Chapter 11
Guided Local Search

Christos Voudouris, Edward P.K. Tsang and Abdullah Alsheddy

Abstract Combinatorial explosion is a well-known phenomenon that prevents complete algorithms from solving many real-life combinatorial optimization problems. In many situations, heuristic search methods are needed. This chapter describes the principles of Guided Local Search (GLS) and Fast Local Search (FLS) and surveys their applications. GLS is a penalty-based metaheuristic algorithm that sits on top of other local search algorithms, with the aim to improve their efficiency and robustness. FLS is a way of reducing the size of the neighbourhood to improve the efficiency of local search. The chapter also provides guidance for implementing and using GLS and FLS. Four problems, representative of general application categories, are examined with detailed information provided on how to build a GLS-based method in each case.

11.1 Introduction

Many practical problems are NP-hard in nature, which means complete, constructive search is unlikely to satisfy our computational demand. For example, suppose we have to schedule 30 jobs on 10 machines, satisfying various production constraints. The search space has 10^{30} leaf nodes. Let us assume that we use a very

Christos Voudouris
Group Chief Technology Office, BT plc, Orion Building, mlb1/pp12, Marthlesham Heath, Ipswich, IP5 3RE, UK
e-mail: chris.voudouris@bt.com

Edward P.K. Tsang
Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK
e-mail: edward@essex.ac.uk

Abdullah Alsheddy
Department of Computer Science, Univeristy of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK
e-mail: aalshe@essex.ac.uk

M. Gendreau, J.-Y. Potvin (eds.), *Handbook of Metaheuristics*,
International Series in Operations Research & Management Science 146,
DOI 10.1007/978-1-4419-1665-5_11,

clever backtracking algorithm that explores only one in every 10^{10} leaf nodes. Let us generously assume that our implementation examines 10^{10} nodes per second (with today's hardware, even the most naïve backtracking algorithm should be expected to examine only 10^5 nodes per second). This still leaves us with approximately 300 years to solve the problem, in the worst case. Many real-life problems cannot be realistically and reliably solved by complete search. This motivates the development of local search or heuristic methods.

In this chapter, we describe GLS, a general metaheuristic algorithm and its applications. GLS sits on top of other heuristic methods with the aim to improve their efficiency or robustness. GLS has been applied to a non-trivial number of problems and found to be efficient and effective. It is relatively simple to implement and apply, with only a few parameters to tune.

The rest of this chapter will be divided into two parts: the first part describes the GLS and surveys its variants. The second part provides guidelines on how to use GLS in practical applications.

Part I: Survey of Guided Local Search

11.2 Background

Local search (LS) is the basis of most heuristic search methods. It searches in the space of candidate solutions, such as the assignment of one machine to each job in the above scheduling example. The solution representation issue is significant, though it is not the subject of our discussion here. Starting from a (possibly randomly generated) candidate solution, LS moves to a "neighbour" that is "better" than the current candidate solution according to the objective function. LS stops when all neighbours are inferior to the current solution.

LS can find good solutions very quickly. However, it can be trapped in local optima—positions in the search space that are better than all their neighbours, but not necessarily representing the best possible solution (the global optimum). To improve the effectiveness of LS, various techniques have been introduced over the years. Simulated Annealing (SA), Tabu Search (TS) and Guided Local Search (GLS) all attempt to help LS escape local optimum. This chapter focuses on GLS [81].

GLS can be seen as a generalization of techniques such as GENET [15, 78, 79, 87, 88] and the min-conflicts heuristic repair method by Minton et al. [56] developed for constraint satisfaction problems. GLS also relates to ideas from the area of Search Theory on how to distribute the search effort (e.g. see [41, 68]).

The principles of GLS can be summarized as follows. As a metaheuristic method, GLS sits on top of LS algorithms. To apply GLS, one defines a set of features for the candidate solutions. When LS is trapped in local optima, certain features are selected and penalized. LS searches the solution space using the objective function augmented by the accumulated penalties. The novelty of GLS is in the way that it selects features to penalize. GLS effectively distributes the search effort in the search space, favouring promising areas.

11.3 Guided Local Search

As mentioned earlier, GLS augments the given objective function with penalties. To apply GLS, one needs to define features for the problem. For example, in the travelling salesman problem [21], a feature could be *whether the candidate tour travels immediately from city A to city B*. GLS associates a cost and a penalty with each feature. The costs can often be defined by taking the terms and their coefficients from the objective function. For example, in the travelling salesman problem, the cost of the above feature can simply be the distance between cities A and B. The penalties are initialized to 0 and will only be increased when the local search reaches a local optimum. Given an objective function g that maps every candidate solution s to a numerical value, GLS defines a function h that will be used by LS (replacing g):

$$h(s) = g(s) + \lambda \times \sum_{i \text{ is a feature}} (p_i \times I_i(s)), \tag{11.1}$$

where s is a candidate solution, λ is a parameter of 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:

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i\text{; 0 otherwise.} \tag{11.2}$$

Sitting on top of local search algorithms, GLS helps them to escape local optima in the following way. Whenever the local search algorithm settles in a local optimum, GLS augments the cost function by adding penalties to selected features. The novelty of GLS is mainly in the way that it selects features to penalize. The intention is to penalize “unfavourable features” or features that “matter most” when a local search settles in a local optimum. A feature with high cost has more impact on the overall cost. Another factor that should be considered is the current penalty value of that feature. The utility of penalizing feature i, util_i, under a local optimum, s_*, is defined as follows:

$$\text{util}_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \tag{11.3}$$

where c_i is the cost and p_i is the current penalty value of feature i. In other words, if a feature is not exhibited in the local optimum (indicated by I_i), then the utility of penalizing it is 0. The higher the cost of this feature (the greater c_i), the greater the utility of penalizing it. Besides, the larger the number of times it has been penalized (the greater p_i), the lower the utility of penalizing it again. In a local optimum, the feature with the greatest util value will be penalized. When a feature is penalized, its penalty value is always increased by 1. The scaling of the penalty is adjusted by λ.

By taking the cost and current penalty into consideration in selecting the feature to penalize, GLS focuses its search effort on more promising areas of the search space: areas that contain candidate solutions that exhibit "good features", i.e. features involving lower cost. On the other hand, penalties help to prevent the search from directing all effort to any particular region of the search space.

Naturally the choice of the features, their costs and the setting of λ may affect the efficiency of a search. Experience shows that the features and their costs normally come directly from the objective function. In many problems, the performance of GLS is not too sensitive to the value λ. It means that not too much effort is required to apply GLS to a new problem. In certain problems, one needs expertise in selecting the features and the λ parameter. Research aiming to reduce the sensitivity of the λ parameter in such cases is reported in [55].

11.4 Implementing Guided Local Search

A local search procedure for the particular problem is required for the algorithm to be implemented. Guided Local Search is repeatedly using this procedure to optimize the augmented objective function of the problem. The augmented objective function is modified each time a local minimum is reached by increasing the penalties of one or more of the features present in the local minimum. These features are selected by using the utility function (11.3). The pseudo-code for implementing a Guided Local Search method is presented and explained in Section 11.4.1.

11.4.1 Pseudo-code for Guided Local Search

The pseudo-code for the Guided Local Search procedure is the following:

```
procedure GuidedLocalSeach(p, g, λ, [I_1,...,I_M], [c_1,...,c_M], M)
begin
    k ← 0;
    s_0 ← ConstructionMethod(p);
    /* set all penalties to 0 */
    for i ← 1 until  M do
        p_i ← 0;
    /* define the augmented objective function */
    h ← g + λ * ∑ p_i * I_i;
    while StoppingCriterion do
    begin
        s_{k+1} ← ImprovementMethod(s_k, h);
        /* compute the utility of features */
        for i ← 1 until M do
```

```
            util_i ← I_i(s_{k+1}) * c_i/(1 + p_i);
        /* penalize features with maximum utility */
        for each i such that util_i is maximum do
            p_i ← p_i + 1;
        k ← k + 1;
    end
    s* ← best solution found with respect to objective function g;
    return s*;
end
```

where p is the problem, g is the objective function, h is the augmented objective function, λ is a parameter, I_i is the indicator function of feature i, c_i is the cost of feature i, M is the number of features, p_i is the penalty of feature i, $ConstructionMethod(p)$ is the method for constructing an initial solution for problem p and $ImprovementMethod(s_k, h)$ is the method for improving solution s_k according to the augmented objective function h.

11.4.2 Guidelines for Implementing the GLS Pseudo-code

To understand the pseudo-code, let us first explain the methods for constructing and improving a solution, as they are both prerequisites for building a GLS algorithm.

11.4.2.1 Construction Method

As with other metaheuristics, GLS requires a construction method to generate an initial (starting) solution for the problem. In the pseudo-code, this is denoted by $ConstructionMethod$. This method can generate a random solution or a heuristic solution based on some known technique for constructing solutions for the particular problem. GLS is not very sensitive to the starting solution given that sufficient time is allocated to the algorithm to explore the search space of the problem.

11.4.2.2 Improvement Method

A method for improving the solution is also required. In the pseudo-code, this is denoted by $ImprovementMethod$. This method can be a simple local search algorithm or a more sophisticated one such as Variable neighbourhood Search [30], Variable Depth Search [50], Ejection Chains [25] or combinations of local search methods with exact search algorithms [60].

It is not essential for the improvement method to generate high-quality local minima. Experiments with GLS and various local heuristics reported in [85] have shown that high-quality local minima take time to produce, resulting in less intervention by GLS in the overall allocated search time. This may sometimes lead to inferior results compared to a simple but more computationally efficient improvement method.

Note also that the improvement method is using the augmented objective function instead of the original one.

11.4.2.3 Indicator Functions and Feature Penalization

Given that a construction and an improvement method are available for the problem, the rest of the pseudo-code is straightforward to apply. The penalties of features are initialized to zero and they are incremented for features that maximize the utility formula, after each call to the improvement method.

The indicator functions I_i for the features rarely need to be implemented. Looking at the values of the decision variables can directly identify the features present in a local minimum. When this is not possible, data structures with constant time deletion/addition operations (e.g. based on double-linked lists) can incrementally maintain the set of features present in the working solution, thus avoiding the need for an expensive computation when GLS reaches a local minimum.

The selection of features to penalize can be efficiently implemented by using the same loop for computing the utility formula for features present in the local minimum (the other features can be ignored) and also placing features with maximum utility in a vector. With a second loop, the features with maximum utility contained in this vector have their penalties incremented by one.

