A Comparison of Encodings and Algorithms for Multiobjective Minimum Spanning Tree Problems

Joshua D. Knowles and David W. Corne
School of Computer Science, Cybernetics and Electronic Engineering
University of Reading, Reading RG6 6AY, UK
j.d.knowles@reading.ac.uk, d.w.corne@reading.ac.uk
http://www.rdg.ac.uk/~ssr97jdk

Abstract-
Finding minimum-weight spanning trees (MST) in graphs is a classic problem in operations research with important applications in network design. The basic MST problem can be solved efficiently, but the degree constrained and multiobjective versions are NP-hard. Current approaches to the degree-constrained single objective MST include Raidl's evolutionary algorithm (EA) which employs a direct tree encoding and associated operators, and Knowles and Corne's encoding based on a modified version of Prim's algorithm. Approaches to the multiobjective MST include various approximate constructive techniques from operations research, along with Zhou and Gen's recent evolutionary algorithm using a Prüfer based encoding. In this paper we apply (appropriately modified) the best of recent methods for the (degree-constrained) single objective MST problem to the multiobjective MST problem, and compare with a method based on Zhou and Gen's approach. Our evolutionary computation approaches, using the different encodings, involve a new population-based variant of Knowles and Corne's PAES algorithm. We find the direct encoding to considerably outperform the Prüfer encoding. And we find that a simple iterated approach, based on Prim's algorithm modified for the multiobjective MST, also significantly outperforms the Prüfer encoding.

1 Introduction

The minimum spanning tree (MST) of a graph is an important concept in network design. Given a graph, with a cost (or weight) associated with each edge, the problem is to find a spanning tree of the graph with minimal total cost. It can be solved in polynomial time, and one of the classic and favoured algorithms [22] is Prim's algorithm [15], which grows a tree gradually by incrementally adding minimal-cost feasible edges. In real-world problems, the vertices (or nodes) are usually subject to a degree constraint. For example, exchanges or switches will typically have a restricted number of communication ports. Unlike the MST, the general degree-constrained MST (d-MST) is NP-hard.

Recent research has seen progress in the development of approximate algorithms for the d-MST. Boldon *et al.*'s 'dual simplex' approach [1], based on Prim's algorithm, was found to perform very well on some benchmarks. More recently, Zhou and Gen [23] described a simple evolutionary algorithm (EA) using a Prüfer based encoding (see Section 3.2) and applied it to some small d-MST problems. In [9], Knowles and Corne improved on these with an EA with a decoder style representation called the randomized primal method (RPM) which employed a modified version of Prim's algorithm called 'd-Prim'. In RPM, a spanning-tree is encoded by an ordered list of integers which guides the series of edge-inclusion choices made by d-Prim.

Meanwhile, Raidl and Julstrom [18], also using an EA, explored a similar approach, but where a chromosome encoded for a list of multiplicative biasing weights; these weights were used to temporarily bias the edge weights of the graph, while d-Prim built the tree. This generally outperformed RPM. The best technique so far, however, seems to be Raidl's more recent work [17] which uses a direct spanning tree encoding, with associated specialized mutation and recombination operators.

In the multiobjective MST (mc-MST), a vector of weights is defined for each edge, and the problem is to find all Pareto optimal (or efficient) spanning trees. This is NP-hard [4] even in its unconstrained form. The mc-MST is commonly found in network-design oriented applications. For example, one set of weights may be associated with financial cost, while another may be associated with link reliability.

There has been much interest in the mc-MST in the operations research literature, e.g. [4, 7, 19]. But, there has, to our knowledge, been only one EA previously proposed for the mc-MST: a Prüfer-encoded multiobjective GA proposed by Zhou and Gen [24], which uses Srinivas and Deb's Non-Dominated Sorting method [20] and a Prüfer based encoding. They apply their method to randomly generated biobjective mc-MSTs ranging from 10 nodes to 50 nodes in size. The goal of their paper is to compare their EA with the results from an enumerative method (which is meant to find all true Pareto optimal solutions) which they propose in the same paper. Although the enumerative method is of course infeasible for large problems, comparisons of their EA with the enumerative method are meant to give an idea of how much of the true Pareto front the EA finds on small problems.

In this paper we describe two evolutionary algorithms for the mc-MST. One employs both the RPM method (see above) and the direct encoding and operators from Raidl's state-of-the-art approach (all with necessary modifications to address the multiobjective case). The other follows Zhou and Gen's approach from [24]. In addition, we describe an algorithm we call 'mc-Prim', which simply iterates Prim's algorithm for different weighted-sum single-objective forms of the given mc-MST. The techniques are all compared on a suite of test mc-MST problems generated in the same way as the test problems used by Zhou and Gen [24].

