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

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

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

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

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

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

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

5.8 Comparison with the CALMA Project Algorithms

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

As it can be seen in Table 5.7, GLS achieves a very good performance compared with the other algorithms and taking into account the values of the best known solutions. In summary, it applies to all problems finding solutions of high quality while it is many times faster than the other algorithms. Algorithms which produce marginally better solutions than GLS (e.g. Genetic Algorithms-LU) were applied to only a subset of the problems and require substantially more time, fine tuning and probably implementation effort. On the other hand, although algorithms such as SA-EUT, extended GENET-KCL and Variable Depth Search-EUT, are applied to most problems and find solutions of good quality, they are between 5 to 100 times slower than GLS (especially on the insoluble instances). This cannot be attributed just to the different machines used in experiments. Besides, although the GA by UEA produces good results for Scenarios 6 and 11, it performs badly in Scenarios 7 and 8; compared to it, GLS is not only much faster, but also more consistent in its performance. Bessiere et al. [BFR95] also applied arc-consistency algorithms to Scenarios 3, 5, 8 and 11. Since only the satisfiability issue (not optimisation) was addressed their results are not comparable with the rest in this section. To conclude, GLS is a highly competitive, if not the best, method amongst the algorithms developed so far for the

problem that are known to us. It is the fastest algorithm which consistently provides quality solutions (never much worse than the best found so far, sometimes the best). Significantly, this is achieved almost without any tuning required.

5.9 Discussion

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

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

5.10 Conclusions

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

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

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

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

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

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

Chapter 6

Workforce Scheduling

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

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

6.1 BT's Workforce Scheduling Problem

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

(Loc, Dur, Type)

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

Each engineer is described by a 5-tuple:

(Base, ST, ET, OT_limit, Skill)

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

serve a job with a standard duration (*Dur*) of 20 then this engineer would actually take 18 minutes to finish the job.

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

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

where:

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

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

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

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

6.2 Local Search for Workforce Scheduling

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

procedure **Evaluation** (input: one particular permutation of jobs)

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

Figure 6.1 Algorithm for mapping job permutations into complete schedules

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

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

6.3 Fast Local Search for Workforce Scheduling

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

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

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

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

6.4 Guided Local Search for Workforce Scheduling

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

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

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

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

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

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

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

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

The best results published so far on the workforce scheduling problem is in Azarmi & Abdul-Hameed [AA95]. Azarmi & Abdul-Hameed have looked at simulated annealing, constraint logic programming [Hen89, LWR95] and genetic algorithms [Hol75, Gol89, Dav91, WT94, ERR94]. The results are based on a benchmark test

problem with 118 engineers and 250 jobs. Each job can be served by 28 engineers on average, which means the search space is roughly 28^{250}, or 10^{360}, in size. This suggests that a complete search is very unlikely to succeed in finding the optimal solution.

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

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

(b) allowing no overtime.

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

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

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

all the results in Group II. The best result in Group III, which is 19,997, was found by GLS when λ was set to 20.

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

Algorithms		Total cost	CPU time (sec)	Travel cost	Cost (number) of unallocated jobs	over-time cost
Group I: Best results reported in the literature (no overtime allowed):						
GA		23,790	N.A.	N.A.	N.A. (67)	disallow
GA + repair		22,570	N.A.	N.A.	N.A. (54)	disallow
CLP - ElipSys + repair		21,292	N.A.	4,902	16,390 (53)	disallow
CLP - CHIP + repair		22,241	N.A.	5,269	16,972 (48)	disallow
SA		21,025	N.A.	4,390	16,660 (56)	disallow
Group II: Best results on FLS and GLS with overtime disallowed:						
Fast Local Search (FLS)		20,732	1,242	4,608	16,124 (49)	disallow
	λ = 10	20,556	5,335	4,558	15,998 (48)	disallow
	λ = 20	20,497	7,182	4,533	15,864 (49)	disallow
Fast GLS	λ = 30	20,486	6,756	4,676	15,810 (50)	disallow
	λ = 40	20,490	5,987	4,743	15,747 (48)	disallow
	λ = 50	20,450	3,098	4,535	15,915 (49)	disallow
	λ = 100	20,433	9,183	4,707	15,726 (48)	disallow
Group III: Best results on FLS and GLS, with a maximum of 10 minutes overtime allowed:						
Fast Local Search (FLS)		20,224	1,244	4,651	15,448 (51)	125
	λ = 10	20,124	4,402	4,663	15,329 (50)	132
	λ = 20	19,997	4,102	4,648	15,209 (49)	140
Fast GLS	λ = 30	20,000	2,788	4,690	15,155 (48)	155
	λ = 40	20,070	4,834	4,727	15,194 (48)	149
	λ = 50	20,055	2,634	4,690	15,197 (49)	168
	λ = 100	20,132	2,962	4,779	15,152 (48)	201

1. GA, CLP and SA results from Azarmi & Abdul-Hameed [AA95], Muller et. al. [MMS93] and Baker [Bak93];
2. FLS and GLS are implemented in C++, all results obtained from a DEC Alpha 3000/600 175MHz machine.
3. The benchmark problem, which has 118 engineers and 250 jobs, was obtained from British Telecom Research Laboratories, UK.

