

A New Evolutionary Approach to the Degree-Constrained  
Minimum Spanning Tree Problem

Joshua Knowles and David Corne

Department of Computer Science, University of Reading, UK

J.D.Knowles@reading.ac.uk, D.W.Corne@reading.ac.uk

July 27, 1999

## Abstract

Finding the degree-constrained minimum spanning tree ( $d$ -MST) of a graph is a well-studied NP-hard problem of importance in communications network design and other network-related problems. In this paper we describe some previously proposed algorithms for solving the problem, and then introduce a novel tree construction algorithm called the Randomised Primal Method (RPM) which builds degree-constrained trees of low cost from solution vectors taken as input. RPM is applied in three stochastic iterative search methods: simulated annealing, multi-start hillclimbing, and a genetic algorithm. While other researchers have mainly concentrated on finding spanning trees in Euclidean graphs, we consider the more general case of random graph problems. We describe two random graph generators which produce particularly challenging  $d$ -MST problems. On these and other problems we find that the genetic algorithm employing RPM outperforms simulated annealing and multi-start hillclimbing. Our experimental results provide strong evidence that the genetic algorithm employing RPM finds significantly lower-cost solutions to random graph  $d$ -MST problems than rival methods.

*Index Terms*—Degree-constrained minimum spanning tree, Prim’s algorithm, network design, hybrid heuristics, simulated annealing, multi-start hillclimbing, evolutionary algorithm.

# 1 Introduction

The minimum spanning tree (MST) of a graph is an important concept in the design of communication networks. It can be solved in polynomial time, and Moret and Shapiro [1] assess the practical performance of several constructive algorithms for solving it. In real networks, however, the vertices (or nodes) are usually subject to a degree-constraint. For example, many exchanges or switches can only be physically connected to a limited number of linking wires. Also, when designing a network for maximum reliability, introducing a degree-constraint limits the damage that may be caused by a single exchange failure. Unlike the MST, the  $d$ -MST of a graph cannot, in general, be found using a polynomial time algorithm. In fact, for a given rational number  $R \geq 1$ , finding a spanning tree of maximum degree at most  $d$ , and of total weight at most  $R$  times that of the optimal solution, is NP-hard [2].

Several different approaches for solving the  $d$ -MST problem have been taken in previous research. Narula and Ho [3] investigated three methods, one of which (a branch and bound algorithm) is guaranteed to converge to a globally optimal solution. In their methods good upper bounds are generated by a modification to Prim's algorithm [4], which we refer to as *d-Prim's* in this paper. Savelsbergh and Volgenant [5] also employed a branch and bound technique with improved heuristics leading to considerably faster run times than those achieved by Narula and Ho. Random, non-Euclidean graphs were found to be far less tractable than Euclidean graphs, however, with the CPU time needed to solve standard instances rising from a mean of 0.2 seconds (SD = 0.7s) for  $n = 30$  to a mean of 74.2 seconds for  $n = 70$  (SD = 369.5s), where  $n$  is the number of vertices

in the graphs. Khuller *et al.* [6] show that for an arbitrary collection of points in the plane there exists a degree-3 spanning tree of at most 1.5 times the MST and a degree-4 spanning tree of at most 1.25 times the MST. They also give algorithms that compute these trees in  $O(n)$  time, given an MST as part of the input. These results are also extended to Euclidean points in  $n$ -dimensional space. Fekete *et al.* [7] report a network flow technique that improves upon these results, working again in linear time. They do not consider the more difficult non-Euclidean case, however. Recent research by Boldon *et al.* [8] describes a ‘dual simplex’ approach, based on Prim’s algorithm (the best in terms of speed and memory usage for finding the MST [1]), which produces good results on random, non-Euclidean graphs where the underlying MST has degree up to 20 and the degree-constraint of 4 is used. Finally, Zhou and Gen [9] propose a genetic algorithm employing a Prüfer numbering system for searching the  $d$ -MST space.

Algorithms for the  $d$ -MST are often designed with Euclidean graphs in mind, and/or tested only on Euclidean graphs. However, although the cost function defined on links can sometimes be closely related to Euclidean distance, this relationship is often confounded in practice. For example, where link costs are defined to be communications costs between nodes, physical distance can be a very minor factor in comparison to others such as the type, quality, maintainability, speed, and corporate provider of the link in question.<sup>1</sup> In general, therefore, it is valuable to test and compare algorithms for the  $d$ -MST on non-Euclidean graphs. In such cases, the algorithms of [6] and [7] are not applicable, while those in [5] are cumbersome.

---

<sup>1</sup>We note that these real-world considerations may best be tackled using a multicriteria approach but reserve such investigations for future work.

We describe a technique designed to achieve fruitful hybridisation of a spanning tree construction algorithm with stochastic iterative search. This method, which we call RPM (Randomised Primal Method) is employed in three stochastic, iterative search techniques: multi-start hillclimbing, simulated annealing [10], and a genetic algorithm [11]. The quality of solutions found by these techniques are compared with each other, and also with those achieved by the dual simplex algorithm of [8] and the upper bound generated by Narula and Ho's  $d$ -Prim's method [3]. These comparisons are made on a variety of randomly generated  $d$ -MST problems of from 50 to 250 nodes. Some of the random graphs used for testing have been created to be particularly difficult and misleading in order to emphasise differences in algorithm performance. The procedures by which they were generated are included so that other researchers may easily compare their results with ours.

The organisation of this paper is as follows: Section 2 describes the  $d$ -MST problem formally and considers the relative difficulty of different versions of the problem. Section 3 describes the main approaches compared in this paper and introduces RPM, a new approach for use in stochastic iterative search techniques. Section 4 details two procedures for producing random graphs, one used by Boldon *et al.* [8], and another we have developed which leads to particularly challenging  $d$ -MST problems. Section 5 provides a brief outline of the three stochastic iterative search techniques that are compared here, together with parameter settings for each. Section 6 records the results achieved when comparing the methods of Boldon *et al.* [8],  $d$ -Prim's [3], and RPM (multi-start hillclimbing, simulated annealing, and genetic algorithm versions) on 20 random graphs with a

range of degree-constraints. The performance of the genetic algorithm is also compared with that proposed by Zhou and Gen [9] on a 9-node graph first presented by Savelsbergh and Volgenant [5]. Finally, some discussion and a summary are provided in Section 7.

## 2 Problem Definition

The degree-constrained minimum spanning tree problem can be stated as follows:

Let  $G = (V, E)$  be a connected weighted undirected graph, where  $V = \{v_1, v_2, \dots, v_n\}$  is the vertex set of  $G$ , and  $E = \{e_1, e_2, \dots, e_m\}$  is the edge set of  $G$ . Let  $W = \{w_1, w_2, \dots, w_m\}$  represent the weight or cost of each edge where the weight is restricted to be a non-negative real number.