Unfortunately, direct comparison with Zhou and Gen's reported results is difficult because of the way their results are presented. As discussed above, Zhou and Gen verify the quality of solutions generated by their proposed GA using an enumeration method that is intended to find all efficient mc-MSTs. However, in a recent submission [11] we show that the proposed enumeration method of Zhou and Gen fails both to guarantee finding *all* Pareto optimal solutions, and to guarantee finding *only* Pareto optimal solutions. So when, for example, it is stated in [24] that their method found 85% of the true Pareto front on a particular problem, this must be considered to be unreliable (and perhaps a considerable overestimate).

Therefore, rather than comparing our results with tables of results in [24], we are limited to comparing with our implementation of their approach. Using a correct enumeration method we are able to find all true Pareto optima for the smaller problems, and using mc-Prim's we are able to find a subset of the true Pareto optima on the larger problems. Comparison of the different EA results with these sets takes the place of the comparisons done by Zhou and Gen with the results of their enumeration method.

In the following, Section 2 defines the mc-MST, and also describes Prim's algorithm [15] and our basic extension of Prim's for multiobjective problems. In Section 3 we further describe the RPM, Prüfer, and Raidl (direct) encodings and operators. Section 4 describes the multiobjective EA we use, and Section 5 describes experiments and results. We conclude in Section 6.

2 MST Problems and Basic Methods

A spanning tree of an undirected, connected graph, $G = (V, E)$, is a subgraph $T = (V, E_T), E_T \subset E$ that contains all vertices in V and connects them with exactly $|V| - 1$ edges, so that there are no cycles. If G is complete, then the set S of spanning trees T of G has $|S| = |V|^{|V|-2}$ members. If each edge $(i,j) \in E$ has $K > 1$ associated non-negative real numbers, representing K attributes defined on it and denoted with $\mathbf{w}_{i,j} = (w^1_{i,j}, w^2_{i,j}, \dots, w^K_{i,j})$, then the dc-MST problem may be defined as:

$$\begin{aligned} \text{“minimize” } \mathbf{W} &= (W^1, W^2, \dots, W^K) \\ \text{with } W^k &= \sum_{(i,j)\in E_T} w^k_{i,j},\ k \in 1..K \end{aligned} \qquad (1)$$

where the term 'minimize' is in quotation marks to indicate that it may not be possible to find a single solution that is minimal on all the components of $\mathbf{W}$. Instead, one is required to find a set of spanning trees $S^* \subset S$, called the Pareto optimal set, with the property that:

$$\forall T^* \in S^* \bullet \ \nexists T \in S \bullet T \prec T^* \qquad (2)$$

where $T \prec T^* \iff \forall k \in 1..K \bullet W^k \leq W^{k^*} \wedge \exists k \in 1..K \bullet W^k < W^{k^*}$. The expression $T \prec T^*$ is read as T *dominates* T^*, and solutions in the Pareto optimal set are also known as efficient or admissible solutions.

We now describe an efficient way of finding at least a subset of the true Pareto optimal solutions to an instance of the mc-MST. First we recall Prim's classic algorithm for the single objective MST, and then we indicate how this can be embedded in a slightly more sophisticated procedure for the multiobjective case.

Prim's algorithm uses two sets of vertices, C (connected) and U (unconnected). Initially, C contains a single, arbitrarily chosen vertex, while U contains all of the remaining vertices. The algorithm proceeds by moving vertices from U to C one at a time until U is empty. Each such move is associated with the addition of a specific edge to the growing tree. When U is empty, the tree is complete. The edge chosen at each step is one of minimal weight among those for which the following is true: it connects vertex $u \in U$ to $c \in C$, and its addition will not introduce a cycle. Clearly, having decided to add edge (u, c), vertex u is the one moved from U to C.

This procedure quickly and efficiently solves the single objective MST. A simple modification, in which we add an extra obvious feasibility test when moving a vertex between the two sets, enables it to build solutions to the dc-MST, but in that case it is an approximate heuristic rather than an exact method.

