Chapter 1

HEURISTIC SEARCH FOR NETWORK DESIGN

Ioannis Gamvros, Bruce Golden, S. Raghavan, and Daliborka Stanojević
The Robert H. Smith School of Business
University of Maryland
College Park, MD 20742-1815
{igamvros,bgolden,rraghava,dstanoje}@rhsmith.umd.edu

Abstract In this chapter, we focus on heuristics for network design problems. Network design problems have many important applications and have been studied in the operations research literature for almost 40 years. Our goal here is to present useful guidelines for the design of intelligent heuristic search methods for this class of problems. Simple heuristics, local search, simulated annealing, GRASP, tabu search, and genetic algorithms are all discussed. We demonstrate the effective application of heuristic search techniques, and in particular genetic algorithms, to four specific network design problems. In addition, we present a selected annotated bibliography of recent applications of heuristic search to network design.

Keywords: Heuristics, Local Search, Network Design, Genetic Algorithms

1. Introduction

Network design problems arise in a wide variety of application domains. Some examples are telecommunications, logistics and transportation, and supply chain management. While the core network design problems such as the minimum spanning tree problem, and the shortest path problem are well-solved, adding additional (or different) restrictions on the network design frequently results in an $\mathcal{NP}$-complete problem. Coincidentally, most network design problems that arise in practice are $\mathcal{NP}$-complete. When it is difficult to solve these network design problems using an exact approach, we are interested in finding good heuristic solutions to them.

In this chapter, we provide practitioners guidelines for designing heuristic search techniques for network design problems. Our guidelines encompass a wide variety of search techniques starting from simple heuristics, to local search and large-scale neighborhood search, and finally to genetic algorithms. We illustrate these guidelines by sharing our experience in developing and applying these heuristic search techniques to four different network design problems: the minimum labeling spanning tree problem, the Euclidean non-uniform Steiner tree problem, the prize collecting generalized minimum spanning tree problem, and the multi-level capacitated spanning tree problem. Finally, we provide an annotated bibliography to selectively describe some recent research work in this area.

We hope to convey through this chapter some of the key ideas in developing good heuristic search techniques for network design problems. In addition, we illustrate the level of success that one can expect when applying heuristic search to specific problems involving network design.

The rest of this chapter is organized as follows. In Section 1.2 we describe a set of general guidelines for developing heuristic search procedures for network design problems. We then illustrate, in Sections 1.3–1.6, the application of these guidelines on four different network design problems. In Section 1.7 we provide a brief annotated bibliography on some recent and successful applications of heuristic search techniques to network design problems. Finally, in Section 1.8 we provide concluding remarks.

2. Guidelines for Network Design Heuristics

In this section we present a set of guidelines to develop heuristic solution approaches for network design problems. Our goal is to inform the reader about popular approaches that have been favored in the literature, discuss their strengths and weaknesses, and provide some notes on how they can be used for related problems. The reader should keep in mind that in almost every case procedures that achieve a high level of performance take advantage of problem-specific structures. However, we believe, that most network design problems share many common characteristics and often the search for an efficient algorithm can follow the same steps and adhere to the same principles regardless of the problem. We will begin by presenting simple heuristic procedures that are usually fairly shortsighted and obtain moderate to poor solutions. We then look at more advanced local search (LS) procedures that rely on the definition of a neighborhood and specify the way in which this neighborhood is explored. In the context of local search we also discuss metaheuris-

tic procedures, such as simulated annealing, GRASP, and tabu search. Finally, we discuss genetic algorithms (GA), which usually require a significant amount of computation time but can find very high quality solutions.

2.1 Simple Heuristics

It is always a good idea to experiment with simple heuristics first, before exploring more complex procedures. They usually provide initial feasible solutions for more advanced procedures, act as a benchmark against which other algorithms are compared, and, if nothing else, help better understand the structure of the problem. Moreover, these simple heuristics are usually employed by more complex procedures for solving smaller subproblems efficiently. Conceptually, we can split the different types of heuristics into three different categories depending on how they find solutions.

Construction. The most popular approach is to attempt to find a solution by starting with the problem graph, without any edges, and to construct the solution by adding edges one at a time. Usually this is done by selecting the *best* edge, according to a specified rule, and adding it to the solution only if it does not violate feasibility. The simplest example of this is Kruskal's algorithm [36] for the Minimum Spanning Tree (MST) problem in which we select the cheapest edge in the graph and add it to the solution if it does not create a cycle. Another equally simple and well-known example is Prim's algorithm [46] for the same problem where the rule is to choose the cheapest edge that connects the nodes in the partially constructed tree T with $T \setminus V$ (V is the node set of the graph, and initially T is a random vertex).

Deletion. A less favored approach works in the opposite way. That is, to start with all the edges in the graph and delete the *worst* edge at each iteration if its deletion results in a network that contains a feasible solution. The procedure stops when no edges in the solution can be deleted. Typically, for minimization problems, the edges are selected for deletion by decreasing order of costs.

Savings. A somewhat different approach starts with a very simple solution and attempts to improve that solution by calculating the savings generated when replacing existing edges with new edges while maintaining feasibility. For tree network design problems these approaches usually start with a MST or star solution (generated by connecting all nodes to a given node). For example, the Esau-Williams heuristic [18]

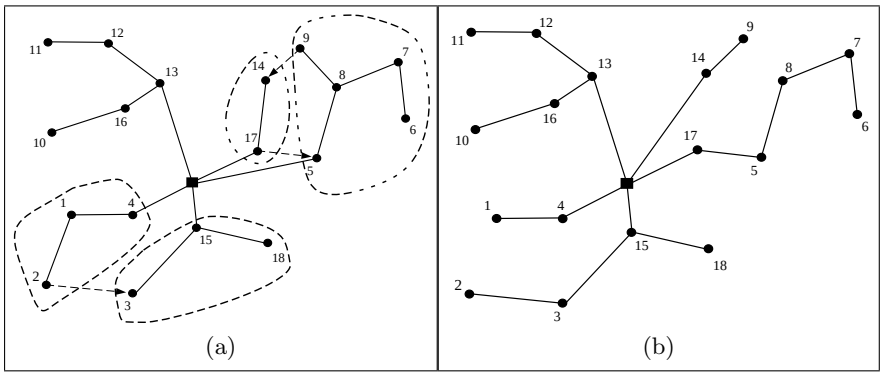

Figure 1.1. Two-exchange example for the CMST problem. (a) Original solution with the proposed exchanges. (b) Final solution after the exchanges.

for the capacitated minimum spanning tree (CMST) problem and the Clarke and Wright heuristic [9] for the vehicle routing problem (VRP) are based on the savings approach.

The first two approaches are easy to understand and implement and usually have very good running times (depending on the feasibility check) but typically result in very poor solutions, especially for more complex problems. The savings approach can give slightly better solutions and has been used extensively in practice. It is advantageous to look into all three types of simple heuristics since their performance is problem dependent (i.e., some of them may work better in some cases than others).

2.2 Local Search

All of the above heuristics, although fast and easy to implement, have the significant drawback of sometimes generating poor quality solutions (especially for large problems). One approach that is widely used to improve on the simple heuristic solutions uses the notion of a neighborhood of feasible solutions and defines a search procedure to find a more promising solution from the neighborhood. In each iteration of a LS algorithm, the neighborhood of the current solution is explored for a suitable alternative solution. If one is found, then it replaces the current solution and the procedure starts over. If none is found, then the procedure terminates. In the case of LS, both the neighborhood structure and the search algorithm play a significant role in the success of the procedure and have to be designed carefully.

Neighborhood. Simple neighborhood structures are usually defined through exchanges between nodes or edges and are evaluated by

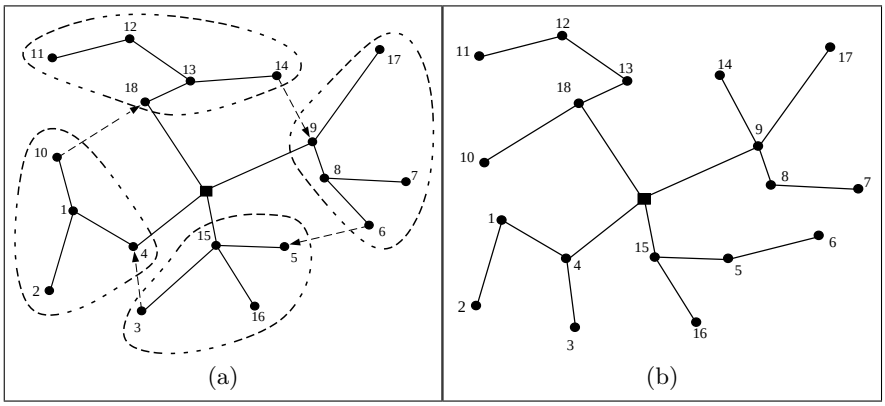

Figure 1.2. Multi-exchange example for the CMST. (a) Original solution with proposed cyclic exchange. (b) Final solution after exchanges.

the savings generated much like the ones used in the savings heuristic we discussed earlier. A popular way of generating neighboring solutions is to first partition the node set in the graph into disjoint sets of nodes and then define exchanges between the different sets. This approach relies on the fact that solving the problem on the partition sets is usually much easier than solving it over the entire set of nodes. An example of a neighborhood used in this context is the so-called node-based, two-exchange neighborhood (e.g., see Amberg et al. [4]) in which we perform a shift move for a single node (i.e., the node moves from its current set to a new one) or an exchange move between two nodes (i.e., the two nodes exchange sets). In Figure 1.1 we give an example of a shift and an exchange move and the resulting tree. In the figure, the sets that participate in the move are highlighted with dashed ellipses while a dashed arc (i, j) indicates that node i moves from its set to node j's set. A similar neighborhood is the tree-based, two-exchange neighborhood (see Shariaha et al.[49]) in which multiple nodes (as opposed to single nodes) shift from one set to another. More involved, multi-exchange neighborhoods can be defined in the same way (see Ahuja et al. [2]). In these neighborhoods, many sets (not just two) participate in the exchange either by contributing a single node or multiple nodes. In Figure 1.2, we present a simple multi-exchange move and the resulting tree. These approaches (i.e., two-exchange, multi-exchange) have been extensively used for problems like the CMST where the nodes in the different subtrees are considered to belong to the different sets of the partition. We note that, in the CMST, after determining the assignment of nodes into sets, the problem of interconnecting the nodes in the same set is a simple MST problem.

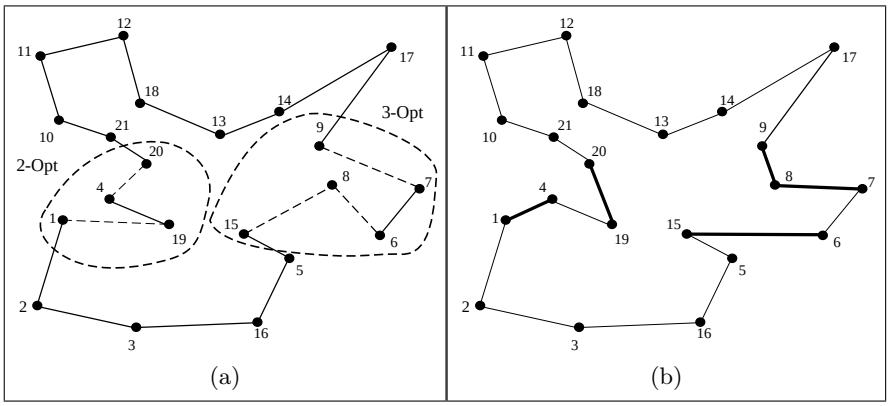

Figure 1.3. 2-opt and 3-opt example for the TSP. (a) Original solution (dashed edges will participate in the exchanges). (b) Final solution after the exchanges (the edges that have been added after the exchanges are noted in bold).

Another way of generating neighborhoods is by changing the placement, number, or type of the edges for a given solution. A simple example of this is the 2-opt neighborhood for the traveling salesman problem (TSP) in which we delete edges (x, y) and (z, w) from the current solution and add edges (x, z) and (y, w). The 3-opt exchange that has also been used for the TSP works in a similar way. Three edges are deleted from the current solution and their endpoints are then connected in a different way that guarantees a feasible tour. Figure 1.3 presents an example of a 2-opt and a 3-opt exchange for a TSP instance. A few indicative examples of the successful application of these ideas are the 2-opt neighborhood that was used on a survivable network design problem by Monma and Shallcross [39] and Park and Han [42], and a k-opt neighborhood for a network loading problem by Gendron et al. [26].