Any subgraph of  $G$  can be described using a vector  $\mathbf{x} = (x_1, x_2, \dots, x_m)$  where each element  $x_i$  is defined by:

$$x_i = \begin{cases} 1 & \text{if edge } e_i \text{ is part of the subgraph} \\ 0 & \text{otherwise} \end{cases}$$

Let  $S$  be a subgraph of  $G$ .  $S$  is said to be a spanning tree in  $G$  if

1.  $S$  contains all the vertices of  $G$ .
2.  $S$  is connected and contains no cycles.

Now let  $T$  be the set of all spanning trees in  $G$ , then a minimum spanning tree is defined as

$$\min\{z(\mathbf{x}) = \sum_{i=1}^m w_i x_i \mid \mathbf{x} \in T\} \quad (1)$$

If  $b_j$  is the constraint on the degree  $d_j$  of each vertex  $v_j$  in the graph  $G$ , then a degree-constrained minimum spanning tree is defined as

$$\min\{z(\mathbf{x}) = \sum_{i=1}^m w_i x_i \mid d_j \leq b_j, v_j \in V, \mathbf{x} \in T\} \quad (2)$$

In this paper we consider only problems where the degree-constraints on all vertices in the graph  $G$  are the same, i.e., *uniform* degree-constraint problems where  $b_j = d$ ,  $2 \leq d \leq n - 1$ .

Whereas the MST problem is solvable in polynomial time using one of several known constructive algorithms (see Section 3), the  $d$ -MST for general  $d$ ,  $2 \leq d \leq n - 1$ , is NP-hard. This is obvious if one considers the special case where  $d = 2$  because the  $d$ -MST then becomes equivalent to the minimum-weight Hamiltonian path problem which is known to be NP-hard [12].<sup>2</sup>

The  $d$ -MST problem can be divided into two categories; Euclidean problems and random table problems. If the graph in which a tree is sought models a network laid out in a Euclidean space, such that the edge weights between each pair of connected vertices represents the Euclidean distance between them, then the problem is Euclidean. The space may have more than two dimensions though the case where the vertices all lie

---

<sup>2</sup>For  $d > 2$  and a *particular* graph the problem of finding the  $d$ -MST may not be in NP, however we may say for general  $d$ , the  $d$ -MST is NP-hard since for *all* graphs finding the  $d$ -MST is NP-hard when  $d = 2$ .

on a plane is particularly relevant to network design, for obvious reasons. In Euclidean problems the edge weights are said to obey the triangle inequality. In contrast, random table problems are generated by assigning a random weight from a given interval to each edge. In such a problem, therefore, the edge weights do not necessarily obey the triangle inequality.

In general, 2-d Euclidean problems are easier to solve than random table problems. This is because for every set of points in the plane there exists a degree-5 MST [13] whereas in random table problems the maximum degree of a vertex is, in theory, bounded only by the number of vertices in the graph. It is thus much more difficult to move from a solution of the unconstrained MST to the solution of the  $d$ -MST in some random table graphs than in any 2-d Euclidean graph.

In fact, for points in the plane, Khuller *et al.* [6] show that there always exists a spanning tree of degree 3 whose weight is at most 1.5 times the weight of a minimum spanning tree. For spanning trees of degree 4 the ratio reduces to 1.25. And for points in higher-dimensional spaces, there exists a spanning tree of degree 3 whose weight is at most  $5/3$  times the weight of a minimum spanning tree. No such guarantees exist for random table problems.

There are thus many good heuristics for finding low-weight degree-constrained spanning trees in Euclidean problems [6, 7, 5] that guarantee to meet the above performance criteria and that work in polynomial time. Random table problems have not been so susceptible to these heuristics, however.

Many researchers have considered only Euclidean problems because of the intuitive

notion that the cost of a link is related to its length. However, the cost of links depends on many factors, not just length. In this paper we therefore concentrate on the more difficult and general problem of finding low cost  $d$ -STs in random graphs. The next section outlines our approach in the context of other methods.

## 3 Approaches

### 3.1 Overview

In this section we describe several heuristic methods for finding low-weight degree-constrained spanning trees. Three of them are based upon alterations or additions to Prim's algorithm [4] for finding an MST. We therefore begin by giving a brief description of the operation of this classic algorithm.

1. Let  $U$  be an unordered list of vertices initially containing all the vertices of  $G$  exactly once, and denoting the unconnected vertices.
2. Let  $C$  be an unordered list of vertices denoting the connected vertices and let  $C$  be initially empty.
3. Let  $E(M)$  be the edge list of the spanning tree  $M$  in  $G$ , to be constructed.
4. Remove the first vertex from  $U$  and place it in  $C$ .
5. Choose a minimum weight edge connecting a vertex in  $C$  with a vertex in  $U$ , and place it in  $E(M)$ .
6. Remove the unconnected vertex chosen in step 5 (as one half of the edge) from the connected list  $U$  and place it in  $C$ .

7. Repeat steps 5 and 6 until there are no vertices left in  $U$ .

Efficient implementation of Prim’s algorithm, making good use of priority queues, leads to a running time of  $O(m \log n)$  where  $m$  is the number of edges and  $n$  is the number of vertices [14].

There are three methods by which Prim’s algorithm can be employed for finding solutions to the  $d$ -MST. The first method is to make an alteration to Prim’s algorithm so that it does not add edges that violate the degree-constraint. The second, a dual simplex approach, begins with a superoptimal solution to the problem calculated using Prim’s algorithm. A constrained solution is then sought by making alterations that force compliance with the degree-constraint. Narula and Ho [3] and Boldon *et al.* [8] both use this approach. The third approach, which we present for the first time, is to perturb Prim’s algorithm through the use of an input solution vector, so that it does not always choose the minimum weight edge at each step in the tree construction. The resulting solutions can then be used as the basis for a stochastic, iterative search.

For comparison, we also describe the Prüfer numbering scheme used by Zhou and Gen [9].

### 3.2 Calculating Upper Bounds Using $d$ -Prim’s

Narula and Ho [3] described three algorithms for the  $d$ -MST problem and compared their performance. One of them, their *primal* method, starts with a feasible  $d$ -ST and then attempts to improve upon this solution by exchanging edges. Of interest to us is the method of generating the initial  $d$ -ST because it is simple, fast, and gives a reasonable

upper bound on the solution to the  $d$ -MST problem. Later, we use the method as a ‘baseline’ against which we compare the other algorithms presented in this paper.

