The First International Conference on The Practical Application of Constraint Technologies and Logic Programming, London, April 1999, 359-383

A Family of Stochastic Methods For Constraint Satisfaction and Optimisation

Edward P K Tsang
Department of Computer Science, University of Essex, Colchester CO4 3SQ
United Kingdom
edward@essex.ac.uk

Chang J Wang
Lehman Brothers
101 Hudson Street
Jersey City, NJ 07302, USA
chwang@lehman.com

Andrew Davenport
Department of Industrial Engineering, University of Toronto M5S 3G9
Canada
andrewdv@interlog.com

.

Christos Voudouris
Intelligent Systems Research Group, Advanced Research & Technology Dept, BT Laboratories, British Telecommunications plc., United Kingdom
chrisv@info.bt.co.uk

Tung Leng Lau
Department of Computer Science, University of Essex, Colchester CO4 3SQ
United Kingdom
tllau@essex.ac.uk

Abstract

Constraint satisfaction and optimisation is NP-complete by nature. The combinatorial explosion problem prevents complete constraint programming methods from solving many real-life constraint problems. In many situations, stochastic search methods, many of which sacrifice completeness for efficiency, are needed. This paper reports a family of stochastic algorithms for constraint satisfaction and optimisation. Developed with hardware implementation in mind, GENET is a class of computation models for constraint satisfaction. Genet is a connectionist approach. A problem is represented by a network with inhibitory connections. The network is designed to converge, in a fashion that resembles the min-conflict repair method. Reinforcement learning is used to bring GENET out of local optima. Building upon GENET as well as ideas from operations research, Guided Local Search (GLS) and Fast Local Search are novel meta-heuristic search methods for constraint optimisation. GLS sits on top of other local-search algorithms. The basic principle of GLS is to penalise features exhibited by the candidate solution when a local search settles in a local optimum. FLS is a way of reducing the size of the neighbourhood so as to improve the efficiency of local search. As a meta-heuristic, GLS is embedded in genetic algorithms to form the Guided Genetic Algorithm (GGA). GGA extends the domain of application by GLS and improves its reliability (i.e. getting good results consistently). GENET, GLS, FLS and GGA have been applied to a non-trivial number of satisfiability and optimisation problems and achieved world-class results. GLS has also been incorporated in ILOG Dispatcher, a commercial package for vehicle routing.

Keywords: constraint satisfaction, optimisation, connectionism, meta-heuristic search, genetic algorithms

1. Introduction

Constraint satisfaction is a very general problem that is required in many real life problems [Tsang 1993, Freuder & Mackworth 1994, Wallace 1996]. Due to its generality, much research effort has been spent in this area in recent years. This has led to technological break-through as well as commercial exploitation, e.g. ILOG Solver [Puget 1995], CHIP [Simonis 1995], ECLiPSe [Lever *el al* 1995], Prolog IV [Colmerauer 1990] and IFProlog [IFProlog]. They have been applied to scheduling, resource allocation, machine vision, logic programming, natural language processing, and many other areas.

Completeness in algorithms is desirable when it can be achieved, but constraint satisfaction problems (CSPs) are NP-complete in general. The practicality of complete search methods is often limited by the combinatorial explosion problem. One often encounters CSPs that cannot be solved using complete methods within the time available. Imagine the application in emergency situations (e.g. in an earthquake) when one needs to schedule resources such as rescue teams and equipment to tasks. The scheduling speed could determine the amount of loss in life and property. Besides, the situation may change so rapidly that spending a lot of time to ensure completeness may not be practical. It may be more appropriate to reacting quickly, even with sub-optimal solutions. In some scheduling problems, one is faced with a large number of possible options. In order to search in a large space of models, the problem solver may be helped by an interactive system that could quickly evaluate the CSPs in the different models. Whether a system responses in minutes or seconds could make all the difference to the user.

In recent years, stochastic methods have received great attention. Most stochastic methods sacrifice completeness for efficiency. This paper describes a family of novel stochastic constraint satisfaction and optimisation algorithms. They are motivated by the kind of applications described above.

2. Constraint satisfaction and optimisation

A constraint satisfaction problem (CSP) comprises three elements:

$$(Z, D, C)$$

where Z is a finite set of variables; D is a function that maps every variable x in Z to a set of objects (of any type), which is called the *domain* of x. Most research in constraint satisfaction deal with discrete and finite domains. C is a set of constraints, which may take any form, which restricts the values that variables may take simultaneously. The task is to assign one value to each variable satisfying all the constraints [Tsang 1993]. A *k-ary constraint* restricts the values that k variables can take simultaneously. A CSP that contains unary and binary constraints only is called a *binary CSP*.

In many constraint satisfaction problems, some solutions are “better” than others, where “better” is defined by some domain-dependent objective functions. The task in such problems is to find the optimal (minimum or maximum) solution. In other problems, constraints are classified as *hard* and *soft* constraints. Hard constraints are not to be violated in any case. Soft constraints can be violated at certain costs. In some problems, assigning different values to different variables involve different utilities. The task is to minimise the cost or maximise the utilities in the solution. A few definitions will help explaining the ideas later: we define a *label* as an assignment of a value v to a variable x, denoted by $<x, v>$. A *compound label* is a set of labels. A compound label that assigns a value to every variable is called a *complete compound label*. A Partial Constraint Satisfaction problem can be defined as:

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

where g is a function which maps every compound label L to a numerical value.[1]

[1] If one is only interested in solutions to (Z, D, C), then one can define g in such a way that g(L) equals infinity (in a minimization problem) when the compound label L is not a solution.

3. GENET for constraint satisfaction

GENET is a connectionist approach to constraint satisfaction. The basic approach of GENET is to represent a CSP by a neural network. The nodes in the network export values to one another following connections defined by the constraints. Associated with the connections are weights, which are subject to changes through reinforcement learning. The nodes switch *on* or *off* depending on the network protocol. The connections and the protocol in GENET are carefully designed to ensure convergence of the network. The aim is to encourage the network to converge to states that represent solutions of the CSP. In this section, we shall describe a network construction for binary CSPs. We refer to this as the *GENET Binary Model*. Later we shall describe models for tackling non-binary CSPs.

