Guided Local Search

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

Contents

A. Alsheddy (✉)
College of Computer and Information Sciences (CCIS), Al-Imam Muhammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia
e-mail: asheddy@imamu.edu.sa

C. Voudouris
Department of Computer Science, University of Essex, Colchester, UK
e-mail: voudcx@essex.ac.uk; christos.voudouris@gmail.com

E.P.K. Tsang
Department of Computer Science, University of Essex, Colchester, UK
e-mail: edward@essex.ac.uk

A. Alhindi
Department of Computer Science, Umm Al-Qura University, Makkah, Saudi Arabia
e-mail: ahhindi@uqu.edu.sa

R. Martí et al. (eds.), *Handbook of Heuristics*,
DOI 10.1007/978-3-319-07153-4_2-1

Abstract

Guided local search (GLS) is a meta-heuristic method proposed to solve combinatorial optimization problems. It is a high-level strategy that applies an efficient penalty-based approach to interact with the local improvement procedure. This interaction creates a process capable of escaping from local optima, which improves the efficiency and robustness of the underlying local search algorithms. Fast local search (FLS) is a way of reducing the size of the neighborhood to improve the efficiency of local search. GLS can be efficiently combined with FLS in the form of guided fast local search (GFLS). This chapter describes the principles of GLS and provides guidance for implementing and using GLS, FLS, and GFLS. It also surveys GLS extensions, hybrids, and applications to optimization, including multi-objective optimization.

Keywords

Heuristic search • Meta-heuristics • Penalty-based methods • Guided local search • Tabu search • Constraint satisfaction

Introduction

Many practical problems are NP-hard in nature, which means complete, constructive search is unlikely to satisfy our computational demand. Many real-life problems cannot be realistically and reliably solved by complete search. This motivates the development of local search or heuristic methods.

Local search (LS) is the basis of most heuristic search methods. It searches in the space of candidate solutions.The solution representation issue is significant, though it is not the subject of our discussion here. In the basic variant of LS, known as hill climbing, LS starts from a (possibly randomly generated) candidate solution and then moves to a "neighbor" that is "better" than the current candidate solution according to the objective function. The search naturally stops when all neighbors 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 neighbors, 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 [79], a general meta-heuristic algorithm, and its applications. GLS has been applied to a nontrivial 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.

GLS can be seen as a generalization of its predecessors GENET [23, 77] which was developed for constraint satisfaction problems. GLS also relates to ideas from the area of search theory on how to distribute the search effort. In particular, incremental distribution of the search effort according to information in a probabilistic framework can be found in a class of methods deriving themselves from the optimal search theory of Koopman [42, 69].

The principles of GLS can be summarized as follows. As a meta-heuristic 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, favoring promising areas. Penalty modifications *regularize* the solutions generated by local search to be in accordance to prior or gathered during search information. The approach taken by GLS is similar to that of regularization methods for "ill-posed" problems [75]. The idea behind regularization methods and GLS up to an extent is the use of prior information to help in solving an approximation problem. Prior information translates to constraints which further define our problem reducing so the number of candidate solutions to be considered.

The structure of the chapter is as follows. In section "Guided Local Search", we describe GLS with details about implementing its components. Similarly, the implementation of combining GLS with fast local search is discussed in section "Guided Fast Local Search". Next, in section "GLS Extensions, Hybrids, and Variations", we review other algorithms that extend GLS or hybridize it with other techniques. We then present an extension of GLS to handle multi-objective optimization problems in section "GLS for Multi-objective Optimization". The applications of GLS to the traveling salesman problem and a workforce scheduling problem are explained in sections "GLS Implementation on the Traveling Salesman Problem" and "GLS/GPLS Implementation on a Workforce Scheduling Problem", respectively. We give comprehensive references to the applications of GLS and its variants in section "Overview of GLS Applications", and we conclude in section "Conclusions".

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 traveling salesman problem [28], 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 traveling 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):

$$h(s) = g(s) + \lambda \times \sum_{i \text{ is a feature}} (p_i \times I_i(s)) \tag{1}$$

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:

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i; 0 \text{ otherwise.} \tag{2}$$

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 "unfavorable 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:

$$util_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \tag{3}$$

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 [54].

Implementation Guideline

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. Each time the local search procedure reaches a local minimum, the augmented objective function is modified 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 (Eq. 3). The section below presents and explains the pseudo-code for implementing a guided local search method.

Algorithm 1 depicts the pseudo-code for the guided local search procedure, 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.

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.

Construction Method

As with other meta-heuristics, 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.

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 neighborhood search [36],

Algorithm 1: The guided local search algorithm

GuidedLocalSearch$(P, g, \lambda, [I_1, \ldots, I_M], [c_1, \ldots, c_M], M)$

```
k ← 0;
s_0 ← ConstructionMethod(P);
{set all penalties to 0}
for i ← 1 until M do
    p_i ← 0;
end for
{define the augmented objective function}
h ← g + λ * Σ p_i * I_i;
while StoppingCriterion do
    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);
    end for
    {penalize features with maximum utility}
    for all i such that util_i is maximum do
        p_i ← p_i + 1;
    end for
    k ← k + 1;
end while
s* ← best solution found with respect to objective function g;
return s*;
```

variable depth descent [49], ejection chains [32], 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 [82] 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.

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.

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 (i.e., tuning α can result in λ values, which work for many instances of a problem class). The recommended formula for λ as a function of α is the following:

$$\lambda = \alpha * g(x^*)/(\text{no. of features present in } x^*) \tag{4}$$

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.

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:

$$\sum_{\text{over all features } j \text{ added}} p_j - \sum_{\text{over all features } k \text{ deleted}} p_k \tag{5}$$

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.

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.

Possible Features for Common Applications

Applying guided local search to a problem requires identifying a *suitable* set of features to guide the search process. 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 modeled as features with 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 toward 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 useful features for several representative problems from various domains.

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 traveling salesman problem, vehicle routing problem, machine scheduling with sequence-dependent setup times, and others.

Travel or setup times are modeled 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 either contains 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.

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, etc.). 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.

Resource Allocation Problem

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 [8], maximum channel assignment problem [68], workforce scheduling problem [9, 15], and others.

Constrained Optimization Problems

Constraints are very important in capturing processes and systems in the real world. A number of combinatorial optimization problems deal 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 modeled 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 modeling include the constraint satisfaction and partial constraint satisfaction problem, the famous SAT and its MAX-SAT variant, graph coloring, various frequency assignment problems [1, 56], and others.

Guided Fast Local Search

In this 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, especially when repeatedly and exhaustively searching the whole neighborhood is computationally expensive.