The method works by modifying step 5 in Prim’s algorithm, above, so that it becomes: ‘Choose a minimum weight edge connecting a vertex in  $C$  with a vertex in  $U$  that would not violate the degree constraint on the vertex chosen from  $C$ , and place it in  $E(M)$ .’ We refer to this method as  $d$ -Prim’s. Note that the cost of a  $d$ -ST generated using  $d$ -Prim’s is dependent on which vertex is used as the starting component.

### 3.3 The Dual Simplex Approach

Boldon *et al.* [8] also utilise Prim’s algorithm in their heuristic approach to solving the  $d$ -MST, arguing that using a good algorithm for computing the MST of a graph gives a good basis for solutions to the  $d$ -MST. Their dual simplex approach involves iteration of two stages until a convergence criterion is reached. Stage 1 is to simply run Prim’s algorithm; this will of course always find a minimal spanning tree, but one which, in early iterations at least, will typically violate the degree-constraint on several nodes. Stage 2 is to adjust the edge weights using a ‘blacklisting function’, which generally increases the weights on edges that were involved in a node degree violation. Every subsequent iteration of stage 1 runs Prim’s algorithm with reference to the newly adjusted weights from the previous Stage 2. This continues until a tree is found that satisfies the degree-constraint (or returns failure if no such tree is found within the maximum number of iterations). This tree, evaluated with respect to the original set of edge-weights, is then returned as the result.

Boldon *et al.* describe five blacklisting functions [8]; the best is found to be their second function, ‘BF2’. It updates edge weights in the following way:

$$newweight = weight + fault * max * (weight - min) / (max - min)$$

where *newweight* is the adjusted weight of the edge, *weight* is the most recent weight of the edge, *fault* is a variable having three allowable values 0, 1, or 2 and denoting the number of vertices incident to the edge which currently violates the degree constraint. The variables *max* and *min* are the maximum and minimum weight in the current spanning tree, respectively. In stage 2 of the method, this function is applied to the weights of all edges involved in a degree violation, with the exception of any that happen to be the minimal weight edge on a violated node.

Using this approach, Boldon *et al.* report *d*-MST’s that are within 0.5% of the MST on a number of large TSP problems. However, solutions are not always found in the allowed 200 iterations.<sup>3</sup> In the following, we will refer to the dual simplex method of Boldon *et al.* [8], using BF2 as the blacklisting function, as simply ‘BF2’.

### 3.4 Prüfer Numbering for Use in a Genetic Algorithm

Zhou and Gen [9] used Prüfer numbers [4] to encode their potential solutions. This has two advantages: every Prüfer number represents a valid spanning tree so that no repairing is necessary, and the degree of each vertex is represented explicitly so it is easy to check whether a solution violates the degree-constraint. However, Zhou and Gen [9] also point out some disadvantages: generating solutions through the standard genetic operators of

---

<sup>3</sup>The number which was regarded by the original authors as a failure of the algorithm to find any valid solution to the problem (presumably arrived at through empirical experience).

crossover and mutation lead to solutions that violate the degree-constraint so that they must be modified. In addition, similar Prüfer numbers may describe spanning trees that are very different, containing few common edges. This makes searching the space very difficult, with the genetic algorithm liable to drift rather than converge.

### 3.5 RPM for Use in Stochastic Iterative Search

In order to apply stochastic iterative search techniques to the  $d$ -MST problem, some means of generating and encoding valid potential solutions to the problem must be devised.

The simplest encoding would be to simply list the  $n - 1$  edges to be included in the spanning tree. However, most lists of  $n - 1$  edges do not constitute a valid spanning tree. In fact, for an 18-node complete graph there are ‘only’  $1.2 \times 10^{20}$  spanning trees compared to  $1.54 \times 10^{22}$  combinations of  $n - 1$  edges. Therefore, if this encoding were used, a great deal of time would have to be spent in checking the validity of the potential solutions.

A better encoding could be based on an algorithm that builds valid spanning trees. The advantage is that no time is spent generating illegal solutions that must then be repaired or discarded. It would also seem a good idea to utilise cost knowledge within the encoding, and thus potentially prevent much more time being spent in the regions of particularly poor spanning trees. An indirect encoding such as this is similar to the method of Zhou and Gen [9] described above. However, their encoding does not exploit any of the information about the costs of edges when generating potential solutions; it also has the additional drawback that similar Prüfer numbers do not correspond to similar spanning trees, leading to a rugged search landscape.

An ideal encoding should have the following properties:

1. Encodes only valid spanning trees.
2. Encodes only trees which meet the degree-constraint.
3. Similar solution vectors (genotypes) should correspond to similar looking trees with many common edges (phenotypes).
4. Edge cost information is utilised so that high cost solutions are rarely generated.

Encoding solutions to incorporate these properties reduces the search space and also leads to a smoother fitness landscape. The algorithm and encoding devised in order to meet these encoding criteria, called the Randomised Primal Method (RPM), is described next.

The RPM employs a dynamic data structure consisting of a table  $T$ , which associates a sorted edge list with each vertex. At initialisation, the entry in  $T$  for any vertex  $i$  is a list of all edges incident on  $i$ , sorted in descending order of weight (Table 1). While RPM runs, the spanning tree is constructed iteratively, and  $T$  is altered dynamically whenever a new edge is added to the growing tree. The idea is that the sorted edge list associated with each vertex  $i$  in  $T$  only contains *valid edges not yet included* in the tree. In other words, an edge only appears in the list associated with vertex  $i$  if it is not already in the growing tree, and if its other incident vertex is also not yet included in the growing tree.

We use  $T$  in our implementation of  $d$ -Prim's algorithm during step 5 when we need to select the lowest weight edge connecting a vertex already in the tree to one not yet in the tree. To execute this step, we look at all of the entries in  $T$  for vertices already in

edge choice	vertex $i$								
	1	2	3	4	5	6	7	8	9
1 (weight)	3 (224)	3 (200)	2 (200)	2 (200)	4 (400)	4 (200)	4 (200)	7 (361)	7 (424)
2 (weight)	2 (224)	4 (200)	1 (224)	7 (200)	7 (447)	7 (283)	6 (283)	9 (500)	8 (500)
3 (weight)	6 (300)	1 (224)	4 (400)	6 (200)	2 (447)	2 (283)	8 (361)	6 (500)	5 (510)
4 (weight)	4 (361)	6 (283)	6 (447)	1 (361)	9 (510)	1 (300)	2 (400)	4 (539)	4 (583)
5 (weight)	7 (539)	7 (400)	5 (566)	3 (400)	3 (566)	3 (447)	9 (424)	2 (728)	6 (707)
6 (weight)	5 (671)	5 (447)	7 (600)	5 (400)	6 (600)	8 (500)	5 (447)	5 (781)	2 (762)
7 (weight)	8 (800)	8 (728)	9 (949)	8 (539)	1 (671)	5 (600)	1 (539)	1 (800)	1 (943)
8 (weight)	9 (943)	9 (762)	8 (992)	9 (583)	8 (781)	9 (707)	3 (600)	3 (992)	3 (949)

Table 1: Example edge choice table  $T$  for a 9-node graph at initialisation. For each vertex a list  $l_i$  of incident edges is given in descending order of weight. The edges choices are referred to by the edge's other incident vertex, and the weight of the edge is shown in brackets.

the tree, and return the top (lowest weight) edge from each of their associated edge lists.