In preparation for the multiobjective version of Prim's, we first observe that a true Pareto optimal solutions to the mc-MST problem can be found by simply using Prim's algorithm (or any algorithm for the single-objective MST), by substituting the weight vector at each vertex with a weighted sum. This may be achieved since, for any scalarizing vector $\boldsymbol{\lambda} = (\lambda^1, \lambda^2, \dots, \lambda^K)$ with $\sum_{k=1}^K \lambda^k = 1$, and $\lambda_k \geq 0,\ k = 1..K$, the following holds:

$$\sum_{(i,j)\in E_T} \boldsymbol{\lambda}.\mathbf{w}_{i,j} = \sum_{k=1}^{K} \lambda^k \Big(\sum_{(i,j)\in E_T} w^k_{i,j} \Big). \qquad (3)$$

A tree which optimizes the term on the left will be a solution to the MST for the single-objective problem defined by the scalarization, and hence easily found by Prim's algorithm. But the rewrite of this term on the right reveals that this tree must also be on the true Pareto front of the corresponding mc-MST. If not, a tree exists which must improve one or more of the inner summed weight terms, contradicting the assumption that this sum was already minimal.

So, we can find an optimal solution to the mc-MST by replacing the vector weights defined on each edge in G by a scalar weight b formed by taking the inner product of $\boldsymbol{\lambda}$ and $\mathbf{w}$, $b = \boldsymbol{\lambda}.\mathbf{w}$, and then finding a spanning tree that minimizes the sum of the scalarized edge weights.

Algorithm: *mc-Prim*

U is the unconnected set of vertices
C is the connected set of vertices
E_T is the edge-set of the spanning tree T
S is the set of nondominated spanning trees

$S \leftarrow \emptyset$
$i \leftarrow 0$
while $(i < \binom{s+K-1}{K-1})$ /* *Main Loop* */
  $\boldsymbol{\lambda} \leftarrow \mathsf{nextvector}(\boldsymbol{\lambda})$
  $E_T \leftarrow \emptyset$
  $U \leftarrow V$
  $C \leftarrow \emptyset$
  $v \leftarrow \mathsf{rand}(V)$ /* *Begin Prim's Algorithm* */
  $C \leftarrow C \cup v$
  $U \leftarrow U \setminus v$
  while $(|C| < |V|)$
    select an edge (u, v) with $u \in C, v \in U$ so that
    $\forall i \in C, j \in U \ \sum_{k=1}^{K} \lambda^k.w_{u,v}^k \leq \sum_{k=1}^{K} \lambda^k.w_{i,j}^k$
    $E_T \leftarrow E_T \cup (u, v)$
    $C \leftarrow C \cup v$
    $U \leftarrow U \setminus v$
  $S \leftarrow S \cup T \leftarrow (V, E_T)$
  $i \leftarrow i + 1$
return (S) /* *Termination* */

Figure 1: mc-Prim

The algorithm, mc-Prim, shown in Figure 1 iteratively changes the scalarizing vector $\boldsymbol{\lambda}$ and uses Prim's algorithm to find a set of optimal solutions to an mc-MST. At each iteration of the main loop, a new scalarizing vector $\boldsymbol{\lambda}$ is generated by the function $\mathsf{nextvector}(\boldsymbol{\lambda})$, which successively generates every normalized scalarizing vector, $\boldsymbol{\lambda}$, with components equal to $l/s, l = 0..s$ where s is a parameter controlling the number of different vectors that will be generated. This gives $\binom{s+K-1}{K-1}$ different scalarizing vectors. For each different $\boldsymbol{\lambda}$ vector, Prim's algorithm is applied. This begins with the selection of a random vertex from V using the function $\mathsf{rand}(V)$, and at each subsequent step a minimum weight edge, that connects a vertex in the connected list to one in the unconnected list, is added to E_T.

For a large number of different scalarizing vectors (in our experiments we set $s = 1000$, giving 1001 different $\boldsymbol{\lambda}$ vectors), mc-Prim may generate an approximation to S^*, the set of Pareto optimal spanning trees, that is satisfactory in many cases. However, in general mc-Prim cannot usually discover the complete set S^* because it can only find solutions on the convex hull of the true Pareto front — the so-called *supported* efficient solutions. Such trees are Pareto optimal, but not minima of any single-objective scalarization.

3 Representations and Operators

We make use of three distinct spanning tree encodings in the experiments which follow, each with associated operators specific to it. They are described here.

3.1 The Randomized Primal Method

Recall that Prim's algorithm (after initialization in which the tree is set to contain a single arbitrary vertex) iterates through $|V| - 1$ steps. At each of these steps, the least cost edge is found which feasibly links an unconnected vertex to the growing tree. In RPM, each of these steps now involves choosing any feasible edge, rather than a minimal-cost edge, although there is a bias towards low-cost edges. For a k-vertex problem, an encoding of a spanning tree in the RPM method is a vector of $k - 1$ integers, each associated with a choice of edge addition. With vertex 1 assumed to be initially in the *connected* set C, each integer in turn is used as an index into a lookup table of feasible edges sorted by cost. The table is initialized by simply sorting all edges in ascending order of cost. After addition of an edge to the growing tree, all edges which could no longer feasibly be added to the tree at the next step are removed. Prim's algorithm corresponds to a vector of all 1s in this representation. In practice (details in Section 4), chromosomes are initialized using an exponential distribution biased towards lower values.