One factor which affects the efficiency of a local search algorithm is the size of the neighborhood. If too many neighbors 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 [12] the *approximate* 2-Opt method to reduce the neighborhood 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) neighbors that are unlikely to lead to improving moves in order to enhance the efficiency of a search.

The neighborhood chosen for the problem is broken down into a number of small sub-neighborhoods, and an activation bit is attached to each one of them. The idea is to scan continuously the sub-neighborhoods in a given order, searching only those with the activation bit set to 1. These sub-neighborhoods are called active sub-neighborhoods. Sub-neighborhoods with the bit set to 0 are called inactive sub-neighborhoods, and they are not being searched. The neighborhood search process does not restart whenever we find a better solution, but it continues with the next

sub-neighborhood in the given order. This order may be static or dynamic (i.e., change as a result of the moves performed).

Initially, all sub-neighborhoods are active. If a sub-neighborhood is examined and does not contain any improving moves, then it becomes inactive. Otherwise, it remains active and the improving move found is performed. Depending on the move performed, a number of other sub-neighborhoods are also activated. In particular, we activate all the sub-neighborhoods where we expect other improving moves to occur as a result of the move just performed. As the solution improves, the process dies out with fewer and fewer sub-neighborhoods being active until all the sub-neighborhood bits turn to 0. The solution formed up to that point is returned as 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 FLS 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-neighborhoods. The associations to be made are such that for each feature we know which sub-neighborhoods contain moves that have an immediate effect upon the state of the feature (i.e., moves that remove the feature from the solution).

By reducing the size of the neighborhood, 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 neighbors is that some improvements may be missed. The hope is that the gain in "search speed" outweighs the loss in "search quality."

Implementation Guideline

Guided fast local search (GFLS) is more sophisticated than the basic GLS algorithm as it uses a number of sub-neighborhoods, 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.

The following 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.

Fast Local Search

The pseudo-code for fast local search is given in Algorithm 2, where s is the solution, h is the augmented objective function, L is the number of sub-neighborhoods, bit_i is the activation bit for sub-neighborhood i, $MovesForSubneighborhood(i)$

is the method which returns the set of moves contained in sub-neighborhood i, and $SubneighborhoodsForMove(m)$ is the method which returns the set of sub-neighborhoods to activate when move m is performed.

Algorithm 2: The fast local search algorithm

FastLocalSearch(s, h, $[bit_1, \ldots, bit_L]$, L)

```
while ∃bit, bit = 1 do
  {i.e., while active sub-neighborhood exists}
  for i ← 1 until L do
    if bit_i = 1 then
      {search sub-neighborhood i}
      Moves ← MovesForSubneighbourhood(i);
      for all move m in Moves do
        s' ← m(s);
        {s' is the result of move m}
        if h(s') < h(s) then
          {spread activation}
          ActivateSet ← SubneighbourhoodsForMove(m);
          for all sub-neighborhood j in ActivateSet do
            bit_j ← 1;
          end for
          s ← s';
          GOTO: ImprovingMoveFound
        end if
      end for
      {no improving move found}
      bit_i ← 0;
    end if
    ImprovingMoveFound: continue
  end for
end while
return s;
```

As explained earlier, the problem's neighborhood is broken down into a number of sub-neighborhoods, and an activation bit is attached to each one of them. The idea is to examine sub-neighborhoods in a given order, searching only those with the activation bit set to 1. The neighborhood search process does not restart whenever we find a better solution, but it continues with the next sub-neighborhood 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-neighborhoods and not the whole neighborhood, which may be a large one.

The procedure has two points that need to be customized. The first is the breaking down of the neighborhood into sub-neighborhoods (*MovesForSubneighborhood* method in pseudo-code). The second is the mapping from moves to sub-neighborhoods 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, graph, etc.). This allows us to use the same move operators for many different problems (e.g., 1-Opt, 2-Opt, swaps, insertions, etc.).

The method for mapping sub-neighborhoods 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 outermost loop in the move generation code can be used to define the sub-neighborhoods. The moves in each sub-neighborhood will be those generated by the inner loops for the particular sub-neighborhood index value at the outermost loop.

In general, the sub-neighborhoods can be overlapping. Fast local search is usually examining a limited number of moves compared to exhaustive neighborhood 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-neighborhood. Since not all sub-neighborhoods are active in the same iteration, if there is no overlapping, some improving moves may be missed.

The method for spreading activation, denoted by *SubneighbourhoodsForMove*, returns a set of sub-neighborhoods to activate after a move is performed. The lower bound for this set is the sub-neighborhood where the move originated. The upper bound (although not useful) is all the sub-neighborhoods 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-neighborhood 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-neighborhoods turn to zero (i.e., no improving move can be found in any of the sub-neighborhoods). 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 neighborhood and, secondly, 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.

Guided Fast Local Search

The pseudo-code for guided fast local search is given in Algorithm 3, where $FastLocalSearch$ is the fast local search method as described in section "Fast Local Search", $SubneighbourhoodsForFeature(i)$ is the method which returns the set of sub-neighborhoods 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 "Implementation Guideline".

Algorithm 3: The guided fast local search algorithm

GuidedFastLocalSearch(p, g, λ, $[I_1, \ldots, I_M]$, $[c_1, \ldots, c_M]$, M, L)

```
k ← 0;
s_0 ← ConstructionMethod(p);
{set all penalties to 0}
for i ← 1 until M do
    p_i ← 0;
end for
{set all sub-neighborhoods to the active state}
for i ← 1 until L do
    bit_i ← 1;
    {define the augmented objective function}
end for
h ← g + λ * Σ p_i * I_i;
while StoppingCriterion do
    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}
    end for
    for all i such that util_i is maximum do
        p_i ← p_i + 1;
        {activate sub-neighborhoods related to penalized feature i}
        ActivateSet ← SubneighbourhoodsForFeature(i);
        for all sub-neighborhood j in ActivateSet do
            bit_j ← 1;
        end for
    end for
    k ← k + 1;
end while
s* ← best solution found with respect to objective function g;
return s*;
```

This pseudo-code is similar to that of GLS explained in section "Implementation Guideline". 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 neighborhood for improving moves. Subsequent calls to fast local search examine only part of the neighborhood and in particular all the sub-neighborhoods that relate to the features penalized by GLS.

Identifying the sub-neighborhoods 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 "Fast Local Search"). 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-neighborhoods, 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.

GLS Extensions, Hybrids, and Variations

GLS is closely related to other heuristic and meta-heuristic methods. In this section, we shall review the different variations, hybrids, and extensions of GLS and FLS that have been developed in recent years.