This gives us a set  $L$  of low-weight edges. The lowest-weighted of these (breaking ties randomly) is then selected.

Now, RPM works in a very similar way to  $d$ -Prim's, with the exception that we do not always choose the *first* edge in each edge-list to go into  $L$ . Instead, we construct the low-weight set  $L$  by choosing the  $a(i, d_i)$ th edge from each appropriate entry in  $T$ . That is, for vertex  $i$  having current degree  $d_i$ , we choose the  $a(i, d_i)$ th lowest cost edge from list  $l_i$ . The allele values  $a$ ,  $1 \leq a \leq n$ , for each vertex  $i$  at each degree level  $d_i < d$  are given as input to RPM in the form of a tabular chromosome or solution vector (see Table 2). So, RPM is a means of decoding a chromosome into a valid low-weight  $d$ -ST. In our implementation, the values of  $a$  stored in the chromosome are randomly initialised with a bias towards lower values. Low values of  $a$  ensure that lower weight edges are selected more often and hence the chromosome will encode for a low-weight  $d$ -ST.

To illustrate RPM, consider the partially constructed 3-ST shown in Figure 1. RPM

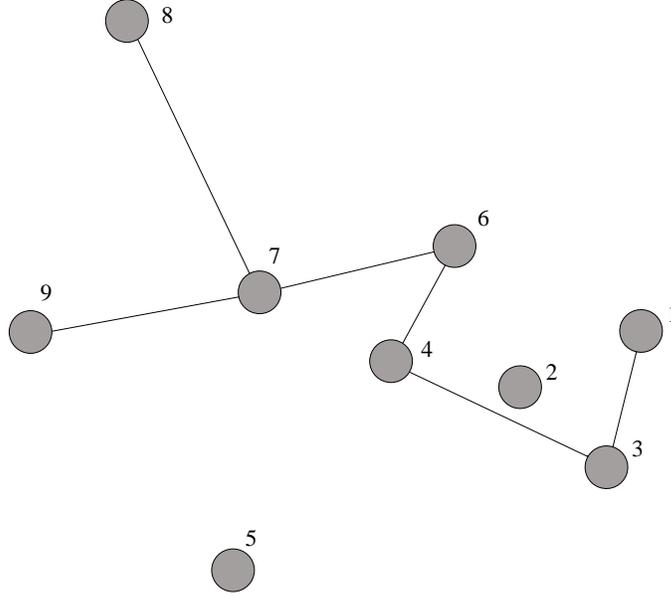


Figure 1: Partially constructed 3-ST in a 9-vertex graph.

has been running, and there are currently 6 edges (7 vertices) in the growing tree. The vertices in the tree so far are: 1 3 4 6 7 8 9 . Now, to add the next edge, the edge list of each of these vertices will be looked at in turn. Table 3 shows the edge choice list for each vertex  $i$ , as they would appear for the graph in Figure 1. All of them contain just two valid edges: the edges that connect to vertices 2 and 5. In  $d$ -Prim's, the weights of the *first* edge in each of the seven lists would be compared, and the lowest of these selected. But in RPM, a tabular chromosome is used as an edge-choice look-up table to guide the choice of edges considered.

A typical chromosome for use in RPM is shown in Table 2. It has a tabular (2-d) form with length  $n$  and depth  $d - 1$ . Each allele value  $a(i, d_i)$ ,  $1 \leq a(i, d_i) \leq n$ , represents the choice of edge that RPM will use when it is considering the sorted edge-list of vertex

current degree of vertex $i$ : $d_i$	vertex $i$								
	1	2	3	4	5	6	7	8	9
1	<b>2</b>	1	4	2	3	2	1	<b>5</b>	<b>1</b>
2	3	1	<b>2</b>	<b>1</b>	1	<b>2</b>	1	3	2

Table 2: Example tabular chromosome used to encode for a degree-3 spanning tree in a 9-node graph via the RPM. At each step in generating the tree the chromosome is consulted: For each vertex  $i$  in the tree having degree  $d_i < d$  the allele  $a(i, d_i)$  is looked up in the chromosome in position  $(i, d_i)$ . The allele represents the choice in the edge list  $l_i$  in table  $T$  which will be selected to go forward into the low-weight edge set  $L$ . The highlighted values in the table represent the values that will be looked up in the iteration where the next edge is being selected in the partially constructed tree given in Figure 1.

edge choice	vertex $i$								
	1	2	3	4	5	6	7	8	9
1 (weight)	2 (224)	–	2 (200)	<b>2 (200)</b>	–	2 (283)	–	2 (728)	<b>5 (510)</b>
2 (weight)	<b>5 (611)</b>	–	<b>5 (566)</b>	5 (400)	–	<b>5 (600)</b>	–	<b>5 (781)</b>	2 (762)

Table 3: Updated edge choice lists for the partially constructed tree in Figure 1. The highlighted choices are those that are encoded for by the chromosome in Table 2. These edges are placed in the low-weight edge set  $L$  and the lightest is chosen. The edge (4, 2) is the lightest edge in  $L$  and so it will be added to the tree in this iteration.

$i$ , the latter having a current degree,  $d_i$ . In Table 2, the alleles that will be used to help in choosing the next edge in the partially constructed tree in Figure 1 are shown in bold face. These allele values are then used to select the edges that will go into the low-cost edge set  $L$ . Table 3 gives the updated version of Table 1 for the partially-constructed tree. The edges that the chromosome instructs RPM to choose, in this table, are shown in bold. Where the chromosome instructs RPM to choose an edge off the bottom of a list (e.g. the 5th edge in vertex 8's list), it simply takes the lowest edge available. Note that genes in columns 2, 5, and 7 are ignored because node 7 in the graph is already at the degree limit, and nodes 2 and 5 are not yet connected. In this example the edge (4, 2) is

selected as the next edge to join the growing tree.

