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

Joshua Knowles David Corne Martin Oates
Department of Computer Science, University of Reading, UK
J.D.Knowles@reading.ac.uk, D.W.Corne@reading.ac.uk, moates@srd.bt.co.uk
FAX: +44 (0) 118 975 1994, TEL: +44 (0) 118 931 8983

Abstract

Finding the degree-constrained minimum spanning tree (d-MST) of a graph is a well studied NP-hard problem which is important in network design. We introduce a new method which improves on the best technique previously published for solving the d-MST, either using heuristic or evolutionary approaches. The basis of this encoding is a spanning-tree construction algorithm which we call the Randomised Primal Method (RPM), based on the well-known Prim's algorithm [6], and an extension [4] which we call 'd-Prim's'. We describe a novel encoding for spanning trees, which involves using the RPM to interpret lists of potential edges to include in the growing tree. We also describe a random graph generator which produces particularly challenging d-MST problems. On these and other problems, we find that an evolutionary algorithm (EA) using the RPM encoding outperforms the previous best published technique from the operations research literature, and also outperforms simulated annealing and multi-start hillclimbing (both also using the RPM encoding). We also note that an EA using the RPM also outperforms a recently published alternative EA approach on a standard test problem.

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. However, in real networks the vertices (or nodes) are usually subject to a degree constraint. For example, exchanges or switches can typically 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 general d-MST is NP-hard; notice that when $d = 2$, it is equivalent to the traveling salesperson problem without the last edge.

Recent research by Boldon *et al.* [2] 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 seems to perform very well on a range of benchmark d-MST problems. We describe an EA approach which outperforms this. It is based on Prim's well-known greedy algorithm for spanning tree construction, in its extended form [4] which deals with degree-constrained versions of the problem. We call this extended form 'd-Prims'. Interestingly, we also find that 'd-Prim's' alone is superior to Boldon *et al.*'s dual simplex approach, but not to the EA. Our goal was to achieve fruitful hybridisation of a spanning tree construction method with stochastic iterative search. This method, which we call the RPM (Randomised Primal Method) runs in reasonable time and significantly outperforms both d-Prim's and Boldon *et al.*'s method in terms of solution quality. It also outperforms a recent alternative EA approach [13]. All methods are compared on a variety of randomly generated networks of from 50 nodes to 3000 nodes. The random graphs used for testing have been created to be particularly difficult and misleading in order to emphasise differences in algorithm performance.

Much of the remainder of the paper provides background on Prim's, d-Prim's, Boldon *et al.*'s 'dual simplex' method, and our procedures for generating random graphs; this detail is essential to support describing our encoding for evolutionary approaches to this problem, understanding the comparative techniques, and also to support replication and furthering of the work by others. Details on our evolutionary approach are concentrated at teh end of section 5, and in Section 6, which describes the results.

The low-level organisation of this paper is as follows: Section 2 provides some background material and presents both the Prim's and d-Prim's algorithms. Section 3 details two procedures for producing random graphs, one used by Boldon *et al.* [2], and another we have developed which leads to particularly challenging d-MST problems. In Section 4, we describe the existing published method for addressing the degree constrained minimal spanning tree problem which seems to have the best results so far; this is Boldon *et al.*'s dual simplex method [2]. In this section we also give notes on searching the space of d-MSTs, and describe our experience in comparing the dual simplex method with d-Prim's. In Section 5 we describe the RPM method, and our evolutionary algorithm (EA) approach which uses it. Section 6 records the results achieved when comparing Boldon *et al.*'s method, d-Prim's, and the RPM (multistart hillclimbing, simulated annealing [8], and EA [9, 10, 11] versions) on thirteen random graphs with a range of degree constraints. This also compares our evolutionary approach with a recently published alternative evolutionary approach [13] on a test problem. Finally, some discussion and a summary are provided in Section 7.

2 Building Spanning Trees

Given space restrictions, we assume the reader has a basic knowledge of graph theory, but state here a few appropriate 'reminders', and then go on to present algorithms for constructing spanning trees.