3.2 The Prüfer Encoding

Cayley [2] proved that there are n^{n-2} distinct labeled trees for a complete graph with n vertices. By providing a constructive proof of this, Prüfer established a bijective mapping between the set of distinct labeled trees and the set of all combinations of $n-2$ n-ary digits from n [16]. The details of the encoding are fairly well known and can be found in [24], for example. For present purposes, it is only necessary to note that this enables an encoding to be used in which a chromosome is simply a string of $n-2$ digits, where each can vary independently between 1 and n. Following [24], our implementation of the Prüfer encoding uses single-gene random mutation (a gene is chosen at random, and changed to any new integer between 1 and n), and uniform crossover [21].

3.3 Direct Tree Encoding and Operators

Raidl [17] uses a direct encoding of a spanning tree. That is, a spanning tree is explicitly represented by the list of edges which comprise it – Raidl calls this the 'edge-set' representation. Raidl's mutation operator simply replaces one of the existing edges of a tree with a new edge, all the time respecting feasibility constraints. In operation, it works as follows. First, an edge currently not in the tree is chosen to insert; this choice is biased towards lower cost edges. Naturally, this will temporarily introduce a cycle. A random choice among the edges in that cycle is then made (excluding the new edge), and the chosen one is removed from the tree.

Raidl's crossover operator is constructed on the principle of inheriting as many edges as possible from two parent trees. Initially, a child is set to contain the intersection of its parents' edge-sets, and hence will almost always be a forest of unconnected non-spanning trees. Next, remaining edges (from those which were in one parent but not both) are successively selected, and included in the tree if such inclusion is feasible. In order to bias this step towards considering the lower cost edges, Raidl uses binary tournament selection. At the end of this process, if the child is not yet a spanning tree, edges between its components are chosen randomly until it is.

Notably, Raidl's technique is challenging to implement efficiently. In particular, a fast way is needed to check whether or not a path exists between two given vertices in a given edge-set. We note, in agreement with Raidl [17], that the use of a union-find data-structure [8] is very helpful in this respect.

Finally, we add here that direct encodings of spanning trees, and associated efficient operators have also been proposed by Li and Bouchebaba [14] and Li [13] for the optimal communication spanning tree problem [6].

4 Algorithms

4.1 AESSEA

An evolutionary algorithm, AESSEA, based closely on procedures already defined for the Pareto archived evolution strategy (PAES) [10] is presented in Figure 2. AESSEA is a steady-state EA, that is, only one new solution is evaluated per 'generation'. It keeps a set of non-dominated solutions in an archive, and uses this set of solutions to estimate the quality of newly generated solutions. The algorithm is elitist in the sense that parents and offspring compete, but the overall selection

Algorithm: *Archived Elitist Steady-State EA*

P is the population
N is the archive of nondominated spanning trees
$\mathbf{a}, \mathbf{b}, \mathbf{c}$, and $\mathbf{x}$ are solution vectors

```
N ← ∅                                /* Initialization */
foreach (x ∈ P)
    init(x)
    evaluate(x)
    archive(x)
i ← 0
while (i < num_evals)                /* Main Loop */
    if rand() < p_c
        a ← rand_mem(P)
        b ← rand_mem(P)
        c ← crossover(a, b)
    else
        a ← rand_mem(P)
        c ← a
    c ← mutate(c)
    evaluate(c)
    if (c ≺ a)
        archive(c)
        P ← P ∪ c \ a
    else if ((∃j ∈ N | j ≺ a) ∧ (∃k ∈ N | k ≺ c))
        P ← P ∪ c \ a
    else if (¬∃j ∈ N | j ≺ c)
        if ((g_pop(c) < g_pop(a)) ∨ (∃j ∈ N | c ≺ j))
            archive(c)
            P ← P ∪ c \ a
    i ← i + 1
return (N)                           /* Termination */
```

Figure 2: AESSEA

pressure of the algorithm is not too strong since selection for mating is purely random, and offspring only replace one of their parents, rather than the weakest member of the population. Some testing of this algorithm and comparison with PESA [3] suggest that it is both an effective and computationally efficient, multiobjective EA.