Extensions to GLS

GLS is closely related to other heuristic and meta-heuristic methods. This motivates the adoption of ideas borrowed from other meta-heuristics in GLS. For example, taboo lists and aspiration ideas from tabu search have been used in later versions of GLS. Resembling the tabu lists idea, a limited number of penalties are used when GLS is applied to the radio link frequency assignment problem [56]. When the list is full, old penalties are overwritten [81]. The motive is that if too many penalties are built up during the search, the local search could be misguided. In another GLS work, aspiration is used to favor promising moves [54].

Tairan and Zhang in [71] studied how to enhance the performance of GLS through designing a cooperative mechanism based on the proximate optimality principle (POP), resulting in a population-based GLS framework. The idea is to run multiple agents of GLS, and during the search the agents exchange their obtained information about the previous search to make their further search more rational. Based on POP, they suggested that common features that appear in many locally optimal solutions of GLS agents are more likely to be parts of the globally optimal solution. Thus, this property should be taken into consideration in the penalization stage during the search. The effectiveness of the proposed cooperative method was demonstrated through high-quality results obtained in the TSP.

GLS Hybrids

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

As a meta-heuristic method, GLS can also sit on top of genetic algorithms (GA). This has been demonstrated in guided genetic algorithm (GGA) [45–47]. 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 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 (Eq. 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 be changed by these two GA operators. This allows GGA to be more focused 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.

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. [88]. 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 [51]. 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 neighborhood of the modified ES in the second stage. The principle of FLS is also incorporated into the idea of penalty variable neighborhood in which the neighborhood considered by the local search is limited to a small set of the neighbors of the penalized feature.

GLS has also been hybridized with variable neighborhood search (VNS) and large neighborhood search (LNS). Kytojoki et al. [43] combine GLS with VNS in an efficient variable neighborhood 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 [84]. 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 meta-heuristic 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 considers the relative arc length according to the rest of customers ($d_{i,j}/\text{avg}_{i,j}$ rather than $d_{i,j}$, where $\text{avg}_{i,j}$ 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. [10] 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. [35]. 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. [34]. 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 subspace 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.

Variations of GLS

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. [11] introduce 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 coloring problem producing promising results compared to other distributed search algorithms.

Hifi et al. [37] 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.

GLS for Multi-objective Optimization

Most real-world optimization problems are multi-objective in nature. The multi-objective optimization problem (MOP) concerns the optimization of two or more objectives simultaneously. Instead of searching for a global optimum solution as in single-objective optimization problems, the search in MOPs targets a set of solutions representing the optimum set of trade-offs between the objectives. This set is known interchangeably as the Pareto optimum set or the efficient solutions, and the objective values of these solutions are located at the Pareto front (PF). Efficient solutions are nondominant solutions in the sense that improving the value of any one of their objectives must be at the expense of degrading the quality of one or more of the other objectives. Thus, all efficient solutions are considered equivalent as long as there is no further information regarding the relative importance of each of the objectives.

Pareto Local Search

A simple, intuitive adaptation of local search to contain multiple objectives, is to apply the Pareto domination as an acceptance criterion when comparing the current solution to the new one. An archive of nondominant solutions discovered during the search is maintained in order to produce an approximation to the PF. The algorithm stops when the neighborhood of all solutions in the archive have been explored, i.e., the archive is a Pareto local optimum set. Recently, this idea was termed Pareto local search (PLS) and has been developed and extended by several researchers such as in [58]. PLS can be outlined, from a high-level perspective, as follows:

1. It starts with a randomly or heuristically generated solution that is added to the $archive$.
2. A non-visited candidate solution in the $archive$ is randomly chosen, and its neighborhood is explored while applying a first-improvement strategy (i.e., any better solution found is accepted immediately, and its neighborhood is explored for further improvement). This is an iterative step which is applied every time a new better solution is found and accepted.

3. The $archive$ is updated with any nondominated neighbor, i.e., the new neighbor is added to the $archive$ only if it is nondominated by another solution in the $archive$, and then all existing solutions that are dominated by the new neighbor are removed from the $archive$.
4. The exploration of the neighborhood of the current solution stops when all of the neighbors are visited. When the exploration stops, the current solution is considered as a Pareto local optimum.
5. The current solution is marked as being visited.
6. The procedure continues at step 2, while there is a solution in the $archive$ that is not visited.

The PLS returns the $archive$, which is a *Pareto local optimal set*.

Guided Pareto Local Search

Guided Pareto local search (GPLS) [4, 6] extends the guidance approach in GLS to guide PLS to escape Pareto local optimum sets. The only change that GPLS makes to the underlying PLS is the replacement of each objective function g_i with an augmented objective function h_i during the evaluation of neighbor solutions. The augmented functions are not used in the updating procedure of the *archive*, which always depends only on the original objective functions.

In GPLS, the definition of features has to be derived from all objectives. Thus, a set (or multiple sets) of features has to be defined, which will be shared by all solutions. In multi-objective optimization problems, one should take into consideration two different scenarios:

1. All objectives have the same structure (e.g., putting an item in all knapsacks in the knapsack problem or an edge in the multi-objective TSP), and therefore they share one defined feature set. However, the cost of a feature varies according to a particular objective. In order to define the cost of a feature in this case, the influence of the feature on each objective has to be considered and modeled into a single cost function. Such models include (weighted) aggregating approaches and other general functions such as min, max, or mean.
2. A distinct feature set needs to be defined for each objective, and a cost is associated with each feature to describe its influence on the corresponding objective. In this case, features from all feature sets should be considered at the penalization phase to pick one or more features to penalize.

The guidance strategy that GLS employs relies on penalizing some features exhibited by the recent local optimum. As described in section "Guided Local Search", the novelty of GLS is mainly in the way it selects features to penalize. The target is to penalize "bad features" when the local search settles in a local optimum. Two factors affect the utility of a feature, namely, its cost and the frequency of penalizing this feature in previous penalizations.

GPLS deals with a Pareto local optimum set instead of a local optimum. A straightforward penalization strategy is to evaluate the utility of all features exhibited by *any* nondominant solution in the *archive* and penalize features with maximum utility (Eq. 3). However, this simple strategy does not incorporate any information from the *archive* (i.e., the Pareto local optimum set). An example of important information that can be extracted from the *archive* is the number of nondominant solutions that exhibit a particular feature. This is another factor that can be incorporated into the utility function, such that the more Pareto local optima that exhibit a feature, the greater its utility of penalization. Recall that when a feature is penalized, only those Pareto local optima that exhibit the penalized feature will have the chance to escape. Therefore, increasing the utility of penalizing a feature that occurs in many Pareto local optima would enhance the chance of escaping a Pareto local optimum set by restarting from more solutions. It would also help to prevent the search from directing all its efforts toward any particular region of the PF, which therefore leads to a better spread over the PF. Bad features, in terms of their cost, are hoped to be removed either during the next calls of PLS or by future penalization.