11.4.2.4 Parameter λ

Parameter λ is the only parameter of the GLS method (at least in its basic version) and in general is instance dependent. Fortunately, for several problems, it has been observed that good values for λ can be found by dividing the value of the objective function of a local minimum with the number of features present in it. In these problems, λ is dynamically computed after the first local minimum and before penalties are applied to features for the first time. The user only provides parameter α, which is relatively instance independent. The recommended formula for λ as a function of α is the following:

$$\lambda = \alpha * g(x_*)/(\text{no. of features present in } x_*), \tag{11.4}$$

where g is the objective function of the problem and x_* is a local minimum. Tuning α can result in λ values, which work for many instances of a problem class.

Another benefit from using α is that, once tuned, it can be fixed in industrialized versions of the software, resulting in ready-to-use GLS algorithms for the end user.

11.4.2.5 Augmented Objective Function and Move Evaluations

With regard to the objective function and the augmented objective function, the program should keep track of the actual objective value in all operations related to storing the best solution or finding a new best solution. Keeping track of the value of the augmented objective value (e.g. after adding the penalties) is not necessary since local search methods will be looking only at the differences in the augmented objective value when evaluating moves.

However, the move evaluation mechanism needs to be revised to work efficiently with the augmented objective function. Normally, the move evaluation mechanism is not directly evaluating the objective value of the new solution generated by the move. Instead, it calculates the difference Δg in the objective function. This difference should be combined with the difference in the amount of penalties. This can be easily done and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only features whose state changes (being deleted or added). The penalties of the features deleted are summed together. The same is done for the penalties of added features. The change in the amount of penalties due to the move is then simply given by the difference:

$$\sum_{\text{over all features } j \text{ added}} p_j - \sum_{\text{over all features } k \text{ deleted}} p_k, \tag{11.5}$$

which then has to be multiplied by λ and added to Δg.

Another minor improvement is to monitor the actual objective value not only for the solutions accepted by the local search but also for those evaluated. Since local search is using the augmented objective function, a move that generates a new best solution may be missed. From our experience, this modification does not improve significantly the performance of the algorithm although it can be useful when GLS is used to find new best known solutions to hard benchmark instances.

11.4.2.6 Stopping Criterion

There are many choices possible for the *StoppingCritetion*. Since GLS is not trapped in local minima, it is not clear when to stop the algorithm. Like other metaheuristics, we usually resort to a measure related to the length of the search process. For example, we may choose to set a limit on the number of moves performed, the number of moves evaluated or the CPU time spent by the algorithm. If a lower bound is known, we can utilize it as a stopping criterion by setting the gap to be achieved between the best known solution and the lower bound. Criteria can also be combined to allow for a more flexible way to stop the GLS method.

In the next section, we look at the combination of Guided Local Search with Fast Local Search, a generalized algorithm for speeding up local search, resulting in the Guided Fast Local Search method. Guided Fast Local Search addresses the issue of slow local search procedures and it is particularly useful when applying GLS to tackle large-scale problem instances.

11.5 Guided Fast Local Search

One factor which affects the efficiency of a local search algorithm is the size of the neighbourhood. If too many neighbours are considered, then the search could be very costly. This is especially true if the search takes many steps to reach a local optimum and/or each evaluation of the objective function requires a significant amount of computation. Bentley presented in [5] the *approximate 2-Opt* method to reduce the neighbourhood of 2-Opt in the TSP. We have generalized this method to a method called *Fast Local Search* (FLS). The principle is to use heuristics to identify (and ignore) neighbours that are unlikely to lead to improving moves in order to enhance the efficiency of a search.

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

Initially, all sub-neighbourhoods are active. If a sub-neighbourhood 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-neighbourhoods are also activated. In particular, we activate all the sub-neighbourhoods 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-neighbourhoods being active until all the sub-neighbourhood bits turn to 0. The solution formed up to that point is returned as an approximate local optimum.

The overall procedure could be 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 settling down, examining fewer moves and saving enormous amounts of time which would otherwise be spent on examining predominantly bad moves.

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

By reducing the size of the neighbourhood, one may significantly reduce the amount of computation involved in each local search iteration. The idea is to enable more local search iterations 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 in “search speed” outweighs the loss in “search quality”.

11.6 Implementing Guided Fast Local Search

Guided Fast Local Search (GFLS) is more sophisticated than the basic GLS algorithm as it uses a number of sub-neighbourhoods, which are enabled/disabled during the search process. The main advantage of GFLS lies in its ability to focus the search after the penalties of features are increased. This can dramatically shorten the time required by an improvement method to re-optimize the solution each time the augmented objective function is modified.

In the following sections, we provide the pseudo-code for the method and also some suggestions on how to achieve an efficient implementation. We first look at the pseudo-code for Fast Local Search, which is part of the overall Guided Fast Local Search algorithm.

11.6.1 Pseudo-code for Fast Local Search

The pseudo-code for Fast Local Search is the following:

procedure FastLocalSeach(s, h, $[bit_1,\ldots,bit_L]$, L)
begin
    while $\exists bit$, $bit = 1$ **do**
    /* i.e. while active sub-neighbourhood exists */
        for $i \leftarrow 1$ until L **do**
        begin
            if $bit_i = 1$ **then**
            /* search sub-neighbourhood i */
            begin
                Moves $\leftarrow MovesForSubneighbourhood(i)$;
                for each move m in Moves **do**
                begin
                    $s' \leftarrow m(s)$;
                    /* s' is the result of move m */
                    if $h(s') < h(s)$ **then**
                    /* minimization case is assumed here */
                    begin
                        /* spread activation */

```
                        ActivateSet ←
                        SubneighbourhoodsForMove(m);
                        for each sub-neighbourhood j in
                        ActivateSet do
                            bit_j ← 1;
                        s ← s';
                        goto ImprovingMoveFound
                    end
                end
                bit_i ← 0; /* no improving move found */
            end
ImprovingMoveFound:
            continue;
        end
    return s;
end
```

where s is the solution, h is the augmented objective function, L is the number of sub-neighbourhoods, bit_i is the activation bit for sub-neighbourhood i, $MovesForSubneighbourhood(i)$ is the method which returns the set of moves contained in sub-neighbourhood i and $SubneighbourhoodsForMove(m)$ is the method which returns the set of sub-neighbourhoods to activate when move m is performed.

11.6.2 Guidelines for Implementing the FLS Pseudo-code

As explained in Section 11.5, the problem's neighbourhood is broken down into a number of sub-neighbourhoods and an activation bit is attached to each one of them. The idea is to examine sub-neighbourhoods in a given order, searching only those with the activation bit set to 1. The neighbourhood search process does not restart whenever we find a better solution but it continues with the next sub-neighbourhood in the given order. The pseudo-code given above is flexible since it does not specify which bits are initially switched on or off, something which is an input to the procedure. This allows the procedure to be focused to certain sub-neighbourhoods and not the whole neighbourhood, which may be a large one.

The procedure has two points that need to be customized. The first is the breaking down of the neighbourhood into sub-neighbourhoods (*MovesForSubneighborhood* method in pseudo-code). The second is the mapping from moves to sub-neighbourhoods for spreading activation (*SubneighbourhoodsForMove* method in pseudo-code). Both points are strongly dependent on the move operator used.

In general, the move operator depends on the solution representation. Fortunately, several problems share the same solution representation which is typically based on some well-known simple or composite combinatorial structure (e.g. selection, permutation, partition, composition, path, cyclic path, tree and graph). This allows us to use the same move operators for many different problems (e.g. 1-Opt, 2-Opt, Swaps and Insertions).

11.6.2.1 Breaking Down the Neighbourhood into Sub-neighbourhoods

The method for mapping sub-neighbourhoods to moves, which is denoted in the pseudo-code by *SubneighbourhoodToMoves*, can be defined by looking at the implementation of a typical local search procedure for the problem. This implementation, at its core, will usually contain a number of nested for-loops for generating all possible move combinations. The variable in the outer-most loop in the move generation code can be used to define the sub-neighbourhoods. The moves in each sub-neighbourhood will be those generated by the inner loops for the particular sub-neighbourhood index value at the outer-most loop.

In general, the sub-neighbourhoods can be overlapping. Fast local search is usually examining a limited number of moves compared to exhaustive neighbourhood search methods and therefore duplication of moves is not a problem. Moreover, this can be desirable sometimes to give a greater range to each sub-neighbourhood. Since not all sub-neighbourhoods are active in the same iteration, if there is no overlapping, some improving moves may be missed.

11.6.2.2 Spreading Activation When Moves Are Executed

The method for spreading activation, denoted by *SubneighbourhoodsForMove*, returns a set of sub-neighbourhoods to activate after a move is performed. The lower bound for this set is the sub-neighbourhood where the move originated. The upper bound (although not useful) is all the sub-neighbourhoods in the problem.

A way to define this method is to look at the particular move operator used. Moves will affect part of the solution directly or indirectly while leaving other parts unaffected. If a sub-neighbourhood contains affected parts then it needs to be activated since an opportunity could arise there for an improving move as a result of the original move performed.

The Fast Local Search loop is settling down in a local minimum when all the bits of sub-neighbourhoods turn to zero (i.e. no improving move can be found in any of the sub-neighbourhoods). Fast Local Search in that respect is similar to other local search procedures. The main differences are that the method can be focused to search particular parts of the overall neighbourhood and second, it is working in an opportunistic way looking at parts of the solution which are likely to contain improving moves rather than the whole solution. In the next section, we look at Guided Fast Local Search, which uses Fast Local Search as its improvement method.

11.6.3 Pseudo-code for Guided Fast Local Search

The pseudo-code for Guided Fast Local Search is given below:

```
procedure GuidedFastLocalSearch(p, g, λ, [I_1,...,I_M], [c_1,...,c_M], M, L)
begin
    k ← 0;
    s_0 ← ConstructionMethod(p);
    /* set all penalties to 0 */
    for i ← 1 until M do
        p_i ← 0;
    /* set all sub-neighbourhoods to the active state */
    for i ← 1 until L do
        bit_i ← 1;
    /* define the augmented objective function */
    h ← g + λ * ∑ p_i * I_i;
    while StoppingCriterion do
    begin
        s_{k+1} ← FastLocalSearch(s_k, h, [bit_1,...,bit_L], L);
        /* compute the utility of features */
        for i ← 1 until M do
            util_i ← I_i(s_{k+1}) * c_i/(1 + p_i);
        /* penalize features with maximum utility */
        for each i such that util_i is maximum do
        begin
            p_i ← p_i + 1;
            /* activate sub-neighbourhoods related
            to penalized feature i */
            ActivateSet ← SubneighbourhoodsForFeature(i);
            for each sub-neighbourhood j in ActivateSet do
                bit_j ← 1;
        end
        k ← k + 1;
    end
    s* ← best solution found with respect to objective function g;
    return s*;
end
```

where $FastLocalSearch(s_k, h, [bit_1, \ldots, bit_L], L)$ is the fast local search method as described in Section 11.6.1, $SubneighbourhoodsForFeature(i)$ is the method which returns the set of sub-neighbourhoods to activate when feature i is penalized, and the rest of the definitions are the same than those used in the pseudo-code for GLS described in Section 11.4.1.