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

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

To evaluate the role of the activation bits in the efficiency of FLS, we compared FLS with a best improvement local search algorithm which used the same moves as FLS,

but without using activation bits to reduce its neighbourhood (we refer to this algorithm as LS). The results are shown in Table 6.2.

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

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

Table 6.2 Evaluation of the efficiency of FLS.

6.7 Remarks

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

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

Table 6.3 Ordering heuristics used in starting permutation.

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

6.8 Conclusions

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

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

Chapter 7

Nonconvex Optimisation

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

7.1 Nonconvex Optimisation and Global Optimisation Methods

Continuous optimisation problems arise in many engineering disciplines (such as electrical and mechanical engineering) in the context of analysis, design or simulation tasks. Particularly difficult problems are those with non-linear multi-extremal cost functions (that is functions with many local minima). These problems, also known as nonconvex optimisation problems [HL95], are difficult to solve using deterministic

gradient-based algorithms used extensively elsewhere in continuous optimisation. Gradient algorithms can be easily trapped in the many local minima of the cost function, so failing to reach the global minimum.

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

7.2 Local Search for Continuous Optimisation Problems

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

In the case of binary encoding, a binary string value is interpreted to represent an integer in base-2 notation. The mapping of the binary string to a real variable works as follows. The binary string value is first converted to the corresponding integer. This integer is then scaled by the appropriate coefficient to give a real value in the desired range (i.e. domain of variable) [Dav91]. One binary string is used for each problem

variable and combinatorial search is utilised to find these binary string configurations which after decoding result in the optimal value for the real-valued cost function. Increasing the number of bits used for representing each variable increases the accuracy of the solution but also results in an increase of the combinatorial search space.

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

7.3 The Sine Envelope Sine Wave (F6) Function

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

Eq. 7.1

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

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

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

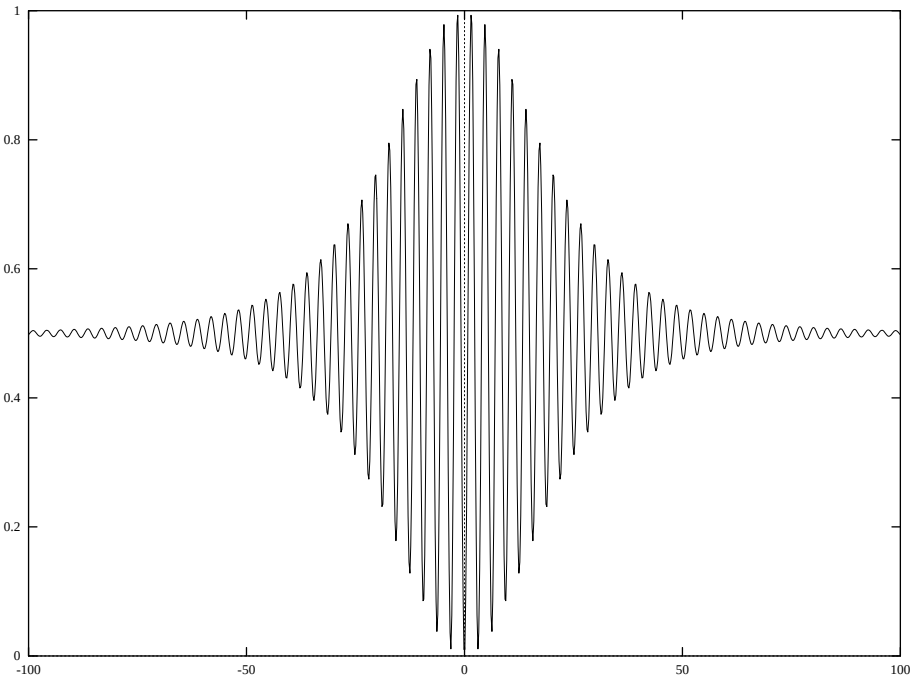

Figure 7.1 Cross section of F6 function

7.4 Guided Local Search for Global Optimisation

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

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

Eq. 7.2

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

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

Eq. 7.3

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

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

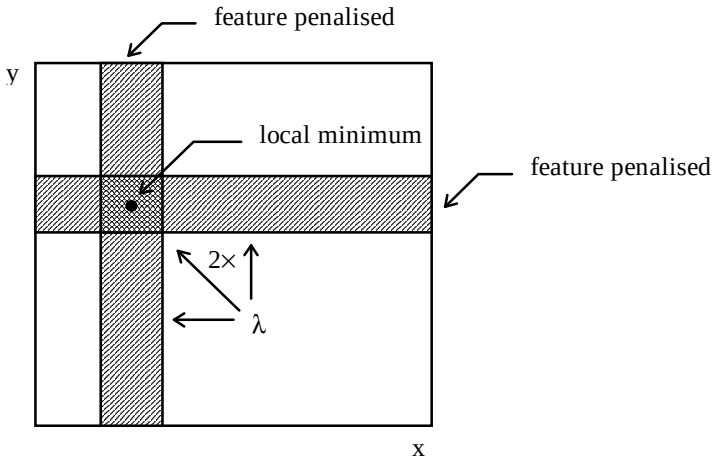

Figure 7.2 Changes in cost due to penalising the features exhibited by a local minimum