In summary, the randomised primal method (RPM) employs a tabular chromosome of length  $n$  and depth  $d - 1$ , where  $n$  is the number of vertices in the presented graph and  $d$  is the required degree constraint. It decodes the chromosome to construct a degree-constrained spanning tree  $S$  iteratively as follows:

*First step* : Place an arbitrary vertex in  $S$ .

*General step* : Add an edge joining a vertex  $i$  that is already in  $S$  and whose degree  $d_i$  is less than the degree constraint  $d$ , to a vertex not in  $S$ .

The particular edge chosen in the *General step* above is dependent upon the edge weights in the underlying graph and the values stored in the tabular chromosome. The *General step* is implemented in the following way:

1. For each vertex  $i$  in  $S$  whose current degree  $d_i$  is less than  $d$ , construct a sorted list  $l_i$  containing each of the vertex's incident edges which join it to vertices not yet in  $S$ , in descending order of weight.
2. For each list  $l_i$  look up the allele value  $a(i, d_i)$  stored in the tabular chromosome in position  $(i, d_i)$ . Place the  $a(i, d_i)$ th edge from list  $l_i$  into a low-cost edge set  $L$ .
3. Select the lowest weighted edge in  $L$ .

When used with our stochastic iterative search techniques, chromosomes are initialised randomly using a negative exponential probability distribution. Using this, most genes

contain allele values close to 1, forcing the RPM to choose one of the lowest-weight available edges nearly all the time. Search is then carried out in the normal way: chromosomes (solution vectors) are decoded using RPM, evaluated, and selection and variation are used to direct the search.

The small change (neighbourhood) operator used in all our search algorithms acts on about 1% of genes and works by setting the gene to a new value from the same distribution as in the initialisation phase. In the genetic algorithm a standard uniform crossover operator [15] was also applied.

## 4 Different Techniques for Generating Random Graphs

Random graphs are those in which the weight of each edge has been generated randomly from a uniform distribution within some predefined range. In theory, this means that the maximum degree of a node in the underlying MST could be quite large. However, Boldon *et al.* [8] have found that when reasonably large graphs are generated this is rarely the case; in fact the degree rarely exceeds four. Hence, they employed a means of generating biased random complete graphs in which a high maximum-node degree is present in the underlying MST. We adopted their procedure, which is outlined below:

The generator has four input parameters:

$n$ , the number of vertices in the graph,

$f$ , the number of vertices with large degree,

$ld$ , the lower bound on the degree of large-degree vertices, and

$ud$ , the upper bound on the degree of large-degree vertices.

It first constructs an  $n$ -vertex tree that will form the (unconstrained) MST of the complete graph, as follows. The tree is constructed by first forming  $f$  different ‘star’ graphs with the degree of each star chosen uniformly at random from the range  $[ld, ud]$ . These stars are then connected by adding  $f - 1$  edges at random, to form a tree. Since the tree constructed so far may have fewer than  $n$  vertices, an  $n$ -vertex tree is formed by adding further vertices and edges at random. This  $n$ -vertex tree forms the MST of the  $n$ -vertex complete graph. The weights of the edges in the MST are given random values in the range  $[0, 0.1]$ . Finally, all the other edges in the graph are added and given random values in the range  $(0.1, 1.0]$ .

In order to emphasise differences in algorithm performance, we developed a further procedure for generating particularly difficult and misleading random graphs (again with large degree in the MST). It is based heavily on the procedure described above. It differs when the additional vertices are added to make the total number of vertices up to  $n$ . At this stage, the additional vertex is connected to one of the vertices at the centre of the stars already generated, with an edge weight that is towards the top end of the range of weights in the underlying MST. Then, all the other edges emanating from the additional vertex are assigned weights that are towards the top end of the range of edges *not* in the MST. Figure 2 shows part of such a graph and the range of values that each type of edge takes.

A misleading graph, generated using the procedure above, from now on called an ‘ $M$ -graph’, will strongly deceive algorithms based upon choosing the lowest-weight edge at

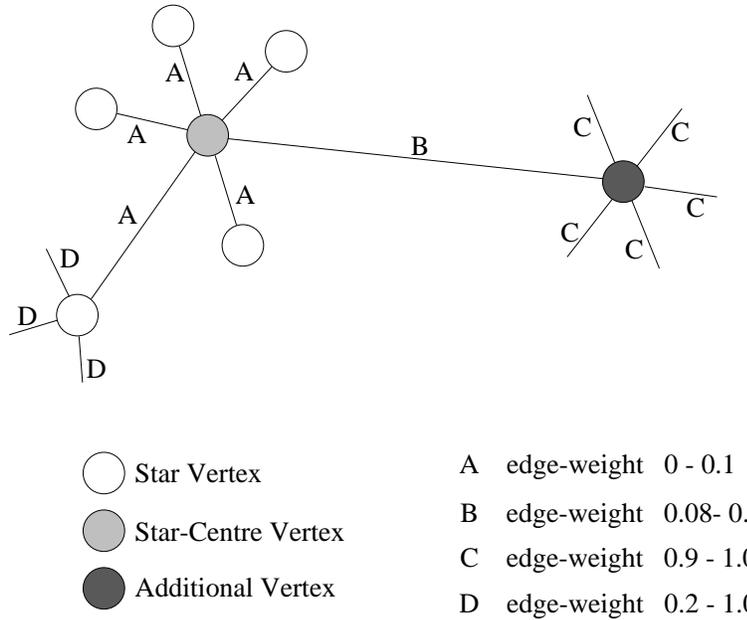


Figure 2: Illustrating Edge-Weight Distributions of the ‘ $M$ -graph’ Generator.

each iteration, such as Prim’s algorithm.<sup>4</sup> This is because when the choice is being made for an edge incident to a star centre vertex the algorithm will usually choose an edge of type A, since these have the lowest weights. Once this has occurred three times for the same star centre vertex (where  $d = 3$ ), no more edges can be connected to it. This means that the neighbouring additional vertex can no longer be connected to the graph by an edge of type B (having a weight  $\leq 0.1$ ) and so it will later be connected by an edge of type C which has a much higher weight ( $\geq 0.9$ ). The graph has misled the algorithm.

Note that standard random table graphs will mislead constructive algorithms as above, which is why the  $d$ -MST problem is NP-hard. Our motivation for the above graph gen-

---

<sup>4</sup>Note that although the  $M$ -graphs have been constructed to deceive algorithms based on greedy heuristics such as those used in Prim’s algorithm, they will present difficulties to any algorithm which uses edge weight information to build a  $d$ -ST.

eration techniques is merely to increase the amount of deception that will normally be encountered to emphasise the differences in the performances of the various algorithms.