The utility function, as stated in Eq. 3, is redefined to incorporate the number of solutions in the Pareto local optimum set that exhibits this feature (γ_i):

$$util_i(archive) = I_i(archive) \times \frac{c_i \times (\gamma_i / |archive|)}{1 + p_i} \tag{6}$$

where *archive* is a Pareto local optimum set, $I_i(archive)$ indicates whether "at least" one solution in the *archive* exhibits feature i, c_i is the feature's cost, p_i is the penalty, and $|archive|$ is the size of the Pareto local optimum set. The feature with the greatest $util$ value will be penalized. When a feature is penalized, its penalty value is always incremented by 1.

This utility function redefines the term "bad feature." If a feature is not exhibited in the Pareto local optimum set (indicated by I_i), then the utility of penalizing it is 0. The higher the cost of this feature (the greater c_i) and the more nondominant solutions exhibiting it (the greater γ_i), the greater the utility of penalizing it. Furthermore, the more times that it has been penalized (the greater p_i), the lower the utility of penalizing it again.

Having penalized a feature, all solutions in the *archive* that exhibit this feature need to be marked as "non-visited," so as to be considered by the PLS method in the next iteration.

The lambda parameter λ is the only parameter to GLS which determines the scaling of the penalty. In multi-objective scenarios each objective ideally has its own lambda, which is calculated as a function of a local optimum with respect to the correspondent objective. Thus, GPLS requires a set of lambda parameters: $[\lambda_1, \cdots, \lambda_k]$, where λ_i is a parameter for the objective h_i. Recall that lambda can be dynamically computed after the first local optimum and before penalties are applied to features for the first time (Eq. 4).

GPLS proves its effectiveness to converge quite quickly, however, at the middle area of the Pareto front [5]. In [6], the speedy convergence property of GPLS is utilized, and the performance of GPLS is further enhanced by coupling it with an efficient initial solution set. Two GPLS-based frameworks are proposed, both of which require such an efficient initial solution set. The first framework applies the standard GPLS, after filling its archive with the initial set. The second framework is a parallel version of GPLS, where each independent GPLS run takes a solution from the initial set as a starting point. GPLS and its frameworks are successfully applied to the 0/1 multi-objective knapsack problem [5], the multi-objective TSP[4], and the empowerment-based workforce scheduling problem [4].

Other Attempts

Apart from GPLS, Alhindi and Zhang [2, 3] propose an idea of using GLS, as a heuristic local search procedure specific for single-objective optimization, in multi-objective evolutionary algorithms (MOEAs). In this work GLS and multi-objective evolutionary algorithm based on decomposition (MOEA/D) [87] are combined to propose a highly efficient algorithm for solving MOPs. To this end, a combination of MOEA/D with GLS, called MOEA/D-GLS, is proposed. In MOEA/D-GLS, a MOP is decomposed into a number of single-objective subproblems. The subproblems are optimized in parallel by using neighborhood information and problem-specific knowledge. In the proposed work, GLS alternates between those subproblems to help them escape local Pareto optimal solutions. More specifically, for a subproblem trapped in a local Pareto optimal solution, the GLS starts from its solution and constructs a transformation function (i.e., augmented function). Then, GLS optimizes the augmented function in order to identify a new better solution. The algorithm has been successfully applied to the multi-objective traveling salesman problem.

GLS Implementation on the Traveling Salesman Problem

For illustration, the implementation details of GLS on the TSP is discussed in this section. The TSP is chosen here as it is the most significant application of GLS. The full details about the application of GLS to the TSP is given in [82].

Problem Description

There are many variations of the 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

$$g(\pi) = \sum_{i=1}^{N} d_{i\pi(i)} \tag{7}$$

and gives the cost function of the TSP.

Local Search

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 i-th element of the vector will contain an identifier for the i-th 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.

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. 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 meta-heuristic 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.

Improvement Method

Most improvement methods for the TSP are based on the k-Opt moves. Using k-Opt moves, neighboring 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 [20], 3-Opt [48], and *Lin-Kernighan (LK)* [49].

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 neighboring 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, e2 are removed and edges e3, e4 are

added with lengths d1, d2, d3, d4, respectively. The change in cost is the following:

$$d3 + d4 - d1 - d2 \tag{8}$$

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.

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 Eq. (7)). As mentioned in section "Routing/Scheduling Problems", 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 "Guided Local Search", 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:

$$D' = D + \lambda \cdot P = [d_{ij} + \lambda \cdot p_{ij}] \tag{9}$$

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 (Eq. 8) 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 (Eq. 8) is as follows:

$$(d3 + d4 - d1 - d2) + \lambda * (p3 + p4 - p1 - p2) \tag{10}$$

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 (Eq. 3), which for the TSP takes the form:

$$Util(tour, e_{ij}) = I_{e_{ij}}(tour) \cdot \frac{d_{ij}}{1 + p_{ij}}, \tag{11}$$

where

$$I_{e_{ij}}(tour) = \begin{cases} 1, & e_{ij} \in tour \\ 0, & e_{ij} \notin tour \end{cases} \tag{12}$$

The only parameter of GLS that requires tuning is parameter λ. Alternatively, we can tune the α parameter which is defined in section "Implementation Guideline" 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 [62]:

$$1/8 \leq \alpha \leq 1/2 \tag{13}$$

The reader may refer to [82] for more details on the experimentation procedure and the full set of results.

Guided Fast Local Search

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

The sub-neighborhoods 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-neighborhoods 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-neighborhoods will include moves that remove one or more of the penalized edges.

GLS/GPLS Implementation on a Workforce Scheduling Problem

The workforce scheduling problem (WSP) is another application of GLS, which is chosen here as another illustrative example. It is chosen since both GLS and GPLS have been applied to two different versions of WSP. The implementation of GLS/GPLS components is provided here, and for full details, the reader is referred to [4, 7, 76].

Problem Description

The WSP is basically the problem of allocating a set of technicians (resources), $R = \{r_1, r_2, \ldots, r_{|R|}\}$, to a set of tasks, $T = \{t_1, t_2, \ldots, t_{|T|}\}$. A task t is described by a 5-tuple:

$$< c_t, dur_t, reqSkill_t, [start_t, end_t], loc_t >$$

where c_t is a predefined priority which determines its importance to the company. The higher the value of c_t, the more important the task will be, and $c_t \in \Re$. dur_t is the expected duration a technician requires to finish this task. Each task requires a technician with a particular skill $reqSkill_t \in SkillSet$, where $SkillSet$ is the set of all skills: $SkillSet = \{skill_1, \ldots, skill_{|S|}\}$. A task t must be serviced within a predefined time window described by $[start_t, end_t]$. Tasks are geographically distributed, and the location of a task is denoted by loc_t.