3.1. GENET for binary CSPs

The structure of a network in the GENET Binary Model is as follows: each value for each variable is represented by a node in the network; such nodes are called *label nodes*. A label node is either in an *on* or *off* status. The label nodes for the same variable are grouped into a cluster. A random label node in each cluster is switched on during initialization. A connection is made between each pair of label nodes that represent two incompatible labels between two different variables. Associated with each connection is a weight, which is a negative value indicating the strength of an inhibitory relationship between the two connected nodes. All weights are initialized to -1 at start. Figure1 shows a CSP and a network that represents it in the GENET Binary Model.

The CSP in figure1 consists of five variables, *A*, *B*, *C*, *D* and *E*, the domains of which all being {1, 2, 3}. The constraints of this CSP are specified on the edges. For example, the constraint between *A* and *B* requires the sum of *A* and *B* to be even. Therefore, the labels <*A*,1> and <*B*,2> are incompatible with each other, so a connection is created between the corresponding label nodes in the network shown in figure2. Other connections are made under the same principle.

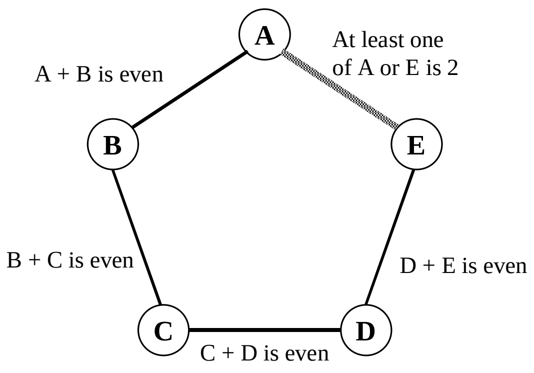

Figure 1. Example of a CSP (all domains are {1, 2, 3})

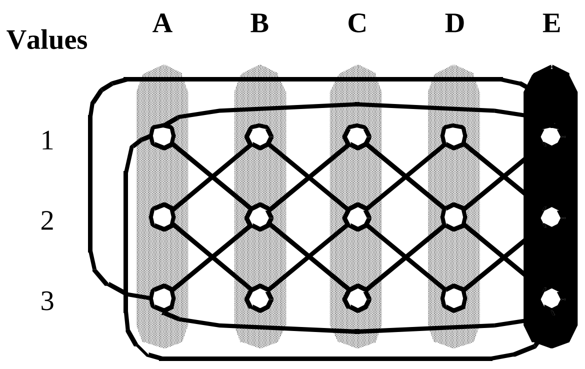

Figure 2. A GENET representation of the problem in Figure 1

Each cluster has a modulator that ensures that exactly one label node in each cluster is switched on at a time. The modulator in all the clusters work in parallel locally and asynchronously. Each label node that is on outputs a unit value to all the nodes that are adjacent (i.e. joined by a connection) to it. This value is multiplied by the weight associated with the connection. A label node that is off gives no output. The input to a label node *x*, which we denote by I_x, is computed as follows:

$$\mathrm{I}_x = \sum_{adjacent(x,y)} w_{x,y} \times s_y \qquad (1)$$

where $w_{x,y}$ is the weight associated with the connection between nodes *x* and *y*, and s_y is the state of *y*, which is 1 if *y* is on and 0 if *y* is off. During initialization, a random label node is switched on in each cluster. Each cluster would continuously examine its own state by choosing the label node that has the highest input to switch on. Since all weights are negative, this effectively means choosing the label that violates the least weighted number of constraints. In this way, the GENET Binary Model resembles the min-conflict heuristic by Minton *et al.* [1990]. Unlike the min-conflict heuristic repair method, which break ties randomly, the GENET Binary Model allows the label node that is currently on to retain its status in tie situations. The purpose of doing so is to ensure that the network converges. If no node in the tie is currently on (which could only happen immediately after the initial state or after learning, see below), then a random node in the tie will be switched on.

In constraint satisfaction, a local minimum can be recognized when the network state does not change but some on nodes receive negative inputs.[2] To escape local minima, the GENET Binary Model incorporates a reinforcement learning mechanism. When the network has converged to a local minimum, the weights associated with the connections between on nodes are reduced by 1. In other words, after converging in local minima, the following rule applies:

$$w_{xy} = w_{xy} + s_x \times s_y \qquad (2)$$

As before, w_{xy} represents the weight associated with the connection between label nodes x and y; s_x and s_y represent the states of x and y.

We define the energy E_b of a particular state of the network (the subscript b indicates that it is for the binary model) as the halved total input to all the nodes, i.e. the total number of weighted constraint violations:

$$E_b = -\frac{1}{2}\sum_i \sum_j w_{i,j} s_i s_j = -\frac{1}{2}\sum_k s_k I_k \qquad (3)$$

where i, j, k range over all variables.[3] A solution to the given CSP, or a global optimal, is therefore represented by a state whose energy equals 0. The higher the energy of a network, the more weighted constraints are violated. It is provable that between learning, the energy in a network under the GENET Binary Model will decrease monotonically, hence the network will always converge. Detailed proofs will not be presented here. Whether the converged state represents a solution to the given CSP is another matter.

GENET is designed for hardware implementation [Wang & Tsang 1994]. In simulation, we assume that the network runs in convergence cycles. In each cycle, each cluster examines its current state and switches on an alternative label node if necessary. We assume that each cluster has the chance to examine its current state once before another convergence cycle starts.

[2] It is worth pointing out that local search in general does not normally know when it has reached local optima. The fact that local optima are recognizable in constraint satisfaction makes it a suitable domain for local search.
[3] The energy is not equivalent to the number of constraints being violated since the weights need not be –1 after learning has taken place.

We also assume that one cluster examines its current state at a time. While these may not be the case in a hardware implementation, the behaviour of the network should not be significantly different. The number of convergence cycles is a useful measure for the efficiency of GENET computation models. The GENET Binary Model has been applied to random binary CSPs and graph colouring problem and demonstrated to be reliable and efficient in solving these problems [Wang & Tsang 1991, Tsang & Wang 1992].

