Partial Constraint Satisfaction Problems and Guided Local Search

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

Technical Report CSM-250
September 1995

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

Abstract

A largely unexplored aspect of Constraint Satisfaction Problem (CSP) is that of over-constrained instances for which no solution exists that satisfies all the constraints. In these problems, mentioned in the literature as Partial Constraint Satisfaction Problems (PCSPs), we are often looking for solutions which violate the minimum number of constraints. In more realistic settings, constraints violations incur different costs and solutions are sought that minimize the total cost from constraint violations and possibly other criteria.

Problems in this category present enormous difficulty to complete search algorithms. In practical terms, complete search has more or less to resemble the traditional Branch and Bound taking no advantage of the efficient pruning techniques recently developed for CSPs. In this report, we examine how the stochastic search method of Guided Local Search (GLS) can be applied to these problems. The effectiveness of the method is demonstrated on instances of the Radio Link Frequency Assignment Problem (RLFAP), which is a real-world Partial CSP. Guided Local Search holds the best known solutions for many of the publicly available instances of RLFAP.

1. Introduction

Guided Local Search is the outcome of a research project with main aim to extend the GENET neural network [27, 4, 20] for constraint satisfaction problems [20] to partial constraint satisfaction [5, 20] and combinatorial optimization problems. Beginning with GENET, we developed a number of intermediate algorithms such as the Tunneling Algorithm [25] to conclude with Guided Local Search (GLS). In contrast to its predecessors, GLS is a general and compact optimization technique suitable for a wide range of combinatorial optimization problems. The algorithm has already been applied to famous problems in combinatorial optimization like the Traveling Salesman Problem and Quadratic Assignment Problem [24]. Other applications of GLS include function optimization [23] and also large scale scheduling [21].

The method takes advantage of problem and search related information to guide local search [13] in a search space. This is made possible by augmenting the cost function of the problem to include a set of penalty terms. Local search is confined by the penalty terms and focuses attention on promising regions of the search space. Iterative calls are made to local search. Each time local search gets caught in a local minimum, the penalties are modified and local search is called again to minimize the modified cost function. Penalty modifications *regularize* the solutions generated by local search to be in accordance to prior or gathered during search information. The approach taken by GLS is similar to that of regularization methods for 'ill-posed' problems [19, 9]. The idea behind regularization methods and GLS up to an extent is the use of prior information to help us solve an approximation problem. Prior information translates to constraints which further define our problem reducing so the number of candidate solutions to be considered.

Generally speaking, GLS objectives are similar to those of a wider class of combinatorial optimization algorithms known as *tabu search* [6, 7, 8]. In fact, GLS could be classified as a tabu search method though there are many and distinct differences with the other methods developed so far within that framework. To mention a few, GLS is a compact algorithm that can be used without any modifications for a range of problems [24, 21, 23]. This contrasts with variants of tabu search which are mainly problem specific. Constraints in tabu search (also called *tabu restrictions*) refer to moves and they are usually implemented as lists (*tabu lists*). In contrast to that, constraints in GLS refer to solutions (solution features to be precise) and take the form of penalties that augment the cost function. In tabu search, constraints are hard and can only be overridden by the complementary mechanism of *aspiration criteria*. On the other hand, GLS constraints are soft and as such can be overridden. Last but not least, GLS provides the necessary mechanisms absent in other methods to capture and exploit problem and search related information. Before proceeding with explaining GLS and its application to Partial CSPs, we formally define Partial CSP and also examine the Radio Link Frequency Assignment Problem [15] for which results are reported in this paper.

2. Partial Constraint Satisfaction Problem

For convenience, we shall first define some terminology. The assignment of a value to a variable is called a *label*. The label which involves the assignment of a value v to the variable x (where v is in the domain of x) is denoted by the pair $\langle x,v \rangle$. A simultaneous assignment of values to a set of variables is called a *compound label* and is represented as a set of labels, denoted by $(\langle x_1,v_1\rangle,\langle x_2,v_2\rangle,\ldots,\langle x_k,v_k\rangle)$. A *complete compound label* is a compound label which assigns a value to every variable in the CSP. The goal in CSP is to find one or all complete compound labels that satisfy the constraints.