A graph or network is *connected* if a path exists along its edges between every pair of nodes. A *spanning tree* of a connected graph G is a connected subgraph of G which includes all of G's vertices, some or all of its edges, and contains no cycles (or 'loops'). Every connected graph contains one or more spanning trees. In a *weighted graph*, there is a weight (or cost) $c(v, w)$, which is a non-negative number, associated with each edge (v, w). If the graph models a network laid out in a Euclidean space, such that the costs represent distances between vertices in this space, then we call the graph *Euclidean*.

A connected graph generally contains very many spanning trees. The cost of a spanning tree is the sum of the costs of the edges it contains. A spanning tree with minimal weight is called a *minimum spanning tree* (MST). If we force a degree constraint d on the vertices of our spanning trees, then a *degree constrained minimum spanning tree* (d-MST) of a weighted graph G is the minimum spanning tree in G in which the degree of no vertex exceeds d.

The problem of finding an MST in a weighted graph has received much attention [1, 2, 3]. Again, the problem is of interest in communications networks where one can imagine that laying wires between pairs of exchanges incurs some cost. The problem is then to find a set of connections which minimises the overall cost. All proposed algorithms for finding an MST are greedy (or 'constructive') algorithms. Moret and Shapiro [1] assess the speed and overhead associated with running these algorithms on a wide range of graphs and conclude that, overall, Prim's algorithm [6] is the best both in terms of speed and memory usage. Since Prim's algorithm forms the basis of much of what follows, we detail it here:

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 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 use of good 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 [3].

Using Prim's algorithm, and then adding further edges in an appropriately weight-biased way, facilitates any search process which needs to find good connected graphs. A simple extension to this which we call 'd-Prim's' was first described by [4]. This differs from Prim's only in step 5, enabling Prim's to be used for constructing degree-constrained MSTs. Step 5 in *d-Prim's* is: 'Choose a minimum weight edge connecting a vertex in C with a vertex in U which would not violate the degree constraint on the vertex chosen from C, and place it in $E(M)$.'

3 Generating Hard d-MST Problems

Example graph problems which have been used to test d-MST algorithms broadly fall into two categories; Euclidean, and random. Of the two, it is generally easier to find good solutions to the d-MST in Euclidean graphs. This fact can be appreciated if one considers the unconstrained MST of a graph. In a 2-d, complete, Euclidean graph, the unconstrained MST never contains vertices of degree greater than five. In fact, for Euclidean graphs it has been shown that the d-MST is NP-hard when $d <= 4$ but polynomial when $d >= 5$ [5]. In a random graph, by contrast, the maximum degree of a vertex contained in the unconstrained MST can be arbitrarily large (up to N-1): Ravi *et al.* [5] state that given any rational number $R >= 1$, 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. Therefore, we consider only random graphs in the remainder of this paper, since they provide a deeper test of the relative quality of proposed d-MST algorithms.

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.* [2] 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 which would 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] and all the other edges are given values in the range (0.1,1.0].

For reasons explained in the next section, a further procedure for generating particularly difficult and misleading random graphs (again with large degree in the MST), was developed. 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 which are towards the top end of the range of edges not in the MST. Figure 1 shows part of such a graph and the range of values that each type of edge takes.

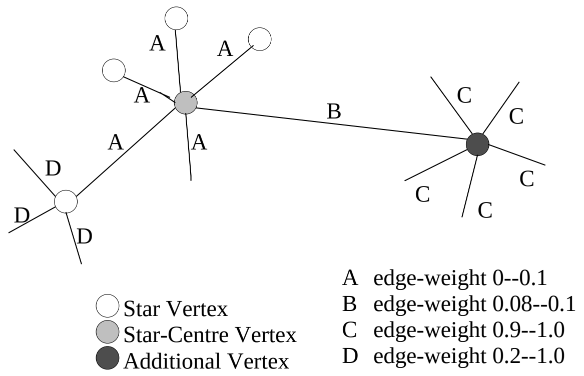

Figure 1: Illustrating Edge-Weight Dsitributions of the "hard graph" generator

Now, let us consider why a graph generated using the procedure above, from now on called a "hard graph", will strongly mislead algorithms based upon choosing the lowest weight available edge at each iteration, such as Prim's algorithm: If the choice is being made for an edge incident to a star centre vertex then the algorithm will usually choose an edge of type A, because 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 cannot be connected to the graph by an edge of type B (having a weight <= 0.1) and so it will later be connected by an edge of type C which has a much higher weight (>= 0.9). The graph has misled the algorithm. (This situation shows why adding a degree constraint to the MST problem can render the use of a greedy approach ineffective.)