In general, the most efficient neighborhood structures are going to be problem specific. However, the ideas of two-exchange, multi-exchange, 2-opt, and 3-opt can serve as valid starting points for a wide variety of problem contexts and lead to more suitable neighborhoods.

Search. It is fairly easy to see that depending on the definition of the neighborhood the number of possible neighbors ranges from exactly one (in the 2-opt case for the TSP) to $O(nK)$ (for the shift move of the node-based two-exchange neighborhood), to $O(n^2)$ (in the tree-based two-exchange case) to a significantly larger $O(n^K)$ (for the multi-exchange case), where n is the number of nodes in the graph and K is the number of sets participating in the exchanges. Because of the large number of possible neighbors, we need to determine a procedure that searches

the neighboring solutions efficiently and finds one with which to replace the current solution. There are a few well-documented and extensively tested procedures that concentrate on this step of the procedure. The most prominent ones are steepest descent, simulated annealing (SA), and the greedy randomized adaptive search procedure (GRASP). The main idea behind most of these procedures is essentially the same. It specifies that the neighboring solution could be chosen at random, characterized as improving or non-improving, and accordingly accepted with a certain probability specific for each case (i.e., improving or not). Compared to steepest descent, SA and GRASP are more sophisticated local search procedures. Due to their advanced search strategies, they are often classified as metaheuristics in the literature.

Steepest descent is the simplest and the most short-sighted of the search procedures. At each iteration, we compute all neighbors for the existing solution and choose the one which reduces (in the case of a minimization problem) the cost of the solution the most. A potential disadvantage of this procedure is that it never accepts *uphill moves* (i.e., neighbors with higher cost than the existing solution) and as a result can potentially terminate after a few iterations at a poor quality local optimum. Moreover, in some cases it is impractical, because of the computational effort required, to compute all neighbors. However, if the neighborhood structure used is well suited to the problem and there is an efficient way to find the best neighbor (i.e., other than enumeration) then a Steepest descent procedure can still achieve a high level of performance and will be computationally efficient. For applications of this approach in the context of the CMST problem, see [2, 3].

Simulated annealing (SA) is a very popular local search procedure. In Figure 1.4 we present the steps of a generic SA algorithm. The main feature of this procedure is that it can accept *uphill moves* with a certain probability that decreases at each iteration. The way in which this acceptance probability decreases (i.e., the so-called cooling schedule) plays an important role in the efficiency of the procedure. If the decrease is too steep then the procedure will terminate early at a poor solution. On the other hand, if it is very gentle we will have to wait a long time before we reach the final solution. However, the slower the cooling schedule, the better the solution obtained by the procedure. In particular, for theoretical convergence to the optimal solution the temperature needs to be reduced infinitesimally in each step. While this result is reassuring, it also implies that there is no finite-time implementation that guarantees the optimal solution. In practice, a balance is to be obtained between the running time of the procedure, quality of solutions obtained, and the cooling schedule. Determining an appropriate cooling schedule is not an

```
Begin
   randomly generate an initial solution f_s
   compute the cost of f_s, C(f_s)
   set the effective temperature, T, to the initial value, T_0
    while T ≠ T_N do
      randomly select a neighbor, f_n, of the current solution, f_s
      compute the cost C(f_n) and Δ = C(f_n) − C(f_s)
      if Δ ≤ 0 then
         f_s ⟵ f_n
      else
         with probability e^{−Δ/T}, f_s ⟵ f_n
      end if
      update the temperature T
   end while
end
```

Figure 1.4. Steps of a Simulated Annealing procedure.

easy task, and most of the time it takes a lot of computational testing to adequately justify a particular selection. Aarts et al. provide a nice synopsis of the simulated annealing procedure in [1].

The GRASP procedure (see [43]) effectively searches the solution space rapidly by implementing local search many times while starting from different points in the solution space. In Figure 1.5 we present the steps of a generic GRASP procedure. The different starting solutions, f_s, that GRASP uses are generated by a greedy construction heuristic that in each iteration selects any edge that is within $100 * \alpha$ percent of the cheapest edge. In this way starting solutions have considerable variability, but are also fairly reasonable. GRASP then proceeds by repeatedly updating the current solution with the best known neighbor, f_n, until there are no more improvements. In the end, the procedure returns the best solution it has encountered, f_b. The total number of iterations or the number of successive iterations without an improvement can both be used as a termination condition. Most of the criticism of GRASP has to do with the fact that successive iterations of the procedure are independent and as a result the algorithm does not learn from solutions found in previous iterations. More advanced versions of GRASP attempt to resolve this issue. Some of them in-

```
Begin
    t ⟵ 0
    f_b = inf
    while (not termination-condition) do
        generate a random greedy solution, f_s
        using local search find f_n from f_s
        if  C(f_n) < C(f_b)  then
        f_b ⟵ f_n
        end if
    end while
    return f_b
end
```

Figure 1.5. Steps of a Greedy Randomized Adaptive Search Procedure.

clude path-relinking approaches (see [37]) that allow for intensification and diversification strategies, and adaptive mechanisms, called Reactive GRASP [45], that change the value of α depending on the results of the search. Another issue with GRASP has to do with the fact that starting points might in fact be identical solutions. An easy way to resolve this issue is to use hash tables (see [10]) to guarantee independent starting points. For an extensive annotated bibliography of GRASP literature (including many applications in network design) see Festa and Resende [21].

All these search procedures have been used successfully in different contexts but, in general, greatest attention should be given to defining the neighborhood structure. In our opinion, in most cases where local search procedures have performed well in the literature, it has been due to the properties of the neighborhood. Often, with an effective neighborhood simple search strategies such as steepest descent can provide results competitive with more sophisticated search strategies such as GRASP and simulated annealing. Additionally, specifying the parameters used in the search procedures—like the rate for decreasing the temperature in SA, or the value of α for GRASP—is critical to their success and a lot of testing is required to find values that work well for a wide range of problem instances.

2.3 Tabu Search

An approach that is similar to LS in some ways but deserves to be noted separately is tabu search (TS). Tabu search, like LS procedures, requires the definition of a neighborhood for each solution in the search space. However, TS differs in the way it searches the neighborhood. The main idea of TS is to incorporate memory in the search process. The notion is that by effectively using memory, one may be better able to guide the search to regions in the search space containing the global optimum or near-optimal solutions to the problem.

In TS, memory concerning the search path is created by the use of a tabu list. This list keeps a record of solutions, or solution attributes that should not be considered (i.e., are tabu) in the next step of the search process. By doing so, TS guides the search process away from local optima and visits new areas in the solution space. Obviously, selecting the rules under which a solution or a specific attribute becomes tabu and the number of iterations for which it will remain tabu (i.e., the tabu tenure) are of vital importance when applying this procedure.

To illustrate the use tabu moves, consider the k-th best spanning tree problem. In this problem we wish to find the k-th best spanning tree in a graph. A feasible solution is a spanning tree, and the neighborhood of a solution can be defined as the set of spanning trees obtained by dropping and adding an edge to the given spanning tree. Here are three different ways in which a move can be classified as tabu. 1) The candidate move is tabu if either the edge added or the edge dropped is tabu. 2) The candidate move is tabu if a specific edge in the solution is tabu. 3) The candidate move is tabu if both the edge added and the edge dropped are tabu. In general, when evaluating choices for tabu moves, the tabu rules should be simple and ensure that the search can escape from local optima and visit new areas in the solution space.

Apart from the definition of the tabu rules another way to control the search is by determining the tabu tenure associated with the different rules. Defining these tenures plays a very important role since it greatly affects the outcome of the search. Small tenures will result in cycling (i.e., revisiting the same solutions) while large tenures will result in a gradual degradation of the search results. It is possible to define tenures as constant, in which case the tenure never changes during the search, or as dynamic with either a deterministic or a random way in which the value of the tenure can change. In any case it is important to experiment until one finds good values for the tabu tenures.

The issues discussed so far although important usually come under the label of *short-term memory* and do not cover the more powerful features

of TS that have to do with *long-term memory*. We will not attempt to give a complete account of all these possibilities nor provide guidelines for their use. However, we would like to bring to the attention of the reader some of the more important features of long-term memory like frequency based measures, strategic oscillation, and path-relinking. The use of long-term memory can significantly increase the efficiency of a TS algorithm. Frequency based measures are built by counting the number of iterations in which a desirable attribute has been present in the current solution. These measures can then be used to guide subsequent explorations of the solution space to regions that are likely to contain near-optimal solutions. Strategic oscillation allows the search to continue past a predetermined boundary which usually coincides with the space of feasible solutions. Once the search process passes the boundary we allow it to proceed to a specified depth beyond the oscillation boundary before forcing it to turn around. This time the boundary is crossed in the other direction (from infeasibility to feasibility) and the search continues normally. Path-relinking generates new solutions by exploring trajectories that connect elite solutions and has been found to be very promising. TS procedures with long-term memory features can significantly outperform simpler heuristic search algorithms, but can be quite complex and intricate to design. For an in depth discussion on TS and all of its features see the text by Glover and Laguna [29].

2.4 Genetic Algorithms

In this section, we describe a popular heuristic search procedure known as an evolutionary algorithm or a genetic algorithm (GA). When carefully applied to a problem, GAs can generate high-quality solutions that are often better than LS procedures.

Genetic algorithms are powerful search procedures motivated by ideas from the theory of evolution. They have been successfully used for a variety of problems. Although many different variations exist, the main steps that GAs follow are essentially the same (see Figure 1.6). At first, an initial population of solutions is generated and the *fitness* of all solutions is evaluated. The fitness essentially describes how good each solution is, and is usually a function of the cost of the solution. A specific set of solutions is then *selected* and on that set different genetic operators are applied. The genetic operators are usually simple heuristic procedures that combine two solutions to get a new one (crossover operator) or transform an existing solution (mutation operator). The new solutions generated by the genetic operators are then included in a new population and the above steps are repeated until a terminat-

```
Begin
   t ⟵ 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
      t ⟵ t + 1
      select r solutions from P(t − 1)
      apply genetic operators on the r solutions
      insert the new r solutions to P(t)
      evaluate P(t)
   end while
end
```

Figure 1.6. Steps of a Genetic Algorithm.

ing condition is satisfied. The selection of solutions from a population is done with respect to their fitness value. Specifically, fitter solutions have a higher chance of being selected than less-fit solutions.

Representation. The most challenging and critical aspect of using a GA is the determination of the way in which solutions are represented. A good representation can lead to near-optimal solutions while a bad one will cause the GA to perform very poorly. Selecting a representation is by no means a trivial task. The main thing to keep in mind when selecting a representation is that it should have meaningful blocks of information that can be related to elements of a desirable solution and that these blocks should be preserved from one generation to the next. These are essentially the main ideas described in the *Building Block Hypothesis* [38] and the guidelines presented by Kershenbaum in [34]. If we are interested in representing trees, then we can look into some of the approaches already developed and tested for various tree problems. Some of the more traditional approaches include the so-called characteristic vectors, predecessor encoding, and Prüfer numbers. Characteristic vectors represent trees by a vector $v \in \{0,1\}^{|E|}$ where $|E|$ is the set of edges in the graph. In predecessor encoding each entry in the encoding corresponds to a particular node and signifies the predecessor of that node, with respect to a predetermined root node in the tree. Finally, Prüfer numbers indirectly encode trees and require specific algorithms to con-

vert a number into a tree and vice versa. All three of these approaches have been criticized (see Palmer and Kershenbaum [41]) because they violate some of the principles we mentioned above.

Palmer and Kershenbaum introduce a link-and-node-bias (LNB) encoding in [41]. In order to encode one tree with the LNB encoding we need to specify one bias, b_i, for each node i in the graph ($|N|$ biases) and one bias, b_{ij} for each edge, $\{i, j\}$, in the graph ($|E|$ biases). The cost matrix is then biased by using: $C'_{ij} = C_{ij} + P_1 b_{ij} C_{max} + P_2(b_i + b_j)C_{max}$, where C_{ij} is the original cost of edge $\{i, j\}$, C_{max} is the maximum edge cost in the graph and P_1, P_2 are two multipliers that are constant for each GA run. The tree can then be found by running a MST algorithm with the new costs. We observe that this representation can encode any tree by setting $b_i = 0$, for all nodes i, and $b_{ij} = 0$ for the edges $\{i, j\}$ that belong to the tree we wish to represent and $b_{ij} = M$, otherwise (where $M > C_{max}$). This encoding satisfies most of the guidelines discussed earlier and has been found to work well for the optimal communication spanning tree problem (OCSTP).