3.2. GENET for non-binary CSPs

The idea of GENET was extended to non-binary CSPs.[4] The major computation model developed for non-binary CSPs was called the GENET Stable Model. It was thus named because, like the Binary Model, it was designed to guarantee convergence. The basic idea is to introduce *constraint nodes* to GENET. Each constraint in the CSP is represented by one constraint node. One type of constraint node is designed for each type of constraint, but the following basic principles (from the GENET Binary Model) apply:

1. If a constraint node *c* represents a constraint that is violated, then *c* sends inhibitory signals to all label nodes concerned to discourage them to remain on.
2. If a constraint node *c* represents a constraint that is not yet violated, but will be violated should any label node *l* is switched from off to on, then *c* sends an inhibitory signal to *l*.
3. Otherwise, a constraint node sends no inhibitory signal out.

Constraint nodes for a number of non-binary constraints have been designed. We shall use the *atmost* constraint to illustrate the principle here. The constraint

$$atmost(m, \{x_1, x_2, \ldots, x_n\}, \{\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_\mathrm{k}\})$$

states that at most *m* of the *n* variables $x_1, x_2, \ldots, x_n$ may take values from the set $\{\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_\mathrm{k}\}$ simultaneously. For example, the compound label (<*w*, a><*x*, b><*y*, c><*z*, b>) violates the

[4] It was unfair for Lee *et al* [1995] to claim that they were the first to extend GENET to non-binary CSPs. Wang & Tsang [1991] first published an idea to apply GENET to non-binary CSPs. It introduced the terms *constraint node* and *label node* which were adopted by Lee *et al* [1995]. Besides, three GENET models for non-binary CSPs were described in Davenport *et al* [1994].

constraint *atmost*(2, {*w*, *x*, *y*, *z*}, {b, c}) because more than 2 variables (namely, *x*, *y* and *z*) are taking values from the set {b, c}.

Given an *atmost* constraint, every label node that represents the assignment of the values v_1, v_2, …, v_k to the variables x_1, x_2, …, x_n is connected to this constraint node, which we refer to as *atm*. Each constraint node *atm* has a weight, denoted by W_{atm}. The weight is initialized to –1, which could be changed through learning. We shall use T_{x_i} to denote a cluster which represents the variable x_i where x_i is a member of the set of constrained variables {x_1, x_2, …, x_n} in the *atmost* constraint. We shall use L_{x_i} to denote the set of label nodes in T_{x_i} which is constrained by *atm*. A *relevant label node* (RLN) is a label node in a L_{x_i} of some cluster T_{x_i}. From *atm*'s point of view, T_{x_i} is an *active* cluster if there exists a label node in L_{x_i} which is on; otherwise this cluster is called an *inactive* cluster. Below we shall describe the structure of the network concerning *atm*.

Connections between RLNs and *atm* all have weights equal to 1, which will remain constant throughout. Each RLN *x* outputs 1 to *atm* when *x* is on, 0 otherwise. Therefore, the input to *atm* (denoted by I_{atm}) is simply the number of RLNs that are on:

$$\text{input to the constraint node } atm,\ I_{atm} = \sum_{y \in RLN} s_y \qquad (4)$$

The *atmost* constraint node *atm* stores a value (we denote the value stored by vs_c), which is the input (I_{atm}) minus the threshold *m*:

$$\text{value of the constraint node, } vs_c = I_{atm} - m \qquad (5)$$

In other words, the *atmost* constraint is violated when the value stored in the constraint node is greater than 0. The constraint node *atm* individually outputs to every RLN. The output from *atm* to every RLN is 0 when vs_c is negative. When vs_c is 0, *atm* outputs W_{atm} to every L_{x_i} in every active cluster T_{x_i}. These RLNs are therefore discouraged to become on. When vs_c is greater than 0, *atm* outputs $vs_c \times W_{atm}$ to every RLN in every active cluster, and $(1+ vs_c) \times W_{atm}$ to every RLN in every inactive cluster. One can interpret this arrangement as: *atm* encourages the active clusters to switch off RLNs, but gives inactive clusters even less incentive to switch on any RLNs. The principle here is that whenever $vs_c \geq 0$, the inactive clusters always receive more inhibitory input than the active clusters.

A network under the Stable Model behaves in the following way: a random label node per cluster is switched on initially. Then each constraint node and each cluster will update its state asynchronously whenever necessary. Each cluster will switch on the label node that has the highest input (i.e. the one that has the least weighted inhibitory input). In a tie situation, the Stable Model will keep the label node that is currently on. If none of the label nodes in the tie is on, then a random node is switched on. If the network converges to a local minimum, the weights between label nodes will be updated in exactly the same way as in the Binary Model. The weights associated with the violated constraint nodes are reduced by 1.

$$\text{New } W_{atm} = \text{Old } W_{atm} - \text{Violated}_{atm} \tag{6}$$

where

$$\text{Violated}_{atm} = \begin{cases} 0 & \text{if } vs_c \leq 0; \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

It is provable that, like the Binary Model, the Stable Model with *atmost* constraint nodes as described only will always converge. Limited by space, the proof will not be presented here.

The *Stable-sideways* (Stable-SW) Model is an important variation of the Stable Model. In tie situations, the modulator of the Stable model keeps the current on label node to be on. The Stable-SW Model breaks ties randomly. We call a move that switches on an alternative label node without reducing the energy of the network a *sideways move*. By allowing sideways moves, the Stable-SW model loses the nice property of guaranteed convergence. The Stable-SW model may continuously change its state (by switching on alternative label nodes) without settling in any state. So it needs a mechanism to escape plateaus and local minima. The Stable-SW model invokes learning when the network stays in the same state for K consecutive cycles, where *K* is a parameter.[5] The Stable-SW Model with $K = 2$ was found to out-perform the Stable Model in many domains tested.

