Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem

Edward Tsang & Chris Voudouris

Technical Report CSM-246
Department of Computer Science
University of Essex
Colchester CO4 3SQ
email: {voudcx, edward}@uk.ac.essex

August 1995

Abstract

This paper reports a Fast Local Search (FLS) algorithm which helps to improve the efficiency of hill climbing and a Guided Local Search (GLS) Algorithm which is developed to help local search to escape local optima and distribute search effort. To illustrate how these algorithms work, this paper describes their application to British Telecom's workforce scheduling problem, which is a hard real life problem. The effectiveness of FLS and GLS are demonstrated by the fact that they both out-perform all the methods applied to this problem so far, which include simulated annealing, genetic algorithms and constraint logic programming.

I. Introduction

Due to their combinatorial explosion nature, many real life constraint optimization problems are hard to solve using complete methods such as *branch & bound* [17, 14, 21, 23]. One way to contain the combinatorial explosion problem is to sacrifice completeness. Some of the best known methods which use this strategy are *local search* methods, the basic form of which often referred to as *hill climbing*. The problem is seen as an optimization problem according to an *objective function* (which is to be minimized or maximized). The strategy is to define a *neighbourhood function* which maps every candidate solution (often called a *state*) to a set of other candidate solutions (which are called *neighbours*). Then starting from a candidate solution, which may be randomly or heuristically generated, the search moves to a neighbour which is 'better' according to the objective function (in a minimization problem, a better neighbour is one which is mapped to a lower value by the objective function). The search terminates if no better neighbour can be found. The whole process can be repeated from different starting points.

One of the main problems with hill climbing is that it may settle in local optima — states which are better than all the neighbours but not necessarily the best possible solution. To over-

come that, methods such as *simulated annealing* [1, 7, 20] and *tabu search* [10, 11, 12] have been proposed. In this paper, we present a method, called *Guided Local Search* (GLS), to guide local search to escape local optima. Like tabu search, this method also allows the software engineer to build knowledge into the algorithm to guide the search towards areas which appear to be more promising.

A factor which limits the efficiency of a hill climbing aglorithm is the size of the neighbourhood. If there are many neighbours to consider, then if the search takes many steps to reach a local optima, and/or each evaluation of the objective function requires a nontrivial amount of computation, then the search could be very costly. In this paper, we present a method, which we called *Fast Local Search* (FLS) to restrict the neighbourhood. The intention is to ignore neighbours which are unlikely to lead to fruitful hill-climbs in order to improve the efficiency of a search.

We shall illustrate these two algorithms by using a real life scheduling problem as an example. We shall also use this example problem to demonstrate the effectiveness and efficiency of these two algorithms. In the next section, we shall explain this example problem.

II. BT's Workforce Scheduling Problem

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

$$(Loc, Dur, Type) \tag{1}$$

where *Loc* is the location of the job (depicted by its x and y coordinates), *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:

$$(Base, ST, ET, OT_limit, Skill) \tag{2}$$

where *Base* is the x and y coordinates 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 duration (*Dur*) 20, say, then this engineer would actually take 18 minutes to finish the job.

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

$$Total\ Cost = \sum_{i=1}^{NoT} TC_i + \sum_{i=1}^{NoT} OT_i^2 + \sum_{j=1}^{NoJ} (Dur_j + Penalty) \times UF_j \tag{3}$$

where: NoT = number of engineers;
NoJ = number of jobs;
TC_i = Travelling Cost of engineer i;
OT_i = Overtime of engineer i;
Dur_j = 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:

$$TC(\,(x_1, y_1),\,(x_2, y_2)\,) = \frac{\Delta_x/2 + \Delta_y}{8} \quad \text{if} \quad \Delta_x > \Delta_y \;;$$

$$= \frac{\Delta_y/2 + \Delta_x}{8} \quad \text{otherwise.} \tag{4}$$

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.

III. Fast Local Search Applied to Workforce Scheduling

III.1. Representation of candidate solutions and hill climbing

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 following (deterministic) algorithm:

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 e.g. [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.

Given a permutation, hill climbing is performed in a simple way: a pair of jobs are looked at 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 the hill climbing 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.

III.2. Fast Hill climbing

So far we have defined an ordinary hill climbing algorithm. Each state in this algorithm has $O(n^2)$ neighbours, where n is the number of jobs in the workforce scheduling problem. The *fast local search* is a general strategy which allows us to restrict the neighbourhood, and consequently improve the efficiency of the hill climbing. We shall explain here how it is applied to the workforce scheduling problem. Later we shall explain that this technique can be (and has successfully been) generalized to other problems.