A *Partial Constraint Satisfaction Problem* (PCSP) is a Constraint Satisfaction Problem in which one is prepared to settle for partial solutions — solutions which may violate some constraints or assignments of values to some, but not all variables — when solutions do not exist (or, in some cases, cannot be found) [5, 20]. This kind of situation often occurs in applications like industrial scheduling where the available resources are not enough to cover the requirements. Under these circumstances, partial solutions are acceptable and a problem solver has to find the one that minimizes an objective function.

The objective function is domain-dependent and may take various forms. In one of its simplest forms, the optimization criterion may be the number of the constraint violations. For more realistic settings, some constraints may be characterized as "hard constraints" and they must be satisfied whilst others, which are referred to as "soft constraints", may be violated if necessary. Moreover, constraints may be assigned violation costs which reflect their relative importance. Partly following Tsang [20], we define the Partial Constraint Satisfaction Problem formally as follows:

Definition 1:

A *partial constraint satisfaction problem* (PCSP) is a quadruple:

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

where

- $Z = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables,
- $D = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$ is a set of finite domains for the variables in Z,

- $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of constraints on an arbitrary subset of variables in Z,
- g is the objective function which maps every compound label to a numerical value.

The goal in a PCSP is to find a compound label (partial or complete) which optimizes (minimizes or maximizes) the objective function g. Given the above definition, standard CSPs and *Constraint Satisfaction Optimization Problems* (CSOPs) (where optimal solutions are required in CSPs, see [20]) can both be cast as PCSPs. Under the Partial CSP formulation, all compound labels (partial or complete) can be characterized as *solutions* since constraint violations are part of the cost function and the problem is considered an unconstrained one.

Versions of branch and bound and other complete methods have been suggested for tackling PCSPs [5, 26]. But complete algorithms are inevitably limited by the combinatorial explosion problem.

3. The Radio Link Frequency Assignment Problem — A Real World PCSP

The *Radio Link Frequency Assignment Problem* was abstracted from the real life application of assigning frequencies (values) to radio links (variables). Eleven instances of RLFAPs, which involve various optimization criteria, were made publicly available by the French Centre d'Electronique l'Armament [15]. Two different types of binary constraints are involved in these RLFAPs:

- The absolute difference between two frequencies must be greater than a given number k (i.e. for two frequencies X and Y, $|X - Y| > k$);
- The absolute difference between two frequencies must be exactly equal to a given number k (i.e. for two frequencies X and Y, $|X - Y| = k$).

The above constraints are either hard or soft constraints. A problem specifies the varibles which are subject to these constraints and the constraint graph is not complete (i.e. not every variable is constrained by every other variable). If all the constraints can be satisfied then either:

- (C1) the solution which assigns the fewest number of different values to the variables,
- (C2) or the solution which largest assigned value is minimal

is preferred. For insoluble problems, *violation costs* are defined for the constraints. Furthermore, for some insoluble problems, default values are defined for some of the variables. If any of the default values is not used in the solution returned, then a predetermined *mobility* cost applies. Table 1 depicts the characteristics of the RLFAP instances.

RLFAP Instance	No. Variables	No. Constraints	Soluble	Minimize
Scen01	916	5,548	Yes	number of different values used (C1)
Scen02	200	1,235	Yes	number of different values used (C1)
Scen03	400	2,760	Yes	number of different values used (C1)
Scen04	680	3,968	Yes	number of different values used (C1)
Scen05	400	2,598	Yes	number of different values used (C1)
Scen06	200	1,322	No	maximum value used (C2)
Scen07	400	2,865	No	weighted constraint violations
Scen08	916	2,744	No	weighted constraint violations
Scen09	680	4,103	No	weighted constraint violations + mobility costs
Scen10	680	4,103	No	weighted constraint violations + mobility costs
Scen11	680	4,103	Yes	number of different values used (C1)

Table 1 Characteristics of RLFAP instances.The domains of variables consist of 6-44 integer values.

In RLFAP, complete compound labels are sought. For PCSPs where partial compound labels are sought, the reader is referred to [21] where GLS is used to tackle a real world workforce scheduling problem belonging to this last category.