The GENET Stable Model represents a class of computation models. It can be varied and instantiated in different ways. Only a few variations have been explored. Davenport *el al* [1994] reported the *Stable1-SW Model*, which was simply called “the GENET model” in that paper. The basic difference between the Stable1 Model and the Stable Model is as follows. In the Stable1 Model, the connections between constraint nodes and label nodes are directional and asymmetric. Besides, the constraint nodes all have weights equal to 1, which is kept constant. An individual weight is associated with each of the connections from constraint nodes to label nodes. These weights are initialized to −1, and subject to changes through learning. The connections from label nodes to constraint nodes all have weights equal to 1, which will remain constant.

4. Guided Local Search for Optimisation

Guided Local Search (GLS) was a generalization of GENET. While GENET was designed for satisfying constraints, GLS was designed for optimisation. GLS also borrows certain ideas from Operations Research (OR): the use of penalties in OR [Luenberger 1984] resembles weight learning in GENET. GLS is a meta-heuristic algorithm that sits on top of hill-climbing algorithms. We shall summarize hill-climbing first so that we can refer to its components later.

4.1. Basic Principles of Hill-climbing

A basic form of local search is often referred to as hill-climbing. To perform hill-climbing, one must define the following:

(a) *representation*: a representation for candidate solutions;

(b) an *objective function*: given any candidate solution, this function returns a numerical value. The problem is seen as an optimisation problem according to this objective function (which

[5] An alternative is to define convergence as remaining in the same energy level for *K* iterations.

is to be minimized or maximized);

(c) a *neighbourhood function* that maps every candidate solution *x* (often called a *state*) to a set of other candidate solutions (which are called the *neighbours* of *x*).

Hill-climbing works as follows: 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, or resources run out. 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 that are better than all their neighbours but not necessarily the best possible solution. To overcome that, methods such as *Simulated Annealing* [Davis 1987] and *Tabu Search* [Glover *et al* 1993] have been proposed.

4.2. Fast Local Search (FLS)

One factor that limits the efficiency of a hill-climbing algorithm is the size of the neighbourhood. If there are many neighbours to consider, then if the search takes many steps to reach a local optimum, and/or each evaluation of the objective function requires a nontrivial amount of computation, then the search could be very costly. Bentley [1992] presented the *approximate 2-Opt* method to reduce the neighbourhood of 2-Opt in the TSP (see, e.g. [Aho *et al* 1983]). We generalised this method to a method that we call *Fast Local Search* (FLS). The intention is to, guided by heuristics, ignore neighbours that are unlikely to lead to fruitful hill-climbs in order to improve the efficiency of a search.

Here we shall use the TSP to show how FLS can be applied to the 2-Opt neighbourhood function. An *activation bit* is associated to each transition position in the tour (i.e. transition between each *k*-th and (*k*+1)-th positions in the tour). All activation bits are switched on at start. Only positions with an *on* activation bit will be examined to see if it can make an improvement move. If no improvement is possible, then this bit is switched off. It will only be switched on again under two conditions:

(1) if a 2-Opt step (initiated by another position) is made which inverts a subsequence which ends in this position. For example, if the subsequence between the fourth and the sixth cities were reversed, then the activation bit for both transitions third-to-fourth and sixth-to-seventh will be switched on.
(2) if this transition is a feature that is penalised (to be explained when we introduce GLS later).

To generalise this to neighbourhood functions other than the 2-Opt, one may define features for candidate solutions. Selecting such features in an application is not difficult because the objective function is often made up of a number of terms, which can be used as features in the candidate solutions.

By reducing the size of the neighbourhood, one may reduce the amount of computation involved in each hill-climbing step. The aim is to enable more hill-climbing steps in a fixed amount of time. The danger of ignoring certain neighbours is that some improvements may be missed. The hope is that the gain out-weighs the loss. We found that FLS combined extremely well with GLS.

4.3. The Guided Local Search Algorithm

GLS is a general meta-heuristic algorithm for optimisation. Like simulated annealing and tabu search, the aim of GLS is to help hill-climbing escape local optima. The basic idea is to augment the objective function with penalties, which direct the search away from local optimum. So one can say that GLS is an algorithm for modifying the behaviour of hill-climbing. To apply GLS, one has to define *features* for the candidate solutions. For example, in the travelling salesman problem, a feature could be “whether the candidate tour travels immediately from city A to city B”.

GLS associates a *cost* and a *penalty* to each feature. The costs should normally take their values from the objective function. For example, in the travelling salesman problem, the cost of the above feature is the distance between cities A and B. The penalties are initialized to 0 and will only be increased when the local search reaches local optimum. This will be elaborated below.

Given an objective function *g* that maps every candidate solution *s* to a numerical value, we define a function *h* which will be used by hill-climbing (replacing *g*).

$$h(s) = g(s) + \lambda \times \Sigma(p_i \times I_i(s)) \tag{8}$$

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

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i\text{; 0 otherwise.} \tag{9}$$

When the local search settles on a local optimum, the penalties of some of the features associated to this local optimum are 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 penalties are imposed. It is worth emphasising that a slight variation in the way that penalties are managed could make all the difference to the effectiveness of a local search.

Our intention is to penalize "unfavourable features" or features that "matter most" when a local search settles in a local optimum. The feature that has high cost affects the overall cost more. Another factor that 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:

$$util_i(s^*) = I_i\,(s^*) \times c_i \,/\, (1 + p_i) \tag{10}$$

where c_i is the cost and p_i is the current panelty value of feature i. 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 (c_i), the greater the utility of penalizing it. Besides, the more times that it has been penalized, 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 increasing its penalty value by 1:

$$p_i = p_i + 1 \tag{11}$$

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 help to prevent all effort be directed to the best features. 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 [1957] and Stone [1983]. Following we shall describe the general GLS procedure:

```
Procedure GLS (input: an objective function $g$; a local search strategy $L$; features
        and their costs; parameter $\lambda$ )
  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 time limit) has been reached:
      3.1. Perform local search (using $L$) according to the function $h$ (which is g
           plus the penalty values, as defined in (8) above) until a local optimum $M$
           has been reached;
      3.2. For each feature $i$ which is exhibited in $M$ 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 $g$.
```

4.4. GLS and FLS Applications

GLS and FLS have been applied to a non-trivial number of problems. They have been applied to *radio link frequency assignment problem* (RLFAP) [Bouju *et al* 1995] and British Telecom's *work force scheduling* problem (WFS) [Baker 1993, Azarmi & Abdul-Hameed 1995]. In the RLFAP, the task is to assign available frequencies to communication channels satisfying constraints that prevent interference. In some RLFAPs, the goal is to minimize the number of frequencies used. GLS+FLS reported the best results when it was published [Voudouris & Tsang 1996]. New and significantly improved results were reported in the NATO Symposium on RLFAP in Denmark, 1998 [Voudouris & Tsang 1998].