4 Existing Methods for the d-MST

4.1 The Dual-Simplex Approach

Boldon *et al.* [2] correctly argue that using a good algorithm for computing the MST of a graph gives a good basis for solutions to the d-MST. With this in mind, they designed a dual simplex approach to the d-MST which works as follows: The method 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 which 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 which 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 [2]; the best is found to be their second function, 'BF2'. It updates edge weights in the following way:

$$\text{newweight} = \\ \text{weight} + \text{fault} * \max * (\text{weight} - \min)/(\max - \min)$$

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 violate the degree constraint. `max` is the maximum weight in the current spanning tree and `min` is the minimum. In stage 2 of Boldon *et al.*'s method, this function is applied to the weights of all edges involved in a degree violation, with the exception of any which happen to be the minimal weight edge on a violated node.

Using this approach, Boldon *et al.* report d-MST's which are within 0.5% of the MST on a number of large TSP problems. However, solutions are not always found in 200 iterations of the program (regarded as failure). In the following, we will refer to Boldon *et al.*'s dual simplex method, using BF2 as the blacklisting function, as simply 'BF2'.

4.2 Notes on Searching MST-Space

In designing a stochastic search-based approach to the d-MST problem it is worth considering first the size and nature of the search space. In a random complete graph having N vertices, the number of edges, L, is $N(N-1)/2$. The total number of possible subgraphs is 2^L, since each edge may be either present or absent. The number of spanning trees is $N^{(N-2)}$. For a graph of 20 vertices this gives a naïve search space of $1.56*10^{57}$ of which a 'mere' $2.62*10^{27}$ are spanning trees. It seems clear, then, that any method which does not produce potential solutions which are spanning trees is going to waste a great deal of time 'repairing' these, especially as checking a graph to ensure it is connected is itself a difficult problem. For this reason, in designing stochastic iterative search approaches for the d-MST, we decided to investigate encodings and operators which centrally incorporate spanning-tree construction.

An early attempt used a very basic spanning-tree construction algorithm which added valid edges randomly to the growing tree (without considering edge-weights). A candidate solution (hereinafter: chromosome), was simply a list of edges in the tree thus generated. Recombination and mutation operators were devised which, again, used a spanning-tree construction algorithm to build offspring, but biassing the choices of edge in the growing tree towards those present in the parent or parents.

This approach proved unsuccessful, even on small d-MST problems where the minimum was known. It soon became clear to us that useful results could only emerge by making use of the edge weights. Our first attempt at this was to use 'd-Prim's' to seed the initial population of an EA which would otherwise continue as described above. Testing was initially carried out on Euclidean graphs; TSP problems for which Boldon *et al.* [2] had published their best solutions. The results were unexpected and important: d-Prim's alone generated solutions in the initial population which were equal to BF2's results on most of these problems.

4.3 d-Prim's

In our early experiments, as described, we found that a single run of d-Prim's performed equally to the more sophisticated method BF2 on several benchmark TSP problems. It is also worth noting that while d-Prim's is always sure to generate a valid tree (one which satisfies the degree constraint), BF2 will occasionally fail. Also, d-Prim's is equivalent to just one iteration of BF2 and hence is always at least as fast as BF2, and usually much faster.

However, as we noted earlier, Euclidean graphs such as TSP benchmarks generally provide less challenging d-MST problems than random graphs. Therefore, to provide further comparison, and to challenge d-Prim's, we then looked at random graph problems.