Sometimes, even when we are interested in constructing trees, it is more efficient to look at other representations that do not specifically deal with tree structures. One example is the CMST problem. As in the case of the LS procedures for the CMST, it may be desirable to find a partition of the nodes in the graph. Once a partition is found it is much easier to construct a feasible solution by solving the subproblems on the partition (for the CMST we find a MST on the union of the central node and the nodes in the partition). In these cases, using a representation that is geared towards groupings/partitions can prove to be far more valuable. In Figure 1.7 we present an example of a capacitated tree encoded using Falkenauer's [19] grouping representation. In this representation there is an item part which defines the assignment of nodes into groups and a group part that specifies the groups in the solution.

Another example of a representation originally developed for problems that are not strictly associated with trees is the network random key encoding (NetKey) originally introduced by Bean [5] for problems where the relative order of tasks is important (e.g., scheduling problems). Rothlauf et al. [47] present a way in which NetKey can be used for tree representations in network design problems, compare it with characteristic vector encoding, and experimentally show that NetKey is better. The idea behind NetKey when used for tree representations is to introduce a relative order in which the edges in the graph will be considered. Based on the order represented in the encoding, edges are added to an empty graph. When the addition of an edge introduces a cycle, we skip

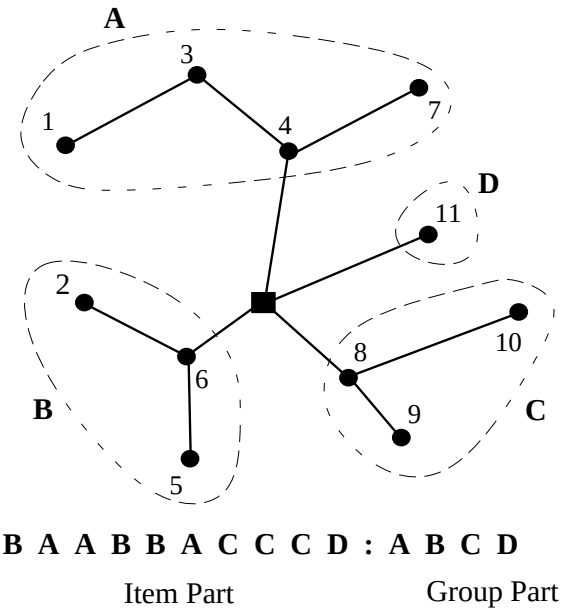

Figure 1.7. Example of a group assignment and the respective representation for a CMST instance.

it and move on to the next edge until we have $|N|-1$ edges in the graph and, therefore, a tree.

Operators. Genetic operators are usually distinguished as crossover operators or mutation operators. Typical crossover operators like single-point or two-point crossover are usually a good choice and can achieve good results depending on the strengths and weaknesses of the representation. On the other hand, mutation operators should be designed carefully to complement what cannot be achieved by the crossover operators. Historically, mutation operators started out as random changes to individual solutions that were applied with very low probability. This approach, however, makes the mutation operators very weak and hardly ever adds true value to the search procedure. It would be much more beneficial to design a mutation operator that is likely to improve the solution and can be applied with a much higher probability. When we think along these lines, a natural choice for a mutation operation is a simple local search procedure. In most cases, the operators should be defined with the specific characteristics of the representation in mind. For example the strength of the grouping representation described earlier lies in the fact that operators are applied on the group part of the

representation and, therefore, allow for meaningful parts of the solution (i.e., the subtrees) to be carried on from one generation to the next.

In general, GAs can be extremely involved procedures and as a result the aim should be to keep the number of parameters as small as possible. In any case, the selected values for the parameters should be able to achieve high quality results regardless of the problem type.

2.5 Remarks

Obviously, the above discussion on heuristics, local search methods, and metaheuristics is by no means exhaustive. However, we believe that it showcases a set of directions that one can follow when approaching a challenging problem in network design. In our opinion, a fruitful attempt at problem solving should begin by the exploration of naive heuristics that are easy to implement and understand. These first attempts can then lead to a neighborhood definition. The development of the neighborhood could possibly start with known and tested ideas and expand to more sophisticated structures that are specifically tailored to the problem. Depending on the efficiency of the neighborhood, different algorithms for the search procedure can then be tested and compared. Generally, strong and complex neighborhoods require very simple search methods (i.e., steepest descent) while simpler neighborhoods could benefit from more advanced search methods (e.g., simulated annealing, GRASP, tabu search). For genetic algorithms we have given a few examples of representations that have been used in network design problems, commented on aspects such as parameter setting and genetic operators, and recommended the use of local search procedures as mutation operators. In the next few sections, based on our experience applying heuristic search, we discuss several $\mathcal{NP}$-complete network design problems, and illustrate the successful application of heuristic search techniques to them.

3. The Minimum Labeling Spanning Tree Problem

A problem in communications network design that has attracted attention recently is the minimum labeling spanning tree (MLST) problem. In this problem, we are given a graph with labeled edges as input. Each label represents a type of edge and we can think of a label as a unique color. For example, in Figure 1.8, there are six nodes, 11 edges, and five labels (denoted by letters). We would like to design a spanning tree using the minimum number of labels. For the example in Figure 1.8 an optimal MLST is indicated in bold.

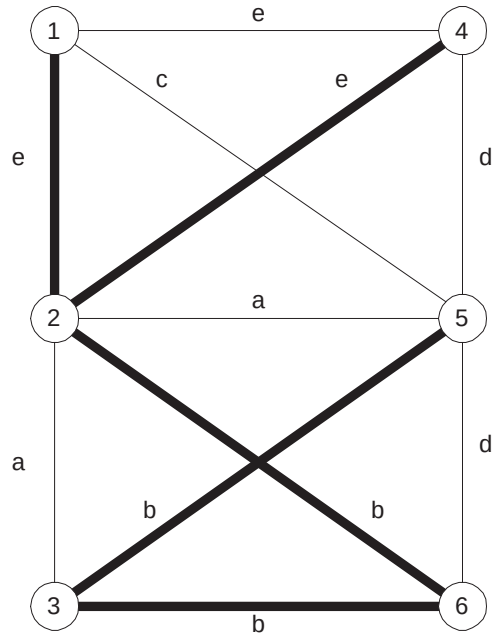

Figure 1.8. An MLST example and solution.

Chang and Leu [8] introduced the problem, proved that it is NP-hard, and proposed two heuristics. One of these, called MVCA, clearly outperformed the other. Krumke and Wirth [35] proposed an improvement to MVCA and proved that the ratio of the (improved) MVCA solution value to the optimal solution value is no greater than $2 \ln n + 1$. Wan et al. [51] improved upon this bound and were able to show that the MVCA solution value divided by the optimal solution value is no greater than $\ln(n-1) + 1$. Brüggeman et al. [7] introduced local search techniques for the MLST and proved a number of complexity results. For example, they were able to show that if no label appears more than twice, the problem is solvable in polynomial time. Building on some of these ideas, Xiong et al. [53] were able to further improve the worst-case bound. In particular, they proved that, in a graph where the label that appears most frequently appears b times, the ratio of the MVCA solution value to the optimal solution value is no greater than the bth harmonic number

$$H_b = 1 + \frac{1}{2} + \ldots + \frac{1}{b} < 1 + \ln b.$$

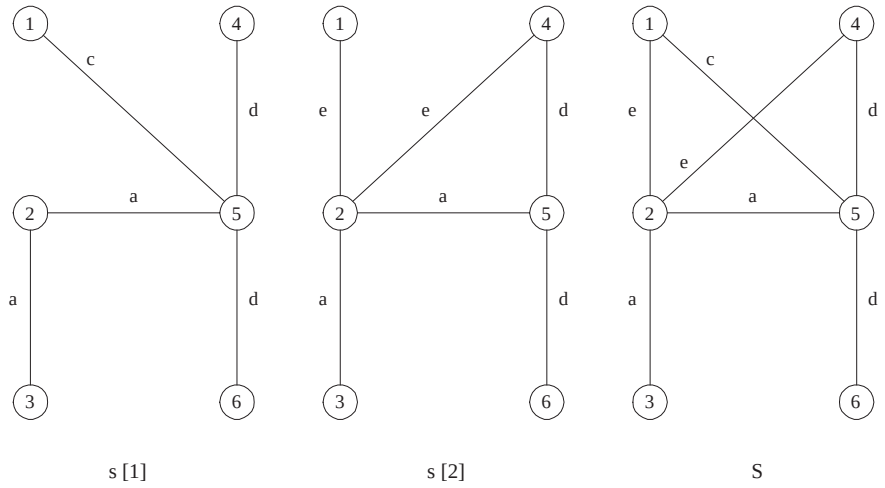

Figure 1.9. The crossover operator in the GA for the MLST problem.

In addition, Xiong et al. present a family of graphs for which this worst-case ratio can be attained. Therefore, the worst-case ratio cannot be improved.

In a second paper, Xiong et al. [52] present a simple genetic algorithm for the MLST problem and they compare it with MVCA over a wide variety of problem instances. The GA consists of crossover and mutation operations.

Before we describe these operations, we must define what we mean by a solution. Suppose we are given a graph $G = (V, E, L)$, where V is the set of nodes, E is the set of edges, and L is the set of labels. A solution $s[i]$ is a subset C of the labels in L such that all the edges with labels in C construct a connected subgraph of G and span all the nodes in G. If we consider the MLST instance in Figure 1.8, we can see that $\{a, c, d\}$ and $\{a, d, e\}$ are two solutions. Thus, any spanning tree of $s[i]$ will be a feasible solution to the MLST problem.

Given an initial population of solutions, the crossover operator works as follows. Two solutions $s[1]$ and $s[2]$ are selected for crossover. First, we combine the labels from the two solutions. For example, if $s[1] = \{a, c, d\}$ and $s[2] = \{a, d, e\}$ in Figure 1.9, then $S = \{a, c, d, e\}$ is the union of these two. Next, we sort S in decreasing order of label frequency, yielding $\{a, d, e, c\}$. Finally, we add labels of S in sorted order to an empty set, T, until T represents a solution. In this case, we output $\{a, d, e\}$ which is the same as $s[2]$.

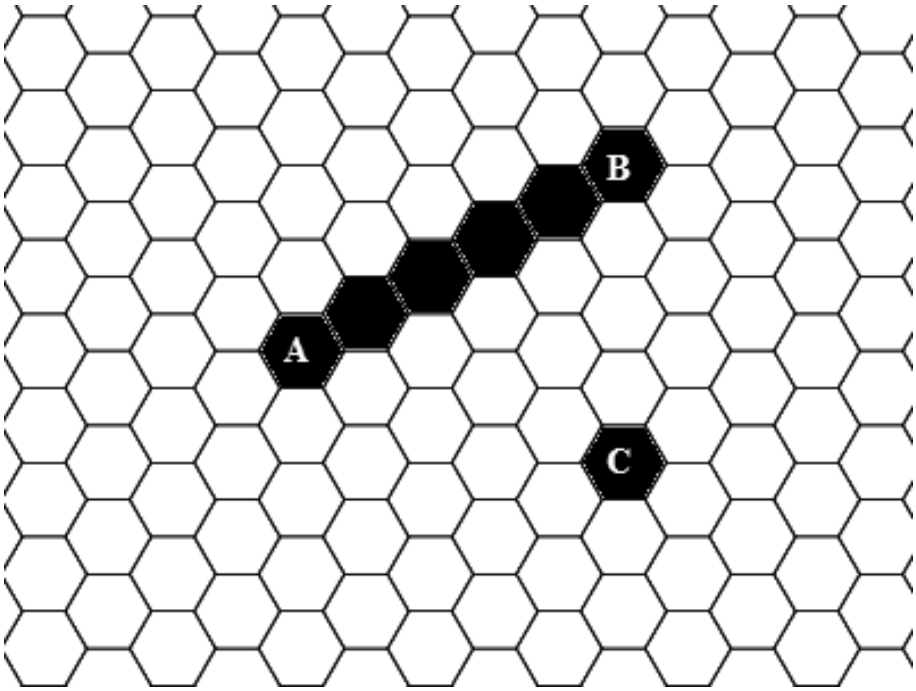

Figure 1.10. Hexagonal tiling of a 2-dimensional space.

The mutation operator is also quite simple. Suppose we begin with solution S. First, we randomly select a label c not in S and let $T = S \cup c$. Next, we sort T in decreasing order of label frequency. Finally, starting at the end of the list, we delete one label at a time from T provided that the result is itself a solution.