## 5 Stochastic Iterative Search Techniques

In Section 3 we introduced a new means of generating good, random solutions to the constrained spanning tree problem, called RPM. Two reproductive operators that generate new solutions from old ones - a small change operator and a crossover operator - were also specified. In this section we outline three stochastic iterative search techniques that can utilise these heuristics.

The first search technique we employ, *hillclimbing*, is strictly a local search method that starts from a random point in the search space and employs a neighbourhood operator to generate nearby points. Each time a potential solution is generated it is evaluated. If the new point is better than or equal to the current point then it replaces the current point, otherwise it is ignored. The process is iterated until some stopping criteria is met. Hillclimbing can be very effective at finding local optima but it cannot escape from them due to its greedy nature. However, a slight variant on the procedure called *multi-start hillclimbing* (MHC), which we employ here, enables a more global search. MHC simply restarts from a new random point if it has failed to replace its current solution in a given number,  $r$  of iterations. The best evaluation found so far is continually stored, and returned at the end. The advantage of MHC is its simplicity, requiring only one parameter to be set in addition to the stopping criteria.

The second technique we employ, *simulated annealing* (SA) [10], is based on an

analogy with the physical process of annealing, in which a solid is driven into a low-energy state by first melting it and then cooling it down slowly. Simulated annealing is characterised by its ability to accept transitions to more expensive solutions (in the case of minimising). This allows it to escape from local minima. As with hillclimbing, SA begins with a random point in the search space and generates new points using a neighbourhood operator. Transitions are accepted according to the following criteria:

$$p\{accept\ j\} = \begin{cases} 1 & \text{if } f(j) \leq f(i) \\ \exp\left(\frac{f(i)-f(j)}{c}\right) & \text{if } f(j) > f(i) \end{cases} \quad (3)$$

where  $p\{accept\ j\}$  is the probability of accepting a new solution  $j$  from the current solution  $i$ , and  $f(i)$  and  $f(j)$  are the respective costs of the two solutions. The control parameter,  $c$  (analogous to the temperature of a physical system) is set to a high initial value,  $c_0$ , at the beginning of a run of the simulated annealing algorithm, allowing most transitions to occur. It is then decremented so that fewer and fewer transitions are allowed. The algorithm terminates when  $c = c_f$ , the final value of the control parameter. The choices of initial and final values of the control parameter and the rate at which it is decremented are collectively known as the cooling schedule.

Our final choice of search technique is the *evolutionary algorithm* [11, 9]. This means using a population-based approach, where gradual improvement over time in a population of candidate solutions is obtained by means of iterated selection and variation of members of the population, coupled with removal of poor solutions to maintain a steady population size. A very wide variety of techniques are known for the selection, variation,

and general population update steps. Our choices for these, described below, are, like most contemporary implementations of evolutionary algorithms, relatively standard and simple methods, drawn from the great variety of research that has produced what are generally accepted to be improvements over those used in early, seminal work.

## 5.1 Parameter Settings

In order for a just comparison of the three search techniques to be made, we require that each executes for the same number of evaluations. Although there are slight differences in the overhead of the different techniques, these are small compared with the computational expense of evaluating a potential solution. Thus equality of the number of evaluations is a fairer test than is runtime, which can be affected by numerous external factors. Therefore, the number of evaluations allowed was set at 10,000 for all three methods.

The reset number,  $r$ , of the multi-start hillclimbing algorithm was set at 500. This number was arrived at empirically after a small number of trials at different values.

The cooling schedule adopted for the simulated annealing algorithm was of the geometric variety; at each iteration the value of the control parameter is updated using  $c = (1 - \delta)c$  where  $\delta$  is a small value. The values of  $c_0$  and  $\delta$  were calculated to produce the following conditions: At  $c_0$  approximately 90% of transitions accepted; at  $c_f < 0.1\%$  of transitions accepted; the number of evaluations is 10,000.

A steady state genetic algorithm was used [11] with one crossover and one mutation per generation. A local mating selection was used following Collins and Jefferson [16]: The chromosomes are mapped onto a two-dimensional grid so that each has a location.

Only chromosomes that are near to each other on the grid can mate. This helps prevent the population from converging too quickly. The population size used was 225 mapped onto a 15x15 grid. Selection of parents for mating is achieved by first choosing a starting grid square completely at random. Each parent is then the fittest individual found along a random walk of three steps, performed each time from the starting square.

## 6 Results

Results are given in terms of the ratio of the best  $d$ -MST weight found to the known weight of the (unconstrained) MST of the graph. The latter is found by simply running Prim's algorithm, which guarantees finding a minimum-weight spanning tree without degree-constraints.

We begin by comparing the MHC, SA, and GA versions of RPM with  $d$ -Prim's and BF2 on an  $M$ -graph of 250 nodes at various degree-constraints. The underlying MST of the graph has maximum node degree 10. The three iterative search algorithms were all run 20 times, each for 10,000 evaluations. The best result from the 20 runs are plotted in figure 3. The  $d$ -Prim's and BF2 algorithms were run only once because they are both deterministic, given the first node to be included. On this graph BF2 clearly exhibits the worst performance and even fails to find a solution (in 200 iterations) when the degree-constraint is three. The  $d$ -Prim's algorithm, which is equivalent to just a single iteration of BF2, is better at all constraints. The three search algorithms, however, find the lowest-cost solutions with the GA and SA exhibiting the best performance.

Table 4 provides a comparison of the performance of the GA and its nearest rival,