11.6.4 Guidelines for Implementing the GFLS Pseudo-code

This pseudo-code is similar to that of Guided Local Search explained in Section 11.4. All differences relate to the manipulation of activation bits for the purpose of focusing Fast Local Search. These bits are initialized to 1. As a result, the first call to Fast Local Search is examining the whole neighbourhood for improving moves.

Subsequent calls to Fast Local Search examine only part of the neighbourhood and in particular all the sub-neighbourhoods that relate to the features penalized by GLS.

11.6.4.1 Identifying Sub-neighbourhoods to Activate When Features Are Penalized

Identifying the sub-neighbourhoods that are related to a penalized feature is the task of *SubneighbourhoodsForFeature* method. The role of this method is similar to that of *SubneighbourhoodsForMove* method in Fast Local Search (see Section 11.6.2.2).

The *SubneighbourhoodsForFeature* method is usually defined based on an analysis of the move operator. After the application of penalties, we are looking for moves which remove or have the potential to remove penalized features from the solution. The sub-neighbourhoods, which contain such moves, are prime candidates for activation. Specific examples will be given later in the chapter and in the context of GLS applications.

Guided Fast Local Search is much faster than basic Guided Local Search especially in large problem instances when repeatedly and exhaustively searching the whole neighbourhood is computationally expensive.

11.7 GLS and Other Metaheuristics

GLS is closely related to other heuristic and metaheuristic methods. In this section, we shall discuss the relationship between GLS and Tabu Search (TS) and GLS and Genetic Algorithms (GA) and also review the different hybrids and extensions of GLS and FLS that have been developed in recent years.

11.7.1 GLS and Tabu Search

GLS is closely related to Tabu Search (TS). For example, penalties in GLS can be seen as soft taboos in TS that guide LS away from local minima. There are many ways to adopt TS ideas in GLS. For example, taboo lists and aspiration ideas have been used in later versions of GLS. Penalties augment the original objective function. They help the local search to escape local optima. However, if too many penalties are built up during the search, the local search could be misguided. Resembling the tabu lists idea, a limited number of penalties are used when GLS is applied to the radio link frequency assignment problem [58]. When the list is full, old penalties are overwritten [83].

In another GLS work, aspiration (inspired by TS) is used to favour promising moves [55].

11.7.2 GLS and Genetic Algorithms

As a metaheuristic method, GLS can also sit on top of genetic algorithms (GA) [27, 33]. This has been demonstrated in Guided Genetic Algorithm (GGA) [44–47].

GGA is a hybrid of GA and GLS. It is designed to extend the domain of both GA and GLS. One major objective is to further improve the robustness of GLS. It can be seen as a GA with GLS to bring it out of local optima: if no progress has been made after a specific number of iterations (this number is a parameter of GGA), GLS modifies the fitness function (which is the objective function) by means of penalties, using the criteria defined in Equation (11.3). 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 contribute more to the penalties are more likely to changed 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 running in parallel from different starting points and exchanging material in a GA manner. The difference is that only one set of penalties is used in GGA whereas parallel GLSs could have used one independent set of penalties per run. Besides, 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.

11.7.3 GLS Hybrids

Being simple and general, GLS ideas can easily be combined with other techniques. GLS has been hybridized with several metaheuristics creating efficient frameworks which were successfully applied to several applications. Below, we review and comment on some of these hybrids of GLS.

GLS was hybridized with two major Evolutionary Computation (EC) techniques, namely Estimate Distribution Algorithm (EDA) and Evolution Strategy (ES). The hybrid of GLS with EDA was introduced by Zhang et al. [92]. They proposed a framework that incorporates GLS within EDA, in which GLS is applied to each solution in the population of EDA. The framework is successfully applied to the

Quadratic Assignment Problem. The results show the superiority of EDA/GLS over GLS alone.

The hybrid of GLS with ES was first studied by Mester and Braysy [52]. The resulting framework combines GLS and ES into an iterative two-stage procedure. GLS is used in both phases to improve the local search in the first stage and to regulate the objective function and the neighbourhood of the modified ES in the second stage. The principle of FLS is also incorporated into the idea of Penalty Variable Neighbourhood in which the neighbourhood considered by the local search is limited to a small set of the neighbours of the penalized feature.

GLS has also been hybridized with Variable Neighbourhood Search (VNS) and Large Neighbourhood Search (LNS). Kytojoki et al. [42] combine GLS with VNS in an efficient variable neighbourhood search heuristic, named Guided VNS (GVNS), which was applied to the vehicle routing problem. The addition to VNS is the use of GLS to escape local minima. The idea of threshold value borrowed from Threshold Accepting (TA) is used as a termination condition for every GLS stage. The hybrid of GLS with LNS is introduced in [89]. In the proposed framework, LNS is applied when the GLS cannot escape a local optimum after a number of penalizations, with the aim of increasing the diversity and exploring more promising parts of the search space. The effectiveness of this hybrid was demonstrated through high-quality results obtained in a planning optimization problem.

Guided Tabu Search (GTS) is a hybrid metaheuristic which combines GLS with TS proposed by Tarantilis et al. [73, 74] to solve the vehicle routing problem with heterogeneous fleet, and then extended to solve another variant of the same general problem. The basic idea is to control the exploration of TS by a guiding mechanism, based on GLS, that continuously modifies the objective function of the problem. The authors propose a new arc (as a feature) selection strategy which consider the relative arc length according to the rest of customers ($d_{i,j}/\mathrm{avg}_{i,j}$ rather than $d_{i,j}$, where $\mathrm{avg}_{i,j}$ is the average value of all outgoing arcs from i and j). They argue that this would lead to a more balanced arc selection, which should improve upon the most frequently employed strategy based on $d_{i,j}$ only. Experimental results confirm the effectiveness of GTS, producing new best results for several benchmarks. De Backer et al. [3] also proposed a Guided Tabu Search hybrid in their work on the VRP.

GLS has been also successfully hybridized with Ant Colony Optimization (ACO) by Hani et al. [29]. This hybrid algorithm was applied to the facility layout problem, a variant of the Quadratic Assignment Problem (QAP). The basic idea is simple: GLS sits on top of the basic LS in the ACO.

The hybridization of GLS and Constraint Programming (CP) was introduced by Gomes et al. [28]. This method, named Guided Constraint Search, borrows ideas from GLS to improve the efficiency of pure CP methods. The basic principle is to use a fitness function to choose at each iteration only the N most promising values of each variable's domain, defining a sub-space for the CP method. The selection strategy is inspired from GLS; for each pair, a utility function, penalty parameter and cost are defined. At each iteration, those features (variable/value pairs) which were considered but did not belong to a new best solution are deemed bad features and are penalized.

11.7.4 Variations and Extensions

The success of GLS motivated researchers to invent new algorithms inspired from GLS, borrowing the ideas of features, penalties and utilities. Below, we briefly describe such GLS-inspired algorithms.

Partially based on GLS, which is a centralized algorithm, Basharu et al. [4] introduced an improved version for solving distributed constraint satisfaction problems. The Distributed Guided Local Search (Dis-GLS) incorporates additional heuristics to enhance its efficiency in distributed scenarios. The algorithm has been successfully applied to the distributed version of the Graph Colouring problem producing promising results compared to other distributed search algorithms.

Hifi et al. [32] introduced a variant of GLS by proposing a new penalization strategy. The principle is to distinguish two phases in the search process, namely the penalty and normal phases. The search process switches between the two phases in order to either escape local optima or diversify the search to explore another feasible space. The computational results confirm the high quality of solutions obtained by the proposed variant.

Tamura et al. [72] propose an improved version of GLS, named the Objective function Adjustment (OA) algorithm which incorporates the idea of features (from GLS) alongside the concept of energy function.

Part II: Applications of Guided Local Search

11.8 Overview of Applications

GLS and its descendents have been applied to a number of non-trivial problems and have achieved state-of-the-art results.

11.8.1 Radio Link Frequency Assignment Problem

In the *radio link frequency assignment problem* (RLFAP), the task is to assign available frequencies to communication channels satisfying constraints that prevent interference [7]. In some RLFAPs, the goal is to minimize the number of frequencies used. Bouju et al. [7] is an early work that applied GENET to radio length frequency assignment. For the CALMA set of benchmark problems, which have been widely used, GLS+FLS reported the best results compared to all work published previously [84]. In the NATO Symposium on RLFAP in Denmark, 1998, GGA was shown to improve the robustness of GLS [46]. In the same symposium, new and significantly improved results by GLS were reported [83]. At the time, GLS and GGA held some of the best known results in the CALMA set of benchmark problems.

11.8.2 Workforce Scheduling Problem

In the *workforce scheduling problem* (WSP) [2], 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 service times at which 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 WSP, GLS+FLS holds the best published results for the benchmark problem available to the authors [77].

11.8.3 Travelling Salesman Problem

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 specialized algorithm for the TSP that has long been perceived as the champion of this problem [50, 51]. We tested GLS+FLS+2Opt against LK [85] on a set of benchmark problems from a public TSP library [61]. Given the same amount of time, GLS+FLS+2Opt found better results than LK on average. GLS+FLS+2Opt also out-performed Simulated Annealing [36], Tabu Search [40] and Genetic Algorithm [23] implementations for the TSP. One must be cautious when interpreting such empirical results as they could be affected by many factors, including implementation details. But given that the TSP is an extensively studied problem, it takes something special for an algorithm to outperform the champions under any reasonable measure ("find the best results within a given amount of time" must be a realistic requirement). It must be emphasized that LK is specialized for the TSP but GLS and FLS are much simpler general-purpose algorithms.

GLS hybrids have also been proposed for the TSP including the combination of GLS with Memetic Algorithms [34] and also with the dynamic-programming based Dynasearch technique with encouraging preliminary results reported in [12].

Padron and Balaguer [59] have applied GLS to the related Rural Postman Problem (RPP), Vansteenwegen et al. [80] applied GLS to the related Team Orienteering Problem (TOP) and Mester et al. [53] applied the Guided Evolution Strategy hybrid metaheuristic to a genetic ordering problem (a Unidimensional Wandering Salesperson Problem, UWSP).

11.8.4 Function Optimization

GLS has been applied to general function optimization problems to illustrate that artificial features can be defined for problems in which the objective function suggests no obvious features. As expected, the results show that 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-optima [82].

11.8.5 Satisfiability and Max-SAT Problem

Given a set of propositions in conjunctive normal form, the Satisfiability (SAT) problem is to determine whether the propositions can all be satisfied. The MAX-SAT problem is a SAT problem in which each clause is given a weight. The task is to minimize the total weight of the violated clauses. In other words, the weighted MAX-SAT problem is an optimization problem. Many researchers believe that many problems, including scheduling and planning can be formulated as SAT and MAX-SAT problems, hence these problems have received significant attention in recent years, e.g. see Gent et al. [24].