The random graph generation procedure of Boldon *et al.*, described earlier in section 3, was used to create two random graphs, one of 500 vertices and one of 3000 vertices, and both having maximum MST degree 20. BF2 and d-Prim's were run on these problems for $d = 19$ down to $d = 3$ in steps of two. The results are presented in the next section (see figures 2 and 3). 10 smaller graphs of 50 nodes each were also generated and results relating to these are given in table 2, which clearly show d-Prim's' superiority to BF2 on problems of this type.

5 The RPM and Chromosome Encoding

Despite the impressive results obtained on these random graph problems achieved simply by using d-Prim's, the approach remains unconvincing. Some graphs should certainly mislead the algorithm, leading to a poor d-MST solution. Therefore, the graph generation procedure described in section 3 for producing 'hard graphs' was developed to provide a greater challenge to d-MST algorithms. We also looked into how we could fruitfully hybridise a Prim's-like algorithm with stochastic iterative search, and came up with the RPM (Randomised Primal Method), which we describe next.

During step 5 of Prim's algorithm, the lowest weight edge which connects a vertex already in the growing tree to one not in it must be selected. In our implementation of this, a sorted list of edges was associated with each vertex, containing each of its incident edges not yet included in the growing tree (but which could validly be included) in ascending order of weight.

When an edge is being selected for inclusion in the tree, the first edge in each such list is examined. The lowest-weighted of these is selected and the lists are updated. The heuristic we devised next, the Randomised Primal Method (the RPM), allows Prim's, instead of always reading the first edge in the ordered edge list for each vertex, to occasionally read the ith edge. So, most of the time it will select the lowest weight edge, but sometimes it will be forced to take a higher weighted edge. Potentially, this will allow the algorithm to avoid misleading situations like those found in the 'hard graphs'.

Before describing the RPM, it is essential to explain the dynamic datastructure which underpins its use. This data structure is 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 ascending order of cost. While the RPM runs, the spanning tree is gradually being constructed, and T is dynamically altered whenever a new edge is added to the growing tree. The idea is that the sorted edge list associated with each vertex i only contains edges *not yet included* in the tree, but which *could validly be included*. 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, and if the inclusion of this edge would not violate a degree constraint.

We use T in our implementation of 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 it. To execute this step, we look at all of the entries in T for vertices already in the tree, and return the top (lowest cost) edge from each of their associated edge lists. This gives us a collection L of low-cost edges. The lowest-weighted of these (breaking ties randomly) is then selected.

The RPM, on the other hand, works very much like Prim's, with the exception that we do not always choose the first edge in each edge-list. Instead, we construct the low-cost set L by choosing the i_jth edge from each appropriate entry in T. That is, for vertex i, we choose the jth lowest cost edge. The values of j for each vertex i are given as input to the RPM. In our implementation, the values of j are biased towards usually being '1', but occasionally they will be higher. So, most of the time it will select the lowest weight edge, but sometimes it will be forced to take a higher weighted edge. This, when allied to a good search algorithm, will allow an effective search to be carried out.

To illustrate the RPM, consider a case of a 9 vertex graph; the RPM 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, the edge list of each of these vertices will be looked at in turn. Table 1 shows a set of edge choice lists for this example. There is an edge choice list for each vertex i. All of them contain just two valid edges: those connecting to vertices 2 and 5. Usually, the weights of the first edge in each of the seven lists would be compared, and the lowest of these selected. But, in the RPM a chromosome is used as an edge-choice look-up table to

vertex i	1	2	3	4	5	6	7	8	9
edge choice 1	**2**	–	2	**2**	–	2	**5**	**5**	5
edge choice 2	5	–	**5**	5	–	**5**	2	2	**2**

Table 1: Edge choice lists for example in text

guide this choice.