4. Local Search for Partial PCSPs

GLS makes iterative calls to a local search procedure modifying the cost function to be minimized between successive calls. A local search procedure for Partial CSPs has already been described in previous work of the authors [25]. The scheme is based on the min-conflicts heuristic of Minton et al. [12] and the computational model of the GENET network [27, 4, 20]. A 1-optimal type move is used which changes the value of one variable at a time. Starting from a random and complete assignment, variables are examined in an arbitrary static order. Each time a variable is examined, the current value of the variable changes to the value which yields the minimum value for the cost function. Ties are randomly resolved allowing moves which transit to solutions with equal cost. These moves, often called sideways moves [16], enable local search to examine plateau of states occurring in the landscapes of many CSPs and Partial CSPs. One problem with sideways moves is that of detecting local minimum. This problem can be overcome using the limited sideways scheme described in [25]. In particular, we characterize a solution as a local minimum when all variables have been examined and now change occurred in the value of the cost function. Although, we allow sideways moves to occur locally, if these moves do not result in a better solution after all variables have been examined then a local minimum is concluded.

```
PROCEDURE LocalSearch(Z, D, g, S_i, S_i+1)
BEGIN
        S ← S_i;
        REPEAT
                g_before ← g(S);
                FOR each variable x in Z DO
                BEGIN
                        S ← S - {<x,v_i>};
                        FOR each value v in Dx DO
                        BEGIN
                                g_v ← g(S + {<x,v>});
                        END
                        BestSet ← set of values with minimum g_v;
                        v_i+1 ← random value in BestSet; (* sideways moves *)
                        S ← S + {<x,v_i+1>};
                END
                g_after ← g(S);
        UNTIL (g_after = g_before) (* local minimum is concluded *)
        S_i+1 ← S;
END
```

Figure 1 Local Search for PCSPs in pseudocode.

The pseudocode in Figure 1 depicts the basic local search procedure for PCSPs. The procedure starts with a solution S_i (which is a compound label as described in section 2) and returns a local minimum solution S_{i+1}.

5. Guided Local Search

5.1. Solution Features

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

Constraints on features are introduced or strengthened on the basis of information about the problem and also the course of the local search. Information pertaining to the problem is the cost of features. The cost of features represents direct or indirect impact of the corresponding solution properties on the solution cost. Feature costs may be constant or variable. Information about the search process pertains to the solutions visited by local search and in particular local minima.

A feature f_i is represented by an indicator function in the following way:

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

5.2. Augmented Cost Function

Constraints on features are made possible by augmenting the cost function g of the problem to include a set of penalty terms. The new cost function formed is called the *augmented cost function* and it is defined as follows:

$$h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s) \tag{2}$$

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

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

$$\boldsymbol{p} = (p_1, \ldots, p_M) \tag{3}$$

each time local search settles in a local minimum. Modifications are made on the basis of information. Initially, all the penalty parameters are set to 0 (i.e. no features are constrained) and a call is made to local search to find a local minimum of the augmented cost function. After the first local minimum and every other local minimum, the algorithm takes a modification *action* on the augmented cost function and re-applies the local search, starting from the previously found local minimum. The modification action is simply incrementing by one the penalty parameter of one or more of the local minimum features. Information is gradually inserted into the augmented cost function by selecting which penalty parameters to increment.

Sources of information are the cost of features and the local minimum itself. Let's consider that each feature f_i defined over the solutions is assigned a cost c_i. This cost may be constant or variable. In order to simplify our analysis, we consider feature costs to be constant and given by the *cost vector* $\boldsymbol{c}$:

$$\boldsymbol{c} = (c_1, \ldots, c_M) \tag{4}$$

which contains positive or zero elements.

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

$$I_i(s_*) = 1 \tag{5}$$

Thus, the vector $\boldsymbol{b}$ of the indicator function values in the local minimum s_* :

$$b(s_*) = (I_1(s_*), \ldots, I_M(s_*)) \tag{6}$$

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

5.3. Penalty Modifications

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

$$util(s_*, f_i) = I_i(s_*) \cdot \frac{c_i}{1 + p_i} \tag{7}$$

In other words, incrementing the penalty parameter of the feature f_i is considered an *action* with utility given by (7). In a local minimum, the actions with *maximum* utility are selected and then performed.