In British Telecom's WFS, the task is to assign technicians from various bases to serve the jobs, which may include customer requests and repairs, at various locations. Customer requirements and working hours restrict the times that certain jobs can be served by certain technicians. The objective is to minimize a function that takes into account the travelling cost, overtime cost and unserved jobs. In the WFS, GLS+FLS holds the best-published results in the benchmark problem available to the authors [Tsang & Voudouris 1997].

The most significant results of GLS and FLS are probably in their application to the travelling salesman problem (TSP). The Lin-Kernighan algorithm (LK) is a specialised algorithm for TSP that has long been perceived as the champion of this problem [Lin & Kernighan 1973, Martin & Otto 1996]. We tested GLS+FLS+2Opt against LK in a set of benchmark problems from the public TSP library [Reinelt 1991]. Given the same amount of time (we tested 5 cpu minutes and 30 cpu minutes on a DEC Alpha 3000/600), GLS+FLS+2Opt found better results than LK in average. GLS+FLS+2Opt also out-performed the best Simulated Annealing [Johnson *et al* 1989], Tabu Search [Knox 1994] and Genetic Algorithm [Freisleben & Merz 1996] implementations reported on TSP so far. One must be cautious when interpreting such empirical results as they could be affected by many factors, including implementation issues. But given that the TSP is an extensively studied problem, it takes something special for an algorithm to out-perform the champions under any reasonable measure ("*find me the best results within a given amount of time*" must be a realistic requirement). It must be emphasized that LK is specialized for TSP but GLS and FLS are much simpler general-purpose algorithms. Details of GLS+FLS applied to the TSP can be found in [Voudouris & Tsang 1998].[6]

[6] We have tried to implement the compared algorithms as efficiently as possible. Even if speed is not the best measure for GLS's success, credit should be given to GLS for its generality and simplicity. A demonstration program for GLS on TSP is available from *http://cswww.essex.ac.uk/CSP/gls.html*

GLS has also been applied to general function optimisation problems to illustrate that artificial features can be defined for problems in which the objective function suggests no obvious features. Results show that, as expected, GLS spreads its search effort across solution candidates depending on their quality (as measured by the objective function). Besides, GLS consistently found solutions in a landscape with many local sub-optimals [Voudouris 1998].

Research in GLS and its predecessor GENET has been followed up by researchers outside our group. GLS has been incorporated in *Dispatcher*; a commercial package for vehicle routing problems developed by ILOG (http://www.ilog.fr/html/products/) [Backer *et al* 1997]. British Telecom is looking into using GLS for scheduling. Bouju *et al* [1995] applied GENET to radio length frequency assignment. Other applications of GENET include rail traffic control [Jose & Boyce 1997] and logic programming [Lee & Tam 1995, Stuckey & Tam 1998]. Having succeeded in so many applications, GLS and FLS are algorithms that anyone interested in optimisation cannot afford to ignore.

5. Guided Genetic Algorithm, an Extension of GLS

5.1. Overview of Guided Genetic Algorithm (GGA)

GLS is a meta-heuristic algorithm. Not only can it sit on top of local search algorithms, it can also be embedded in *Genetic algorithms* (GAs). GAs borrow their ideas from evolution [Holland 1975, Davis 1987]. The idea is to maintain a *population* of candidate solutions. The candidate solutions are given individual chances to produce offspring depending on their "*fitness*". Fitness is measured by the objective function in optimisation. GAs have been applied to constraint satisfaction [Eiben *et al* 1994, Ruttkay *et al* 1995, Chu & Beasley 1997, Warwick & Tsang 1994, 1995].

Guided Genetic Algorithm (GGA) is a hybrid of GA and GLS. It aims to widen the applicability of GAs (which is limited by epistasis) and further improve the robustness of GLS. It can be seen as a GA with GLS to bring it out of local optimum: if no progress has been made after a number of iterations (this number is a parameter to GGA), GLS modifies the fitness function (which is the objective function) by means of penalties. GA will then use the modified fitness function in future generations. The penalties are also used to bias crossover and mutation in GA – genes that are involved in more penalties are made more susceptive to changes by these two GA operators. This allows GGA to be more focussed in its search.

On the other hand, GGA can roughly be seen as a number of GLSs from different starting points running in parallel, exchanging partial solutions in a GA manner. The difference is that only one set of penalties is used in GGA whereas parallel GLS would have used one independent set of penalties per run. One can say that learning in GGA is more selective than parallel GLS: the updating of penalties is only based on the best chromosome found at the point of penalization.

5.2. Features and Penalties in GGA

Like GLS, GGA runs with the augmented objective function (though it records the best solutions so far according to the original objective function). As in GLS, one needs to define features in GGA. In GGA, a feature θ_i is defined by the assignment of a fixed set of (possibly single) assignments, $Cover(\theta_i) = \{x_{i1}, x_{i2}, ..., x_{in}\}$ to specific values. For example, a feature may be exhibited if variable x takes values from {1, 2, 3}. To conform with GLS notations, we define an indicator function τ_i for each feature θ_i:

$$\tau_i(\gamma_p) = 1 \text{ if chromosome } \gamma_p \text{ exhibits feature } \theta_i\text{; 0 otherwise}$$

Penalties are used to change the objective function used by GA in order to bring it out of local optima. They are also used to bias crossover and mutation, as will be described later.