GLSSAT, an extension of GLS, was applied to both the SAT and the weighted MAX-SAT problems [54]. On a set of SAT problems from DIMACS, GLSSAT produced more frequently better or comparable solutions than those produced by WalkSAT [64], a variation of GSAT [65], which was specifically designed for the SAT problem.

On a popular set of benchmark weighted MAX-SAT problems, GLSSAT produced better or comparable solutions, more frequently than state-of-the-art algorithms, such as DLM [66], WalkSAT [64] and GRASP [63].

11.8.6 Generalized Assignment Problem

The Generalized Assignment Problem is a generic scheduling problem in which the task is to assign agents to jobs. Each job can only be handled by one agent, and each agent has a finite resource capacity that limits the number of jobs that it can be assigned to. Assigning different agents to different jobs bear different utilities. On the other hand, different agents will consume different amounts of resources when doing the same job. In a set of benchmark problems, GGA found results as good as those produced by a state-of-the-art algorithm (which was also a GA algorithm) by Chu and Beasley [11], with improved robustness [47].

GLS hybrids have been proposed for the related QAP. Zhang et al. [92] proposed the GLS/EDA hybrid metaheuristic. In addition, the hybrid of GLS with ACO (ACO_GLS) has been applied to a variation of the QAP [29].

11.8.7 Processor Configuration Problem

In the Processor Configuration Problem, one is given a set of processors, each of which with a fixed number of connections. In connecting the processors, one objective is to minimize the maximum distance between processors. Another possible objective is to minimize the average distance between pairs of processors [9]. In applying GGA to the Processor Configuration Problem, representation was a key issue. To avoid generating illegal configurations, only mutation is used. GGA found configurations with shorter average communication distance than those found by other previously reported algorithms [45, 46].

11.8.8 Vehicle Routing Problem

In a vehicle routing problem, one is given a set of vehicles, each with its specific capacity and availability, and a set of customers to serve, each with specific weight and/or time demand on the vehicles. The vehicles are grouped at one or more depots. Both the depots and the customers are geographically distributed. The task is to serve the customers using the vehicles, satisfying time and capacity constraints. This is a practical problem which, like many practical problems, is NP-hard.

Kilby et al. applied GLS to vehicle routing problems and achieved outstanding results [38, 39]. As a result, their work was incorporated in Dispatcher, a commercial package developed by ILOG [3].

Recently, the application of GLS and its hybrids to the VRP have been considerably extended to several variants of the problem. GLS has been applied to the vehicle routing problem with backhauls and time windows [93], and to the capacitated arc routing problem [6]. Guided Tabu Search has been applied to the VRP with time window [73, 74] and also extended to other variants of the VRP, namely the VRP with two-dimensional loading constraints [90], the VRP with simultaneous pick up and delivery [91] and the VRP with Replenishment Facility [74]. GLS with VNS [42], as well as GLS with ES [52] hybrids, has been proposed to solve large-scale VRPs.

11.8.9 Constrained Logic Programming

Lee and Tam [48] and Stuckey and Tam [69] embedded GENET in logic programming languages in order to enhance programming efficiency. In these logic programming implementations, unification is replaced by constraint satisfaction [76]. This enhances efficiency and extends applicability of logic programming. Hoos and Tsang [35] provide a good overview of local search in constraint programming.

11.8.10 Other Applications of GENET and GLS

We have experimented with GLS and FLS on a variety of other problems, including the Maximum Channel Assignment problem, a Bandwidth Packing problem variant, graph colouring and the car sequencing problem. Some of these works are available for download over the Internet from Essex university's website [26] but are largely undocumented due to lack of time during the original development phase of the algorithm.

GLS and FLS have been successfully applied to the three-dimensional Bin Packing Problem and its variants [18, 19, 43], VLSI design problems [20] and network planning problems [22, 89]. GLS has been applied to the natural language parsing problem [14], Graph Set T-colourings Problem [10], query reformulation [57]. Variations of GLS have been applied to graph colouring [4] and the Multidimensional Knapsack problem [32]. Other applications of GENET include rail traffic control [37].

GLS and FLS have been incorporated into new software packages, namely iOpt which is a software toolkit for heuristic search methods [86] and iSchedule [17], which is an extension of iOpt for planning and scheduling applications (e.g. for solving problems such as the VRP [16]).

11.9 Useful Features for Common Applications

Applying Guided Local Search or Guided Fast Local Search to a problem requires identifying a *suitable* set of features to guide the search process. As explained in Section 11.3, features need to be defined in the form of indicator functions that, given a solution, return 1 if the feature is present in the solution or 0 otherwise.

Features provide the heuristic search expert with quite a powerful tool since any solution property can be potentially captured and used to guide local search. Usually, we are looking for solution properties, which have a direct impact on the objective function. These can be modelled as features with feature costs equal or analogous to their contribution to the objective function value. By applying penalties to features, GLS can guide the improvement method to avoid costly ("bad") properties, converging faster towards areas of the search space, which are of high quality.

Features are not necessarily specific to a particular problem and they can be used in several problems of similar structure. Real-world problems, which sometimes incorporate elements from several academic problems, can benefit from using more than one feature set to guide the local search in better optimizing the different terms of a complex objective function.

Below, we provide examples of features that can be deployed in the context of various problems. The reader may find them helpful and use them in his/her own optimization application.

11.9.1 Routing/Scheduling Problems

In routing/scheduling problems, one is seeking to minimize the time required by a vehicle to travel between customers or for a resource to be set-up from one activity to the next. Problems in this category include the Travelling Salesman Problem, Vehicle Routing Problem and Machine Scheduling with Sequence Dependent Set-up Times.

Travel or set-up times are modelled as edges in a path or graph structure commonly used to represent the solution of these problems. The objective function (or at least part of it) is given by the sum of lengths for the edges used in the solution.

Edges are ideal GLS features. A solution contains either an edge or not. Furthermore, each edge has a cost equal to its length. We can define a feature for each possible edge and assign a cost to it equal to the edge length. GLS quickly identifies and penalizes long and costly edges guiding local search to high-quality solutions, which use as much as possible the short edges available.

11.9.2 Assignment Problems

In assignment problems, a set of items has to be assigned to another set of items (e.g. airplanes to flights, locations to facilities people to work). Each assignment of item i to item j usually carries a cost and depending on the problem, a number of constraints are required to be satisfied (e.g. capacity or compatibility constraints). The assignment of item i to item j can be seen as a solution property which is either present in the solution or not. Since each assignment also carries a cost, this is another good example of a feature to be used in a GLS implementation.

In some variations of the problem such as the Quadratic Assignment Problem, the cost function is more complicated and assignments have an indirect impact on the cost. Even in these cases, we found that GLS can generate good results by assigning the same feature costs to all features (e.g. equal to 1 or some other arbitrary value). Although, GLS is not guiding the improvement method to good solutions (since this information is difficult to extract from the objective function), it can still diversify the search because of the penalty memory incorporated and it is capable of producing results comparable to popular heuristic methods.

11.9.3 Resource Allocation Problems

Assignment problems can be used to model resource allocation applications. A special but important case in resource allocation is when the resources available are not sufficient to service all requests. Usually, the objective function will contain a sum of costs for the unallocated requests, which is to be minimized. The cost incurred when a request is unallocated will reflect the importance of the request or the revenue lost in the particular scenario.

A possible feature to consider for these problems is whether a request is unallocated or not. If the request is unallocated then a cost is incurred in the objective function, which we can use as the feature cost to guide local search. The number of features in a problem is equal to the number of requests that may be left unallocated, one for each request. There may be hard constraints which state that certain requests should always be allocated a resource, in which case there is no need to define a feature for them. Problems in this category include the Path Assignment Problem [1], Maximum Channel Assignment Problem [67] and Workforce Scheduling Problem [2].

11.9.4 Constrained Optimization Problems

Constraints are very important in capturing processes and systems in the real world. A number of combinatorial optimization problems deals with finding a solution, which satisfies a set of constraints or, if that is not possible, minimizes the number of constraint violations (relaxations). Constraint violations may have costs (weights) associated with them, in which case the sum of constraint violation costs is to be minimized.

Local search usually considers the number of constraint violations (or their weighted sum) as the objective function even in cases where the goal is to find a solution which satisfies all the constraints. Constraints by their nature can be easily used as features. They can be modelled by indicator functions and they also incur a cost (i.e. when violated/relaxed), which can be used as their feature cost. Problems which can benefit from this modelling include the Constraint Satisfaction and Partial Constraint Satisfaction Problem, the famous SAT and its MAX-SAT variant, Graph Colouring and various Frequency Assignment Problems [58].

The features exposed in the past sections will be used in the following case problems. In particular, we examine the application of GLS to the following problems:

- Travelling Salesman Problem (Routing/Scheduling category),
- Quadratic Assignment Problem (Assignment Problem category),
- Workforce Scheduling Problem (Resource Allocation category),
- Radio Link Frequency Assignment Problem (Constrained Optimization category).

For each case problem, we provide a short problem description along with guidelines on how to build a basic local search procedure, implement GLS and also GFLS when applicable.

11.10 Travelling Salesman Problem (TSP)

11.10.1 Problem Description

There are many variations of the Travelling Salesman Problem (TSP). Here, we examine the classic symmetric TSP. The problem is defined by N cities and a symmetric distance matrix $D = [d_{ij}]$ which gives the distance between any two cities

i and j. The goal is to find a tour (i.e. closed path), which visits each city exactly once and is of minimum length. A tour can be represented as a cyclic permutation π on the N cities if we interpret $\pi(i)$ to be the city visited after city i, $i = 1,\ldots,N$. The cost of a permutation is defined as

$$g(\pi) = \sum_{i=1}^{N} d_{i\pi(i)} \tag{11.6}$$

and gives the cost function of the TSP.

11.10.2 Local Search

11.10.2.1 Solution Representation

The solution representation usually adopted for the TSP is that of a vector which contains the order of the cities in the tour. For example, the ith element of the vector will contain an identifier for the ith city to be visited. Since the solution of the TSP is a closed path there is an edge implied from the last city in the vector to the first one in order to close the tour. The solution space of the problem is made of all possible permutations of the cities as represented by the vector.

11.10.2.2 Construction Method

A simple construction method is to generate a random tour. If the above solution representation is adopted then all that is required is a simple procedure, which generates a random permutation of the identifiers of the cities. More advanced TSP heuristics can be used if we require a higher quality starting solution to be generated [62]. This is useful in real-time/online applications where a good tour may be needed very early in the search process in case the user interrupts the algorithm. If there are no such concerns, then a random tour generator suffices since the GLS metaheuristic tends to be relatively insensitive to the starting solution and capable of finding high-quality solutions even if it runs for a relatively short time.