To apply the fast local search to workforce scheduling, each job permuation position is associated with it an *activation bit*, which takes binary values (0 and 1). These bits are manipulated in the following way:

1. All the activation bits are set to 1 (or "*on*") when hill climbing 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 another job 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 hill climbing, only those job permutation positions which 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 involved in a successful swap recently will be examined more. 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.

IV. Guided Local Search Applied to Workforce Scheduling

IV.1. The GLS algorithm

Like all other hill climbing algorithms, FLS suffers from the problem of settling in local optima. Guided local search (GLS) is a method for escaping local optima. GLS is built upon our experience in a connectionist method called GENET (it is a generalization of the GENET computation models) [27, 6, 26].

GLS is a algorithm for modifying local search algorithms. The basic idea is that *costs* and *penalty* values are associated to selected features of the candidate solutions. Selecting such features in an application is not difficult because the objective function is often made up of a number of features in the candidate solutions. If the costs should normally take their values from the objective function. The penalties are initialized to 0.

Given an objective function g which maps every candidate solution s to a numerical value, we define a function h which will be used by the local search algorithm (replacing g).

$$h(s) = g(s) + \quad \lambda \sum_{i=1,F} p_i \cdot I_i(s) \tag{5}$$

where s is a candidate solution, λ is a parameter to the GLS algorithm, F is the number of features, p_i is the penalty for feature i (which are initialized to 0) and I_i is an indication of whether the candidate solution s exhibits feature i:

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i\text{; 0 otherwise.} \tag{6}$$

When the local search settles on a local optimum, the penalty of some of the features associated to this local optimum is increased (to be explained below). This has the effect of changing the objective function (which defines the "*landscape*" of the local search) and driving the search towards other candidate solutions. The key to the effectiveness of GLS is in the way that the penalties are imposed.

Our intention is to penalize "bad features", or features which "matter most", when a local search settles in a local optima. The feature which has high cost affects the overall cost more. Another factor which should be considered is the current penalty value of that feature. We define the utility of penalizing feature i, $util_i$, under a local optimum s_*, as follows:

$$util_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \tag{7}$$

In other words, if a feature is not exhibited in the local optimum, then the utility of penalizing it is 0. The higher the cost of this feature, the greater the utility of penalizing it. Besides, the more times that it has been penalized, the greater $(1 + p_i)$ becomes, and therefore, the lower the utility of penalizing it again.

In a local optimum, the feature(s) with the greatest *util* value will be penalized. This is done by adding 1 to its penalty value:

$$p_i = p_i + 1 \tag{8}$$

The λ parameter in the GLS algorithm is used to adjust the weight of penalty values in the objective function. As it will be shown later, results in applying GLS to the workforce scheduling problem is not very sensitive to the setting of this parameter.

By taking cost and the current penalty into consideration in selecting the feature to penalize, we are distributing the search effort in the search space. Candidate solutions which exhibit "good features", i.e. features involving lower cost, will be given more effort in the search, but penalties may lead the search to explore candidate solutions which exhibit features with higher cost. The idea of distributing search effort, which plays an important role in the success of GLS, is borrowed from Operations Research, e.g. see Koopman [16] and Stone [22].

Following we shall describe the general GLS procedure:

Procedure **GLS** (input: an objective function g and a local search strategy $\mathcal{L}$)

1. Generate a starting candidate solution randomly or heuristically;
2. Initialize all the penalty values (p_i) to 0;
3. Repeat the following until a termination condition (e.g. a maximum number of iterations or a time limit) has been reached:
   3.1. Perform local search (using $\mathcal{L}$) according to the function h (which is g plus the penalty values, as defined above) until a local optimum $\mathcal{M}$ has been reached;
   3.2. For each feature i which is exhibited in $\mathcal{M}$ compute $util_i = c_i / (1 + p_i)$
   3.3. Penalize every feature i such that $util_i$ is maximum: $p_i = p_i + 1$;
4. Return the best candidate solution found so far according to the objective function g.

IV.2. GLS applied to BT's workforce scheduling problem

To apply GLS to a problem, we need to identify the objective function and a local search algorithm. In earlier sections, we have described the objective function and a local search algorithm, FLS, for BT's workforce scheduling problem.