As in GLS, a user-defined cost c_i is associated to each feature *i*. Internally, GGA associates one penalty counter p_i to each feature *i*. Costs will remain constant but penalties could be updated during the run. Penalties are all initialized to 0. Penalties are global. In other words, one set of penalty values is applied to all chromosomes in GGA.

GGA invokes GLS whenever the best chromosome in the population remains unchanged for *k* GA cycles, where *k* is a parameter to GGA. The penalty operator, a special operator in GGA, selects features θ_i in the best chromosome γ_p in the current population in the same way as GLS, i.e. the features θ_i which have the highest $util_i$ value (equation 10).

5.3. Fitness Templates, The Key to GGA's Performance

One novel element of GGA is the *fitness template*. It is the bridge between GLS and GA. It enables the penalties to affect GA's crossover and mutation operators. Every chromosome γ_p is associated with a fitness template, δ_p, which is made up of *weights*. One weight $\delta_{p,q}$ corresponds to each gene $\gamma_{p,q}$. Each weight $\delta_{p,q}$ defines how susceptive to change the corresponding gene $\gamma_{p,q}$ is during crossover and mutation: the greater the value $\delta_{p,q}$ ("heavier") the more susceptive to change $\gamma_{p,q}$ is, as it will be elaborated later.

The fitness templates are computed from the penalties when a chromosome is created. They are updated when the penalties are changed. The weights of a fitness template are computed in the following way: all weights are initialized to zero. Then for each feature θ_i that is exhibited by the chromosome, the weight $\delta_{p,q}$ is increased by the value in the penalty counter for θ_i if the gene $\gamma_{p,q}$ is in *Cover*(θ_i).

5.4. GGA Operators and Population Dynamics

In this section, we shall explain the operations of a generic GGA.[7] First we shall explain the special operators in GGA. Then we shall explain GGA's control flow.

5.4.1. GGA's Crossover Operator

Crossover produces new members of the population by combining genetic information from two parents. GGA's crossover operator makes use of the fitness templates of the parents. Two parents γ_p and $\gamma_{p'}$ are selected to produce one offspring γ_c. The probability of performing crossover on a pair of parents is called the *crossover rate*. If crossover were to be performed,

[7] The operators can be specialized to suit individual problems.

then each gene $\gamma_{p,q}$ in parent γ_p competes against the corresponding gene $\gamma_{p',q}$ in parent $\gamma_{p'}$ for a place in the offspring. This competition is in reverse proportion to the corresponding weights, $\delta_{p,q}$ and $\delta_{p',q}$. The probability of genes $\gamma_{p,q}$ and $\gamma_{p,q}$ being selected are $\frac{\delta_{p',q}}{\delta_{p,q} + \delta_{p',q}}$ and $\frac{\delta_{p,q}}{\delta_{p,q} + \delta_{p',q}}$ respectively. In other words, the "lighter" gene (the one with the smaller weight in its fitness template) has a better chance of being passed to the offspring. The selection of each gene is independent of the others during crossover. Since the offspring may exhibit features different from those exhibited by the parents, the fitness template are computed afresh for the offspring rather than inherited.

5.4.2. GGA's Mutation Operator

Mutation produces variations in the population through altering the information that genes carry. Whether mutation is performed or not is determined by the mutation rate, which is a GGA parameter. In GGA, the mutation rate is defined as a fraction of the length of each chromosome. The number of genes in a chromosome to mutate is the product of the mutation rate and the length of that chromosome.

For every chromosome, GGA selects the required number of genes to mutate. The selection is done using the *roulette wheel* method. In this selection method, the probability for each gene $\gamma_{c,q}$ to be picked is directly proportional to its weight $\delta_{c,q}$. In other words, the "heavier" a gene $\gamma_{c,q}$ (i.e. the greater the value $\delta_{c,q}$), the more chance it has of being selected for mutation.

Having picked a gene $\gamma_{c,q}$ for mutation, the next step is to seek a value for it. This could have been done randomly. In GGA, the value that yields the best fitness (i.e. the best local improvement) will be selected. Ties are broken randomly.

After a gene $\gamma_{c,q}$ has been mutated, its weight $\delta_{c,q}$ may optionally be reduced by one unit in GGA. This will reduce the chance of $\gamma_{c,q}$ being selected for mutation again. GGA allows the same gene to mutate more than once because the next time it is picked for mutation, some other genes may have been mutated. So the best value in the last mutation may no longer be the best when the same gene is picked again.

5.4.3. The Control Flow in GGA

Having described the components of GGA, we are now in a position to describe its control flow. Like any standard GA, given a problem, one needs to define a representation for candidate solutions. For GGA, one also needs to define features and their costs. To start, an initial population is generated randomly. A fitness template is computed for each chromosome in the population. Then GGA cycles ensue.

In each GGA cycle, the reproduction operator picks chromosomes from the population, using the roulette wheel method, to form the mating pool. Pairs of chromosomes are picked from the mating pool randomly to form parents. Whether crossover, as described above, is performed is determined by the crossover rate, which is a GGA parameter. The offspring is potentially mutated using the mutation operator described above, depending on the mutation-rate.

Potentially mutated offspring are then given a chance to join the population in the next generation. The old population and the offspring are ranked by their fitness. The fittest *n* chromosome in the ranking are used to form the population in the next generation, where *n* is the population size which is a GGA parameter.

New elements of the GGA come into play at this point. The new population is surveyed for the possibility of being trapped in local optima. If the best solution in the population remains unchanged for *k* iterations, where *k* is a GGA parameter, then the search is concluded to be in local optimum. This invokes the penalty operator, which modifies the objective function in exactly the way that GLS does based on the fittest string in the population. To reflect the change of the objective function, the fitness templates for individual chromosomes are updated. This will affect the crossover and mutation operators in the next GGA cycle.

GGA cycles until it reaches the pre-defined termination conditions, which could be defined in terms of a maximum number of cycles or computation time.