From the above description it should be evident that the GA is fairly simple. To compare the performance of the GA to existing heuristics for the MLST problem, the GA and MVCA were tested over 81 problem instances. The largest of these involved 200 nodes and 250 labels. Both procedures are extremely fast. The GA outperformed MVCA in 53 cases (sometimes by a substantial margin), MVCA outperformed the GA in 12 cases (by a very small amount in each case), and there were 16 ties.

4. The Euclidean Non-uniform Steiner Tree Problem

In Steiner tree problems, we are given a set of terminal nodes which we seek to connect at minimum cost. Additional nodes, known as Steiner

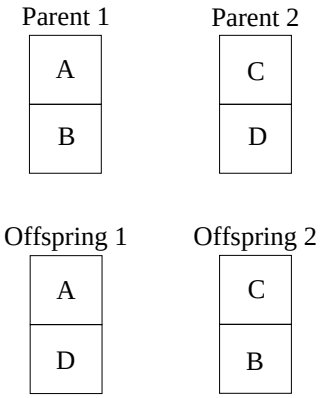

Figure 1.11. Horizontal crossover operator in the GA for the Euclidean non-uniform Steiner tree problem.

points, may be added to the terminal nodes in order to reduce the overall construction cost. The problem requires that specific Steiner points and connecting edges be determined.

Many variants of the Steiner tree problem have been studied in the literature including the Euclidean Steiner problem, the Steiner problem in graphs, and the rectilinear Steiner problem (see Du et al. [15] for recent work on different variants of the Steiner tree problem). In most variants, the cost of an edge typically depends on its distance only. That is, costs are uniform with respect to the location of the edge.

Coulston [11] recently introduced an interesting variant. In this problem, the cost of an edge depends on its location on the plane as well as its distance. Suppose the problem involves laying cable, constructing pipeline, or routing transmission lines. In such a setting, certain streets (routes) are more expensive to rip apart and re-build than others. The Euclidean non-uniform Steiner tree problem reflects this realism. To begin with, the 2-dimensional region is covered with a hexagonal tiling. Each tile has an associated cost and terminal nodes (located at specific tiles), which must be connected, are given as input. Other nodes (equivalently, tiles) may be selected for use as Steiner nodes. Two nodes can be directly connected if and only if a straight line of tiles can be drawn between the nodes (see Figure 1.10). The total cost of the Euclidean non-uniform Steiner tree is the sum of the cost of tiles in the tree. Coulston uses heuristic search techniques such as genetic algorithms and ant colony optimization in his computational work.

In Frommer et al. [22], a simple genetic algorithm is introduced that exploits the geometric nature of the problem (i.e., it makes use of (x, y)

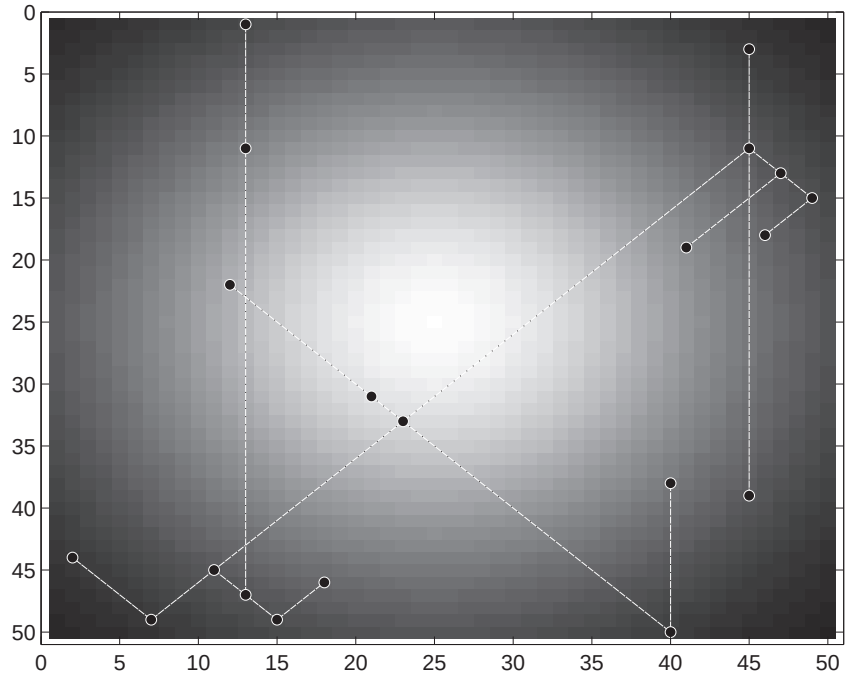

Figure 1.12. MST solution (cost = 56.24) for the Euclidean non-uniform Steiner tree problem.

coordinates). In addition, Frommer et al. allow an additional charge for each Steiner node.

At the heart of their genetic algorithm is a spatial crossover operator. In its simplest implementation, the grid is split in half horizontally. Let A, B, C, and D be sets of Steiner nodes. A and C can have some common nodes, as can B and D. The parents and offspring are shown in Figure 1.11. In each of the four cases, there are a set of terminal nodes also (not shown). If we combine Steiner nodes and terminal nodes and solve a minimal spanning tree problem over these nodes, we can obtain a Steiner tree. (Note, we must eliminate degree-1 Steiner nodes first.)

The genetic algorithm is tested over a wide variety of problem instances and is compared with a heuristic based on the progressive addition of Steiner nodes. The overall performance is comparable, but progressive addition is significantly slower as problem size grows.

The GA behaves quite reasonably as indicated by Figures 1.12 and 1.13. In Figure 1.12, a minimal spanning tree solution is presented. The

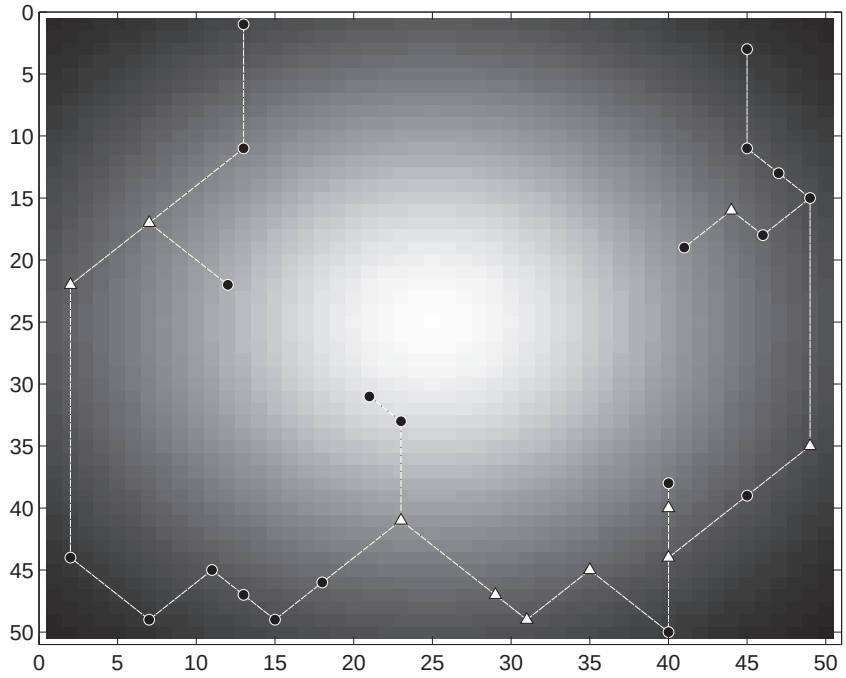

Figure 1.13. GA solution (cost = 29.01) for the Euclidean non-uniform Steiner tree problem.

lightest regions have the highest costs. The terminal nodes are connected in a least cost way, without using any Steiner nodes. In Figure 1.13, the GA finds a better solution, by using Steiner nodes in order to avoid the lightest (most expensive) regions.

5. The Prize-Collecting Generalized Minimum Spanning Tree Problem

The prize-collecting generalized minimum spanning tree (PCGMST) problem occurs in the regional connection of local area networks (LAN), where several LANs in a region need to be connected with one another. For this purpose, one gateway node needs to be identified within each LAN, and the gateway nodes are to be connected via a minimum spanning tree. Additionally, nodes within the same cluster are competing to be selected as gateway nodes, and each node offers certain compensation (a prize) if selected.

Formally, the PCGMST problem is defined as follows: Given an undirected graph G = (V, E), with node set V, edge set E, cost vector $c \in R_+^{|E|}$ on the edges E, prize vector $p \in R_+^{|V|}$ on the nodes V, and a set of K mutually exclusive and exhaustive node sets V_1, ..., V_K (i.e., $V_i \bigcap V_j = \emptyset$, if $i \neq j$, and $\bigcup_{k=1}^{K} V_k = V$), find a minimum cost tree spanning exactly one node from each cluster. This problem represents a generalization of the $\mathcal{NP}$-hard generalized minimum spanning tree (GMST) problem, where all nodes are equally important, that is, each node offers equal compensation if selected for the regional network design.

Several exact and heuristic procedures have been developed for the GMST problem so far. Myung et al. [40] developed a dual-ascent procedure based on a multicommodity flow formulation for the problem. Feremans [20] developed a branch-and-cut algorithm that uses a tabu search algorithm for the initial upper bound, and a local search algorithm that improves upper bounds found during the course of the branch-and-cut procedure. Pop [44] developed a relaxation procedure based on a polynomial-size MIP formulation for the GMST problem. Golden et al. [31] developed several upper bounding procedures including simple local search and genetic algorithm heuristics.

Building upon our previous work in Golden et al. [31] for the GMST problem, we describe the application of a local search procedure and a genetic algorithm to the PCGMST problem.

5.1 Local Search Procedure

The local search (LS) procedure we developed in [31] for the GMST problem may be directly applied to the PCGMST problem. It differs only in that an additional objective function term for node prizes needs to be taken into account here.

The LS procedure is an iterative 1-opt procedure. It visits clusters in a wraparound fashion following a randomly defined order. In each cluster visit, the neighborhood of a feasible generalized spanning tree is explored by examining all feasible trees obtained by replacing the node (in the tree) from the current cluster. In other words, a GST of least cost (the cost of the tree is the sum of the edge costs minus the rewards on the nodes) is found by trying to use every node from that cluster, while fixing nodes in other clusters. The local search procedure continues with visiting clusters until no further improvement is possible. The procedure is applied to a pre-specified number of starting solutions (denoted by t). The steps of the procedure are outlined in Figure 1.14.

Local Search Heuristic (LS):

Step 0. Specify the number of feasible solutions to be generated - t. Repeat Steps 1 through 3 t times.

Step 1. Generate a feasible solution as follows. Randomly select a single node from each cluster, and build an MST on the selected nodes (using any MST algorithm). If no MST exists on the selected nodes, repeat Step 1. Otherwise, continue to Step 2.

Step 2. Randomly define an order in which clusters will be searched.

Step 3. Follow the order defined in Step 2 in visiting clusters. Repeat the following steps until K sequential cluster visits result in no improvement.

- While visiting a cluster, consider each node in the cluster as a replacement for the current node in the cluster contained in the GST. Compute the cost of the solution for each replacement.
- Among the solutions computed in the previous step, identify the one giving the greatest improvement in the objective function. If there is an improvement, implement it by replacing the node representing the cluster, and update the current tree.

Figure 1.14. Steps in the Local Search procedure for the PCGMST problem.

5.2 Genetic Algorithm

We present a genetic algorithm for the PCGMST problem that is similar to the one we developed in [31], with a few differences in the initial population and genetic operators applied. Figure 1.15 shows an outline of our genetic algorithm. The initial population is created by randomly generating a pre-specified number of feasible GSTs. Before adding a new chromosome to the population $P(0)$, we apply local search and add the resulting chromosome as a new population member. Within each generation t, new chromosomes are created from population $P(t-1)$ using two genetic operators: local search enhanced crossover and random mutation. The total number of offspring created using these operators is equal to number of chromosomes in the population $P(t-1)$, with $\alpha P(t-1)$ offspring created using crossover, and $\beta P(t-1)$ offspring created using

```
Begin
   t ⟵ 0
   initialize P(t)
   while (not termination-condition) do
      t ⟵ t + 1
      Generate αP(t − 1) offspring using local search enhanced
         crossover operator
      Generate βP(t − 1) offspring using random mutation
         operator
      Select new generation P(t), with population size equal
         to θ(1 + α + β)P(t − 1)
   end
end
```

Figure 1.15. Steps of our Genetic Algorithm for the PCGMST problem.

mutation (fractions α and β are experimentally determined). Once the pre-specified number of offspring is generated, a subset of chromosomes is selected to be carried over to the next generation. The algorithm terminates when the termination condition, a pre-specified number of generations, is met.