Our next task is to define solution features to penalize. 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 minimized. Therefore, we intend to penalize the inability to serve a job in a permutation. To do so, we associate a penalty value to each job. 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 the assignment of each job x to each engineer g, with the cost of this feature reflecting the travelling cost. This penalty is increased if the schedule in a local minimum uses engineer g to do job x.) Integrated into GLS, FLS will switch on (i.e. switching from 0 to 1) the activation bits associated to the positions where the penalized jobs currently lie.

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

V. Experimental Results

The best results published so far on the workforce scheduling problem is in Azarmi & Abdul-Hameed [3]. Apart from simulated annealing which we mentioned above, they have looked at constraint logic programming [24, 18] and *genetic algorithms* [15, 13, 8, 28, 9]. 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 [3] 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.* [19] for the GA approach and Baker [4] for the SA approach. Results obtained by GA and CLP were "repaired" (i.e. hill climbed) upon. 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 so far was 21,025, which was obtained by the SA approach. No timing was reported on the these tests. These results are shown in Table 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 1. FLS obtained a result of 20,732, which is better than all the reported results. This result is slightly 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!)

Table 1: Results obtained in BT's benchmark workforce scheduling problem

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
Fast GLS	$\lambda = 10$	20,556	5,335	4,558	15,998 (48)	disallow
	$\lambda = 20$	20,497	7,182	4,533	15,864 (49)	disallow
	$\lambda = 30$	20,486	6,756	4,676	15,810 (50)	disallow
	$\lambda = 40$	20,490	5,987	4,743	15,747 (48)	disallow
	$\lambda = 50$	20,450	3,098	4,535	15,915 (49)	disallow
	$\lambda = 100$	**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
Fast GLS	$\lambda = 10$	20,124	4,402	4,663	15,329 (50)	132
	$\lambda = 20$	**19,997**	4,102	4,648	15,209 (49)	140
	$\lambda = 30$	20,000	2,788	4,690	15,155 (48)	155
	$\lambda = 40$	20,070	4,834	4,727	15,194 (48)	149
	$\lambda = 50$	20,055	2,634	4,690	15,197 (49)	168
	$\lambda = 100$	20,132	2,962	4,779	15,152 (48)	201

Notes:
1. GA, CLP and SA results from Azarmi & Abdul-Hameed [3], Muller *et. al.* [19] and Baker [4];
2. FLS and GLS are implemented in C++, all results obtained from a DEC Alpha 3000/600 175MHz machine; results are available in http://cswww.essex.ac.uk/CSP/wfs;
3. The benchmark problem, which has 118 engineers and 250 jobs, is obtained from British Telecom Research Laboratories, UK.

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 in limiting overtime to 10 minutes per engineer, is shown in Group III of Table 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.

Test data[1] and results reported in this paper by FLS and GLS are included in Essex's world wide web (http://cswww.essex.ac.uk/CSP/wfs) for reference and verification.

VI. Discussion

In BT's workforce scheduling problem, an *activation bit* is associated to each job permutation postion. Although the definition of activation bits in FLS is domain dependent, it is not difficult to define them in an application. Hints can often be obtained from the objective function or the representation of candidate solutions.

FLS is a generalization of Bentley's *approximate 2-opting* algorithm [5], which is applied to the travelling salesman problem (TSP). Voudouris & Tsang [26] report the successful application of FLS and GLS in the TSP. Apart from these prolems, FLS has been applied successfully to the radio link frequency assignment problem, which will be reported in detail in another occasion (this problem and early results were reported in Voudouris & Tsang [25]).

To evaluate the role of the activation bits in the efficiency of FLS, we compare FLS with a local search algorithm which uses the same hill climbing strategy as FLS, but without using activation bits to reduce its neighbourhood (we refer to this algorithm as LS). The results are shown in Table 2.

Table 2: Evalutation of the efficiency of FLS

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 aglorithms implemented in C++, all results obtained from a DEC Alpha 3000/600 175MHz machine

1. with the permission of British Telecom

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.

In BT's workforce scheduling problem, penalties are associated to each job, because they are the subject of allocation. The selection of features to be penalized is (like the definition of activation bits) not difficult. It can often be done through observation in the objective function. Bits can be associated to elements which make up the total cost. (In fact, a similar exercise is often done in genetic algorithms in defining building blocks in a representation.)

We have also experimented with random starting permutations and a starting permutation with the jobs ordered by the ratio between their durations and the number of qualified engineers. Their results are shown in Table 3.

Table 3: Ordering heuristics used in starting permutation

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

In Table 3, an (almost) arbitrary λ value of 100 has been chosen to give the readers 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 3 shows that the result of FLS can be affected by the initial ordering of the job, though even the worse 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. This helps to show the robustness of GLS. More evidence of the effectiveness of GLS will be given in other occasions.