5.5. GGA Applications

The Royal Road Function is a function with many large plateaus; hill climbing in general can be very inefficient in it [Mitchell 1996]. GGA has been applied to this problem and demonstrated to be more efficient than both GAs [Mitchell 1996] and GLS [Lau 1998].

GGA has been applied to the Processors Configuration Problem in which one is to find configurations of processors minimizing the communication distance [Chalmers 1994]. GGA found configurations with shorter average communication distance than those found by other algorithms reported so far [Lau & Tsang 1997, 1998a]. In the General Assignment Problem, GGA has been found to be as good as one of the best algorithm proposed for this problem so far (which is a GA algorithm) [Chu & Beasley 1997], with even more remarkable robustness [Lau & Tsang 1998b]. In the *radio link frequency assignment problem* (RLFAP, described above), GGA found results as good as GLS, but solution costs fall into a narrower range [Lau & Tsang 1998c].

To summarize, GGA has been found to be effective in a wide range of problems. It can improve the robustness of GLS, which itself improves the robustness (as well as effectiveness) of the hill-climbing algorithms that it sits on [Lau 1998]. As as general algorithm, GGA has found results comparable to, if not better than, those obtained by other world class algorithms in the tested problems.

Acknowledgements

The authors appear in chronological order in their joining the group. Jim Doran, Terry Warwick, James Borrett, Kangmin Zhu, Alvin Kwan, John Ford, Patrick Mills, Paul Scott, Richard Williams and Nathan Barnes have all made valuable contributions to this project. This project was partially sponsored by EPSRC funded projects GR/H75275 and GR/L20122. Andrew Davenport was sponsor by an EPSRC studentship for his PhD study. Chris Voudouris is being supported by British Telecom in this research. Tung-Leng Lau was sponsored by the University of Essex’s Scholarship. The anonymous reviewers gave useful comments. Unfortunately, not all of them can be incorporated in this paper due to space limitation.

References

1. Aho, A.V., Hopcroft, J.E. & Ullman, J.D., Data structures and algorithms, Addison-Wesley, 1983
2. Azarmi N. and Abdul-Hameed W., *Workforce scheduling with constraint logic programming*, British Telecom Technology Journal, Vol.13, No.1, January, 1995, 81-94
3. Backer, B.D., Furnon, V., Kilby, P., Prosser, P. & Shaw, P., *Solving vehicle routing problems using constraint programming and metaheuristics*, Technical Report, GreenTrip Project, http://www.cs.strath.ac.uk/~ps/GreenTrip/, 1997
4. Baker S., *Applying simulated annealing to the workforce management problem,* ISR Technical Report, British Telecom Laboratories, Martlesham Heath, Ipswich, 1993
5. Bentley J.J., *Fast algorithms for geometric traveling salesman problems*, ORSA Journal on Computing, Vol.4, 1992, 387-411
6. Bouju, A., Boyce, J.F., Dimitropoulos, C.H.D., vom Scheidt, G. & Taylor, J.G., *Intelligent search for the radio link frequency assignment problem*, Proceedings of the International Conference on Digital Signal Processing, Cyprus 1995
7. Boyce, J.F., Dimitropoulos, C.H.D., vom Scheidt, G. & Taylor, J.G., *GENET and tabu search for combinatorial optimization problems*, World Congress on Neural Networks, Washighton D.C., 1995
8. Chalmers, A.G., *A minimum path parallel processing environment*, Research Monographs in Computer Science, Alpha Books, 1994
9. Chu, P. & Beasley, J.E., *Genetic algorithms for the generalized assignment problem*, Computers and Operations Research, Vol.24, 1997, 17-23
10. Colmerauer, A., *An introduction to Prolog III*, CACM Vol.33, No7, July 1990, 69-90
11. Davenport A., Tsang E.P.K., Wang C.J. and Zhu K., *GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement*, Proc., 12th National Conference for Artificial Intelligence (AAAI), 1994, 325-330
12. Davenport, A., *Extensions and evaluation of GENET in constraint satisfaction*, PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK, July 1997
13. Davis L. (ed.), *Genetic algorithms and simulated annealing*, Research notes in AI, Pitman/Morgan Kaufmann, 1987.
14. Davis L. (ed.), *Handbook of genetic algorithms*, Van Nostrand Reinhold, 1991
15. Eiben A.E., Raua P-E., and Ruttkay Zs., *Solving constraint satisfaction problems using genetic algorithms*, Proc., 1st IEEE Conference on Evolutionary Computing, 1994, 543-547
16. Mitchell, M., *An introduction to genetic algorithms,* MIT Press, Cambridge, Massachusetts, 1996
17. Freisleben, B., and Merz, P., *A Genetic Local Search Algorithm for Solving the Symmetric and Asymmetric TSP*, Proceedings of IEEE International Conference on Evolutionary