The utility function (7) makes full use of the information carrying vectors $\boldsymbol{b}(s_*)$ and $\boldsymbol{c}$. The penalty parameter p_i is incorporated in (7) to prevent the scheme from being totally biased towards penalizing features of high cost. The role of the penalty parameter in (7) is that of a counter which counts how many times a feature has been penalized. If a feature is penalized many times over a number of iterations then the term $\frac{c_i}{1 + p_i}$ in (7) decreases for the feature diversifying choices and giving the chance to others features to be also penalized. The policy implemented is that features are penalized with a frequency proportional to their cost. Due to (7), features of high cost are penalized more frequently than those of low cost and the opposite. The search effort is distributed according to *promise* as it is expressed by the feature costs and the already visited local minima. The basic GLS algorithm as described so far is given in Figure 2.

```
PROCEDURE GuidedLocalSearch(Z, D, g, λ, [I1, ...,IN], [c1,...,cN], M)
BEGIN
        k ← 0;
        S0 ← arbitrary assignment of values to variables;
        FOR i ← 1 until M DO
                pi ← 0;
        h ← g + λ * Σpi*Ii ;
        WHILE StoppingCriterion DO
        BEGIN
                LocalSearch(Z,D,h,Sk,Sk+1);
                FOR i ← 1 until M DO
                        utili ← Ii(Sk+1) * ci / (1+pi);
                FOR each i such that utili is maximum DO
                        pi ← pi + 1;
                k ← k+1;
        END
        S* ← best solution found with respect to cost function g;
        return S*;
END
```

Figure 2 The GLS algorithm in pseudocode.

Incremental distribution of the search effort according to information in a probabilistic framework can be found in a class of methods deriving themselves from the *optimal search theory* of Koopman [11, 17]. Also, counter based schemes for search diversification like that of GLS are used under the name *counter-based exploration* in reinforcement learning [18].

5.4. Regularization Parameter

Something that has been left out from the so far analysis is the regularization parameter λ in the augmented cost function (2). This parameter determines the degree up to which constraints on features are going to affect local search. Let's examine how the regularization parameter is going to affect the moves performed by a local search method. A move alters the solution, adding new features, removing existing features whilst leaving other features unchanged. In the general case, the difference Δh in the value of the augmented cost function due to a move is given by the following difference equation:

$$\Delta h = \Delta g + \lambda \cdot \sum_{i=1}^{M} p_i \Delta I_i \tag{8}$$

As we can see in (8), if λ is large then the selected moves will solely remove the penalized features from the solution and the information will fully determine the course of local search. This introduces risks because information may be wrong. Conversely, if λ is 0 then local search will not be able to escape from local minima. However, if λ is small and comparable to Δg then the moves selected will aim at the combined objective to improve the solution (taking into account the gradient) and also remove the penalized features (taking into account the information). Since the difference Δg is problem dependent, the regularization parameter is also problem dependent. Our experience shows that GLS can be quite tolerant to the choice of the λ value [21, 23, 24].

5.5. Fast Local Search and Other Improvements

There are both minor and major alterations that improve the method. For example, instead of calculating the utilities (7) for all the features, we can restrict ourselves to the local minimum features because for non-local minimum features the utility function (7) takes the value 0. Also, the evaluation mechanism for moves should be changed. Usually, this mechanism is not evaluating directly the cost of the new solution generated by the move but it calculates the difference Δg caused to the cost function. This difference in cost should be combined with the difference in penalty as this is shown in (8). This can be done easily and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only the features that change state (being removed or added). The penalty parameters of the features removed are summed together. The same is done for the penalty parameters of features added. The change in penalty due to the move is then simply given by difference:

$$- \sum_{\text{over all features j added}} p_j + \sum_{\text{over all features k removed}} p_k \tag{9}$$

Leaving behind the minor improvements, we turn our attention to the major improvements. In fact, these improvements do not directly refer to GLS but to local search. Greedy local search selects the best solution in the whole neighborhood. This is very time consuming especially if we are dealing with large instances of problems. In [24], we have presented *fast local search* which drastically speeds up the neighborhood search process. The method is a generalization of the *approximate 2-opt* method proposed by Bentley [1] for the traveling salesman problem.

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

Initially, all sub-neighborhoods are active. If a sub-neighborhood is examined and does not contain any improving moves then it becomes inactive. Otherwise, it remains active and the improving move found is performed. Depending on the move performed, a number of other sub-neighborhoods are also activated. In particular, we activate all the sub-neighborhoods where we expect other improving moves to occur as a result of the move just performed. As the solution improves the process dies out with fewer and fewer sub-neighborhoods being active until all the sub-neighborhood bits turn to 0. The solution formed up to that point is returned as a local minimum.