We now give a more detailed description of the genetic algorithm.

Representation. We represent a chromosome by an array of size K, so that the gene values correspond to the nodes selected for the generalized spanning tree. Figure 1.16 provides an example of a generalized spanning tree and its corresponding chromosome representation.

Initial Population. The initial population is generated by random selection of nodes for each cluster. If possible, a minimum spanning tree is built over the selected nodes. Otherwise, the chromosome is discarded, since it represents an infeasible solution. Each feasible minimum spanning tree built in this way is then used as input for the local search procedure. The resulting solution is then added to the initial population as a new chromosome with a fitness value defined as the difference between the cost of the MST and the prizes associated with the nodes in the chromosome.

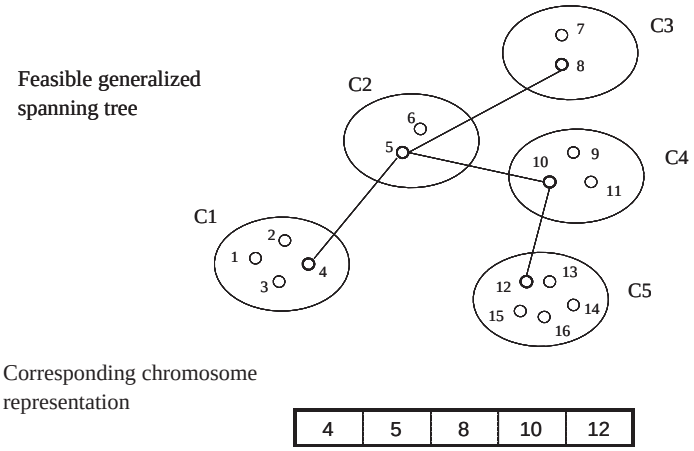

Figure 1.16. An example of a feasible GST and the corresponding chromosome representation

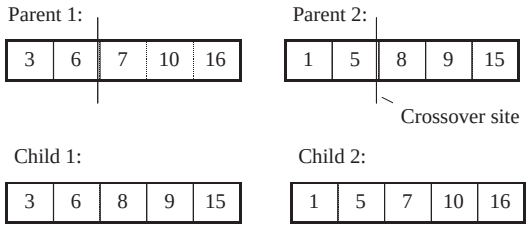

Figure 1.17. One-point crossover operator example.

Crossover. We apply a standard one-point crossover operation as shown in Figure 1.17. As in the initial population, only the feasible solutions are accepted. Each child chromosome created using this operator is used as input to the local search procedure, and the resulting chromosome is added to the population.

Random Mutation. A random mutation operator randomly selects a cluster to be modified and replaces its current node by another, randomly selected, node from the same cluster. The new chromosome is accepted if it results in a feasible GST. In order to maintain diversity of the population, we do not apply local search to new chromosomes created by random mutation.

Selection/Replacement. At the end of each generation, a fraction, θ, of the current population is selected to be carried to the next generation, while the remaining chromosomes are discarded. This selec-

tion is a combination of elitism and rank-based selection, where the top 10% of the current population is selected using elitism and the remaining 90% is selected using rank-based selection (see Michalewicz [38]).

5.3 Computational Experiments

In this subsection, we provide a summary of some computational experiments with the local search (LS) heuristic and the genetic algorithm (GA) procedure. Both LS and GA were coded in Microsoft Visual C++ on a workstation with 2.66 GHz Xeon processor and 2GB RAM. The heuristics were tested on two classes of instances—those from TSPLIB (identical to those in [20], with edge costs satisfying the triangle inequality) and those with random edge costs (identical to those in [31]). For these experiments we have added integer node prizes generated randomly in the range [0, 10].

The TSPLIB instances consist of 5 sets of problems differing in terms of a user-defined parameter μ that may be thought of as the average number of nodes per cluster. The size of these instances varies from 7 to 84 clusters, with 47 to 226 nodes, and up to 25,118 edges. The size of random instances varies from 15 to 40 clusters, with 120 to 200 nodes, and up to 15,000 edges.

Parameter settings for LS and GA. Based on the initial testing over a separate set of large instances with random edge costs, we selected the GA parameters as follows. The size of the initial population $P(0)$ was set to 100. α and β were both set to 0.5. θ, the fraction of the population to survive, was set to 0.5. The stopping criterion was 15 generations. For LS the number of starting solutions was set to 500.

Computational results. For the TSPLIB instances, we were able to find optimal solutions for 133 of 169 instances using a branch-and-cut algorithm for the PCGMST problem that we proposed in [30]. The branch-and-cut procedure was able to find optimal solutions for all random instances. The summary of results for these instances is shown in Table 1.1.

The first two columns in the table represent the type of problem set and number of instances within each set. The third column in the table indicates the number of instances where the optimal solution is known (from the branch-and-cut procedure). The first column under LS and GA indicates the number of instances where these procedures found the optimal solution. In all TSPLIB instances where the optimal solution is known (there are 133 such instances), our GA and LS found the optimum. In the remaining 36 TSPLIB instances, GA always provided

Table 1.1. Summary of computational results for the PCGMST problem.

Problem Set	*Inst*	*Opt*	*LS*		*GA*	
		Known	*Opt*	*Time (sec)*	*Opt*	*Time (sec)*
TSPLIB, geog	41	35	35	14.44	35	7.48
TSPLIB, $\mu = 3$	32	22	22	70.95	22	24.34
TSPLIB, $\mu = 5$	32	22	22	28.82	22	11.13
TSPLIB, $\mu = 7$	32	27	27	8.61	27	5.35
TSPLIB, $\mu = 10$	32	27	27	2.60	27	3.09
TSPLIB	169	133	133	24.51	133	10.25
Random	42	42	40	6.33	40	5.96
Summary	211	175	173	20.89	173	9.39

solutions that were at least as good as the ones obtained by LS. In 8 of these instances, GA provided a solution better than the one obtained by LS. In the case of random instances, where the optimum solution is known for all 42 instances, both LS and GA did not find the optimum in 2 of 42 instances. In one of these instances, GA provided a better solution than LS, and in the other instance LS was better than GA. The branch-and-cut algorithm took an average of 726 seconds per instance to solve the 175 instances to optimality, while LS and GA take an average of 20.89 and 9.39 seconds per instance respectively for all 211 problem instances.

5.4 Remarks

The computational results with the two heuristic search techniques, local search and the genetic algorithm, presented in this section have shown that relatively simple heuristic search techniques can provide high quality results for the PCGMST problem, while requiring only a small fraction of time compared to an exact procedure like branch-and-cut.

It is noteworthy that we obtained similar compelling results with local search and a genetic algorithm for the GMST problem in [31]. The LS procedure provided results that were on average 0.09% from optimality (where the optimal solution was known), while the genetic algorithm (with a different structure than the one described here) provided results that were on average only 0.01% from optimality.

Finally, we note that there is a slightly different version of the GMST problem that has been studied in the literature. In this version the clusters are collectively exhaustive, but not necessarily mutually exclusive, and *at least* one node from each cluster needs to be selected for the GMST. Several heuristic search procedures have been applied to this at least version of the GMST problem. Dror et al. [14] developed a genetic

algorithm for the *at least* version of the GMST problem. Shyu et al. [50] developed an ant colony optimization procedure for this version of the GMST problem, with faster but similar results (in terms of solution quality) to the GA developed by Dror et al. More recently, Duin and Voß [17] use a transformation of the *at least* version of the GMST problem to the Steiner problem in graphs, and show that the problem can be efficiently solved using heuristic pilot methods (the pilot method is a technique proposed by Duin and Voß [16] that is designed to improve solutions obtained by other heuristics, referred to as sub-heuristics or pilots, through tailored repetitions of these heuristics).

6. The Multi-Level Capacitated Minimum Spanning Tree Problem

In this section, we describe the multi-level capacitated minimum spanning tree (MLCMST) problem, a generalization of the well-known capacitated minimum spanning tree (CMST) problem, and discuss two heuristics for it. In the CMST problem, we are given a set of terminal nodes, each with a specific traffic requirement that we wish to transport to a given *central* node. Furthermore, a single type of facility with a fixed capacity is available for installation between any two nodes. We wish to design a feasible, minimum cost, *tree* network to carry the traffic. The CMST is a fundamental problem in network design and has been extensively studied by many researchers over the years (see [25] for a nice survey).

In many practical applications in telecommunications, utility networks, etc., there is more than a single facility type available to the network planner. The MLCMST addresses this practical issue and generalizes the CMST by allowing for multiple facilities with different costs and capacities in the design of the network. Other than our own work [23, 24], little has been done on the MLCMST problem.

Formally, the MLCMST problem can be defined as follows. Given a graph $G = (V, E)$, with node set $V = \{0, 1, 2, \ldots, N\}$ and edge set E. Node 0 represents the central node (which we will also denote by c) and the rest are terminal nodes. W_i is the traffic requirement (or weight) of node i to be transported to the central node c. We are also given a set of facility types $\Lambda = \{0, 1, \ldots, L\}$ with capacities $Z_0 < Z_1 < \ldots < Z_L$ and cost functions C_{ij}^l denoting the cost of a facility of type l installed between nodes i and j. We wish to find a minimum cost *tree* network on G to carry the traffic from the terminal nodes to the central node.

We will restrict our attention to (realistic) cost functions that exhibit economies of scale and are typical in communication networks. In other

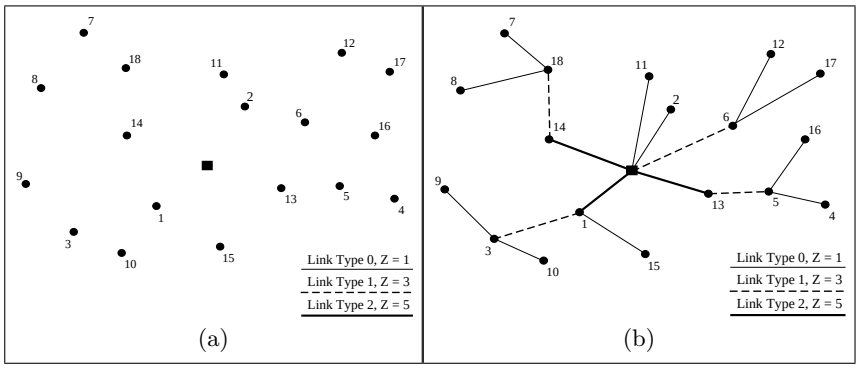

Figure 1.18. Multi-Level Capacitated Minimum Spanning Tree. (a) Nodes in the network. (b) Feasible Multi-Level Capacitated Spanning Tree.

words, the cost of each facility satisfies the relationship $C_{ij}^{y} \leq \frac{Z_y}{Z_x} C_{ij}^{x}$ for every edge $\{i, j\} \in E$, and $x < y$. We also impose the condition that only a single facility type is permitted to be installed on an edge. This condition is actually not restrictive as a problem without this restriction can be transformed to one with this restriction (see [48]). Finally, in the following discussion, we only deal with the homogenous demand case in which $W_i = \kappa$, $\forall i \in V$. Like most of the capacitated network design problems, the MLCMST is very difficult to solve and since the CMST is a special case for $|\Lambda| = 1$, it is NP-hard.

6.1 Savings Heuristic

We first briefly describe a savings heuristic for the MLCMST problem (complete details of this heuristic procedure are in a working paper [23]). This heuristic starts with a feasible star solution in which all nodes in V are connected to the central node c, with the lowest capacity link (i.e., type 0). The central idea behind the heuristic is to upgrade the connection of a node i that is currently connected with a type 0 link, to a higher capacity link, while connecting other nodes to i. Obviously, we would like to reduce the overall cost of the network after the upgrade and the reconnections. In order to ensure this for each node i, we compute the savings, D_i^l introduced by upgrading to a type l link,

$$D_i^l = C_{i\text{pred}(i)}^{0} - C_{i\text{pred}(i)}^{l} + \sum_{j \in H} d_{ij}, \tag{1.1}$$

where pred(i) denotes the first node on the path from node i to the central node. The first two terms in (1.1) represent the change in cost

involved in upgrading the current connection of i to a type l link. The third term is the sum of the savings introduced when reconnecting the nodes j in H, to i. Note that D_i^l is only defined for nodes that are currently connected with a type 0 link (for all other nodes, we assume $D_i^l = 0$). For a given node j, $d_{ij} = C^0_{j\text{pred}(j)} - C^0_{ji}$ is the savings obtained by reconnecting node j from its current connection to node i. Again, notice that the reconnection is only possible if node j is currently connected with a type 0 link. We would like to define the set H in a way that maximizes the sum of the savings in (1.1). This can be done by finding all j for which $d_{ij} > 0$ and sorting them in decreasing order. Next, consider all j in sorted order and attempt to connect them to i. If j can be connected to i without violating any of the capacity constraints of the links that are on the path from i to the central node, we add j to H. Otherwise, we consider the next node.