In AESSEA, the function rand() returns a uniformly distributed deviate in $[0, 1)$, and the function rand_mem(P) returns with uniform probability a member of the current population, P. The function archive($\mathbf{c}$) updates the nondominated solutions archive N with $\mathbf{c}$, that is $\mathbf{c}$ is added to N if it is nondominated with respect to the other solutions in N, and if $\mathbf{c}$ dominates any members of N the dominated members are removed. The archive N has a finite capacity *arcsize* and in the case where adding $\mathbf{c}$ to N would cause $|N| > arcsize$, a solution in the most crowded region in N is removed. This archiving strategy is exactly the same as in the

PAES algorithm, described in greater detail in [10]. The function $\mathsf{g\_pop}(\mathbf{x})$ returns the number of solutions in the same grid location in objective space as the solution $\mathbf{x}$. This is part of a crowding strategy described in detail in [10].

The functions, $\mathsf{init}(\mathbf{x})$, $\mathsf{evaluate}(\mathbf{x})$, $\mathsf{crossover}(\mathbf{a}, \mathbf{b})$, and $\mathsf{mutate}(\mathbf{c})$, are to be defined depending on the application of the algorithm. In the following, we describe each of these functions for the two different representations that we use with AESSEA: the Prüfer coding, and the direct coding.

4.2 AESSEA+Prüfer

In the AESSEA+Prüfer algorithm, we closely follow the operators used in [24]. The chromosome is an integer string of length $|V|-2$, and cardinality $|V|$. This is decoded using a function described in [24], so that there is a one to one correspondence between chromosomes and spanning trees. The resultant spanning tree is then evaluated by summing the weights over the edges in the spanning tree. Initialization is performed by simply setting each gene to a uniformly random value in $[1, |V|]$, for every chromosome in the population. Mutation is applied to every gene with probability p_m, and sets the value of the mutated gene to a uniformly random value in $[1, |V|]$. The crossover used is uniform crossover [21].

4.3 AESSEA+Direct/RPM

In the AESSEA+Direct/RPM algorithm, the direct representation and specialized operators used follow those described in detail in [17], but they are adapted to deal with multiple weights on each edge. We also use the randomized primal method (RPM) [9] in the initialization process.

As indicated previously, a chromosome for the RPM procedure is a string of $|V|$ integers used as lookups into a dynamically adjusted table of edges ordered by weight. The integers are biased towards low values, and hence towards choosing lower cost edges, by setting each gene to an allele value $q = \lfloor e^{x^p \log(|V|)} \rfloor$, where x is a uniformly distributed random number in $[0,1)$, and p is a parameter that controls how strong is the bias towards lower allele values. We set $p=2$ in our experiments.

In addition to the chromosome, a scalarizing vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_K)$ is initialized from a uniformly random distribution such that $\sum_{k=1}^{K} \lambda_k = 1$ and $\forall k \in 1..K,\ \lambda_k \geq 0$. Each chromosome is then decoded into a spanning tree using the RPM and with the vector of edge weights replaced by a set of scalar edge weights, $b_{i,j}$, produced by taking the inner product of the scalarizing vector $\boldsymbol{\lambda}$ and $\mathbf{w}$: $\forall (i,j) \in E,\ b_{i,j} = \boldsymbol{\lambda}.\mathbf{w}_{i,j}$.

After the initialization procedure, the direct tree encoding only is used. Heuristic edge crossover and heuristic edge mutation, described briefly above and more fully in [17], are adapted to operate with vector edge weights. In our adapted heuristic edge crossover, the offspring inherits the scalarizing vector $\boldsymbol{\lambda}$ from parent $\mathbf{a}$. Then the scalarized weights $b_{i,j}$ are used in the tournament selection of edges used in the edge union operation of the heuristic edge crossover.