11.10.2.3 Improvement Method

Most improvement methods for the TSP are based on the k-Opt moves. Using k-Opt moves, neighbouring solutions can be obtained by deleting k edges from the current tour and reconnecting the resulting paths using k new edges. The k-Opt moves are the basis of the three most famous local search heuristics for the TSP, namely *2-Opt* [13], *3-Opt* [49] and *Lin–Kernighan (LK)* [50].

The reader can consider using the simple 2-Opt method, which in addition to its simplicity is very effective when combined with GLS. With 2-Opt, a neighbouring solution is obtained from the current solution by deleting two edges, reversing one of the resulting paths and reconnecting the tour. In practical terms, this means reversing the order of the cities in a contiguous section of the vector or its remainder depending on which one is the shortest in length.

Computing incrementally the change in solution cost by a 2-Opt move is relatively simple. Let us assume that edges $e1$ and $e2$ are removed and edges $e3$ and $e4$ are added with lengths $d1$, $d2$, $d3$, $d4$, respectively. The change in cost is the following:

$$d3 + d4 - d1 - d2. \tag{11.7}$$

When we discuss the features used in the TSP, we will explain how this evaluation mechanism is revised to account for penalty changes in the augmented objective function.

11.10.3 Guided Local Search

For the TSP, a tour includes a number of edges and the solution cost (tour length) is given by the sum of the lengths of the edges in the tour (see Equation (11.6)). As mentioned in Section 11.9.1, edges are ideal features for routing problems such as the TSP. First, a tour either includes an edge or not and second, each edge incurs a cost in the objective function which is equal to the edge length, as given by the distance matrix $D = [d_{ij}]$ of the problem. A set of features can be defined by considering all possible undirected edges e_{ij} $(i = 1 \ldots N, j = i+1 \ldots N, i \neq j)$ that may appear in a tour with feature costs given by the edge lengths d_{ij}. With each edge e_{ij} connecting cities i and j is attached a penalty p_{ij} initially set to 0 which is increased by GLS during the search. When implementing the GLS algorithm for the TSP, the edge penalties can be arranged in a symmetric penalty matrix $P = [p_{ij}]$. As mentioned in Section 11.3, penalties have to be combined with the problem's objective function to form the augmented objective function which is minimized by local search. We therefore need to consider the auxiliary distance matrix:

$$D' = D + \lambda \cdot P = [d_{ij} + \lambda \cdot p_{ij}]. \tag{11.8}$$

Local search must use D' instead of D in move evaluations. GLS modifies P and (through that) D' whenever the local search reaches a local minimum.

In order to implement this, we revise the incremental move evaluation formula (11.7) to take into account the edge penalties and also parameter λ. If $p1$, $p2$, $p3$, $p4$ are the penalties associated with edges $e1$, $e2$, $e3$, and $e4$, respectively, the revised version of Equation (11.7) is as follows:

$$(d3 + d4 - d1 - d2) + \lambda * (p3 + p4 - p1 - p2). \tag{11.9}$$

Similarly, we can implement GLS for higher order k-Opt moves.

The edges penalized in a local minimum are selected according to the utility function (11.3), which for the TSP takes the form

$$\mathrm{util}(\mathrm{tour}, e_{ij}) = I_{e_{ij}}(\mathrm{tour}) \cdot \frac{d_{ij}}{1+p_{ij}}, \tag{11.10}$$

where

$$I_{e_{ij}}(\mathrm{tour}) = \begin{cases} 1, & e_{ij} \in \mathrm{tour} \\ 0, & e_{ij} \notin \mathrm{tour}. \end{cases} \tag{11.11}$$

The only parameter of GLS that requires tuning is parameter λ. Alternatively, we can tune the parameter α parameter which is defined in Section 11.4.2 and is relatively instance independent. Experimenting with α on the TSP, we found that there is an inverse relation between α and local search effectiveness. Not so effective local search heuristics such as 2-Opt require higher α values compared to more effective heuristics such as 3-Opt and LK. This is probably because the amount of penalty needed to escape from local minima decreases as the effectiveness of the heuristic increases explaining why lower values for α (and consequently for λ which is a function of α) work better with 3-Opt and LK. For 2-Opt, the following range for α generates high-quality solutions for instances in the TSPLIB [61]:

$$1/8 \leq \alpha \leq 1/2. \tag{11.12}$$

The reader may refer to [85] for more details on the experimentation procedure and the full set of results.

11.10.4 Guided Fast Local Search

We can exploit the way local search works on the TSP to partition the neighbourhood in sub-neighbourhoods as required by Guided Fast Local Search. Each city in the problem may be seen as defining a sub-neighbourhood, which contains all 2-Opt edge exchanges removing one of the edges adjacent to the city. For a problem with N cities, the neighbourhood is partitioned into N sub-neighbourhoods, one for each city in the instance.

The sub-neighbourhoods to be activated after a move is executed are those of the cities at the ends of the edges removed or added by the move.

Finally, the sub-neighbourhoods activated after penalization are those defined by the cities at the ends of the edge(s) penalized. There is a good chance that these sub-neighbourhoods will include moves that remove one or more of the penalized edges.

11.11 Quadratic Assignment Problem (QAP)

11.11.1 Problem Description

The Quadratic Assignment Problem (QAP) is one of the most difficult problems in combinatorial optimization. The problem can model a variety of applications but it is mainly known for its use in facility location problems. In the following, we describe the QAP in its simplest form.

Given a set $N = \{1, 2, \ldots, n\}$ and $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{kl}]$, the QAP can be stated as follows:

$$\min_{p \in \Pi_N} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{p(i)p(j)}, \tag{11.13}$$

where p is a permutation of N and Π_N is the set of all possible permutations. There are several other equivalent formulations of the problem. In the facility location context, each permutation represents an assignment of n facilities to n locations. More specifically, each position i in the permutation represents a location and its contents $p(i)$ the facility assigned to that location. The matrix A is called the distance matrix and gives the distance between any two of the locations. The matrix B is called the flow matrix and gives the flow of materials between any two of the facilities. For simplicity, we only consider the Symmetric QAP case for which both the distance and flow matrices are symmetric.

11.11.2 Local Search

QAP solutions can be represented by permutations to satisfy the constraint that each facility is assigned to exactly one location. A move commonly used for the problem is simply to exchange the contents of two permutation positions (i.e. swap the facilities assigned to a pair of locations). A best improvement local search procedure starts with a random permutation. In each iteration, all possible moves (i.e. swaps) are evaluated and the best one is selected and performed. The algorithm reaches a local minimum when there is no move, which improves further the cost of the current permutation.

An efficient update scheme can be used in the QAP which allows evaluation of moves in constant time. The scheme works only with best improvement local search. Move values of the first neighbourhood search are stored and updated each time a new neighbourhood search is performed to take into account changes from the move last executed, see [71] for details. Move values do not need to be evaluated from scratch and thus the neighbourhood can be fully searched in roughly $O(n^2)$

time instead of $O(n^3)$[1]. To evaluate moves in constant time, we have to examine all possible moves in each iteration and have their values updated. Because of that, the scheme cannot be easily combined with Fast Local Search, which examines only a number of moves in each iteration therefore preventing the problem to benefit substantially from GFLS.

11.11.3 Guided Local Search

A set of features that can be used in the QAP is the set of all possible assignments of facilities to locations (i.e. location–facility pairs). This kind of feature is general and can be used in a variety of assignment problems as explained in Section 11.9.2. In the QAP, there are n^2 possible location–facility combinations. Because of the structure of the objective function, it is not possible to estimate easily the impact of features and assign to them appropriate feature costs. In particular, the contribution in the objective function of a facility assignment to a location depends also on the placement of the other facilities with a non-zero flow to that facility.

Experimenting with the problem, we found that if all features are assigned the same cost (e.g. 1), the algorithm is still capable of generating high-quality solutions. This is due to the ability of GLS to diversify search using the penalty memory. Since features are considered of equal cost, the algorithm is distributing search efforts uniformly across the feature set. Comparative tests we conducted between GLS and the Tabu Search of [70] indicate that both algorithms are performing equally well when applied to the QAPLIB instances [8] with no clear winner across the instance set. GLS, although not using feature costs in this problem, is still very competitive to state-of-the-art techniques such as Tabu Search.

To determine λ in the QAP, one may use the formula below, which was derived experimentally:

$$\lambda = \alpha * n * (\text{mean flow}) * (\text{mean distance}), \tag{11.14}$$

where n is the size of the problem and the flow and distance means are computed over the distance and flow matrices, respectively (including any possible 0 entries which are common in QAP instances). Experimenting with QAPLIB instances, we found that optimal performance is achieved for $\alpha = 0.75$.

[1] To evaluate the change in the cost function (11.13) caused by a move normally requires $O(n)$ time. Since there are $O(n^2)$ moves to be evaluated, the search of the neighbourhood without the update scheme requires $O(n^3)$ time.

11.12 Workforce Scheduling Problem

11.12.1 Problem Description

We now look at how GLS can be applied to a real-word resource allocation problem with unallocated requests called the Workforce Scheduling problem (WSP), see [77] for more details. The problem is to schedule a number of engineers to a set of jobs, minimizing the total cost according to a function, which is to be explained below. Each job is described by a triple:

$$(\text{Loc, Dur, Type}), \tag{11.15}$$

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

Each engineer is described by a 5-tuple:

$$(\text{Base, ST, ET, OT\_limit, Skill}), \tag{11.16}$$

where Base is the x and y co-ordinates of the engineer location, ST and ET are this engineer's starting and ending time, OT_limit is his/her overtime limit and Skill is a skill factor between 0 and 1 which indicates the fraction of the standard duration that this engineer needs to accomplish a job. The cost function to be minimized is defined as follows:

$$\text{TotalCost} = \sum_{i=1}^{\text{NoT}} \text{TC}_i + \sum_{i=1}^{\text{NoT}} \text{OT}_i^2 + \sum_{j=1}^{\text{NoJ}} (\text{Dur}_j + \text{Penalty}) \times \text{UF}_j, \tag{11.17}$$

where

NoT = number of engineers,
NoJ = number of jobs,
TC_i = Travelling Cost of engineer i,
OT_i = Overtime of engineer i,
Dur_j = Standard duration of job j,
UF_j = 1 if job j is unallocated; 0 otherwise,
Penalty = constant (which is set to 60 in the tests).