Each technician $r \in R$ is described by the following triple:

$$< [start_r, end_r], skills_r, loc_r >$$

Each technician has limited shift hours where the beginning and end of the shift are expressed by $[br_r, er_r]$. $skills_r$ denotes the skill(s) a technician has, where $skills_r \subseteq SkillSet$. There is no single depot for technicians to start from, as they can start from home or from one of predefined depots. The location of a technician is denoted by loc_r.

There are two sets of decision variables in the WSP: the allocation variables $X = \{x_{rt} | r \in R; t \in T; x_{rt} \in \{0, 1\}\}$ and the service times $ServTime = \{st_t | t \in T\}$. A variable x_{rt} is set to 1 if the technician r is allocated to the task t and 0 otherwise. A variable st_t denotes the start time of service for task t.

Having decided these variables, a set of routes $\pi = \{\pi_r | r \in R\}$ are defined. A route π_r is a sequence of tasks ($\subseteq T$) that are to be visited by technician r; $\pi_r = (\pi_{r1}, \cdots, \pi_{r\sigma_r}), 1 \leq \sigma_r \leq |T|$.

A main objective is to maximize the number of allocated tasks with respect to their priorities, while satisfying all assignment and routing constraints.

The WSP can, then, be mathematically modeled as follows:

$$\max \qquad \frac{\sum_{r \in R} \sum_{t \in T} c_t x_{rt}}{\sum_{t \in T} c_t} \tag{14}$$

subject to

$$\sum_{r \in R} x_{rt} \leqslant 1 \quad \forall t \in T \tag{15}$$

$$reqSkill_t \in skills_r \quad \forall x_{rt} = 1, t \in T, r \in R \tag{16}$$

$$start_t \leqslant st_t \quad \forall t \in T \tag{17}$$

$$st_t + dur_t \leqslant end_t \quad \forall t \in T \tag{18}$$

$$start_r \leqslant st_{\pi_{ri}} \quad \forall r \in R, i = \{1..\sigma_r\} \tag{19}$$

$$st_{\pi_{ri}} + dur_{\pi_{ri}} \leqslant end_r \quad \forall r \in R, i = \{1..\sigma_r\} \tag{20}$$

$$start_r + trvl_{loc_r loc_{\pi_{r1}}} \leqslant st_{\pi_{r1}} \forall r \in R \tag{21}$$

$$st_{t_i} + dur_{t_i} + trvl_{loc_{t_i} loc_{t_i+1}} \leqslant st_{t_{i+1}} \quad \forall t_i \in \pi_r, \pi_r \in \pi \tag{22}$$

The objective function is represented by (1) which is normalized by dividing by the sum of all the costs associated with tasks. Constraint (2) imposes that each task is visited at most once. The skill constraint is expressed in (3). The time window constraints of all tasks are assured by (4) and (5). (6) and (7) ensure that all tasks which are assigned to a technician must be within the technician's working time. Finally, constraints (8) and (9) ensure route validity by considering the traveling time between a technician's base location and the first task, as well as between subsequent tasks in technician's route.

Empowerment in Workforce Scheduling

Empowerment scheduling [4, 7] is a term introduced to refer to scheduling approaches that incorporate the empowerment concept in the scheduling system by involving technicians in the task assignment decision. The hope of empowering human resources is to boost their contribution and commitment to the organization. In [4], an empowerment scheduling model is proposed for the workforce scheduling problem, named as the EmS-WSP, by enabling technicians to express their requests/preferences and submit plans for consideration in the allocation process. Naturally all plans can be expressed as a constraint relationship. These constraints will be considered to be satisfied in the scheduling process. Workforce scheduling problems are very complex, and thus the promise of satisfying all the plans cannot be guaranteed. One direct way to handle employees' work plans is to treat them as soft constraints and associate a cost to each plan when it is not satisfied. In this case the task involves finding a schedule that satisfies the majority of constraints, rather than all the constraints. A key property of the empowerment scheduling model proposed in [7] is in the efficient strategy that dynamically determines the incurred cost of violating each plan (see [7] for more details).

Formally, every technician $r \in R$ is given an opportunity to provide a work plan (wp_r) every day. A plan is basically a constraint (ρ) by which a technician can describe the jobs he/she wants to undertake. For instance, a technician r might want to be allocated to jobs of a particular set of skills or jobs in a particular region. Each plan is associated with a weight (ω); in our case this is equal to the EP value of r.

An indicator y_r ($\forall r \in R$) is defined which is equal to 1 if the plan ρ_r is satisfied and 0 otherwise.

The WSP is reformulated in the EmS-WSP to not only maximize the productivity in terms of the number of served jobs (Eq. 14) but also to satisfy the maximum number of working plans. The formulation of the additional objective is as follows:

$$wp_r = < \rho_r, \omega_r > \quad \forall r \in R, \quad 0 \leqslant \omega_r \leqslant 1 \tag{23}$$

$$\max \quad \frac{\sum_{r \in R} y_r \omega_r}{\sum_{r \in R} \omega_r} \tag{24}$$

Local Search

Before going to the implementation details, it is worth mentioning that the two objectives in the EmS-WSP can be transformed to a single-objective using an aggregation approach and then solved using a single-objective optimizer such as GLS. Otherwise, a multi-objective optimizer such as GPLS can be used to optimize the two objectives simultaneously. Since both GLS and GPLS are similar in the definition of their basic components, the following implementation details are for both algorithms.

Solution Representation

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 "GLS Implementation on the Traveling Salesman Problem"), modified in 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 costs of the input permutation, which is the cost (as defined in Eqs. 14 and 24) of the schedule thus created, are returned. These costs will be somehow aggregated in a single cost by GLS or maintained simultaneously by GPLS.

Improvement Method

Given a permutation, the 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 to be evaluated (using the evaluation procedure above) and compared to the current 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, swaps, etc. In this case, the solution representation and construction methods need to be revised. The reader may refer to other works (e.g., [10]) for more information on the application of GLS to the VRP.

Feature Definition

Since there are two objectives of different natures in the EmS-WSP, two sets of features need to be defined. As maximizing the total allocated tasks is an objective, each feature in the first set represents the inability to serving a task, so as to bias the local search to serve tasks of high importance. Thus, for each task t, we define a feature $I_t(schedule)$, which is equal to 1 if t is unallocated in $schedule$ and 0 otherwise. The cost of this feature is given by the task priority (c_t) which is equal to the cost incurred in the cost function (Eq. 14) when a task is unallocated.

