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.

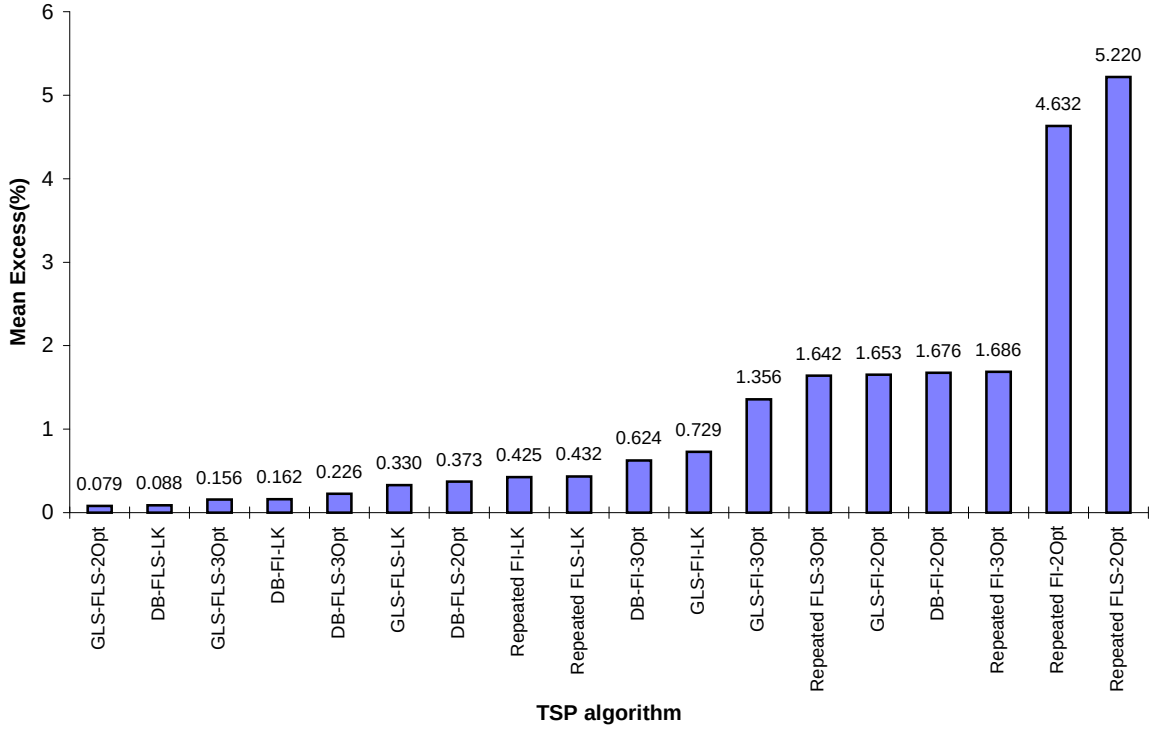

Figure 3.9 Overall ranking of the algorithms in terms of solution quality when tested on a set of 20 TSPLIB problems

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

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

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 $\rightarrow$ satisfied or satisfied $\rightarrow$ 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