The travelling cost between $(x1, y1)$ and $(x2, y2)$ is defined as follows:

$$TC((x_1, y1), (x2, y2)) = \begin{cases} \dfrac{\frac{\Delta_x}{2} + \Delta_y}{8}, & \Delta_x > \Delta_y \\ \dfrac{\frac{\Delta_y}{2} + \Delta_x}{8}, & \Delta_y \geq \Delta_x \end{cases}. \tag{11.18}$$

Here Δ_x is the absolute difference between $x1$ and $x2$, and Δ_y is the absolute difference between $y1$ and $y2$. The greater of the x and y differences is halved before summing. The formula above was specifically designed for the benchmark used in [77] to convert distances into approximate travel times as observed in realistic trips conducted by engineers. Engineers are required to start from and return to their base everyday. An engineer may be assigned more jobs than he/she can finish.

11.12.2 Local Search

11.12.2.1 Solution Representation

We represent a candidate solution (i.e. a possible schedule) by a permutation of the jobs. Each permutation is mapped into a schedule using the deterministic algorithm described below:

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

1. For each job, order the qualified engineers in ascending order of the distances between their bases and the job (such orderings only need to be computed once and recorded for evaluating other permutations).
2. Process one job at a time, following their ordering in the input permutation. For each job x, try to allocate it to an engineer according to the ordered list of qualified engineers:
   - 2.1. to check if engineer g can do job x, make x the first job of g; if that fails to satisfy any of the constraints, make it the second job of g, and so on;
   - 2.2. if job x can fit into engineer g's current tour, then try to improve g's new tour (now with x in it): the improvement is done by a simple 2-opt algorithm (see Section 11.10), modified in a such a way that only better tours which satisfy the relevant constraints will be accepted;
   - 2.3. if job x cannot fit into engineer g's current tour, then consider the next engineer in the ordered list of qualified engineers for x; the job is unallocated if it cannot fit into any engineer's current tour.
3. The cost of the input permutation, which is the cost of the schedule thus created, is returned.

11.12.2.2 Construction Method

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

11.12.2.3 Improvement Method

Given a permutation, local search is performed in a simple way: the pairs of jobs are examined one at a time. Two jobs are swapped to generate a new permutation if the new permutation is evaluated (using the Evaluation procedure above) to a lower cost than the original permutation. Note here that since the problem is also close to the Vehicle Routing Problem (VRP), one may follow a totally different approach considering VRP move operators such as insertions and swaps. In this case, the solution representation and construction methods need to be revised. The reader may refer to other works (e.g. [3]) for more information on the application of GLS to the VRP.

11.12.3 Guided Local Search

In the workforce scheduling problem, we use the feature type recommended for resource allocation problems in Section 11.9.3. In particular, the inability to serve jobs incurs a cost, which plays the most important part in the objective function. Therefore, we intend to bias local search to serve jobs of high importance. To do so, we define a feature for each job in the problem:

$$I_{\mathrm{job}_j}(\mathrm{schedule}) = \begin{cases} 1, & \mathrm{job}_j \text{ is unallocated in schedule} \\ 0, & \mathrm{job}_j \text{ is allocated in schedule.} \end{cases} \tag{11.19}$$

The cost of this feature is given by $(\mathrm{Dur}_j + \mathrm{Penalty})$ which is equal to the cost incurred in the cost function (11.17) when a job is unallocated.

The jobs penalized in a local minimum are selected according to the utility function (11.3) which for workforce scheduling takes the form

$$\mathrm{util}(\mathrm{schedule},\, \mathrm{job}_j) = I_{\mathrm{job}_j}(\mathrm{schedule}) \cdot \frac{(\mathrm{Dur}_j + \mathrm{Penalty})}{1 + p_j}. \tag{11.20}$$

WSP exhibits properties found in resource allocation problems (i.e. unallocated job costs) and also in routing problems (i.e. travel costs). In addition to the above feature type and for better performance, we may consider introducing a second feature type based on edges as suggested in Section 11.9.1 for routing problems and explained in Section 11.10.3 for the TSP. This feature set can help to aggressively optimize the travel costs also incorporated in the objective function (11.17). Furthermore, one or both feature sets can be used in conjunction with a VRP-based local search method.

11.12.4 Guided Fast Local Search

To apply Guided Fast Local Search to workforce scheduling, each job permutation position defines a separate sub-neighbourhood. The activation bits are manipulated according to the general FLS algorithm of Section 11.5. In particular

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

Mapping penalized jobs to sub-neighbourhoods is straightforward. We simply activate the sub-neighbourhoods corresponding to the permutation positions of the penalized jobs. This essentially forces Fast Local Search to examine moves, which swap the penalized jobs.

11.13 Radio Link Frequency Assignment Problem

11.13.1 Problem Description

The *Radio Link Frequency Assignment Problem* (RLFAP) [58, 75] is abstracted from the real-life application of assigning frequencies to radio links. The problem belongs to the class of constraint optimization problems mentioned in Section 11.9.4. In brief, the interference level between the frequencies assigned to the different links has to be acceptable; otherwise communication will be distorted. The frequency assignments have to comply with certain regulations and physical characteristics of the transmitters. Moreover, the number of frequencies is to be minimized, because each frequency used in the network has to be reserved at a certain cost. In certain cases, some of the links may have pre-assigned frequencies which may be respected or preferred by the frequency assignment algorithm. Here, we examine a simplified version of the problem considering only the interference constraints. Information on the application of GLS to the full problem can be found in [83]. A definition of the simplified problem is the following.

We are given a set L of links. For each link i, a frequency f_i has to be chosen from a given domain D_i. Constraints are defined on pairs of links that limit the choice of frequencies for these pairs. For a pair of links $\{i, j\}$ these constraints are either of type

$$|f_i - f_j| > d_{ij}, \tag{11.21}$$

or of type

$$|f_i - f_j| = d_{ij}, \tag{11.22}$$

for a given distance $d_{ij} \geq 0$. Two links i and j involved in a constraint of type Equation (11.21) are called *interfering* links, and the corresponding d_{ij} is the interfering distance. Two links bound by a constraint of type Equation (11.22) are referred to as a pair of *parallel* links; every link belongs to exactly one such pair.

Some of the constraints may be violated at a certain cost. Such restrictions are called *soft*, in contrast to the *hard* constraints, which may not be violated. The constraints of type Equation (11.22) are always hard. Interference costs c_{ij} for violating soft constraints of type Equation (11.21) are given. An assignment of frequencies is complete if every link in L has a frequency assigned to it. We denote by C the set of all soft *interference* constraints.

The goal is to find a complete assignment that satisfies all hard constraints and is of minimum cost:

$$\min \sum_{C} c_{ij} \cdot \delta(|f_i - f_j| \leq d_{ij}) \tag{11.23}$$

subject to hard constraints:

$|f_i - f_j| > d_{ij}$: for all pairs of links $\{i, j\}$ involved in the hard constraints,
$|f_i - f_j| = d_{ij}$: for all pairs of parallel links $\{i, j\}$,
$f_i \in D_i$: for all links $i \in L$,

where $\delta(.)$ is 1 if the condition within brackets is true and 0 otherwise.

We look next at a local search procedure for the problem.

11.13.2 Local Search

11.13.2.1 Using an Alternate Objective Function

When using heuristic search to solve a combinatorial optimization problem, it is not always necessary to use the objective function as dictated in the problem formulation. Objective functions based on the original one can be devised which result in smoother landscapes. These objective functions can sometimes generate solutions of higher quality (with respect to the original objective function) than if the original one is used.

In the RLFAP, we can define and use a simple objective function g, which is given by the sum of all constraint violation costs in the solution with all the constraints contributing equally to the sum instead of using weights as in Equation (11.23). This objective function is as follows for a given solution s:

$$g(s) = \sum_{C \cup C^{\text{Hard}}} \delta(|f_i(s) - f_j(s)| \leq d_{ij}), \tag{11.24}$$

subject to hard constraints:

$f_i(s) \in D_i'$: for all links $i \in L$,

where $\delta(.)$ is 1 if the condition within brackets is true and 0 otherwise, $f_i(s)$ is the frequency assigned to link i in solution s, C^{Hard} is the set of hard inequality constraints, C is the set of soft inequality constraints and D'_i is the reduced domain for link i containing only frequencies which satisfy the hard equality constraints.

A solution s with cost 0 with respect to g is satisfying all hard and soft constraints of the problem.

The motivation to use an objective function such as Equation (11.24) is closely related to the rugged landscapes formed in RLFAP, if the original cost function is used. In particular, high and very low violation costs are defined for some of the soft constraints. This leads to even higher violation costs to be defined for hard constraints. The landscape is not smooth but full of deep local minima mainly due to the hard and soft constraints of high cost. Soft constraints of low cost are buried under these high costs.

A similar objective function replacement approach has been used successfully by [54] in the MAX-SAT problem suggesting the universal appeal of the idea in constrained optimization problems.

11.13.2.2 Solution Representation

An efficient solution representation for the problem takes into account the fact that each link in RLFAP is connected to exactly one other link via a hard constraint of type Equation (11.22). In particular, we can define a decision variable for each pair of parallel links bound by an equality constraint Equation (11.22). The domain of this variable is defined as the set of all pairs of frequencies from the original domains of the parallel links that satisfy the hard equality constraint.

11.13.2.3 Construction Method

A construction method can be implemented by assigning to each decision variable (which assigns values to a pair of links) a random value from its domain. In large problem instances, it is beneficial to consider a domain pre-processing and reduction phase. Sophisticated techniques based on Arc-Consistency can be utilized during that phase to reduce the domain based on the problem's hard constraints. These domains can then be used instead of the original ones for the random solution generation and also by the improvement method.

11.13.2.4 Improvement Method

An improvement method can be based on the min-conflicts heuristic of Minton et al. [56] for Constraint Satisfaction Problems. A 1-optimal type move is used which changes the value of one variable at a time. Starting from a random and complete assignment of values to variables, variables are examined in an arbitrary static order. Each time a variable is examined, the current value of the variable changes to the value (in the variable's domain) which yields the minimum value for the objective function. Ties are randomly resolved allowing moves to solutions with equal cost. These moves are called *sideways moves* [65] and enable the local search to examine plateaus of solutions that occur in the landscapes of many constrained optimization problems.

11.13.3 Guided Local Search

The most important cost factor in the RLFAP is the constraint violation costs defined for soft inequality constraints. Inequality constraints can be used to define a basic feature set for the RLFAP. Each inequality constraint is interpreted as a feature with the feature cost given by the constraint violation cost c_{ij} as defined in the problem's original cost function (11.23).

Hard inequality constraints are also modelled as features by assigning to them an infinite cost. This results in their utility to be penalized to also tend to infinity. To implement this in the code, hard constraints are simply given priority over soft constraints when penalties are applied. This basically forces local search to return back to a feasible region where penalizing soft constraints can resume.

