. E. Burkard, “Quadratic Assignment Problem”, *European Journal of Operations Research* 15, 283-289 (1984).

[4] R. E. Burkard, S. E. Karisch, and F. Rendl, “QAPLIB - A Quadratic Assignment Library”, Technical Report 299, Graz University of Technology, Austria (1994). (Problems and report available by ftp at ftp.tu-graz.ac.at:/pubs/papers/qaplib)

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

[6] E. C. Freuder and R. J. Wallace, “Partial constraint satisfaction”, *Artificial Intelligence* 58, 21-70 (1992).

[7] F. Glover, “Tabu search Part I”, *ORSA Journal on Computing*, Vol. 1, 190-206 (1989).

[8] F. Glover, “Tabu search Part II”, *ORSA Journal on Computing*, Vol. 2, 4-32 (1990).

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

[10] S. Haykin, *Neural Networks*, Macmillan, 1994.

[11] D. Johnson, “Local Optimization and the Traveling Salesman Problem”, *In Proceedings of the 17th Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science , 443, 446-461, Springer-Verlag (1990).

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

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

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

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

[16] M. W. Padberg and G. Rinaldi, "Optimization of a 532 City Symetric Traveling Salesman Problem by Branch and Cut", *Operations Research Letters* 6, 1-7 (1987).

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

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

[19] G. Reinelt, "A Traveling Salesman Problem Library", *ORSA Journal on Computing*, Vol. 3, 376-384 (1991).

[20] G. Reinelt, "The Traveling Salesman: Computational Solutions for TSP Applications", *Lecture Notes in Computer Science* 840, Springer-Verlag (1994).

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

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

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

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

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

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

[27] E. Tsang and C. Voudouris, "Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem", Technical Report CSM-426, Department of Computer Science, University of Essex (1995).

[28] C. Voudouris and E. Tsang, "Tunnelling Algorithm for Partial CSPs and Combinatorial Optimization Problems", Technical Report CSM-213, Department of Computer Science, University of Essex (1994).

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