The heuristic starts with $l = L$. Once the calculation of all savings D_i^l, for $i = V\backslash\{c\}$, is complete, we select the largest one and implement the upgrade and the reconnections. We then repeat the computations and implement the changes as long as $D_i^l > 0$. When we can no longer find positive savings, we set $l := l - 1$ and repeat the same steps while $l \geq 0$.

6.2 Genetic Algorithm

The solutions provided by the savings heuristic, although reasonable, can be significantly improved. We now discuss the main characteristics of a genetic algorithm (GA) developed for the MLCMST that improves upon the savings heuristic. Conceptually, our genetic algorithm approach splits the MLCMST problem into two parts: a grouping problem and a network design problem. The grouping problem aims at finding the lowest cost, feasible partition, $S_1, S_2, \ldots, S_m$ of the set $V\backslash\{c\}$. The partition is feasible if $\sum_{i \in S_\mu} W_i \leq Z_L$, for all $\mu = 1, 2, \ldots, m$. The cost of a partition is determined by finding a minimum cost tree T_μ on all of the sets $S_\mu \cup \{c\}$ and then summing the costs of the individual trees. Although the network design problem on the partition (finding T_μ) is also an MLCMST problem, in practice, we will have to solve it on a much smaller graph than the original. Our GA algorithm searches through possible partitions and evaluates their cost by calling on the savings heuristic we presented earlier.

The steps of the GA procedure are described in Figure 1.19. First, an initial population of feasible partitions (or individuals) $P(t)$ is created. Next, the fitness of each individual in that population is evaluated. In this case, the fitness of a partition is equal to $\sum_{\mu=1\ldots m} c(T_\mu)$

```
Begin
   t ⟵ 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
      t ⟵ t + 1
      select r parents from P(t − 1)
      let the r parents crossover and generate r offspring
      insert the r offspring to P(t)
      select (pop_size − r − m) individuals from P(t − 1) and
         copy them to P(t)
      take the best m individuals of P(t − 1) and mutate them
      insert the mutated individuals to P(t)
      evaluate P(t)
   end while
end
```

Figure 1.19. Steps of the Genetic Algorithm for the MLCMST problem.

($c(T_\mu)$ denotes the cost of T_μ). The evaluation of the fitness of the population is required to check the termination condition of the algorithm and during the selection of individuals. At first, we *select* r individuals (parents) from the old population $P(t-1)$ for reproduction (crossover). The crossover operation results in r new individuals (children) which are assigned to the new population, $P(t)$. Next, $(pop\_size - r - m)$ individuals are *selected* from $P(t-1)$ and are copied to $P(t)$. Additionally, the best m individuals are chosen from $P(t-1)$ to be mutated. The mutated individuals then become part of $P(t)$. Finally, the new population is evaluated and the procedure starts over. Note that all selections throughout the algorithm are done with regard to the fitness values of the individuals (i.e., fitter individuals have a higher chance of being selected than less-fit individuals). We now give a brief overview of the most important aspects of the GA. Additional details can be found in our working paper [23].

Initial Population. The initial population is generated by running our savings heuristic and the Esau-Williams [18] heuristic (with link capacity Z_L) on perturbed versions of the problem graph.

Selection. All selections are done with the typical roulette wheel mechanism on fitness values that were transformed through a *Sigma Truncation* function [38]. This function, $f'(i) = f(i)+(\overline{f}-\gamma\sigma)$, computes the transformed fitness value $f'(i)$ of an individual i, from the original $f(i)$, the population average $\overline{f}$, and the population standard deviation σ. γ is a constant that can be used to control selective pressure.

Representation. Our genetic algorithm uses a representation proposed by Falkenauer [19] for grouping problems in general. This representation consists of an item part and a group part. The item part consists of the group labels assigned to each node and the group part contains a list of the group labels. The representation is shown in Figure 1.7 in the context of the CMST problem. In that case, nodes that belong in the same group are connected with a MST. In our problem, nodes in the same group have to still be connected by a multi-level capacitated spanning tree.

Crossover. Crossovers are handled as suggested by Falkenauer [19]. The crossover operator is a typical two-point crossover but acts only on the group part of the representation. The item part of the representation is updated once the crossover is complete based on the changes done in the group part.

Mutation. Mutations can be handled in many different ways. We use an elaborate local search procedure, which we briefly discuss in Section 1.6.4.

6.3 Computational Results

Extensive computational testing on both the savings heuristic and the genetic algorithm was done on problem instances with different problem sizes (N = 20, 30, 50, and 100) and different central location positions (center, edge, and random). Each problem set (combination of size and central location position) contained 50 randomly generated problem instances. The results of the heuristics were compared to optimal solutions for smaller problems (for which $N = 20$ and some of the instances for which $N = 30$) and to lower bounds. The lower bounds were generated by linear programming relaxations of mathematical formulations for the MLCMST problem. Both procedures were coded in C++ and all runs were done on a Dual Processor Pentium III PC running Windows 2000 with 1GHz clock speed and 512MB of RAM.

When compared to the optimal solutions, the savings heuristic and the GA found solutions that were on average, within 4.29% and 0.25% of

the optimal solutions, respectively. Specifically, the GA solutions were never more than 2.12% away from optimality. For the savings heuristic, however, in the worst case the results were almost 12% away from optimality. When compared to the lower bounds, the savings heuristic was, on average, within 9.91% of the bound with 20.00% being the worst case. The GA was within 6.09% of the lower bound, on average, and never exceeded 11.18%. It is important to note here that for smaller problems the lower bounds were found to be within 6.14% of the optimal solution, on average. In terms of running times, the savings heuristic runs within 0.05 of a second for all problems and the GA runs on average within 45 minutes for larger problems (i.e., $N = 100$).

6.4 Remarks

The MLCMST problem is a challenging network design problem that arises in practice. The results presented here reveal a fairly strong performance from the GA. The savings heuristic provides weaker solutions than the GA, which might be expected from a greedy heuristic for a challenging problem like this. One possible avenue for research is to develop local search (LS) procedures that use neighborhood structures similar to those originally developed for the CMST problem. As we mentioned earlier Ahuja et al. [2, 3] present a multi-exchange neighborhood structure that defines a very extensive neighborhood for CMST problems, and present efficient search procedures for this neighborhood structure. Based on this neighborhood, and in conjunction with our savings heuristic, we have developed a LS algorithm for the MLCMST problem that performs quite well. Complete details on this procedure are in our working paper [23].

There are many potential directions for future research. Different possibilities include improving the savings heuristics, genetic algorithms based on tree (instead of group) representations, and alternative local search procedures. Finally, the heterogenous demand case is even more challenging and many promising directions for a solution procedure exist.

7. Selected Annotated Bibliography

In this section, we provide an annotated bibliography of recent and interesting applications of heuristic search to network design. While the list is certainly not complete, it includes representative work in network design. We have also limited the list to the less extensively studied problems, so we did not include problems such as the Steiner tree problem. The following selection of papers is classified into five groups depending upon the heuristic search technique used: local search, tabu search, ge-

netic algorithms, simulated annealing, and finally approaches that use multiple heuristic search techniques in one place. Under each classification we describe a few papers (usually containing a novel idea or with very good computational results) that are representative of the recent work in that particular heuristic search area.

Interested readers may find more on the general use of these heuristic search techniques in the following references. Surveys of different strategies related to local search, such as iterated local search, variable neighborhood, and guided local search, along with applications of these procedures can be found in the recently edited handbook on metaheuristics by Glover and Kochenberger [28]. More recent applications of tabu search can be found in the survey paper by Hindsberger and Vidal [33], while additional selected applications of heuristic approaches in telecommunication networks can be found in the special issue of the *Journal of Heuristics* edited by Doverspike and Saniee [13].

Applications of Local Search

R. K. Ahuja, J. B. Orlin, and D. Sharma, 2001. "A composite neighborhood search algorithm for the capacitated minimum spanning tree problem," Operations Research Letters, Vol. 31, pp. 185–194.
Ahuja et al. present a neighborhood structure for the CMST problem that allows for multiple exchanges between the subtrees of a feasible tree. Each subtree can be part of the exchange by contributing one of its nodes or one of its subtrees. A new, directed graph is constructed on which all the exchanges and their savings are represented by cycles. The authors also present an exact algorithm that finds the best multi-exchange by searching this new graph. Their computational results indicate that a local search procedure based on this large-scale multi-exchange neighborhood is able to find the best-known solutions for a set of benchmark problems and improve upon the best-known solution in 36% of the cases.

B. Gendron, J.-Y. Potvin, and P. Soriano, 2002. "Diversification strategies in local search for a nonbifurcated network loading problem," European Journal of Operational Research, Vol. 142, pp. 231–241.
Gendron et al. studied the impact of different diversification strategies on the performance of the local search procedure for a nonbifurcated network loading problem. This problem occurs in telecommunications networks, where we are given a central supply node and a set of demand nodes. We are also given multiple facilities of different capacity and cost

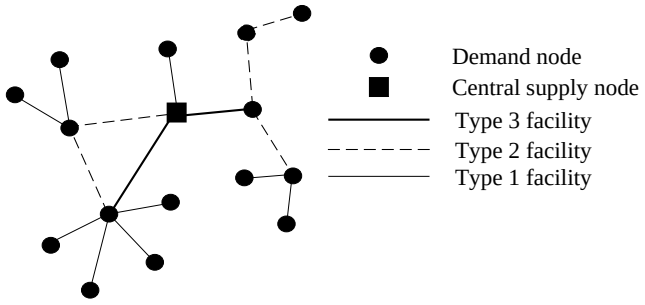

Figure 1.20. An example of a nonbifurcated network loading problem.

that can be installed on links between the nodes. The goal is to lay out facilities in the network so that all demand requirements are met and the total network cost is minimized. Figure 1.20 shows an example of this problem. The proposed solution procedure alternates between construction and local search phase. Gendron et al. presented four diversification strategies: adaptive memory [6], greedy multistart, 2-opt neighborhood, and greedy k-opt. Computational experiments on networks with up to 500 nodes indicate that the best diversification strategy is greedy k-opt, which significantly outperforms other diversification strategies.

M. G. C. Resende, and C. C. Ribeiro, 2003. "A GRASP with path-relinking for private virtual circuit routing," Networks, Vol. 41, No. 3, pp. 104–114.

Resende and Ribeiro have developed a version of GRASP for the private virtual circuit (PVC) routing problem, where the goal is to route off-line a set of PVC demands over a backbone network, so that a linear combination of propagation and congestion-related delays is minimized. The PVC problem occurs in a frame relay service, which provides private virtual circuits for customers connected through a large backbone network. Each customer's request can be seen as a commodity specified by its origin and destination, and required bandwidth, and must be routed over a single path, that is, without bifurcation. The backbone network may consist of multiple links between pairs of nodes and any given link can support a limited bandwidth and a limited number of PVCs. Resende and Ribeiro discussed different path-relinking strategies (a technique described in Glover [27]) that can be used along with their GRASP procedure, and tested their performance on networks with up to 100 nodes and 9900 commodities. These tests showed that, among the tested variants, the most effective variant of GRASP is backward path-relinking from an elite solution to a locally optimal solution. These

experiments also showed that the proposed version of GRASP for the PVC routing significantly outperforms other, simpler, heuristics used by network planners in practice.

D. Fontes, E. Hadjiconstantinou, and N. Christofides, 2003. "Upper bounds for single-source uncapacitated concave minimum-cost network flow problems," Networks, Vol. 41, No. 4, pp. 221–228.

The single-source uncapacitated concave min-cost network flow problem (SSU concave MCNFP) is a network flow problem, with a single source node and concave costs on the arcs. This models many real-world applications. Fontes et. al. proposed a local search algorithm for the SSU concave MCNFP. It starts by finding a lower bound obtained by a state-space relaxation of a dynamic programming formulation of the problem. This lower bound is improved by using a state-space ascent procedure (that considers penalties for constraint violations and state-space modifications). The solutions obtained by the lower bound procedure are first corrected for infeasiblities by an upper-bounding procedure and then iteratively improved by applying local search over extreme flows (extreme flows correspond to spanning tree solutions rooted at the source node and spanning all demand vertices). The extensive tests for two types of cost functions (linear fixed-charge and second-order concave polynomial) over a large set of instances with up to 50 nodes and 200 arcs have shown that the new procedure provides optimal solutions in most instances, while for the remaining few, the average percent gap is less than 0.05%.