The second objective concerns maximizing the total satisfied employees' plans, and therefore each feature in the second feature set represents the inability to satisfy a work plan, so as to bias the (Pareto) local search to satisfy plans of high weight. Thus, for each technician r in the problem, we define a feature $I_r(schedule) = y_r$, where y_r is equal to 1 if the work plan of r (ρ_r) is satisfied and 0 otherwise. The cost of this feature is given by the working plan's weight (ω) which is equal to the cost incurred in the cost function (Eq. 24) when a plan is unsatisfied.

Overview of GLS Applications

GLS and its descendants have been applied to a number of nontrivial problems and have achieved state-of-the-art results.

Routing/Scheduling Problems

One of the most successful application of GLS and FLS is the *traveling 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 [49, 50]. We tested GLS + FLS + 2Opt against LK [82] on a set of benchmark problems from a public TSP library [62]. Given the same amount of time, GLS + FLS + 2Opt found better results than LK on average. The outstanding performance of GLS+FLS+2Opt was also demonstrated in comparison to simulated annealing [39], tabu search [41], and genetic algorithm [30] 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 extensions and hybrids have also been proposed for the TSP, including a population-based GLS [71] and the combination of GLS with memetic algorithms [38] and also with the dynamic programming-based dynasearch technique with encouraging preliminary results reported in [18]. The multi-objective TSP has been tackled by GPLS [4] and a GLS hybrid with MOEA/D [3].

Padron and Balaguer [57] have applied GLS to the related rural postman problem (RPP), Vansteenwegen et al. [78] applied GLS to the related team orienteering problem (TOP), and Mester et al. [52] applied the guided evolution strategy hybrid meta-heuristic to a genetic ordering problem (a Unidimensional Wandering Salesperson Problem, UWSP).

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 [40]. As a result, their work was incorporated in Dispatcher, a commercial package developed by ILOG [10]. 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 [89] and to the capacitated arc routing problem [13]. 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 [85], the VRP with simultaneous pickup and delivery [86], and the VRP with replenishment facility [74]. GLS with VNS [43] and GLS with ES [51] hybrids have been proposed to solve large-scale VRPs. A new hybridization between GLS and GA has been devised to effectively tackle the transit network design problem in [61]. The deployment of fiber optics in the telecommunication industry has been successfully optimized using GLS in [19] and [66].

Assignment Problems

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 [17], with improved robustness [47].

GLS hybrids have been proposed for the related QAP. Zhang et al. [88] proposed the GLS/EDA hybrid meta-heuristic. In addition, the hybrid of GLS with ACO (ACO_GLS) has been applied to a variation of the QAP [35]. This encourages Daoud et al. [21] to couple GLS with ACO, GA, and PSO into three frameworks which are successfully applied to the robotic assembly line balancing problem.

In the *radio link frequency assignment problem* (RLFAP), the task is to assign available frequencies to communication channels satisfying constraints that prevent interference [14]. In some RLFAPs, the goal is to minimize the number of frequencies used. Bouju et al. [14] is an early work that applied GENET to radio length frequency assignment. For the CALMA set of benchmark problems, which has been widely used, GLS+FLS reported the best results compared to all work published previously [80]. 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 [81]. At the time, GLS and GGA held some of the best-known results in the CALMA set of benchmark problems.

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 coloring, and the car sequencing problem. Some of their work are available for download over the Internet from Essex University's website [33] but are largely undocumented due to lack of time during the original development phase of the algorithm.

GGA was also successfully applied to the processor configuration problem, with new better results than those found by other previously reported algorithms [45,46].

GLS and FLS have been successfully applied to the 3-D bin packing problem and its variants [26, 44], VLSI design problems [27], and network planning problems [29, 84]. GLS has been applied to the natural language parsing problem [22], graph set T-coloring problem [16], query reformulation [55], and ligand docking problems via drug design [59]. Variations of GLS have been applied to graph coloring [11] and the multidimensional knapsack problem [37, 70]. The multi-objective 0/1 knapsack problem has been tackled using GPLS in [6].

Resource Allocation Problems

In the *workforce scheduling problem* (WSP) [9], 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 traveling cost, overtime cost, and unserved jobs. In the WSP, GLS + FLS holds the best-published

results for the benchmark problem available to the authors [76]. GPLS has also been applied to a multi-objective WSP in [4].

Constrained Optimization 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. [31]).

GLSSAT, an extension of GLS, was applied to both the SAT and weighted MAX-SAT problem [53]. 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 [67], WalkSAT [64], and GRASP [63].

GLS in Commercial Packages

GLS and FLS have been incorporated into commercial software packages, namely, iOpt, which is a software toolkit for heuristic search methods [83], and iSchedule [24], which is an extension of iOpt for planning and scheduling applications (e.g., for solving problems such as the VRP [25]).

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 [32] 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. Solutions 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 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 neighborhood 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.

Despite all the years of development, further research is possible to improve GLS and its related techniques 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, define sub-neighborhoods, etc.) This makes GLS and GFLS easier to use by nonspecialist software engineers.

Cross-References

▸ Iterated Local Search
▸ Simulated Annealing
▸ Tabu Search
▸ Variable Neighborhood Search