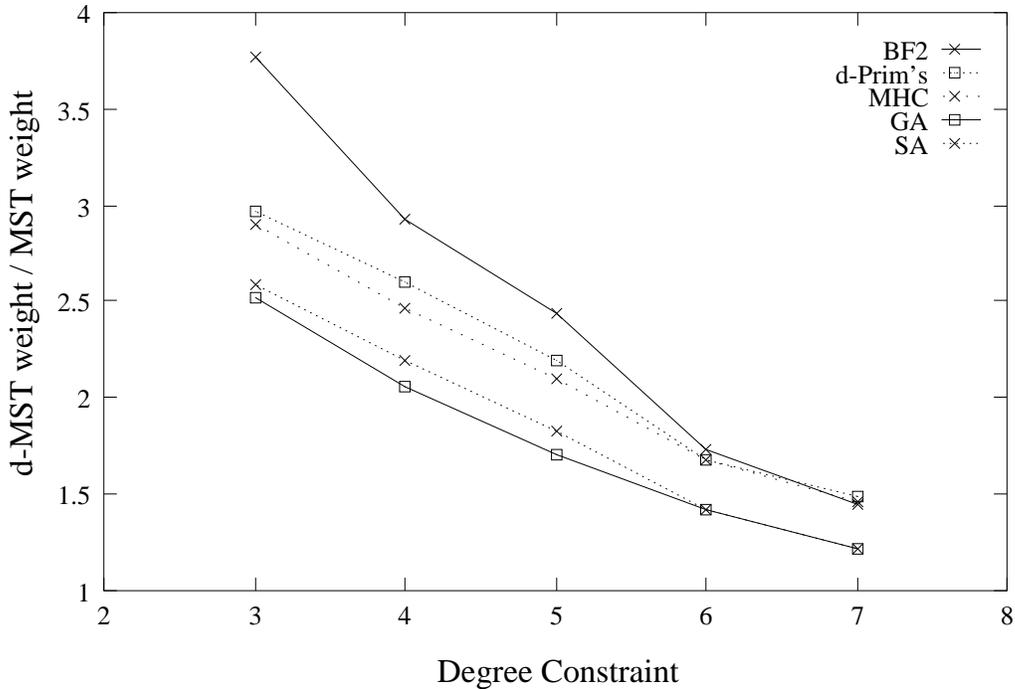


Figure 3: Comparing MHC, SA, GA (all using RPM), BF2, and  $d$ -Prim's on a 250-node  $M$ -graph.

SA. The mean of the 20 runs are shown for each constraint,  $d$ , as well as the  $t$ -test result from comparing the (approximately normal) distributions of results achieved by the two algorithms. In this case,  $t > 2.33$  yields statistically significant evidence in the superiority of GA over SA ( $P < 0.01$ ); so the GA is clearly superior to SA on this problem at all degree-constraint levels tested.

Next, the five algorithms are compared on nine further  $M$ -graphs but with the degree-constraint set at 5 for all of them. As before, each of the RPM based methods are run 20 times for 10,000 evaluations, and  $d$ -Prim's and BF2 just once. Three graphs at each of three sizes (50, 100 and 200 nodes) having up to a maximal node degree of 15 in the unconstrained MST were employed. Table 5 presents the best and mean solutions found

Degree-Constraint:	3	4	5	6	7
MST-weight	12.84	12.84	12.84	12.84	12.84
mean $d$ -MST weight (GA)	33.72	27.57	22.8	18.34	15.7
mean $d$ -MST weight (SA)	35.75	29.21	24.06	19.53	16.50
$t$ -value	8.44	6.9	7.1	8.25	6.5

Table 4: Comparing GA-RPM with SA-RPM on a 250-node  $M$ -graph with various degree-constraints.

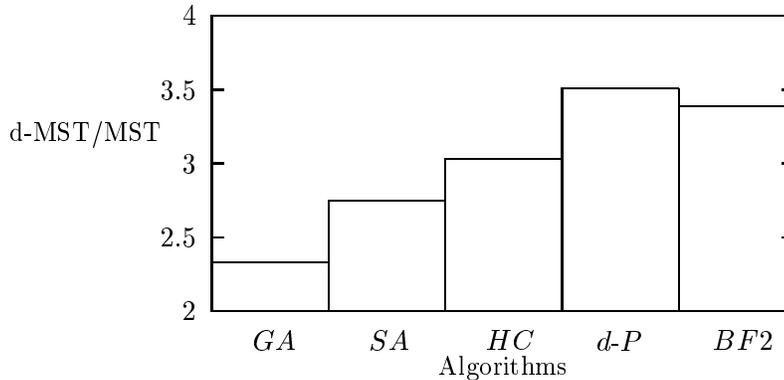


Figure 4: Mean over all  $M$ -graphs of best solutions found.

by each of the algorithms in terms of the  $d$ -MST/MST ratio.

It is clear from the above results that the GA employing RPM finds lower-cost solutions than any of the other algorithms. This result is presented graphically in Figure 4, which shows the mean (over all nine graphs) of the best solution found by each algorithm.

The above analysis is repeated on a set of nine random graphs produced using the method of Boldon *et al.* [8]. The difference in the costs of solutions found by the five algorithms is less marked using these less challenging graphs but the GA employing RPM still finds the lightest trees. Table 6 shows the mean and best solutions found and Figure 5 presents the mean of the best results for each algorithm, as before.

Graph	Vertices	MST degree		GA	SA	HC	<i>d</i> -P	BF2
1	50	11	Best	2.78	3.40	3.68	4.35	3.30
			Mean	3.15	3.71	3.87	—	—
2	50	10	Best	2.51	2.97	3.50	4.22	3.84
			Mean	2.99	3.46	3.75	—	—
3	50	11	Best	2.37	2.96	2.71	3.24	3.94
			Mean	2.58	3.29	2.91	—	—
4	100	12	Best	2.26	2.68	2.99	3.44	2.82
			Mean	2.44	2.74	3.10	—	—
5	100	12	Best	2.50	3.07	3.42	4.13	4.71
			Mean	2.84	3.29	3.10	—	—
6	100	13	Best	2.73	3.14	3.51	3.90	4.55
			Mean	2.86	3.26	3.60	—	—
7	200	11	Best	1.92	2.14	2.55	2.75	2.24
			Mean	2.04	2.26	2.63	—	—
8	200	11	Best	2.25	2.61	2.97	3.34	2.99
			Mean	2.47	2.74	3.13	—	—
9	200	9	Best	1.68	1.77	2.07	2.26	2.12
			Mean	1.71	1.84	2.14	—	—

Table 5: Comparing RPM Search with *d*-Prims and BF2 on 9 different *M*-graphs.

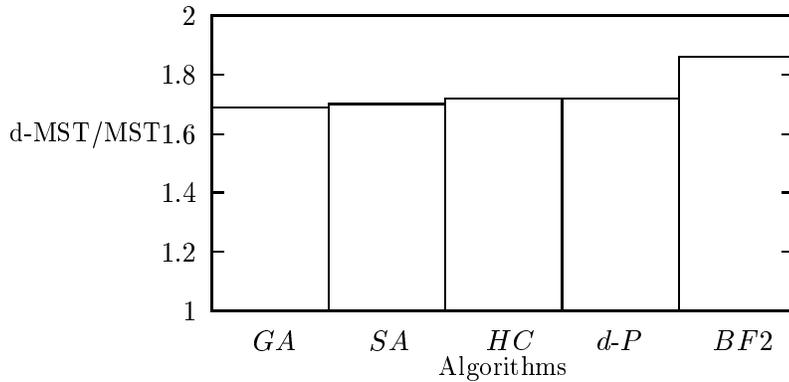


Figure 5: Mean over all Random graphs of best solutions found.