The overall procedure is many times faster than conventional local search. The bit setting scheme encourages chains of moves that improve specific parts of the overall solution. As the solution becomes locally better the process is fading down examining fewer moves saving enormous amounts of time otherwise spent on examining predominantly bad moves.

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

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

A concrete example of fast local search will be given later and in the context of the Radio Link Frequency Assignment Problem. Other examples can be found in [21] and also [24].

6. Using Guided Local Search for Partial CSPs

Applying guided local search to a problem simply requires the existence of a local search procedure preferably a version of fast local search and also a set of features which will be used to bias local search. Both prerequisites are domain dependent allowing the GLS algorithm to adapt to particular combinatorial optimization problems. A local search procedure for PCSPs has already been described in section 4. Fast local search for PCSPs will be explained later in this paper. For the moment, we focus our attention on the features to be used in PCSPs. In particular, we examine the features used to bias local search in the RLFAP instances. The same or similar features can be used in many other problems in the PCSP class.

Features should preferably capture structural properties of solutions which account for part of the solution cost. GLS distributes the search effort according to feature costs biasing local search to avoid specific features and favor others. In most cases, a glance at the cost function is enough to suggest which features are appropriate to be used in a problem.

6.1. Constraints

The main cost factor in PCSPs is constraint violation costs (sometimes mentioned as relaxation costs). In a simple setting, all the problem's constraints have violation costs defined (high for hard constraints) which denote their relative importance. The sum of violation costs gives the cost function of the problem. Constraints can be used to define a basic feature set for PCSPs. Each constraint is interpreted as a feature with the feature cost given by the violation cost of the constraint. Note here, that feature costs are not incorporated in the cost function. They are solely used along with the other pieces of information to determine which features are to be penalized in a local minimum (see utility function (8)).

For example, in the RLFAP, a set of constraints is given for each instance. Apart from relaxing each constraint and including its violation cost in the cost function, each constraint defines a feature which is used to guide local search. Feature costs are set equal to the corresponding violation costs and the cost function is augmented with a set of modifiable penalty parameters. Initially, the penalty parameters are set to 0 and each constraint accounts only for its violation cost. Each time local search settles in a local minimum, the penalties for some of the constraints violated (the corresponding features are exhibited) are increased according to the general scheme described in section 5.3. Constraints with high costs are penalized more frequently than those with low costs. In the short term, local search escapes from the local minimum while in the long term, it is biased to spend more time on solutions that satisfy high cost constraints than low cost constraints.

6.2. Assignment Costs.

Some of the insoluble RLFAP instances (Scen09 and Scen10) involve assignment costs. In particular, a cost is incurred when a variable is assigned a value which is different from a default value provided. These costs are called *mobility costs* and apply to only some of the variables. RLFAP mobility costs are comparable to constraint violation costs and are linearly combined with constraint violation costs to form the objective function.

Local search remains unchanged for these problems. If GLS was also to remain unchanged then the distribution of the search effort would only be determined by the constraint violation costs ignoring the extra mobility costs to be minimized. This will not result to the best possible performance. Extra information pertaining to mobility costs may be exploited to affect the distribution of the search effort. The set of features based on constraints is augmented with extra features that detect assignments of particular values to variables which incur mobility costs. The costs of these new features are set equal to the corresponding mobility costs. GLS operates on the combined set of features which now includes both constraints and assignments.

6.3. Minimize the Number of Different Values Used

In resource allocation problems, a main concern is the efficient utilization of resources. In many cases, this translates into satisfying all requests using the minimum number of resources possible. Frequencies are the resources in RLFAP. As mentioned in section 3. Some of the RLFAP instances as soluble (Scen01-03 and Scen11). For these instances, solutions are sought that satisfy all constraints and also use as few frequencies as possible. In other words, the goal is to find a solution which satisfies all constraints and also minimizes the number of different values used. The problem is similar to finding the minimum number of colors (i.e. chromatic number) needed to color a graph.

One possibility is to include this criterion in the cost function as it is described for graph coloring by Johnson [10]. The alternative approach examined here is not to include this criterion in the objective function but instead to bias local search using penalties such that this criterion is minimized. In particular, a feature is defined for each value in the union of domains. This feature is exhibited only when the corresponding value is assigned to at least one of the variables. By penalizing the feature, we can prevent the associated value from being assigned to any of the variables.