References

1. Aardal K, Van Hoesel S, Koster A, Mannino C, Sassano A (2007) Models and solution techniques for frequency assignment problems. Ann Oper Res 153(1):79–129
2. Alhindi A (2015) Multiobjective evolutionary algorithm based on decomposition with advanced local search methods. PhD thesis, Department of computer science, University of Essex, Colchester
3. Alhindi A, Zhang Q (2013) MOEA/D with guided local search: some preliminarily experimental results. In: 5th computer science and electronic engineering conference (CEEC), Colchester, pp 109–114
4. Alsheddy A (2011) Empowerment scheduling: a multi-objective optimization approach using guided local search. PhD thesis, Department of computer science, University of Essex, Colchester
5. Alsheddy A, Tsang EPK (2009) Guided pareto local search and its application to the 0/1 multi-objective knapsack problems. In: Metaheuristics international conference (MIC2009), Hamburg
6. Alsheddy A, Tsang EPk (2010) Guided pareto local search based frameworks for biobjective optimization. In: IEEE congress on evolutionary computation (CEC), Barcelona, pp 1–8
7. Alsheddy A, Tsang EPk (2011) Empowerment scheduling for a field workforce. J Sched 14(6):639–654
8. Anderson CA, Fraughnaugh K, Parker M, Ryan J (1993) Path assignment for call routing: an application of tabu search. Ann Oper Res 41:301–312
9. Azarmi N, Abdul-Hameed W (1995) Workforce scheduling with constraint logic programming. BT Technol J 13(1):81–94
10. Backer BD, Furnon V, Shaw P, Kilby P and Prosser P (2000) Solving vehicle routing problems using constraint programming and metaheuristics. J Heuristics 6(4):501–523
11. Basharu M, Arana I, Ahriz H (2005) Distributed guided local search for solving binary DisCSPs. In: Proceedings of FLAIRS 2005. AAAI Press, Clearwater Beach, pp 660–665
12. Bentley JJ (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Comput 4:387–411
13. Beullens P, Muyldermans L, Cattrysse D, Van Oudheusden D (2003) A guided local search heuristic for the capacitated arc routing problem. Eur J Oper Res 147(3):629–643
14. Bouju A, Boyce JF, Dimitropoulos CHD, vom Scheidt G, Taylor JG (1995) Intelligent search for the radio link frequency assignment problem. In: Proceedings of the international conference on digital signal processing, Limassol
15. Castillo-Salazar A, Landa-Silva D, Qu R (2016) Workforce scheduling and routing problems: literature survey and computational study. Ann Oper Res 239(1):39–67
16. Chiarandini M, Stutzle T (2007) Stochastic local search algorithms for graph set T-colouring and frequency assignment. Constraints 12(3):371–403
17. Chu P, Beasley JE (1997) A genetic algorithm for the generalized assignment problem. Comput Oper Res 24:17–23
18. Congram RK, Potts CN (1999) Dynasearch algorithms for the traveling salesman problem. In: Presentation at the travelling salesman workshop, CORMSIS, University of Southampton, Southampton
19. Cramer S, Kampouridis M (2015) Optimising the deployment of fibre optics using guided local search. In: Proceedings of the 2015 IEEE congress on evolutionary computation (CEC). IEEE Press, Sendai, pp 799–806
20. Croes A (1958) A method for solving traveling-salesman problems. Oper Res 5:791–812
21. Daoud S, Chehade H, Yalaoui F, Amodeo L (2014) Solving a robotic assembly line balancing problem using efficient hybrid methods. J Heuristics 20(3):235–259
22. Daum M, Menzel W (2002) Parsing natural language using guided local search. In: Proceedings of 15th European conference on artificial intelligence (ECAI-2002), Lyon, pp 435–439

23. Davenport A, Tsang EPK, Wang CJ, Zhu K (1994) GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In: Proceedings of 12th national conference for artificial intelligence (AAAI), Seattle, 325–330
24. Dorne R, Voudouris C, Liret A, Ladde C, Lesaint D (2003) iSchedule – an optimisation tool-kit based on heuristic search to solve BT scheduling problems. BT Technol J 21(4):50–58
25. Dorne R, Mills P, Voudouris C (2007) Solving vehicle routing using iOpt. In: Doerner KF et al (eds) Metaheuristics: progress in complex systems optimization. Operations research/computer science interfaces series, vol 39. Springer, New York, pp 389–408
26. Egeblad J, Nielsen B, Odgaard A (2007) Fast neighbourhood search for two- and three-dimensional nesting problems. Eur J Oper Res 183(3):1249–1266
27. Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for final placement in VLSI design. J Heuristics 9:269–295
28. Flood MM (1956) The traveling-salesman problem. In: Operations research, vol 4. Columbia University, New York, pp 61–75
29. Flores Lucio G, Reed M, Henning I (2007) Guided local search as a network planning algorithm that incorporates uncertain traffic demands. Comput Netw 51(11):3172–3196
30. Freisleben B, Merz P (1996) A genetic local search algorithm for solving symmetric and asymmetric travelling salesman problems. In: Proceedings of the 1996 IEEE international conference on evolutionary computation. IEEE Press, Piscataway, pp 616–621
31. Gent IP, van Maaren H, Walsh T (2000) SAT2000, highlights of satisfiability research in the year 2000. Frontiers in artificial intelligence and applications. IOS Press, Amsterdam/Washington, DC
32. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Boston
33. GLS Demos (2008). http://cswww.essex.ac.uk/CSP/glsdemo.html
34. Gomes N, Vale Z, Ramos C (2003) Hybrid constraint algorithm for the maintenance scheduling of electric power units. In: Proceeding Of international conference on intelligent systems application to power systems (ISAP 2003), Lemnos
35. Hani Y, Amodeo L, Yalaoui F, Chen H (2007) Ant colony optimization for solving an industrial layout problem. Eur J Oper Res 183(2):633–642
36. Hansen P, Mladenović N, Todosijević R (2016) Variable neighborhood search: basics and variants. EURO J Comput Optim 1–32
37. Hifi M, Michrafy M, Sbihi A (2004) Heuristic algorithms for the multiple-choice multidimensional Knapsack problem. J Oper Res Soc 55:1323–1332
38. Holstein D, Moscato P (1999) Memetic algorithms using guided local search: a case study. In: Corne D, Glover F, Dorigo M (eds) New ideas in optimisation. McGraw-Hill, London, pp 235–244
39. Johnson D (1990) Local optimization and the traveling salesman problem. In: Proceedings of the 17th colloquium on automata languages and programming. Lecture notes in computer science, vol 443. Springer, London, pp 446–461
40. Kilby P, Prosser P, Shaw P (1999) Guided local search for the vehicle routing problem with time windows. In: Voss S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic, Boston, pp 473–486
41. Knox J (1994) Tabu search performance on the symmetric traveling salesman problem. Comput Oper Res 21(8):867–876
42. Koopman BO (1957) The theory of search, part III, the optimum distribution of search effort. Oper Res 5:613–626
43. Kytöjoki J, Nuortio T, Bräysy O, Gendreau M (2007) An efficient variable neighbourhood search heuristic for very large scale vehicle routing problems. Comput Oper Res 34(9): 2743–2757
44. Langer Y, Bay M, Crama Y, Bair F, Caprace JD, Rigo P (2005) Optimization of surface utilization using heuristic approaches. In: Proceedings of the international conference COMPIT'05, Hamburg, pp 419–425
45. Lau TL (1999) Guided genetic algorithm. PhD thesis, Department of computer science, University of Essex, Colchester