Graph	Vertices	MST degree		GA	SA	HC	d-P	BF2
1	50	15	Best	1.74	1.76	1.77	1.78	1.94
			Mean	1.74	1.77	1.78	—	—
2	50	15	Best	1.83	1.86	1.84	1.84	2.00
			Mean	1.83	1.86	1.87	—	—
3	50	14	Best	1.77	1.79	1.79	1.81	1.89
			Mean	1.78	1.79	1.82	—	—
4	100	15	Best	1.73	1.76	1.78	1.76	1.94
			Mean	1.73	1.76	1.81	—	—
5	100	15	Best	1.60	1.61	1.62	1.64	1.68
			Mean	1.60	1.61	1.64	—	—
6	100	14	Best	1.68	1.67	1.71	1.71	1.88
			Mean	1.69	1.68	1.74	—	—
7	200	15	Best	1.63	1.63	1.66	1.67	1.85
			Mean	1.64	1.64	1.68	—	—
8	200	15	Best	1.59	1.60	1.62	1.60	1.79
			Mean	1.59	1.60	1.64	—	—
9	200	15	Best	1.65	1.65	1.68	1.68	1.79
			Mean	1.65	1.66	1.79	—	—

Table 6: Comparing RPM Search with  $d$ -Prims and BF2 on 9 different Random graphs.

$j$	1	2	3	4	5	6	7	8	9
1	—	224	224	361	671	300	539	800	943
2		—	200	200	447	283	400	728	762
3			—	400	566	447	600	992	949
4				—	400	200	200	539	583
5					—	600	447	781	510
6						—	283	500	707
7							—	361	424
8								—	500
9									—

Table 7: Edge weights of the 9-vertex  $d$ -MST problem.

Finally, we demonstrate that our GA employing RPM is superior to the GA employing Prüfer numbering, proposed by Zhou and Gen [9]. To this end, we present results on a 9-node Euclidean Graph which was first given as an example by Volgenant and Savelsbergh, who solved it heuristically, and was used by Zhou and Gen to show that their GA could also solve it. The edge weights of the graph are given in Table 7.

The optimal solution to the graph is 2256 when the degree-constraint  $d = 3$ . Zhou and Gen [9] report that their GA finds the optimal solution 66.7% of the time in 25,000 evaluations (a population of 50 for 500 generations). A GA employing the RPM was found to discover the optimal solution on this problem with greater than 99% probability using just 500 evaluations.

## 7 Discussion and Summary

In this paper we have presented a new heuristic, RPM, for searching the  $d$ -ST space of random-weight, non-Euclidean graphs. It is based on Prim’s classic algorithm for building

minimum spanning trees but includes extra features to ensure that the degree-constraint is not violated and to allow more costly edges to be occasionally chosen. This heuristic was tested within three stochastic iterative search algorithms on a large number of different graphs. Results show that a genetic algorithm can take advantage of RPM most effectively in terms of solution cost returned.

A new method of generating particularly challenging  $d$ -MST problems was also presented. The method is based heavily on that proposed by Boldon *et al.* [8] for creating graphs with a high underlying MST degree. However, the new method additionally produces graphs ( $M$ -graphs) that penalise algorithms which take a purely greedy approach to constructing low-weight  $d$ -ST's. The new  $M$ -graphs and the graphs proposed in [8] were both used to test the algorithms considered.

Comparison of the genetic algorithm employing the RPM heuristic to BF2 (the dual simplex approach of Boldon *et al.* [8]) and to  $d$ -Prim's (the upper bound technique of Narula and Ho [3]) reveals that the GA consistently finds better solutions, and more so on the more challenging graphs.

A genetic algorithm employing RPM was also compared with a Prüfer number encoded GA proposed by Zhou and Gen [9]. The RPM encoding method, which utilises edge cost information to reduce the search space, helps the GA find the optimal solution in far fewer evaluations and far more reliably than the GA using a Prüfer number encoding.

## Acknowledgements

The work presented in this paper is part of a larger project aimed at developing methods for optimisation in communication networks. The project is particularly concerned with the optimisation of growing networks, both in terms of network design and the management of traffic, and is sponsored by BT Labs Plc. The first author would like to express his thanks to BT for sponsorship of his Ph.D.

## References

- [1] B.M.E. Moret and H.D. Shapiro, “An empirical analysis of algorithms for constructing a minimum spanning tree,” *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 15, pp. 99–117, 1994.
- [2] R. Ravi, M.V. Marathe, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt, “Many birds with one stone: Multi-objective approximation algorithms,” in *Proceedings of 25th Annual ACM STOCs*, 1993, pp. 438–447.
- [3] S.C. Narula and C.A. Ho, “Degree-Constrained Minimum Spanning Tree,” *Computers and Operations Research*, vol. 7, no. 4, pp. 39–49, 1980.
- [4] R.J. Wilson and J.J. Watkins, *Graphs: an introductory approach: a first course in discrete mathematics*, John Wiley and Sons. Inc., 1990.
- [5] M. Savelsbergh and T. Volgenant, “Edge Exchanges in the Degree-Constrained Minimum Spanning Tree Problem,” *Computers and Operations Research*, vol. 12, no. 4,

- pp. 341–348, 1985.
- [6] S. Khuller, B. Raghavachari, and N. Young, “Low-Degree Spanning Trees of Small Weight,” *SIAM Journal of Computing*, vol. 25, no. 2, pp. 355–368, 1996.
- [7] S.P. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari, and N. Young, “A Network-Flow Technique for Finding Low-Weight Bounded-Degree Spanning Trees,” *Journal of Algorithms*, vol. 24, pp. 310–324, 1997.
- [8] B. Boldon, N. Deo, and N. Kumar, “Minimum-Weight Degree-Constrained Spanning Tree Problem: Heuristics and Implementation on an SIMD Parallel Machine,” Tech. Rep. CS-TR-95-02, Department of Computer Science, University of Central Florida, Orlando, FL 32816, 1995.
- [9] G. Zhou and M. Gen, “A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems,” *Networks*, vol. 30, pp. 91–95, 1997.
- [10] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, Netherlands, 1987.
- [11] D. E. Goldberg, *Genetic Algorithms in Search, Optimisation, and Machine Learning*, Addison-Wesley, 1989.
- [12] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
- [13] C. Monma and S. Suri, “Transitions in geometric minimum spanning trees,” *Discrete Computational Geometry*, vol. 8, pp. 265–293, 1992.

- [14] D. Cheriton and R. E. Tarjan, "Finding Minimum Spanning Trees," *SIAM Journal of Computing*, vol. 5, no. 4, pp. 724–741, December 1976.
- [15] G. Syswerda, "Uniform Crossover in Genetic Algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 2–9.
- [16] R.J. Collins and D.R. Jefferson, "Selection in Massively Parallel Genetic Algorithms," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker, Eds., San Mateo, 1991, pp. 249–256, Morgan Kaufmann.