Consider the chromosome (1,1,4,1,2,3,1,1,2); the value of the ith gene represents the choice of edge which the RPM will use when it is considering the sorted edge-list for vertex i. So, in this example, the RPM will choose the lowest-weighted edge from the following set: the 1st edge in vertex 1's list, the 4th edge in vertex 3's list, the 1st edge in vertex 4's list, the 3rd edge in vertex 6's list, and so on. These edges are highlighted in Table 1. Where the chromosome instructs the RPM to choose an edge off the bottom of a list (e.g. the 4th edge in vertex 3's list), it simply selects the lowest one it can. Note that it ignores the 2nd and 5th genes, since, in this example, vertices 2 and 5 are not yet connected.

The representation we actually use is slightly more involved than this; the chromosome is a collection of d such edge-choice tables, where d is the degree constraint. When the edge list of a vertex is considered, the current degree of the vertex is first checked. If this is k, then the kth edge-choice table is used. The rationale for this is as follows: for a particular vertex we may wish to choose, say, the 5th or 6th best edge the first time it is connected to another vertex, the third best the second time it is connected and the best when we are connecting it for the third time.

Chromosomes are initialised randomly using a negative exponential probability distribution. Using this, most edge-choice table genes contain 1's. Mutation is applied to about 1% of genes and works by setting the gene to a new value, from the same distribution as in the initialisation phase. The recombination operator used is uniform crossover. These two operators were applied in an EA using a steady-state version of Collins and Jeffersons' Local Mating method [7], with a population of 225 (a 15 × 15 grid) and a random-walk length of 3. For comparison, the initialisation and mutation operations were also used in a hill climber and a simulated annealing algorithm. The hill climber was of the multiple-restart variety i.e. if it has not found a better chromosome in a given number of evaluations (in this case, 500) it re-initialises and begins hill climbing again, keeping a record of the best evaluation found so far. The simulated annealing algorithm employed a geometric cooling schedule with initial and final temperatures T_I and T_F engineered to allow 90% acceptance at T_I and 0.1% at T_F. In all cases, trials were run for 10,000 chromosome evaluations.

These three methods were compared, along with d-Prim's and BF2, on a 'hard graph' of 250 nodes, for a variety of degree constraints. The results can be seen in the next section.

6 Results

Results are generally 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 just by running Prim's, which guarantees finding a minimal weight spanning tree without degree constraints. We begin by comparing d-Prim's with BF2 on a random graph of 500 vertices. Results are plotted on the left in Figures 2 and 3. The graph's (unconstrained) MST has maximal node-degree of 20. Both BF2 and d-Prim's are deterministic given the choice of the first node, so the results are for a single run of each. The plots show the ratio of the d-MST weight to the MST weight for a range of degree constraints. BF2 was allowed a maximum of 200 iterations in which to find a valid solution. Bear in mind that using d-Prim's is equivalent to just one iteration of BF2. Note also that BF2 found no solution in 200 iterations when the degree constraint was 3.

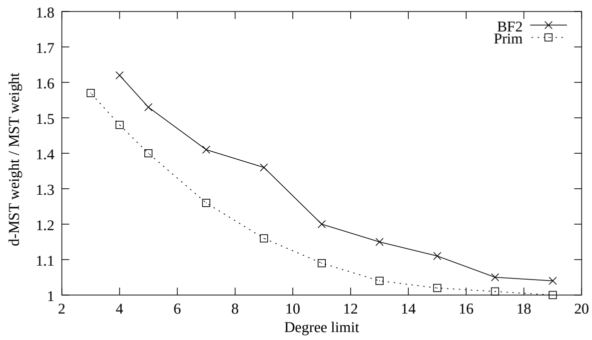

Figure 2: Comparing BF2 and d-Prim's on a 500-node graph

The second set of results are for an identical set of conditions but with a larger graph of 3000 nodes; these are plotted on the right of Figures 2 and 3. There is little difference between this plot and the last, confirming that both methods scale equally well in terms of the quality of solutions that they find. In both cases, d-Prim's is far superior.