The costs of these features should be such that we prefer to penalize values that are assigned to only few of the variables. The motivation is that values that are assigned to only few of the variables could be swapped for values that are assigned to many of the variables, decreasing so the total number of values used. The fewer the number of variables that are assigned a value the higher should the cost of the related feature be. For a value v in the union of domains the cost of the associated feature f_v is given by:

$$c(s_*, f_v) = \frac{\text{total number of variables}}{(\text{number of variables assigned value } v \text{ in } s_*) + 1} \tag{10}$$

where s_* is the local minimum solution in the context of which the feature cost is evaluated. The above feature costs are not constant like those in sections 6.1 and 6.2. This is because we can not be sure which value can be avoided unless a solution has been found that satisfies all the constraints. If the solution violates some of the constraints, these constraints are penalized first taking precedence over the value features in the penalizing scheme.

6.4. Minimize Maximum Value Used

This last criterion is involved in only one of the RLFAP instances (Scen05). The approach taken for this criterion was to penalize constraints first and if these were satisfied to penalize the maximum value used without considering the utility function (7).

7. Using Fast Local Search for PCSPs

A greedy local search for PCSPs evaluates all possible 1-optimal moves over all variables before selecting and performing the best move. The local search procedure described in section 4 is already a faster alternative to greedy local search since the neighborhood is confined to the values of each variable. In spite of that, further improvements may be introduced in the algorithm of Figure 1 using the activation bits technique of fast local search described in section 5.5. The reader is also referred to [24] and also [21] for more information on fast local search and its other applications.

In the case of PCSPs, a bit is attached to each problem variable. If the bit of a variable is 1 then the variable is called *active* and it is examined for improving moves otherwise it is called *inactive* and it is discarded by local search. Whenever a variable is examined and a move is performed the activation bit of the variable remains to 1 otherwise it turns to 0 and the variable is not examined in future iterations. Additionally, if a move is performed activation spreads to other variables which have their bits set to 1. In particular, we set to 1 the bit of variables where improving moves may occur as a result of the move just performed. In general, such variables are those that are connected via a constraint to the variable where the move performed.

Three main schemes for the spreading of activation may be used. The schemes determine which variables are to be activated when the value of a variable changes and they are the following:

- S1. Activate all variables connected via a constraint to the variable which changed value.
- S2. Activate only variables that are connected via a constraint which is violated.
- S3. Activate only variables that are connected via a constraint that changed state (i.e. violated $\rightarrow$ satisfied or satisfied $\rightarrow$ violated) as a result of the move.

S2 and S3 are the more approximate schemes among the three activating fewer variables than S1.

The overall procedure starts with all the bits set to 1. The variables are continuously scanned from first to last. Only variables with the bit set to 1 are being searched. Each time a variable is searched and its value is changed, the variable remains active and also activation spreads to other related variables according to one of the activation schemes (S1, S2, or S3). On the other hand, if the value of the variable is not changed the variable becomes inactive (i.e. the bit is set to 0). The process stops under the same conditions that apply to local search without activation bits depicted in section 4.

Each time local search settles in a local minima, GLS penalizes some of the features. A limited number of variables are activated and a fresh fast local search cycle starts. Depending on the features penalized, we activate variables relating to these features such that moves examined aim at removing the penalized features from the solution. Table 2 gives the relation between features penalized and variables activated.

Feature penalized	Activate
Constraint	Variables associated to the constraint
Assignment	Variable the assignment refers to
Value	Variables assigned the value

Table 2 Associations between features penalized and variables activated

In the continue, we report results for GLS. GLS is compared with two other methods suggested for the RLFAP. Some of the results reported for GLS are the best known solutions for these instances.

8. Results

Experiments performed on the RLFAP using either one of the three activation schemes showed that all schemes perform equally well in terms of solution quality with S3 having a slight advantage in run times over scheme S2 and being much faster than scheme S1. The results reported here give the average performance of the algorithm using the activation scheme S3.

Ten runs performed on each instance starting from random initial solutions. In each run, the algorithm was allowed to complete 100K penalty cycles (i.e. GLS iterations as in Figure 2) before being stopped. Hard constraints in all instances were assigned a high violation cost equal to 1,000,000. The regularization parameter λ was also set to this value though values of λ in the range $[2\times10^5, 2\times10^6]$ also performed well. Table 3 presents the results obtained. Experiments were performed on a DEC Alpha 3000/600 with the GLS algorithm implemented in C++.