GLS is especially suited to use the alternate objective function (11.24) because of the definition of the feature costs described above. The application of penalties can still force the local search toward solutions which satisfy constraints with high violation costs while the algorithm is benefiting from the smoother landscape introduced by (11.24).

The λ parameter can be set to 1 provided that we use (11.24) as the objective function. The same value for λ has also been used in MAX-SAT problems in [54] where the same approach is followed with respect to smoothing the landscape.

A variation of the GLS method which seems to significantly improve performance in certain RLFAP instances is to decrease penalties and not only increase them [83]. More specifically, the variant uses a circular list to retract the effects of penalty increases made earlier in the search process, in a way that very much resembles a tabu list. In particular, increased penalties are decreased after a certain number of increases. The scheme uses an array of size t where the t most recent features penalized are recorded. The array is treated as a circular list, adding elements in sequence in positions 1 through t and then starting over at position 1. Each time the penalty of a feature is increased (by one unit), the feature is inserted in the array and the penalty of the feature previously stored in the same position is decreased (by one unit). The rationale behind the strategy is to allow GLS to return to regions of the search visited earlier in the search process, so introducing a search intensification mechanism.

11.13.4 Guided Fast Local Search

Best improvement local search for the RLFAP as used in the context of Tabu Search (for an example, see [31]), evaluates all possible 1-optimal moves over all variables before selecting and performing the best move. Given the large number of links in real-world instances, greedy local search is a computationally expensive option. This is especially the case for the RLFAP where we cannot easily devise an incremental move update mechanism (such as the one for the QAP) for all the problem's variations. The local search procedure described in Section 11.13.2 is already a faster alternative than best improvement. Using Guided Fast Local Search, things can be improved further.

To apply Guided Fast Local Search to RLFAP, each decision variable defines a sub-neighbourhood and has a bit associated with it. Whenever a variable is examined and its value is changed (i.e. the variable's parallel links are assigned to another pair of frequencies because of an improving or sideway move) the activation bit of the variable remains to 1 otherwise it turns to 0 and the variable is excluded in future iterations of the improvement loop. 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 for which improving moves may occur as a result of the move just performed. They are the variables for which one of their links is connected via a constraint to one of the links of the variable with a modified value. There are five potential schemes for propagating activation after changing the value of a variable. They are the following:

1. Activate all variables connected via a constraint to the variable with a modified value.
2. Activate only variables that are connected via a constraint which is violated. This resembles CSP local search methods where only variables in conflict have their neighbourhood searched.
3. Activate only variables that are connected via a constraint which has become violated as a result of the move (subset of condition 2 and also condition 4).
4. Activate only variables that are connected via a constraint that changed from violated to satisfied or from satisfied to violated, as a result of the move (superset of condition 3).
5. Activate variables that fall under either condition 2 or 4.

Experimentation suggests that scheme 5 tends to produce better results for the real-world instances of RLFAP available in the literature. Fast local search stops when all the variables are inactive or when a local minimum is detected by other means (i.e. a number of sideway moves is performed without an improving move found).

Finally, when a constraint is penalized we activate the variables connected via the constraint in an effort to find 1-Opt moves which will satisfy the constraint.

11.14 Summary and Conclusions

For many years, general heuristics for combinatorial optimization problems, with prominent examples such as Simulated Annealing and Genetic Algorithms, heavily relied on randomness to generate good approximate solutions to difficult NP-Hard problems. The introduction and acceptance of Tabu Search [25] by the Operations Research community initiated an important new era for heuristic methods where deterministic algorithms exploiting historical information started to appear and to be used in real-world applications.

Guided local search described in this chapter follows this trend. While Tabu search is a class of algorithms (where a lot of freedom is given to the management of the tabu list), GLS is more prescriptive (the procedures are more concretely defined). GLS heavily exploits information (not only the search history) to distribute the search effort in the various regions of the search space. Important structural properties of solutions are captured by solution features. Solution features are assigned costs and local search is biased to spend its efforts according to these costs. Penalties on features are utilized for that purpose.

When local search settles in a local minimum, the penalties are increased for selected features present in the local minimum. By penalizing features appearing in local minima, GLS not only escapes the local minima visited (exploiting historical information) but also diversifies the choices, with regard to the various structural properties of solutions, as captured by the solution features. Features of high costs are penalized more often than features of low cost: the diversification process is directed and deterministic rather than undirected and random.

In general, several penalty cycles may be required before a move is executed out of a local minimum. This should not be viewed as an undesirable situation. It is caused by the uncertainty in the information as captured by the feature costs which forces the GLS to test its decisions against the landscape of the problem.

The penalization scheme of GLS is ideally combined with FLS which limits the neighbourhood search to particular parts of the overall solution leading to the GFLS algorithm. GFLS significantly reduces the computation times required to explore the area around a local minimum to find the best escape route allowing many more penalty modification cycles to be performed in a given amount of running time.

The GLS and GFLS methods are still in their early stages and future research is required to develop them further. The use of incentives implemented as negative penalties, which encourage the use of specific solution features, is one promising direction to be explored. Other interesting directions include *fuzzy features* with indicator functions returning real values in the [0, 1] interval, automated tuning of the λ or α parameters, definition of effective termination criteria, alternative utility functions for selecting the features to be penalized and also studies about the convergence properties of GLS.

It is relatively easy to adapt GLS and GFLS to the different problems examined in this chapter. Although local search is problem dependent, the other structures of GLS and also GFLS are problem independent. Moreover, a mechanical, step-by-step procedure is usually followed when GLS or GFLS is applied to a new problem (i.e. implement a local search procedure, identify features, assign costs and define sub-neighbourhoods). This makes GLS and GFLS easier to use by non-specialist software engineers.

References

1. Anderson, C.A., Fraughnaugh, K., Parker, M., Ryan, J.: Path assignment for call routing: An application of tabu search. Ann. Oper. Res. **41**, 301–312 (1993)
2. Azarmi, N. and Abdul-Hameed, W.: Workforce scheduling with constraint logic programming. BT Technol. J. **13**:1, 81–94 (1995)
3. Backer, B.D., Furnon, V., Shaw, P., Kilby, P., Prosser, P.: Solving vehicle routing problems using constraint programming and metaheuristics. J. Heuristics **6**:4, 501–523 (2000)
4. Basharu, M., Arana, I., Ahriz, H.: Distributed guided local search for solving binary DisCSPs. Proceedings of FLAIRS 2005, Florida, USA, AAAI Press, pp. 660–665 (2005)
5. Bentley, J.J.: Fast algorithms for geometric traveling salesman problems. ORSA J. Comput. **4**, 387–411 (1992)
6. Beullens, P., Muyldermans, L., Cattrysse, D., Van Oudheusden, D.: A guided local search heuristic for the capacitated arc routing problem. Eur. J. Oper. Res. **147**:3, 629–643 (2003)
7. 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)
8. Burkard, R.E., Karisch, S.E., Rendl F.: QAPLIB - A Quadratic assignment problem library. J. Global Optim **10**, 391–403 (1997)
9. Chalmers, A.G.: A minimum path parallel processing environment. Research Monographs in Computer Science, Alpha Books (1994)
10. Chiarandini, M. and Stutzle, T.: Stochastic local search algorithms for graph set T-colouring and frequency assignment. Constraints **12**(3), 371–403 (2007)
11. Chu, P., Beasley, J.E.: A genetic algorithm for the generalized assignment problem. Comput. Oper. Res. **24**, 17–23 (1997)
12. Congram, R.K., Potts, C.N.: Dynasearch Algorithms for the traveling salesman problem. Presentation at the Travelling Salesman Workshop, CORMSIS, University of Southampton, Southampton, UK (1999)
13. Croes, A.: A method for solving traveling-salesman problems. Oper. Res. **5**, 791–812 (1958)
14. Daum, M., Menzel, W.: Parsing natural language using guided local search. Proceedings of 15th European Conference on Artificial Intelligence (ECAI-2002), pp. 435–439 (2002)
15. Davenport, A., Tsang, E.P.K., Wang, C.J., Zhu, K.: GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. Proceedings of 12th National Conference for Artificial Intelligence (AAAI), 325–330, Seattle, WA, USA (1994)
16. Dorne, R., Mills, P., Voudouris, C.: Solving vehicle routing using iOpt. In: Doerner, K.F. et al. (eds.) Metaheuristics: Progress in Complex Systems Optimization. Operations Research/Computer Science Interfaces Series, vol. **39**, pp. 389–408, Springer, New York (2007)
17. Dorne, R., Voudouris, C., Liret, A., Ladde, C., Lesaint, D.: iSchedule an optimisation toolkit based on heuristic search to solve BT scheduling problems. BT Technol. J. **21**:4, 50–58 (2003)
18. Egeblad, J., Nielsen, B., Odgaard, A.: Fast neighbourhood search for two- and three-dimensional nesting problems. Eur. J. Oper. Res. **183**(3), 1249–1266 (2007)
19. Faroe, O., Pisinger, D., Zachariasen, M.: Guided local search for the three-dimensional bin packing problem. Tech. Rep. 99-13, Department of Computer Science, University of Copenhagen. (1999)
20. Faroe, O., Pisinger, D., Zachariasen, M.: Guided local search for final placement in VLSI design. J. Heuristics **9**, 269–295 (2003)
21. Flood, M.M.: The traveling-salesman problem. Oper. Res. **4**, 61–75 (1956)
22. Flores Lucio, G., Reed, M., Henning, I.: Guided local search as a network planning algorithm that incorporates uncertain traffic demands. Comput. Net. **51**(11), 3172–3196 (2007)
23. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric travelling salesman problems. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 616–621, Nayoya University, Japan (1996)