Adapting the heuristic edge mutation operator is a little more difficult. In Raidl's operator, the edges are sorted in order of weight (once at the beginning of the algorithm) and the heuristic mutation operator uses this sorted list to bias the selection of an edge to be inserted, towards cheaper edges. In the mc-MST problem, it is not possible to order the edges by weight, so that method cannot be applied without adaption. However, if the vector weights of edges are replaced by scalar edge weights $c_{i,j}$, produced by taking the inner product of the edge weight vectors with a scalarizing vector $\boldsymbol{\mu}$, with $\sum_{k=1}^{K} \mu_k = 1$ and $\forall k \in 1..K, \mu_k \geq 0$: $\forall (i,j) \in E,\ c_{i,j} = \boldsymbol{\mu}.\mathbf{w}_{i,j}$, then the edges can be sorted in increasing order of c, and the bias can be applied using this sorted list. We could use the scalarizing vector $\boldsymbol{\lambda}$ that is associated with each solution vector, putting $\boldsymbol{\mu} = \boldsymbol{\lambda}$, to generate all the $c_{i,j}$, but we do not wish to have to re-sort the edges for each different $\boldsymbol{\lambda}$ vector, as this would strongly compromise the computational efficiency of the mutation operator. To get around this, we pre-sort the edges for a small sample of different, evenly distributed $\boldsymbol{\mu}$ vectors at the beginning of the algorithm. In the mutation operator itself, we first mutate the $\boldsymbol{\lambda}$ vector of the current solution vector by adding a Gaussian random deviate of mean zero and standard deviation $\sigma = 0.05$ to each of the components of $\boldsymbol{\lambda}$ and then re-normalizing it. Then, when choosing an edge to insert into the current tree, the sorted edge list that was generated using a $\boldsymbol{\mu}$ vector which most closely matches the updated $\boldsymbol{\lambda}$ vector of the solution undergoing mutation is used. In our experiments, 5 different representative $\boldsymbol{\mu}$ vectors: (0,1),(0.25,0.75),(0.5,0.5),(0.75,0.25),(1,0) are defined, so that just 5 edge-sorts must be performed (just once) at the beginning of the algorithm. Thus, our adaptions of Raidl's operators preserve their linear time complexity whilst also maintaining the heuristic nature of their biasing edge choices towards lower-weighted edges.

5 Experiments

5.1 Problem instances

Following [24], we generate biobjective MST instances using complete graphs of size 10, 20, 30, 40, and 50 vertices with uniformly randomly distributed edge weights $w^1_{i,j} \in [10, 100],\ w^2_{i,j} \in [10, 50], \forall (i,j) \in E$.

5.2 Method

Three different instances at size 10 are generated, and on these instances the complete space of solutions is enumerated so that we obtain all Pareto optimal solutions. For these problems we then present results in terms of the proportion of the efficient set that is discovered by the genetic algorithms, over 30 independent runs. This is a similar method to that used by Zhou and Gen for verifying the effectiveness of their GA. However, we enumerate the set of efficient spanning trees using a straightforward technique where every possible spanning tree is generated using a Prüfer code, and then those that are dominated are discarded. Although better techniques exist [19], we have been unable to obtain source code for these methods to date. Nonetheless, our enumeration does correctly guarantee to find all and only nondominated spanning trees, in contrast to the method used by Zhou and Gen [11].

For the other sizes of network, just one instance at each size is used. Because of the larger size of these problem instances, we cannot use our enumeration method. Therefore, we instead compute a subset of the supported efficient solutions using our mc-Prim algorithm for 1001 different $\boldsymbol{\lambda}$ vectors: $(0,1),(0.001,0.999),\ldots,(0.999,0.001),(1,0)$.

On the small problems, since we know the complete true Pareto front, we evaluate algorithm results by looking at the percentage of the true Pareto front found by the algorithm. We in fact look at the percentages of both the full Pareto front and its complete supported subset. For the larger problems, however, only a subset of the true supported Pareto front is available to us. In these cases we look at the size of the total discovered region [12]. This is simply the area (since our problems are two-dimensional) contained by the approximation to the Pareto front which the algorithm discovers.

The parameters used in the AESSEA algorithms are given in Table 1. The mutation rate is quoted as $1/L$, where L is the length of the chromosome. For the Prüfer encoding $L=|V|-2$. For the AESSEA+Direct/RPM, this parameter is not actually set but it refers to the fact that using Raidl's edge encoding and heuristic edge mutation, exactly one edge in the tree is changed per mutation, which is equivalent to a mutation rate of $1/L$ on a direct edge encoding.

5.3 Results

Table 3 shows the results on the 10-node problems. On the small problems, AESSEA+Direct/RPM finds all optima on all individual runs, and in only 20000 evaluations. The AESSEA+Prüfer method performs considerably worse, often finding no true Pareto optima at all. On the larger problems, with results summarised in Table 4, the discovered Pareto front of AESSEA+Prüfer is considerably smaller than that from either of the other methods.

Such findings are illustrated in Figure 3, clearly indicating how well AESSEA+Direct/RPM approximates the true Pareto front in comparison to AESSEA+Prüfer. As we note from similar figures (not displayed here) for the other problems, this difference is increasingly marked as problem size increases.

The runtime figures given in Table 2 seem to indicate that both EAs scale well with increasing problem size.