Computation, Nagoya, Japan, 1996, 616-621

18. Freuder, E.C. & Mackworth, A., (ed.), *Constraint-based reasoning*, MIT Press, 1994
19. Glover F., E. Taillard, and D. de Werra, *A user's guide to tabu search*, Annals of Operations Research, Vol.41, 1993, 3-28
20. Holland J.H., *Adaptation in natural and artificial systems*, University of Michigan press, Ann Arbor, MI, 1975
21. IFProlog, *http://www.biz.isar.de/ifcomputer/*
22. Johnson, D., *Local optimization and the traveling salesman problem*, Proceedings of the 17th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science, 443, 446-461, Springer-Verlag, 1990
23. Johnson, D., Aragon, C., McGeoch, L., and Schevon, C., *Optimization by simulated annealing: an experimental evaluation, part I, graph partitioning*, Operations Research, 37, 865-892, 1989
24. Jose, R. & Boyce, J., *Appication of connectionist local search to line management rail traffic control*, Proceedings, Practical Applicaiton of Constraint Technology (PACT'97), London, April 1997
25. Kilby, P., Prosser, P. & Shaw, P., *Guided local search for the vehicle routing problem*, Proc., 2nd International Conference on Metaheuristics (MIC97), Sophia-Antipolis, France, July 1997, 21-24
26. Knox, J., *Tabu search performance on the symmetric traveling salesman problem*, Computers Ops Res., Vol. 21, No. 8, pp. 867-876, 1994
27. Koopman B.O., *The theory of search, part III, the optimum distribution of searching effort*, Operations Research, Vol.5, 1957, 613-626
28. Lau, T.L. & Tsang, E.P.K., *Solving the processor configuration problem with a mutation-based genetic algorithm*, International Journal on Artificial Intelligence Tools (IJAIT), http://www.wspc.com.sg/journals/journals.html, World Scientific, Vol.6, No.4, December 1997, 567-585
29. Lau, T.L. & Tsang, E.P.K., *Solving large processor configuration problems with the guided genetic algorithm*, IEEE 10th International Conference on Tools with Artificial Intelligence (ICTAI'98), Taiwan, November 1998
30. Lau, T.L. & Tsang, E.P.K., *The guided genetic algorithm and its application to the general assignment problem,* IEEE 10th International Conference on Tools with Artificial Intelligence (ICTAI'98), Taiwan, November 1998
31. Lau, T.L. & Tsang, E.P.K., *Guided genetic algorithm and its application to the radio link frequency allocation problem*, Proceedings, NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Denmark, October 1998
32. Lau, T.L., *Guided Genetic Algorithm*, PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK, submitted September 1998

33. Lee, J.H.M., Leung, H.F. & Won, H.W., *Extending GENET for non-binary CSP's*, Proceedings, Seventh International Conference on Tools with Artificial Intelligence, 1995, 338-342

34. Lee, J.H.M. & Tam, V.W.L., *A framework for integrating artificial neural networks and logic programming*, International Journal on Artificial Intelligence Tools, Vol.4, Nos.1&2, June 1995, 3-32

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

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

37. Luenberger, D., *Linear and nonlinear programming*, Addison-Wesley Publishing Co., Inc., 1984

38. Martin, O., and Otto, S.W., *Combining simulated annealing with local search heuristics*, in: G. Laporte and I. H. Osman (eds.), Metaheuristics in Combinatorial Optimization, Annals of Operations Research, Vol. 63, 1996

39. Minton, S., Johnston, M., Philips, A.B. & Laird, P., *Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems*, Artificial Intelligence, Vol.58, Nos.1-3 (Special Volume on Constraint Based Reasoning), 1992, 161-205

40. Puget, J-F., *Applications of constraint programming*, in Montanari, U. & Rossi, F. (ed.), Proceedings, Principles and Practice of Constraint Programming (CP'95), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg & New York, 1995, 647-650

41. Reinelt, G., *A Traveling Salesman Problem Library*, ORSA Journal on Computing, 3, 1991, 376-384

42. Ruttkay, Zs., Eiben, A.E. & Raue, P.E., *Improving the performances of GAs on a GA-hard CSP*, Proceedings, CP95 Workshop on Studying and Solving Really Hard Problems, 1995, 157-171

43. Simonis, H., *The CHIP system and its applications*, in Montanari, U. & Rossi, F. (ed.), Proceedings, Principles and Practice of Constraint Programming (CP'95), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg & New York, 1995, 643-646

44. Stone L.D., *The process of search planning: current approaches and continuing problems*, Operations Research, Vol.31, 1983, 207-233

45. Stuckey, P. & Tam, V., *Semantics for using stochastic constraint solvers in constraint logic programming*, Journal of Functional and Logic Programming, January 1998

46. Tsang, E.P.K., *Foundations of constraint satisfaction*, Academic Press, 1993

47. Tsang E.P.K., *Scheduling techniques — a comparative study*, British Telecom Technology Journal, Vol.13, No.1, 1995, 16-28

48. Tsang, E.P.K. & Voudouris, C., *Fast local search and guided local search and their application to British Telecom's workforce scheduling problem*, Operations Research Letters, Elsevier Science Publishers, Amsterdam, Vol.20, No.3, March 1997, 119-127

49. Tsang, E.P.K. & Wang, C.J., *A generic neural network approach for constraint satisfaction problems*, in Taylor, J.G. (ed.), Neural network applications, Springer-Verlag, 1992, 12-22

50. Voudouris, C. & Tsang, E.P.K., *Partial constraint satisfaction problems and guided local search*, Proc., Practical Application of Constraint Technology (PACT'96), London, April, 1996, 337-356

51. Voudouris, C., *Guided Local Search for Combinatorial Optimisation Problems*, PhD Thesis, Department of Computer Science, University of Essex, UK, July 1997

52. Voudouris, C., *Guided Local Search -- An illustrative example in function optimisation*, BT Technology Journal, Vol.16, No.3, July 1998, 46-50

53. Voudouris C. & Tsang, E.P.K., *Guided local search and its application to the traveling salesman problem*, European Journal of Operational Research, Vol.113, Issue 2, November 1998, 80-110

54. Voudouris, C. & Tsang, E.P.K., *Solving the radio link frequency assignment problem using guided local search*, Proceedings, NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Denmark, October 1998

55. Wallace, M., *Practical applications of constraint programming, Journal of Constraints*, Kluwer Academic Publishers, Boston, Vol.1, Nos.1&2, 1996, 139-168

56. Wang, C.J. & Tsang, E.P.K., *Solving constraint satisfaction problems using neural-networks*, Proceedings of IEE Second International Conference on Artificial Neural Networks, 295-299, 1991

57. Wang, C.J. & Tsang, E.P.K., *A cascadable VLSI design for GENET*, in Delgado-Frias, J.G. & Moore, W.R. (ed.), VLSI for Neural Networks and Artificial Intelligence, Plenum Press, New York, 1994, 187-196

58. Warwick T., and Tsang, E.P.K., *Using a genetic algorithm to tackle the processors configuration problem*, Proc., ACM Symposium on Applied Computing, 1994, 217-221

59. Warwick, T. & Tsang, E.P.K., *Tackling car sequencing problems using a generic genetic algorithm*, Evolutionary Computation, Vol.3, No.3, 1995, 267-298