Applications of Tabu Search

S. Chamberland, B. Sanso, and O. Marcotte, 2000. "Topological design of two-level telecommunication networks with modular switches," Operations Research, Vol. 48, No. 5, pp. 745–760.

Chamberland et al. have developed a tabu search algorithm that improves upon a greedy heuristic for the design of two-level telecommunication networks. In this problem we are given a set of possible switch sites that need to be connected by high capacity links (OC-192 links) into a tree or ring backbone network. We are also given a set of user nodes that need to be connected to a backbone network using low capacity links (OC-3 links) and a star network structure. Figure 1.21 shows an example of a two-level telecommunication network with a tree backbone network. The low capacity (OC-3) links from users are connected

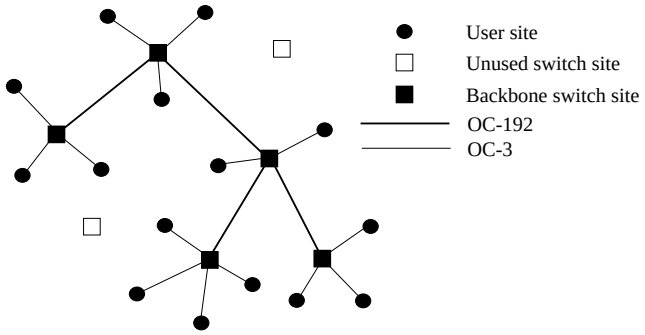

Figure 1.21. An example of a two-level telecommunication network with a tree-backbone network and a star-access network.

to the switch either directly or via a multiplexer. In either case a "port" of the appropriate capacity must also be installed on the switch (in a "slot" on the switch). There are multiplexers of different capacities and costs, and each switch port can connect to at most one multiplexer. By using a multiplexer, multiple users can be connected to a single port (for example an OC-12 multiplexer can handle 4 users). A backbone link between two switches occupies a port on each switch. Each port is installed into a slot on the switch. There is a constraint on the number of slots at each switch site. The network design must be determined so that the total traffic requirements are met and the total network cost is minimized. Chamberland et al. created initial solutions using a greedy heuristic, which were then improved by a tabu search algorithm where the neighborhood structure is defined by the states of slots at switch sites (these states indicate whether a slot is occupied, and if so by what type of port). The tabu search algorithm was tested on networks with up to 500 users and 30 potential switch sites. The proposed algorithm provided results that were, on average, 0.64% and 1.07% from the optimum for the ring and tree topologies, respectively.

D. Berger, B. Gendron, J. Potvin, S. Raghavan, and P. Soriano, 2000. "Tabu search for a network loading problem with multiple facilities," Journal of Heuristics, Vol. 6, No. 2, pp. 253–267.

Berger et al. presented the first tabu search algorithm for a network loading problem with multiple facilities. (This is the same problem as in [26], and an example can be seen in Figure 1.20). The neighborhood of the search space is explored by using the k^{th} shortest path algorithm to find alternative paths for demand nodes in a given solution. Computational experiments on networks with up to 200 nodes and 100 demand

nodes indicated that tabu search provides solutions better than using a steepest-descent local search heuristic based on the 1-opt and 2-opt neighborhoods.

L. A. Cox and J. R. Sanchez, 2000. "Designing least-cost survivable wireless backhaul networks," Journal of Heuristics, Vol. 6, No. 4, pp. 525–540.
Cox and Sanchez developed a short-term memory tabu search heuristic, with embedded knapsack and network flow problems for the design of survivable wireless backhaul networks. In this problem we need to select hub locations and types of hubs along with types of links used in the network so that capacity and survivability constraints are satisfied and network cost is minimized. The tabu search algorithm solved to optimality all problems where the optimum is known in less than 0.1% of time needed to solve the corresponding MIP. On the real-world problems, this algorithm provided 20% improvement over the previously best known designs.

M. Vasquez and J.-K. Hao, 2001. "A heuristic approach for antenna positioning in cellular networks," Journal of Heuristics, Vol. 7, No. 5, pp. 443–472.
Vasquez and Hao developed a three-step procedure utilizing a tabu search algorithm for antenna positioning in cellular networks. In this problem, we need to determine the position, number, and type of antennas at a set of potential locations, while satisfying a set of constraints and optimizing multiple objectives (minimization of the number of the sites used, minimization of the level of the noise within the network, maximization of the total traffic supported by the network, and maximization of the average utilization of each base station). The authors developed a heuristic procedure that was successful in finding feasible solutions for problems where no feasible solution was known previously.

T. G. Crainic and M. Gendreau, 2002. "Cooperative parallel tabu search for capacitated network design," Journal of Heuristics, Vol. 8, pp. 601–627.
Crainic and Gendreau developed a parallel cooperative tabu search method for the fixed charge, capacitated multicommodity flow network design problem. In this problem we are given a set of demand nodes, and we need to route all the traffic through links of limited capacity. However, aside from routing cost, which is proportional to the number of units of each commodity transported over any given link, there is also a fixed charge cost for the use of a link. The proposed method is based

on using multiple tabu search threads, which exchange their search information through a common pool of good solutions. Computational experiments indicate that cooperative parallel tabu search outperforms sequential tabu search algorithms, including the version of tabu search developed by the authors previously in [12]. (The previous tabu search algorithm works in combination with column generation, and, in the earlier paper, was found to significantly outperform other solution approaches in terms of the solution quality.)

I. Ghamlouche, T. G. Crainic, and M. Gendreau, 2003. "Cycle-based neighborhoods for fixed-charge capacitated multicommodity network design," Operations Research, Vol. 51, No. 4, pp. 655–667.
Ghamlouche et al. have defined a new neighborhood structure that can be used for heuristics for fixed-charge capacitated multicommodity network design. The new neighborhood structure differs from the previous ones in the sense that moves are implemented on cycles in the network, instead of individual arcs. More specifically, the proposed neighborhood structure is based on identifying two paths for a given node pair (thus identifying a cycle) and deviating flow from one to another path in this cycle so that at least one currently used arc becomes empty. The exploration of the neighborhood is based on the notion of γ-residual networks, which together with a heuristic for finding low-cost cycles in these networks, is designed to identify promising moves. The new neighborhood structure was implemented within a simple tabu search algorithm, and computational experiments on problems with up to 700 arcs and 400 commodities show that the proposed tabu search algorithm outperforms all existing heuristics for fixed-charge capacitated multicommodity networks.

Applications of Genetic Algorithms

L. Davis, D. Orvosh, L. Cox, and Y. Qiu, 1993. "A genetic algorithm for survivable network design," in Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 408–415, Morgan Kaufmann, San Mateo, CA.
Although this paper is more than a decade old, we include it in this bibliography, since it represents a nice example of how a chromosome structure can be used to simplify evaluation of chromosomes in problems with complex constraints/requirements. Davis et al. developed a GA with a hybrid chromosome structure for the survivable network design problem where a set of links in the network needs to be assigned a

bandwidth (capacity) so that all the demand is routed and survivability requirements are met (these requirements may impose only a certain percentage of the traffic be routed in case of a single link failure in the network). The proposed hybrid chromosome structure contains three groups of genes designed to reduce the complexity of chromosome evaluation with respect to routing and survivability requirements. The first group is the list of capacities assigned to each link, while the remaining two groups represent ordered lists (permutations) of traffic requirements used for the approximate evaluation of the chromosome using greedy heuristics. Computational experiments indicate that the GA provides near optimal solutions (obtained by solving an MIP for this problem) on small problems with 10 nodes and up to 21 links. On the larger problems, with 17 nodes and up to 31 links, where the optimum is not known, the GA outperforms greedy heuristics.

C. C. Palmer and A. Kershenbaum, 1995. "An approach to a problem in network design using genetic algorithms," Networks, Vol. 26 pp. 151–163.
In the optimal communication spanning tree problem (OCSTP) we are given a set of nodes, flow requirements between pairs of nodes, and an underlying graph with the cost per unit of flow for each edge. We would like to design a minimum cost tree network to carry the flow. The cost of an edge in this tree network is the product of cost per unit flow of the edge and the total flow (over all demand pairs) carried on that edge in the tree network design. Palmer and Kershenbaum proposed a set of criteria that can be used for selection of adequate genetic encoding for the OCSTP. They discuss applicability of existing encodings (Prüfer number, characteristic vector, and predecessor encoding), and present a new type of chromosome encoding called link and node bias (LNB) that meets desired criteria for encodings to a greater extent. The GA with the LNB encoding was tested on networks with up to 98 nodes. The results indicate that the proposed procedure provides results up to 7.2% better than results obtained by pure random search or a previously developed heuristic for the OCSTP.

Y. Li and Y. Bouchebaba, 1999. "A new genetic algorithm for the optimal communication spanning tree problem," in Artificial Evolution, 4th European Conference, Dunkerque, France, November 3-5, 1999, Lecture Notes in Computer Science, 1829, Springer, 2000.
Li and Bouchebaba developed a new genetic algorithm for the optimal communication spanning tree problem, where a chromosome is repre-

sented by a tree structure. The genetic operators are specific for the proposed tree chromosome representation, and work directly on the tree structure through exchange of paths and subtrees (crossover operators), or insertion of randomly selected paths or subtrees (mutation operators). Experiments over a small set of instances on complete graphs with up to 35 nodes indicate that the proposed GA provides higher quality solutions than previously developed GAs.

H. Chou, G. Premkumar, and C.-H. Chu, 2001. "Genetic algorithms for communications network design–An empirical study of the factors that influence performance," IEEE Transactions on Evolutionary Computation, Vol. 5, No. 3, pp. 236–249.
Chou et. al. looked at the impact of three factors on the performance of the GA developed for the degree-constrained minimum spanning tree problem: two forms of genetic encoding (Prüfer and determinant encoding) and different versions of mutation and crossover operators. Computational experiments indicated that, in terms of solution quality, the best combination of genetic factors is determinant encoding, exchange mutation (randomly selects two positions in a given chromosome and exchange genes), and uniform crossover (a set of positions, called a mask, is chosen for a chromosome, and their alleles are exchanged with each other based on the generated positions). The results also indicate that, among tested factors, genetic encoding has the greatest impact on solution quality.

F. G. Lobo and D. E. Goldberg. "The parameter-less genetic algorithm in practice," to appear in Information Sciences.
Lobo and Goldberg provided an overview of a parameter-less genetic algorithm that was introduced by Harik and Lobo [32]. The notion behind a parameter-less genetic algorithm is to minimize or eliminate the specification of parameters for a genetic algorithm. Typically, a significant amount of testing is necessary to come up with a good choice of values for GA parameters. The parameter-less genetic algorithm solely uses the crossover operator. It uses a fixed value for the selection rate and probability of crossover. These values are shown to comply with the schema theorem and ensure the growth of building blocks (see [38] for more on the schema theorem). The parameter-less genetic algorithm continuously increases the population size until a solution of desired quality is obtained. This is done in a novel fashion, by running the genetic algorithm on multiple populations. Larger number of iterations are run on smaller populations, while a smaller number of iterations are run on larger populations (in particular if population a is 2^k times larger than

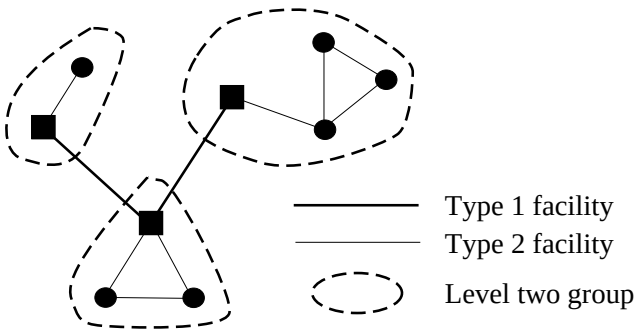

Figure 1.22. An example of a two-level hierarchical network. Type 1 facilities have higher capacity than type 2 facilities.

population b, the algorithm runs 4^k iterations on population b for every iteration on population a). When the average fitness of a small population is lower than the average fitness of a larger population, the smaller population is eliminated. The algorithm is run until a user-specified stopping criteria, is met. The authors illustrate the parameter-less genetic algorithm by applying it to an electrical utility network expansion problem.