Parameter	AESSEA+Prüfer	AESSEA+D/RPM
$\lvert P\rvert$	200	200
arcsize	100	100
p_c	0.2	0.2
p_m	$1/L$	$1/L$
num_evals	100000	20000
grid squares	1024	1024

Table 1: Parameter settings for the two AESSEA algorithms, all kept constant across the different problem instances. The parameter *arcsize* is the capacity of the nondominated solutions archive, and the grid squares parameter refers to the number of grid locations the objective space is divided into for the crowding strategy (see Section 4.1).

Problem	AESSEA+Prüfer	AESSEA+D/RPM
20v1	$19.7s$	$14.2s$
50v1	$49.6s$	$28.0s$

Table 2: Run-times for the two algorithms on two different-sized problems. The algorithms were run on a Sun SPARC Ultra 5 300MHz with 256MB RAM. The timings are the mean of 10 runs. On each run, AESSEA+Prüfer performed 100000 function evaluations, and AESSEA+Direct/RPM performed 20000, as in the experiments reported.

6 Concluding Discussion

The results indicate that a direct encoding is more effective than a Prüfer-based encoding when using an EA to address the mc-MST. We should point out that our conclusions differ from those made by Zhou and Gen for their Prüfer GA, which seemed to show that their method was able to find high proportions of the true Pareto front on all sizes of problem, which is surprising in light of the results we find using the Prüfer encoding on large problems. We suggest the discrepancy arises from Zhou and Gen's use of an unreliable Pareto front enumeration method, as noted earlier. In addition to the results we report, reported theoretical notes provide additional evidence for direct encodings being a better choice than the Prüfer encoding for MST problems. For example, Ehrgott and Klamroth [5] noted that, in 50 random instances in which they generated a complete set of efficient spanning trees, the set was ergodic with respect to a single-edge exchange operator (although they prove

Problem instance	$\|SPF_t\|$	$\|PF_t\|$	AESSEA+Prüfer $\%\|SPF_t\|$	$\%\|PF_t\|$	$\|PF\|$ mean	$\|PF\|\ \sigma$	AESSEA+Direct/RPM $\%\|SPF_t\|$	$\%\|PF_t\|$	$\|PF\|$ mean	$\|PF\|\ \sigma$
10v1	10	42	80	76	2.47	1.33	100	100	42.00	0.00
10v2	12	28	67	71	0.50	0.78	100	100	28.00	0.00
10v3	12	25	83	84	0.57	0.82	100	100	25.00	0.00

Table 3: Results on three 10-vertex mc-MST instances using the two different EAs. $|SPF_t|$ and $|PF_t|$, give the number of supported efficient points and the total number of efficient points on the *true* Pareto front, respectively, for each problem instance. $\%|SPF_t|$ gives the percentage of the supported efficient points that were found over the 30 runs *combined*, for each algorithm. Similarly, $\%|PF_t|$ gives the percentage of the total number of efficient points that were found over the 30 runs *combined*, for each algorithm. $|PF|$ mean, and $|PF|\,\sigma$, give respectively the mean and standard deviation of the number of efficient points found *per* algorithm run, for each algorithm.

Problem instance	$\|SPF_{Pr}\|$	Upper limit of bounding box	Size of the total discovered region: mc-Prim	AESSEA+Prüfer	AESSEA+Direct/RPM
20v1	25	832 x 574	0.2949	0.1488	0.2986
30v1	54	1280 x 838	0.3780	0.1249	0.3792
40v1	63	1882 x 1181	0.4200	0.1150	0.4196
50v1	85	2425 x 1494	0.4473	0.0943	0.4451

Table 4: The size of the total discovered region of the AESSEA algorithms and mc-Prim on four mc-MST problem instances of increasing size. The column, $|SPF_{Pr}|$, gives the number of supported efficient solutions found by mc-Prim, using 1001 different $\boldsymbol{\lambda}$ vectors. The upper limit of the bounding box is determined, in each dimension, by taking the maximum value (over all algorithms) of any point on the combined Pareto front of each algorithm. The lower limit is taken to be zero in each dimension.