Table 2 presents the results of comparing BF2 with d-Prim's on 10 different 50 node random graphs to find the 5-MST. Each graph's unconstrained MST had

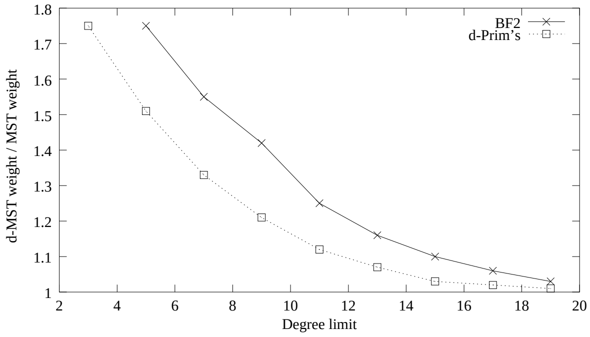

Figure 3: Comparing BF2 and d-Prim's on a 3000-node graph

Graph No.:	1	2	3	4	5
MST-weight	2.34	2.33	2.2	2.33	2.39
5-MST (BF2)	4.13	4.09	3.98	4.43	3.64
5-MST (d-Prim's)	3.9	3.81	3.63	4.01	3.37
Graph No.:	6	7	8	9	10
MST-weight	2.52	2.14	2.68	2.56	2.37
5-MST (BF2)	4.17	3.81	4.4	4.21	4.54
5-MST (d-Prim's)	3.81	3.61	4.22	4.21	3.78

Table 2: Comparison of BF2 and d-Prim's on 50-node random graphs for 5-MST

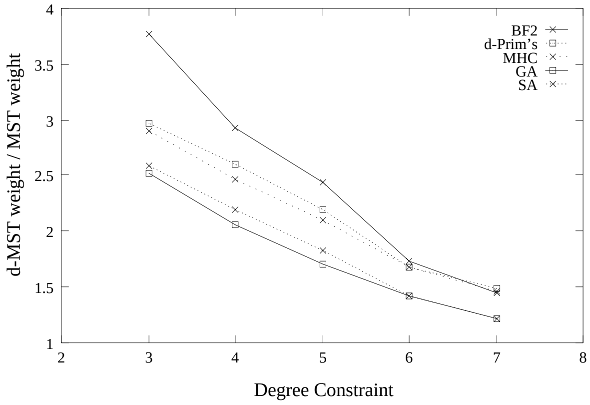

Figure 4: Comparing MHC, SA, EA (all using the RPM), BF2, and d-Prim's on a 250-node hard random graph

DC:	3	4	5	6	7
MST	12.84	12.84	12.84	12.84	12.84
EA	33.72	27.57	22.8	18.34	15.7
SA	35.75	29.21	24.06	19.53	16.50
t-value	8.44	6.9	7.1	8.25	6.5

Table 3: Comparing EA-RPM with SA-RPM on a 250-node hard random graph with various degree constraints

maximal node degree greater than 10. A t-test showed that d-Prim's was significantly better than BF2 (with $t = 2.68$, where $t > 2.55$ needed for greater than 99% confidence), and furthermore requires only a fraction of the time to achieve this level of performance.

Figure 4 compares both BF2 and d-Prim's with the three algorithms employing the RPM. Again BF2 and Prim's were run just once. The EA, SA and HC were all run 20 times, each for 10,000 evaluations. The best result of the 20 runs is plotted. The graph problem was a 'hard graph' of 250 vertices and with an unconstrained MST of maximal node degree 10. The greater difficulty of this graph problem can be appreciated by observing the performance of BF2, which only manages to find a solution of d-MST/MST ratio 3:1 when $d = 4$.

The EA is clearly superior in terms of the best solutions found. Table 3 compares the EA with its nearest rival, simulated annealing. The mean of the 20 runs are shown for each constraint, d, as well as the t-test result from comparing the EA to SA. In this case, $t > 2.33$ yields 99% confidence in the superiority of EA over SA; so, the EA is clearly superior to SA on this problem at all degree-constraint levels tested.

The header row refers to degree constraints (DC), and the results rows give, for each degree constraint, the weight of the MST (ignoring the degree constraint), the mean MST-weight found by EA-RPM (respecting the degree constraint) and the mean MST-weight found by SA-RPM (respecting the degree constraint). Each EA-RPM or SA-RPM mean value is teh mean of 20 trial runs. The final row gives the t-test result from comparing the EA to SA. In this case, $t > 2.33$ yields 99% confidence in the superiority of EA over SA; so, the EA is clearly superior to SA on this problem at all degree-constraint levels tested.