Applications of Simulated Annealing

T. Thomadsen and J. Clausen, 2002. "Hierarchical network design using simulated annealing," Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Thomadsen and Clausen define a problem in which a minimum cost hierarchical network has to be designed to satisfy demands between origin/destination pairs of nodes. In this problem, we are given a set of different facility types $\Gamma = \{1, 2, \ldots, L\}$ with varying capacities that can be installed between a pair of nodes. We are also given the cost of installing these facilities. We would like to design a minimum cost *hierarchical network* with sufficient capacity to route the traffic. The requirements of a hierarchical network are specified as follows. The network must be connected. Further, each connected subgraph defined by facilities of type l defines a group of level l. A hierarchical network requires that every group, other than those groups defined by the highest capacity facility in the network, contains exactly one node that is connected to a higher capacity facility. In other words, a group of level l contains exactly one node that is connected to a facility with capacity

greater than that of facility type l. Figure 1.22 provides an example of a two-level hierarchical network. Thomadsen and Clausen develop a simulated annealing algorithm that is based on a two-exchange neighborhood and report solutions with up to 100 nodes and two levels in the network hierarchy.

K. Holmberg and D. Yuan, 2004, "Optimization of Internet protocol network design and routing," Networks, Vol. 43, No. 1, pp. 39–53.
Holmberg and Yuan developed a simulated annealing (SA) heuristic for the network design and routing of Internet Protocol traffic. In this problem, we need to determine both the topology of the network and the capacity of the links so that the total network cost is minimized. (In this version of the problem, two types of costs are encountered: link costs represented by a fixed charge and linear capacity expansion cost, and a penalty cost for lost demand.) Computational experiments on networks with up to 24 nodes and 76 arcs have shown that, in general, SA provides results as good as a MIP-based sequential procedure developed in the same paper. SA, however, proves to be much better in instances with a large number of commodities.

Other Applications

F. Altiparmak, B. Dengiz, and A. E. Smith, 2003, "Optimal design of reliable computer networks: A comparison of metaheuristics," Journal of Heuristics, Vol. 9, pp. 471–487.
Altiparmak et al. looked at the design of reliable computer networks, where network topology is fixed, i.e., the node locations and links connecting nodes are already specified. However, the type of equipment (differing in reliability and cost) used at each node and link needs to be determined, so that subject to a given budget, the total network reliability is maximized. This paper provides a comparison between three heuristic search techniques—steepest descent, simulated annealing, and genetic algorithm—and hybrid forms of SA and GA (referred to as seeded SA and GA), where both SA and GA use the best solution found by steepest descent at the initial stage of the search. Additionally, a memetic algorithm, a GA with steepest descent applied to all new solutions generated in the reproduction phase, was tested against other proposed heuristic search techniques. Computational experiments performed on graphs with up to 50 nodes and 50 links indicated that SA and GA outperform steepest descent, while seeded SA and GA provide only minor improvements over the original SA and GA. The memetic

algorithm, on the other hand, turned out to be the best heuristic search technique for this problem in terms of solution quality.

G. Carello, F. Della Croce, M. Ghirardi, and R. Tadei, "Solving the hub location problem in telecommunication network design: a local search approach," to appear in Networks.
Carello et al. developed a two-step solution procedure for the hub location problem in telecommunication networks. In this problem, we are given a set of access nodes with certain traffic demand, and a set of candidate transit nodes that are used to interconnect the access nodes. The selected transit nodes represent a fully connected backbone network, and the access nodes along with the links connecting these nodes to the backbone network form the tributary network. The costs encountered in this problem include fixed costs for opening a given transit node and installing the necessary node equipment, and the connection costs for all links deployed in the network. More formally, the network hub location problem can be defined as follows: Given a set of access nodes, a set of candidate transit nodes, and a traffic matrix for each type of traffic, select a set of transit nodes and route all the traffic in the network so that all capacity constraints are satisfied and the total network cost is minimized. In the first step of the solution procedure, where only transit nodes are selected, four heuristic procedures were proposed: local search, tabu search, iterated local search, and a random multistart procedure. For the second step, a local search procedure with a choice of limited or complete neighborhood search was proposed. Computational experiments over 19 test instances indicated that a version of tabu search with quadratic neighborhood and with a complete neighborhood search in the second step produces the best results with average error of 0.2%.

8. Conclusion

The number of applications of heuristic search approaches to network design problems in the literature has significantly increased over the last decade. There are several reasons for this. First, these procedures are often simpler to implement compared to more complex exact procedures. Second, researchers have made significant advances in developing neighborhoods and searching effectively through them. Third, there have also been significant advances in the design of genetic algorithms, and in particular in combining genetic algorithms with other heuristic search procedures. Finally, there has been a rapid and steady increase in the average processor speed of desktop/personal computers. Due to all of these factors, when artfully applied, heuristic search techniques enable

us to obtain near-optimal (i.e., high-quality) solutions for difficult problems where existing exact procedures fail or require prohibitively long computation times.

In this chapter, we have tried to provide specific guidelines applicable to a wide class of network design problems. First, we have described some general guidelines, where we have tried to cover a broad range of solution approaches, starting from the most basic ones to the more complex procedures such as local search, steepest descent, GRASP, simulated annealing, tabu search, and genetic algorithms. We have paid special attention to guidelines for using genetic algorithms in network design, since in our experience, this heuristic search technique can provide very good results for this class of problems. To illustrate the application of GAs, we have provided four examples of GAs applied to different network design problems. These four applications differ in the complexity of the GA design, which varies with the nature of the problem. Our sample applications show that for certain problems, such as MLST and PCGMST, very simple genetic algorithms can provide excellent results. More complex design problems, such as the Euclidean nonuniform Steiner tree problem and MLCMST, on the other hand, require more complex GA designs tailored to specific problem characteristics. Finally, we have provided an annotated bibliography that includes a selection of recent applications of heuristic search techniques for numerous network design problems found in the literature.

As a final remark, we would like to emphasize once again that although heuristic search techniques can be a powerful tool for network design problems, the degree to which they are successful depends upon a careful selection of procedures used in the algorithm design. In order to develop a good algorithm, one must keep in mind the specific properties of a particular problem as well as general guidelines for the intelligent application of heuristics.

Acknowledgments

We thank Ian Frommer for assistance with the figures in Section 1.4. The first, third, and fourth authors research was partially supported by the DoD, Laboratory for Telecommunications Sciences, through a contract with the University of Maryland Institute for Advanced Computer Studies.

References

[1] E. H. L. Aarts, J. H. M. Kost, and P. J. M van Laarhoven. Simulated annealing. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial*

Optimization, pages 91–120. Wiley, New York, 1997.

[2] R. K. Ahuja, J. B. Orlin, and D. Sharma. A composite neighborhood search algorithm for the capacitated minimum spanning tree problem. *Operations Research Letters*, 31:185–194, 2001.

[3] R. K. Ahuja, J. B. Orlin, and D. Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.

[4] A. Amberg, W. Domschke, and S. Voß. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice*, 1:9–40, 1996.

[5] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[6] D. Berger, B. Gendron, J. Y. Potvin, S. Raghavan, and P. Soriano. Tabu search for a network loading problem with multiple facilities. *Journal of Heuristics*, 6(2):253–267, 2000.

[7] T. Brüggeman, J. Monnot, and G. J. Woeginger. Local search for the minimum label spanning tree problem with bounded color classes. *Operation Research Letters*, 31:195–201, 2003.

[8] R. S. Chang and S. J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.

[9] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[11] C. S. Coulston. Steiner minimal trees in a hexagonally partitioned space. *International Journal of Smart Engineering System Design*, 5:1–6, 2003.

[12] T. G. Crainic and M. Gendreau. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12(3):223–236, 2000.

[13] R. Doverspike and I. Saniee, editors. Heuristic approaches for telecommunications network management, planning and expansion. *Journal of Heuristics*, 6(1):1–188, 2000.

[14] M. Dror, M. Haouari, and J. Chaouachi. Generalized spanning trees. *European Journal of Operational Research*, 120:583–592, 2000.

[15] D. Du, J. H. Rubenstein, and J. M. Smith, editors. *Advances in Steiner Trees*. Kluwer Academic Publishers, 2000.

[16] C. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks*, 34(3):181–191, 1999.

[17] C. Duin and S. Voß. Solving group Steiner problems as Steiner problems. *European Journal of Operational Research*, 154:323–329, 2004.

[18] L. R. Esau and K. C. Williams. On teleprocessing system design. *IBM System Journal*, 5:142–147, 1966.

[19] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[20] C. Feremans. *Generalized Spanning Trees and Extensions*. PhD thesis, Université Libre de Bruxelles, 2001.

[21] P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In P. Hansen and C. C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2001.

[22] I. Frommer, B. Golden, and G. Pundoor. Heuristic methods for solving Euclidean non-uniform Steiner tree problems. Working paper, Smith School of Business, University of Maryland, 2003.

[23] I. Gamvros, B. Golden, and S. Raghavan. The multi-level capacitated minimum spanning tree problem. Working paper, Smith School of Business, University of Maryland, 2003.

[24] I. Gamvros, S. Raghavan, and B. Golden. An evolutionary approach to the multi-level capacitated minimum spanning tree problem. In G. Anandalingam and S. Raghavan, editors, *Telecommunications Network Design and Management*, chapter 6, pages 99–124. Kluwer Academic Publishing, Norwell, MA, 2003.

[25] B. Gavish. Topological design of telecommunications networks - local access design methods. *Annals of Operations Research*, 33:17–71, 1991.

[26] B. Gendron, J. Y. Potvin, and P. Soriano. Diversification strategies in local search for a nonbifurcated network loading problem. *European Journal of Operational Research*, 142:231–241, 2002.

[27] F. Glover. Tabu search and adaptive memory programming - advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, Boston, 1996.

[28] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, 2002.

[29] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, 1997.

[30] B. Golden, S. Raghavan, and D. Stanojević. The prize-collecting generalized minimum spanning tree problem. Working paper, Smith School of Business, University of Maryland, 2004.

[31] B. Golden, S. Raghavan, and D. Stanojević. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, to appear.

[32] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 258–265, San Francisco, CA, USA, 1999. Morgan Kaufmann.

[33] M. Hindsberger and R. V. V. Vidal. Tabu search - a guided tour. *Control and Cyberentics*, 29(3):631–661, 2000.

[34] A. Kershenbaum. When genetic algorithms work best. *INFORMS Journal on Computing*, 9(3):254–255, 1997.

[35] S. O. Krumke and H. C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[36] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. American Math. Society*, volume 2, pages 48–50, 1956.

[37] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

[38] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Heidelberg, 1996.

[39] C. L. Monma and D. F. Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531–541, 1989.

[40] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26:231–241, 1995.

[41] C. C. Palmer and A. Kershenbaum. An approach to a problem in network design using genetic algorithms. *Networks*, 26:151–163, 1995.

[42] J. C. Park and C. G. Han. Solving the survivable network design problem with search space smoothing. *in Network Optimization, Lecture Notes in Economics and Mathematical Systems, Springer Verlag*, 450, 1997.

[43] L. S. Pitsoulis and M. G. C. Resende. Greedy randomized adaptive search procedure. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, chapter 3, pages 168–183. Oxford University Press, 2002.

[44] P. C. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, 2002.

[45] M. Prais and C. C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.

[46] R. C. Prim. Shortest connecting networks and some generalizations. *Bell System Tech. J.*, 36:1389–1401, 1957.

[47] F. Rothlauf, D. E. Goldberg, and A. Heinzl. Network random keys - a tree representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, 10(1):75–97, 2002.

[48] F. Salman, R. Ravi, and J. Hooker. Solving the local access network design problem. Working paper, Krannert Graduate School of Management, Purdue University, 2001.

[49] Y. M. Sharaiha, M. Gendreau, G. Laporte, and I. H. Osman. A tabu search algorithm for the capacitated shortest spanning tree problem. *Networks*, 29:161–171, 1997.

[50] S. J. Shyu, P. Y. Yin, B. M. T. Lin, and M. Haouari. Ant-tree: an ant colony optimization approach to the generalized minimum spanning tree problem. *Journal of Experimental and Theoretical Artificial Intelligence*, 15:103–112, 2003.

[51] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84:99–101, 2002.

[52] Y. Xiong, B. Golden, and E. Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. Working paper, Smith School of Business, University of Maryland, 2003.

[53] Y. Xiong, B. Golden, and E. Wasil. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters*, to appear.