46. Lau TL, Tsang EPK (1998) Guided genetic algorithm and its application to the radio link frequency allocation problem. In: Proceedings of NATO symposium on frequency assignment, sharing and conservation in systems (AEROSPACE), Aalborg, AGARD, RTO-MP-13, paper No.14b
47. Lau TL, Tsang EPK (1998) The guided genetic algorithm and its application to the general assignment problem. In: IEEE 10th international conference on tools with artificial intelligence (ICTAI'98), Taiwan, pp 336–343
48. Lin S (1965) Computer solutions of the traveling-salesman problem. Bell Syst Tech J 44: 2245–2269
49. Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
50. Martin O, Otto SW (1966) Combining simulated annealing with local search heuristics. Ann Operat Res 63(1):57–75
51. Mester D, Bräysy O (2005) Active guided evolution strategies for large-scale vehicle routing problems with time windows. Comput Oper Res 32(6):1593–1614
52. Mester DI, Ronin YI, Nevo E, Korol AB (2004) Fast and high precision algorithms for optimization in large-scale genomic problems. Comput Biol Chem 28(4):281–290
53. Mills P, Tsang EPK (2000) Guided local search for solving SAT and weighted MAX-SAT problems. J Autom Reason 24:205–223
54. Mills P, Tsang E, Ford J (2003) Applying an extended guided local search to the quadratic assignment problem. Ann Ope Res 118:1–4/121–135
55. Moghrabi I (2006) Guided local search for query reformulation using weight propagation. Int J Appl Math Comput Sci (AMCS) 16(4):537–549
56. Murphey RA, Pardalos PM, Resende MGC (1999) Frequency assignment problems. In: Du D-Z, Pardalos P (eds) Handbook of combinatorial optimization. vol 4, Kluwer Academic, Boston
57. Padron V, Balaguer C (2000) New methodology to solve the RPP by means of isolated edge. In: Tuson A (ed) Cambridge conference tutorial papers. Young OR, vol 11. Operational Research Society, UK
58. Paquete L, Chiarandini M, Stützle T (2004) Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. In: Gandibleux X (ed) Metaheuristics for multiobjective optimisation, vol 535. Springer, Berlin/New York, pp 177–199
59. Peh S, Hong J (2016) GLSDock – drug design using guided local search. In: Proceedings of the 2016 international conference on computational science and its applications, Beijing, pp 11–21
60. Pesant G, Gendreau M (1999) A constraint programming framework for local search methods. J Heuristics 5(3)255–279
61. Rahman MK, Nayeem MA, Rahman MS (2015) Transit network design by hybrid guided genetic algorithm with elitism. In: Proceedings of the 2015 conference on advanced systems for public transport (CASPT), Rotterdam
62. Reinelt G (1991) A traveling salesman problem library. ORSA J Comput 3:376–384
63. Resende MGC, Feo TA (1996) A GRASP for satisfiability. In: Johnson DS, Trick MA (eds) Cliques, coloring, and satisfiability: second DIMACS implementation challenge. DIMACS series on discrete mathematics and theoretical computer science, vol 26. American Mathematical Society, Providence, pp 499–520
64. Selman B, Kautz H (1993) Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: Proceeding of 13th international joint conference on AI, Chambéry, pp 290–295
65. Selman B, Levesque HJ, Mitchell DG (1992) A new method for solving hard satisfiability problems. In: Proceedings of AAAI-92, San Jose, pp 40–446
66. Shaghaghi A, Glover T, Kampouridis M, Tsang E (2013) Guided local search for optimal GPON/FTTP network design. In: Chaki N et al (eds) Computer networks & communications (NetCom): proceedings of the fourth international conference on networks & communications. Springer, New York, pp 255–263

67. Shang Y, Wah BW (1998) A discrete Lagrangian-based global-search method for solving satisfiability problems. J Glob Optim 12(1):61–99
68. Simon HU (1989) Approximation algorithms for channel assignment in cellular radio networks. In: Proceedings 7th international symposium on fundamentals of computation theory. Lecture notes in computer science, vol 380. Springer, Berlin/New York, pp 405–416
69. Stone LD (1983) The process of search planning: current approaches and continuing problems. Oper Res 31:207–233
70. Tairan N, Algarni A, Varghese J, Jan M (2015) Population-based guided local search for multidimensional knapsack problem. In: Proceedings of the 2015 fourth international conference on future generation communication technology (FGCT), Luton, pp 1–5
71. Tairan N, Zhang Q (2013) P-GLS-II: an enhanced version of the population-based guided local search. In: Proceedings of the 13th annual conference on genetic and evolutionary computation (GECCO), Dublin, pp 537–544
72. Tamura H, Zhang Z, Tang Z, Ishii M (2006) Objective function adjustment algorithm for combinatorial optimization problems. IEICE Trans Fundam Electron Commun Comput Sci E89-A:9:2441–2444
73. Tarantilis CD, Zachariadis EE, Kiranoudis CT (2008) A guided tabu search for the heterogeneous vehicle routeing problem. J Oper Res Soc 59(12):1659–1673
74. Tarantilis CD, Zachariadis EE, Kiranoudis CT (2008) A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. INFORMS J Comput 20(1)154–168
75. Tikhonov AN, Arsenin VY (1977) Solutions of ill-posed problems. Wiley, New York
76. Tsang EPK, Voudouris C (1997) Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Oper Res Lett 20(3):119–127
77. Tsang EPK, Wang CJ, Davenport A, Voudouris C, Lau TL (1999) A family of stochastic methods for constraint satisfaction and optimisation. In: Proceedings of the first international conference on the practical application of constraint technologies and logic programming (PACLP), London, pp 359–383
78. Vansteenwegen P, Souffriau W, Berghe G, Oudheusden D (2009) A guided local search metaheuristic for the team orienteering problem. Eur J Oper Res 196(1):118–127
79. Voudouris C (1997) Guided local search for combinatorial optimisation problems. PhD thesis, Department of computer science, University of Essex, Colchester
80. Voudouris C, Tsang EPK (1996) Partial constraint satisfaction problems and guided local search. In: Proceedings of PACT'96, London, pp 337–356
81. Voudouris C, Tsang E (1998) Solving the radio link frequency assignment problems using guided local search. In: Proceedings of NATO symposium on frequency assignment, sharing and conservation in systems (AEROSPACE), Aalborg, AGARD, RTO-MP-13, paper No. 14a
82. Voudouris C, Tsang EPK (1999) Guided local search and its application to the travelling salesman problem. Eur J Oper Res 113(2):469–499
83. Voudouris C, Dorne R, Lesaint D, Liret A (2001) iOpt: a software toolkit for heuristic search methods. In: Walsh T (ed) Practice of constraint programming – CP 2001, Paphos. Lecture notes in computer science, vol 2239, pp 716–729
84. Xiaohu T, Haubrich H-J (2005) A hybrid metaheuristic method for the planning of medium-voltage distribution networks. In: Proceedings of 15th power systems computation conference (PSCC 2005), Liege
85. Zachariadis E, Tarantilis C, Kiranoudis C (2009) A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. Eur J Oper Res 195(3):729–743
86. Zachariadis E, Tarantilis C, Kiranoudis C (2009) A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. Expert Syst Appl 36(2):1070–1081

87. Zhang Q, Hui L (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11:712–731
88. Zhang Q, Sun J, Tsang EPK, Ford J (2003) Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem. In: Bird of a feather workshops, genetic and evolutionary computation conference, Chicago
89. Zhong Y, Cole MH (2005) A vehicle routing problem with backhauls and time windows: a guided local search solution. Transp Res E Logist Transp Rev 41(2):131–144