Finally, we compare EA-RPM to a recent EA developed for this problem, employing an inspired representation based on Prüfer numbering, proposed by Zhou and Gen [13]. To this end, we present results on a nine node Euclidean Graph which was first given as an example by Savelsbergh and Volgenant [12], who solved it heuristically, and was used by Zhou and Gen to show that their EA could also solve it. The edge weights of the graph are given in Table 4.

The optimal solution to the graph is 2256 when the degree constraint $d = 3$. Zhou and Gen report that their GA finds the optimal solution 66.7% of the time

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 4: Edge weights of the 9-vertex dc-MST problem

in 25,000 evaluations (a population of 50 for 500 generations). We find that our GA employing the RPM finds the optimal solution on this problem with greater than 99% probability using just 500 evaluations.

7 Discussion and Summary

We have found that the RPM method is a very effective way of searching the d-MST space. When applied in an EA, it is clearly able to find solutions that more closely approach optimality, compared with d-Prim's or BF2. Results so far comparing the EA to SA and multiple re-start hill-climbing, indicate that the EA seems the best suited to searching the d-MST when using the RPM. In addition, several runs of 40,000 iterations for both the EA and SA produced no improvement in the quality of d-MST solutions found. We thus conclude that our results, believed to be the best yet published, suggest evolutionary techniques to be superior on this problem. We have also compared our EA-RPM method with an alternative EA developed by [13], and also find it significantly better to that. Although so far we have only been able to compare EA-RPM with the latter method on a single test problem, it is worth noting that this is a particularly simple problem compared to the others we have addressed in this paper.

Acknowledgements

We thank BT Plc for their continuing sponsorship of the first author.

References

[1] Moret, B.M.E., Shapiro, H.D.: An empirical analysis of algorithms for constructing a minimum spanning tree. DIMACS Series in Discrete Math. and Theoretical Computer Science **15** (1994), 99–117

[2] Boldon, B., Deo N., Kumar, N.: Minimum-Weight Degree-Constrained Spanning Tree Problem: Heuristics and Implementation on an SIMD Parallel Machine. Technical Report CS-TR-95-02, Department of Computer Science, University of Central Florida (1995)

[3] Cheriton, D., Tarjan, R.E.: Finding Minimum Spanning Trees. SIAM Journal of Computing **5**(4), (1976) 724–741

[4] Narula, S.C., Ho, C.A.: Degree-Constrained Minimum Spanning Tree. Computers and Operations Research **7**(4), (1980), 39–49

[5] Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt, H.B.: Many birds with one stone: Multi-objective approximation algorithms. Proc. of 25th Annual ACM STOCS (1993), 438–447

[6] Wilson, R.J., Watkins, J.J.: Graphs: an introductory approach: a first course in discrete mathematics. John Wiley and Sons. Inc. (1990)

[7] Collins, R.J., Jefferson, D.R.: Selection in Massively Parallel Genetic Algorithms. in Belew, R.K., Booker, L.B. (eds.), Proceedings of the Fourth International Conference on Genetic Algorithms (1991), San Mateo: Morgan Kaufmann, 249–256

[8] van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated Annealing: Theory and Applications. D. Reidel Publishing Company, Dordecht, Netherlands (1987)

[9] Goldberg, D.E.: Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley, (1989)

[10] Fogel, D.B.: Evolutionary Computation: Toward a New Philosophy of Machine Intelligence IEEE, New York (1995)

[11] Bäck, T.: Evolutionary Algorithms in Theory and Practice Oxford University Press, New York (1996)

[12] Savelsbergh, M., Volgenant, T.: Edge Exchanges in the Degree-Constrained Minimum Spanning Tree Problem. Computers and Operations Research **12**(4) (1985), 341–348

[13] Zhou, G., Gen, M.: A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems. Networks **30** (1997), 91–95