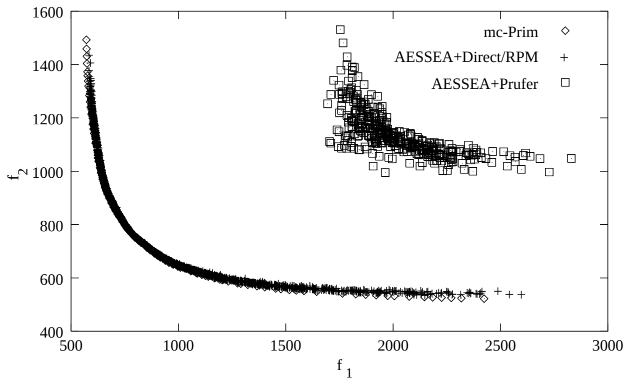

Figure 3: Combined discovered points found by the two versions of AESSEA and mc-Prim on a 50-vertex problem

that this will not always be true). A direct encoding such as Raidl's, which naturally allows for such operators (as indeed used here) seems thus well-designed for the problem, while the high non-locality of the Prüfer encoding seems ill-chosen in light of this. A further difficulty with the Prüfer encoding is the fact that many real problems will have sparse graphs. That is, although the problem may still be very large in terms of allowed edges, most edges will not be allowable. This could be dealt with in Prim's algorithm, for example, by giving those edges massive weight (vectors), large enough to guarantee that they would never be chosen in tree construction. A far better approach, however, is simply not to include these edges in the edge array accessed by the algorithm. The latter natural and effective way to deal with sparse problems is trivial for both RPM and Raidl's direct encoding, but is unavailable to the Prüfer encoding. This is because it is impossible to control which edges are coded for by a Prüfer number, so the decoding process will typically yield infeasible trees (containing edges not members of the sparse graph).

It is also worth noting that mc-Prim's alone is capable of finding well-spread sets of true Pareto optima for the mc-MST much faster than an EA. However, mc-Prim's is incapable of finding unsupported Pareto optima, of which there may be many between any two neighbouring supported optima. For these reasons, mc-Prim's is able to display impressive results in Table 4, but will always fail to offer a fine-grained view of the tradeoff surface, such as would be desired in many applications. Also, note that although mc-Prim's is easily adaptable to degree-constrained problems, it certainly no longer guarantees to find true optima in those cases.

We suspect that many real-world problems, as well as having degree-constraints, are also quite sparse, and have a considerable number of unsupported solutions.

Acknowledgments

We are grateful to British Telecommunications for continuing sponsorship. Thanks also to the anonymous reviewers for insightful comments.

Bibliography

[1] B. Boldon, N. Deo, and N. Kumar. 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, Orlando, FL 32816, 1995.

[2] A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.

[3] D. W. Corne and J. D. Knowles. The Pareto-envelope based selection algorithm for multiobjective optimization. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN VI)*, pages 839–848, Berlin, 2000. Springer-Verlag.

[4] M. Ehrgott. *Multicriteria Optimization*. Number 491 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, 2000.

[5] M. Ehrgott and K. Klamroth. Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research*, 97:159–166, 1997.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[7] H. W. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52:209–230, 1994.

[8] J. Kingston. *Algorithms and Data Structures*. Addison Wesley, Singapore, 1990.

[9] J. Knowles and D. Corne. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134, July 2000.

[10] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[11] J. D. Knowles and D. W. Corne. Enumeration of all optimal multi-criteria spanning trees - a proof of the incorrectness of Zhou and Gen's proposed algorithm. (submitted as a short communication to the *European Journal of Operational Research*), December 2000.

[12] J. D. Knowles, D. W. Corne, and M. J. Oates. On the Assessment of Multiobjective Approaches to the Adaptive Distributed Management Problem. In *Proceedings of the Sixth International Conference of Parallel Problem Solving From Nature (PPSN VI)*, pages 869–878, Berlin, 2000. Springer-Verlag.

[13] Y. Li. An effective implementation of a GA using a direct tree representation for constrained minimum spanning tree problems. Technical Report 2000-15, LaRIA, Université de Picardie Jules Verne, France, 2000.

[14] Y. Li and Y. Bouchebaba. A new genetic algorithm for the optimal communication spanning tree problem. In *Artificial Evolution: 4th European conference AE'99*, pages 162–173. Springer, 1999.

[15] R. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, 1957.

[16] H. Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys.*, 27:742–744, 1918.

[17] G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*, pages 104–111, Piscataway, NJ, 2000. IEEE Press.

[18] G. R. Raidl and B. A. Julstrom. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning problem. In *Proceedings of the 2000 ACM Symposium on Applied Computing*. ACM Press, 2000.

[19] R. M. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111:617–628, 1998.

[20] N. Srinivas and K. Deb. Comparative study of vector evaluated GA and NSGA applied to multiobjective optimization. In P. K. Roy and S. D. Mehta, editors, *Proceedings of the Symposium on Genetic Algorithms*, pages 83–90, 1995.

[21] G. Syswerda. Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

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

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

[24] G. Zhou and M. Gen. Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. *European Journal of Operational Research*, 114(1), April 1999.