24. Gent, I.P., van Maaren, H., Walsh, T.: SAT2000, Highlights of satisfiability research in the year 2000. Frontiers in Artificial Intelligence and Applications, IOS Press. (2000)
25. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
26. GLS Demos: http://cswww.essex.ac.uk/CSP/glsdemo.html (2008)
27. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Pub. Co., Inc., Reading, MA (1989)
28. Gomes, N., Vale, Z., Ramos, C.: Hybrid Constraint algorithm for the maintenance scheduling of electric power units. Proceedings of International Conference on Intelligent Systems Application to Power Systems (ISAP 2003), Lemnos, Greece (2003)
29. Hani, Y., Amodeo, L., Yalaoui, F., Chen, H.: Ant colony optimization for solving an industrial layout problem. Eur. J. Oper. Res. **183**(2), 633–642 (2007)
30. Hansen, P., Mladenovic, N.: An introduction to variable neighbourhood search. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) pp. 433–458. Kluwer, Boston (1999)
31. Hao J.-K., Dorne, R., Galinier, P.: Tabu search for frequency assignment in mobile radio networks. J. Heuristics **4**(1), 47–62 (1998)
32. Hifi, M., Michrafy, M., Sbihi, A.: Heuristic algorithms for the multiple-choice multidimensional knapsack problem. J. Oper. Res. Soc. **55**, 1323–1332 (2004)
33. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
34. Holstein, D., Moscato, P.: Memetic algorithms using guided local search: a case study. Corne, D., Glover, F., Dorigo, M., (eds.) New Ideas in Optimisation, pp. 235–244, McGraw-Hill, London (1999)
35. Hoos, H., Tsang, E.P.K.: Local search for constraint satisfaction, Chapter 5. Rossi, F., van Beek P., Walsh T. (eds.), Handbook of Constraint Programming, pp. 245–277 Elsevier, Amsterdam, The Netherlands (2006)
36. Johnson, D.: Local optimization and the traveling salesman problem. Proceedings of the 17th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science, vol **443**, pp. 446–461, Springer, Berlin (1990)
37. Jose, R., Boyce, J.: Application of connectionist local search to line management rail traffic control. Proceedings of International Conf. on Practical Applications of Constraint Technology (PACT'97), London (1997)
38. Kilby, P., Prosser, P., Shaw, P.: Guided local search for the vehicle routing problem with time windows. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 473–486 Kluwer Academic Publishers, (1999)
39. Kilby, P., Prosser, P., Shaw, P.: A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. Constraints **5**(4), 389–414 (2000)
40. Knox, J.: Tabu search performance on the symmetric traveling salesman problem. Comput. Ops. Res. **21**(8), 867–876 (1994)
41. Koopman, B.O.: The theory of search, part III, the optimum distribution of search effort. Oper. Res. **5**, 613–626 (1957)
42. Kytjoki, J., Nuortio, T., Brysy, O., Gendreau, M.: An efficient variable neighbourhood search heuristic for very large scale vehicle routing problems. Comput. Oper. Res. **34**:9, 2743–2757 (2007)
43. Langer, Y., Bay, M., Crama, Y., Bair, F., Caprace, J.D., Rigo, P.: Optimization of surface utilization using heuristic approaches. Proceedings of the International Conference COMPIT'05, pp. 419–425 (2005)
44. Lau, T.L.: Guided Genetic Algorithm. PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK. (1999)
45. Lau, T.L., Tsang, E.P.K.: Solving the processor configuration problem with a mutation-based genetic algorithm. Int. J. Artif. Intell. Tools (IJAIT) **6**(4), 567–585 (1997)
46. Lau, T.L., Tsang, E.P.K.: Guided genetic algorithm and its application to the radio link frequency allocation problem. Proceedings of NATO symposium on Frequency Assignment,

Sharing and Conservation in Systems (AEROSPACE), AGARD, RTO-MP-13, paper No. 14b. (1998)
47. 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, 336–343 (1998)
48. Lee, J.H.M., Tam, V.W.L.: A framework for integrating artificial neural networks and logic programming. Int. J. Artif. Intell. Tools 4, 3–32 (1995)
49. Lin, S.: Computer solutions of the traveling-salesman problem. Bell Syst. Tech. J. **44**, 2245–2269 (1965)
50. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res. **21**, 498–516 (1973)
51. Martin, O., Otto, S.W.: Combining simulated annealing with local search heuristics. Laporte G., Osman I.H. (eds.), Metaheuristics in Combinatorial Optimization, Ann. Oper. Res. 63 (1966)
52. Mester, D., Brysy, O.: Active guided evolution strategies for large-scale vehicle routing problems with time windows. Comput. Oper. Res. **32**(6), 1593–1614 (2005)
53. Mester, D.I., Ronin, Y. I., Nevo, E., Korol, A.B.: Fast and high precision algorithms for optimization in large-scale genomic problems. Comput. Biol. Chem. **28**(4), 281–290 (2004)
54. Mills, P., Tsang, E.P.K.: Guided local search for solving SAT and weighted MAX-SAT problems. J. Automat Reas **24**, 205–223 (2000)
55. Mills P., Tsang E., Ford J.: Applying an extended guided local search to the quadratic assignment problem. Ann. Oper. Res. **118**(1-4), 121–135 (2003)
56. Minton S., Johnston, M.D., Philips A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artif. Intell. 58(1-3), (Special Volume on Constraint Based Reasoning), 161–205 (1992)
57. Moghrabi, I.: Guided local search for query reformulation using weight propagation. Int. J. Appl. Mathe. Comput. Sci. (AMCS) **16**(4), 537–549 (2006)
58. Murphey, R.A., Pardalos, P.M., Resende, M.G.C.: Frequency assignment problems. Du D.-Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization vol. 4, Kluwer Academic Publishers, Dordrecht, The Netherlands (1999)
59. Padron, V., Balaguer, C.: New methodology to solve the RPP by means of isolated edge. 2000 Cambridge Conference Tutorial Papers. In: Tuson, A. (ed.) Young OR 11, UK Operational Research Society (2000)
60. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. J. Heuristics **5**(3), 255–279 (1999)
61. Reinelt, G.: A traveling salesman problem library. ORSA J. Comput. 3, 376–384 (1991)
62. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science 840, Springer, Berlin (1995)
63. Resende, M.G.C., Feo, T.A.: A GRASP for satisfiability. Cliques, coloring, and satisfiability: Second DIMACS implementation challenge. In: Johnson D.S., Trick, M.A. (eds.) DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 499–520. American Mathematical Society, (1996)
64. Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. Proceedings of 13th International Joint Conference on AI, pp. 290–295, Chambery, France (1993)
65. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. Proceedings of AAAI-92, pp. 440–446, San Jose, CA, USA (1992)
66. Shang, Y., Wah, B.W.: A discrete lagrangian-based global-search method for solving satisfiability problems. J. Global Optim. **12**(1), 61–99 (1998)
67. Simon, H.U.: Approximation algorithms for channel assignment in cellular radio networks. Proceedings of 7th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Computer Science 380, pp. 405–416, Springer, Berlin (1989)
68. Stone, L.D.: The process of search planning: current approaches and continuing problems. Oper. Res. **31**, 207–233 (1983)

69. Stuckey, P., Tam, V.: Semantics for using stochastic constraint solvers in constraint logic programming. J. Funct. Logic Programming 2 (1998)
70. Taillard, E.: Robust taboo search for the QAP. Parallel Comput. **17**, 443–455 (1991)
71. Taillard, E.: Comparison of iterative searches for the quadratic assignment problem. Location Sci. 3, 87–105 (1995)
72. Tamura, H., Zhang, Z., Tang, Z., Ishii, M.: Objective function adjustment algorithm for combinatorial optimization problems. IEICE Trans. Fundamentals of Electronics, Communications Comput. Sci. **E89-A**:9, 2441–2444 (2006)
73. Tarantilis, C.D., Zachariadis, E.E., Kiranoudis, C.T.: A guided tabu search for the heterogeneous vehicle routeing problem. J. Oper. Res. Soc. **59**, 1659–1673 (2008)
74. Tarantilis, C.D., Zachariadis, E.E., Kiranoudis, C.T.: A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. INFORMS J. Comput. **20**(1), 154–168 (2008)
75. Tiourine, S., Hurkins, C., Lenstra, J.K.: An overview of algorithmic approaches to frequency assignment problems. EUCLID CALMA Project Overview Report, Delft University of Technology, The Netherlands (1995)
76. Tsang, E.P.K.: Foundations of constraint satisfaction, Academic Press, London (1993)
77. Tsang, E.P.K., Voudouris, C.: Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Oper. Res. Lett. **20**(3), 119–127 (1997)
78. Tsang, E.P.K., Wang, C.J.: A generic neural network approach for constraint satisfaction problems. Taylor, J.G. (ed.) Neural Network Applications, pp. 12–22 Springer, Berlin (1992)
79. Tsang, E.P.K., Wang, C.J., Davenport, A., Voudouris, C., Lau, T.L.: A family of stochastic methods for constraint satisfaction and optimisation. Proceedings of the First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP), London, pp. 359–383 (1999)
80. Vansteenwegen, P., Souffriau, W., Berghe, G., Oudheusden, D.: A guided local search metaheuristic for the team orienteering problem. Eur. J. Oper. Res. **196**(1), 118–127 (2009)
81. Voudouris, C.: Guided Local Search for Combinatorial Optimisation Problems. PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK (1997)
82. Voudouris, C.: Guided local search an illustrative example in function optimisation. BT Technol. J. **16**(3), 46–50 (1998)
83. Voudouris, C., Tsang, E.: Solving the radio link frequency assignment problems using guided local search. Proceedings of NATO symposium on Frequency Assignment, Sharing and Conservation in Systems (AEROSPACE), AGARD, RTO-MP-13, paper No. 14a (1998)
84. Voudouris, C., Tsang, E.P.K.: Partial constraint satisfaction problems and guided local search. Proceedings of PACT'96, London, pp. 337–356 (1996)
85. Voudouris, C., Tsang, E.P.K.: Guided local search and its application to the travelling salesman problem. Eur. J. Oper. Res. **113**(2), 469–499 (1999)
86. Voudouris, C., Dorne, R., Lesaint, D., Liret, A.: iOpt: A Software Toolkit for Heuristic Search Methods. Principles and Practice of Constraint Programming – CP 2001 In: Walsh, T. (ed.) Lecture Notes in Computer Science, vol. **2239**, pp. 716–729 Springer, Heidelberg (2001)
87. Wang, C.J., Tsang, E.P.K.: Solving constraint satisfaction problems using neural-networks. Proceedings of the IEE Second International Conference on Artificial Neural Networks, pp. 295–299, Baurnmouth, UK (1991)
88. Wang, C.J., Tsang, E.P.K.: A cascadable VLSI design for GENET. In: VLSI for Neural Networks and Artificial Intelligence, Delgado-Frias J.G., Moore, W.R. (eds.) pp. 187–196 Plenum Press, New York (1994)
89. Xiaohu, T., Haubrich, H.-J.: A hybrid metaheuristic method for the planning of medium-voltage distribution networks. Proceedings of 15th Power Systems Computation Conference (PSCC 2005), Liege, Belgium (2005)
90. Zachariadis, E., Tarantilis, C., Kiranoudis, C.: A Guided Tabu Search for the Vehicle Routing Problem with two-dimensional loading constraints. Eur. J. Oper. Res. **195**(3), 729–743 (2009)
91. Zachariadis, E., Tarantilis, C., Kiranoudis, C.: A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. Exp. Syst. Appl. **36**(2), 1070–1081 (2009)

92. Zhang, Q., Sun, J., Tsang, E.P.K, Ford, J.: Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem. Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference. (2003)
93. Zhong, Y., Cole, M. H.: A vehicle routing problem with backhauls and time windows: a guided local search solution. Transport Res. E: Logistics Transport Rev. **41**(2), 131–144 (2005)