RLFAP Instance	Best Solution	Average Cost (Std. Dev.)	Worst Solution	Average Iterations	Average Time (CPU sec.)
Scen01	16	18.6 (2.3)	22	1,895	8.77
Scen02	14	14 (0.0)	14	233	0.59
Scen03	14	15.4 (1.3)	18	1,626	5.62
Scen04	46	46 (0.0)	46	60	0.46
Scen05	792	792 (0.0)	792	1,584	8.50
Scen06	3,628	4,333.8 (766.0)	6,042	34,365	120.87
Scen07	427,054	530,641.1 (79,666.7)	700,685	20,412	78.79
Scen08	294	335.7 (34.7)	377	50,626	232.88
Scen09	15,805	15,999.7 (194.7)	16,340	31,150	129.4
Scen10	31,533	31,686.6 (146.1)	31,942	64,258	297.29
Scen11	28	not applicable	Not solved	21,577	93.97

Table 3 Average performance of GLS on the RLFAP instances.

The tunneling algorithm a predecessor of GLS significantly improved the best known solutions on the RLFAP that accompanied the initial release of the instances (see [25]). GLS with fast local search found even better solutions improving over tunneling algorithm for many of the instances. Table 4 summarizes the best known solutions for RLFAP as found by GLS and its variants including tunneling algorithm. For all instances, the solutions reported in Table 4 are the best known so far. Note here that solutions in Table 4 for Scen02-Scen05 have been proved to be optimal by complete search techniques [14].

RLFAP Instance	Best known solutions found by GLS variants
Scen01	16
Scen02	14
Scen03	14
Scen04	46
Scen05	792
Scen06	3,570
Scen07	374,705
Scen08	282
Scen09	15,680
Scen10	31,517
Scen11	28

Table 4 Best known solutions for RLFAP found by GLS variants.

Recently and independently from this work another method also based on GENET has been developed for the RLFAP by G. vom Scheidt [22]. The method is described in the paper by Boyce et al. [2] where it is compared with tabu search. For convenience, we shall call this method *extended GENET*. Extended GENET has many similarities as well as differences with GENET [27, 4, 20] and GLS. Although it uses an augmented cost function, it only penalizes constraint violations by increasing penalties proportionally to the constraint violation costs. No scheme is used for distributing the search effort though a similar effect is attempted by varying penalty increments amongst constraints. Minimization of the number of different values used is achieved by incorporating an additional cost term to the cost function weighted by an appropriate coefficient. The algorithm has not been applied to instances involving mobility costs (Scen09 and Scen10) while instance Scen11 was approached as a pure satisfiability problem. Extended GENET makes use of a fast local search procedure using an activation scheme similar to S1 but does not consider sideways moves.

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[1]	28 values	0 viol.	0 viol.	80%	60%	60%	1.6min	25sec	54min

Table 5 Comparison of GLS with tabu search and extended GENET. Results for tabu search and extended GENET are from Boyce et al. [2].

Table 5 contrasts the results reported in Boyce et. al [2] for tabu search and extended GENET with those reported for GLS in Table 3. Experiments in [2] were also performed on

[1] 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.

DEC Alpha machines with the algorithms implemented in C++ and therefore a direct comparison in running times can be made. As one can see in Table 5, GLS outperforms both extended GENET and tabu search. For problems (Scen01-Scen05), GLS succeeded more times in finding the optimum than either tabu search or extended GENET. Moreover, GLS found better solutions than these two methods in all insoluble instances (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 minimize 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.

RLFAP	**Average Solution Cost**		**Percentage excess of Ext. GENET solutions over GLS solutions**
Instance	**GLS**	**Extended GENET**	
Scen06	4,333.8	5,076	17%
Scen07	530,641.1	727,458	37%
Scen08	335.7	451	34%

Table 6 GLS and extended GENET on insoluble instances. Results for Extended GENET are from [22].

Table 6 provides further evidence on the superiorioty 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 Extened GENET are from [22].

Recently, Bessiere et al. [2] applied arc-consistency algorithms to Scen3, Scen5, Scen8, and Scen11. Since only the satisfiability issue (not optimization) was addressed their results are not comparable with ours.