VII. Concluding Summary

In this paper, we have described two general local algorithms, namely the fast local search (FLS) algorithms and the guided local search (GLS) algorithm, for tackling constraint optimization problems. We have demonstrated their effectiveness in a case study using British Telecom's workforce scheduling problem. On the benchmark problem, our algorithms obtained results convincingly better than those published, which include simulated annealing, genetic algorithms and constraint logic programming.

The FLS algorithms is designed to speed up local search by restricting the neighbourhood to those which are more likely to contain better neighbours. This allows one to speed up a hill

climbing search. The GLS algorithm helps local search to escape local optima by adding a penalty term in the objective function. The penalties also help the search to distribute its effort according to the promise of the selected features in candidate solutions.

Acknowledgement

The Computer Service Department provided the DEC Alpha machines. The benchmark problem was obtained from British Telecom (BT) Research Laboratories. Edward Tsang was supported by a BT Short Term Research Fellowship in 1992, under the guidance of Barry Crabtree. Chris Voudouris is employed by the EPSRC funded project, ref GR/H75275.

References

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, 1989.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data structures and algorithms*, Addison-Wesley, 1983.

[3] N. Azarmi and W. Abdul-Hameed, "Workforce scheduling with constraint logic programming", *British Telecom Technology Journal*, Vol.13, No.1, January, 81-94 (1995).

[4] S. Baker, "Applying simulated annealing to the workforce management problem", ISR Technical Report, British Telecom Laboratories, Martlesham Heath, Ipswich (1993).

[5] J.J. Bentley, "Fast algorithms for geometric traveling salesman problems", *ORSA Journal on Computing*, Vol.4, 387-411 (1992).

[6] A. Davenport, E.P.K. Tsang, C.J. Wang and K. Zhu, "GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement", *Proc., 12th National Conference for Artificial Intelligence (AAAI)*, 325-330 (1994).

[7] L. Davis (ed.), *Genetic algorithms and simulated annealing*, Research notes in AI, Pitman/Morgan Kaufmann, 1987.

[8] L. Davis (ed.), *Handbook of genetic algorithms*, Van Nostrand Reinhold, 1991.

[9] A.E. Eiben, P-E. Raua, and Zs. Ruttkay, "Solving constraint satisfaction problems using genetic algorithms", *Proc., 1st IEEE Conference on Evolutionary Computing*, 543-547 (1994).

[10] F. Glover, "Tabu search Part I", *ORSA Journal on Computing* 1, 109-206 (1989).

[11] F. Glover, "Tabu search Part II", *ORSA Journal on Computing* 2, 4-32 (1990).

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

[13] D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley Pub. Co., Inc., 1989.

[14] P.A.V. Hall, "Branch and bound and beyond", *Proc. International Joint Conference on AI*, 641-650 (1971).

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

[16] B.O. Koopman, "The theory of search, part III, the optimum distribution of searching effort", *Operations Research*, Vol.5, 1957, 613-626 (1957).

[17] E.W. Lawler and D.E. Wood, "Branch-and-bound methods: a survey",

Operations Research 14, 699-719 (1966).

[18] J. Lever, M. Wallace, and B. Richards, "Constraint logic programming for scheduling and planning", *British Telecom Technology Journal*, Vol.13, No.1., 73-80 (1995).

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

[20] R.H.J.M. Otten and L.P.P.P. van Ginneken, *The annealing algorithm*, Kluwer Academic Publishers, 1989.

[21] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial algorithms: theory and practice*, Englewood Cliffs, N.J., Prentice hall, 1977.

[22] L.D. Stone, "The process of search planning: current approaches and continuing problems", *Operations Research*, Vol.31, 207-233 (1983).

[23] E.P.K. Tsang, "Scheduling techniques — a comparative study", *British Telecom Technology Journal*, Vol.13, No.1, 16-28 (1995).

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

[25] C. Voudouris and E.P.K. Tsang, "The tunnelling algorithm for partial constraint satisfaction problems and combinatorial optimization problems", Technical Report CSM-213, Department of Computer Science, University of Essex (1994).

[26] C. Voudouris and E.P.K. Tsang, "Guided Local Search", Technical Report CSM-247, Department of Computer Science, University of Essex (1995).

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

[28] T. Warwick and E.P.K. Tsang, "Using a genetic algorithm to tackle the processors configuration problem", *Symposium on Applied Computing* (1994).