9. Conclusions

Guided local search is a novel approach that enables intelligent search schemes to be built that exploit information to guide a local search algorithm in a search space. Constraints on solution features are introduced and dynamically manipulated to distribute the search effort over the regions of a search space.

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 of the method. The technique generated the best solutions known so far for a set of benchmark RLFAPs, outperforming alternative schemes suggested for the problem. Given the generality and effectivess of the approach, GLS can be considered a promising optimization technique for real world applications.

Acknowledgments

We would like to thank the Centre d'Electronique l'Armament for making available the RLFAP instances. We would also like to thank Thomas Schiex for answering our queries and providing us with information on alternative approaches tried on the problem. This project and Chris Voudouris are funded by the EPSRC grant (GR/H75275).

References

1. J. J. Bentley, “Fast Algorithms for Geometric Traveling Salesman Problems”, *ORSA Journal on Computing*, Vol. 4, 387-411 (1992).
2. C. Bessiere, E. C. Freuder, and J. Regin, “Using Inference to Reduce Arc Consistency Computation”, In *Proceedings of IJCAI-95,* 592-598 (1995).
3. J.F. Boyce, C.H.D. Dimitropoulos, G. vom Scheidt, J.G. Taylor, “GENET and Tabu Search for combinatorial optimization problems”, *World Congress on Neural Networks*, Washighton D.C. (1995).
4. A. Davenport, E. Tsang, C. J. Wang, and K. Zhu, “GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement”, In *Proceedings of AAAI-94*, 325-330 (1994).
5. E. C. Freuder and R. J. Wallace, “Partial constraint satisfaction”, *Artificial Intelligence* 58, 21-70 (1992).
6. F. Glover, “Tabu search Part I”, *ORSA Journal on Computing*, Vol. 1, 190-206 (1989).
7. F. Glover, “Tabu search Part II”, *ORSA Journal on Computing*, Vol. 2, 4-32 (1990).
8. F. Glover, E. Taillard, and D. de Werra, “A user's guide to tabu search”, *Annals of Operations Research* 41, 3-28 (1993).
9. S. Haykin, *Neural Networks*, Macmillan, 1994.
10. D. Johnson, C. Aragon, L. McGeoch, and C. Schevon, “Optimization by simulated annealing: an experimental evaluation, part II, graph coloring and number partitioning”, *Operations Research* 39, 378-406 (1991).
11. B. O. Koopman, “The Theory of Search, Part III. The Optimum Distribution of Searching Effort”, *Operations Research* 5, 613-626 (1957).
12. S. Minton, M.D. Johnston, A.B. Philips, and P. Laird, “Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems”, *Artificial Intelligence* 58, 161-205 (1992).
13. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall,1982.
14. Personal communication with Thomas Schiex who originally released the RLFAP instances.
15. The RLFAP instances are available from listserver@saturne.cert.fr. To get more information and the instances send the command "get csp-list CELAR.blurb" to the above list server. The problems are also available via ftp from the Constraints Archive maintained by Michael Jampel (ftp.cs.city.ac.uk:/pub/constraints/csp-benchmarks/celar/).
16. B. Selman, H.J. Levesque, and D.G. Mitchell, “A new method for solving hard satisfiability problems”, In *Proceedings of AAAI-92*, 440-446 (1992).
17. L. D. Stone, “The Process of Search Planning: Current Approaches and Continuing Problems”, *Operations Research* 31, 207-233 (1983).
18. S. B. Thrun, “Efficient exploration in reinforcement learning”, Technical report CMU-CS-92-102, School of Computer Science, Carnegie-Mellon University (1992).

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

20. E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, 1993.

21. E. Tsang and C. Voudouris, "Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem", Technical Report CSM-246, Department of Computer Science, University of Essex (1995).

22. G. vom Scheidt, "Extension of GENET to Partial Constraint Satisfaction Problems and Constraint Satisfaction Optimization Problems", Master's Thesis, Dept. of mathematics, King's College London, UK (1995).

23. C. Voudouris and E. Tsang, "Function Optimization using Guided Local Search", Technical Report CSM-249, Department of Computer Science, University of Essex (1995).

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

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

26. R.J. Wallace and E.C. Freuder, "Conjunctive Width Heuristics for Maximal Constraint Satisfaction", In *Proceedings of AAAI-93*, 762-768 (1993).

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