An Efficient Evolutionary Algorithm for the Degree-Constrained Minimum Spanning Tree Problem

Günther R. Raidl
Institute of Computer Graphics
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
raidl@apm.tuwien.ac.at

Abstract- The representation of candidate solutions and the variation operators are fundamental design choices in an evolutionary algorithm (EA). This paper proposes a novel representation technique and suitable variation operators for the degree-constrained minimum spanning tree problem. For a weighted, undirected graph $G(V, E)$, this problem seeks to identify the shortest spanning tree whose node degrees do not exceed an upper bound $d \geq 2$. Within the EA, a candidate spanning tree is simply represented by its set of edges. Special initialization, crossover, and mutation operators are used to generate new, always feasible candidate solutions. In contrast to previous spanning tree representations, the proposed approach provides substantially higher locality and is nevertheless computationally efficient; an offspring is always created in $O(|V|)$ time. In addition, it is shown how problem-dependent heuristics can be effectively incorporated into the initialization, crossover, and mutation operators without increasing the time-complexity. Empirical results are presented for hard problem instances with up to 500 vertices. Usually, the new approach identifies solutions superior to those of several other optimization methods within few seconds. The basic ideas of this EA are also applicable to other network optimization tasks.

1 Introduction

The problem of identifying a *minimum spanning tree* (MST) of a connected, undirected graph is a classical combinatorial optimization problem which can be solved efficiently in polynomial time by greedy heuristics like Kruskal's [14] or Prim's [22]. Unfortunately, there are several practically relevant variants of the MST problem that have been shown to be $\mathcal{NP}$-complete and therefore computationally expensive to solve in exact ways. One of these related problems is the *degree-constrained MST* problem, in which a minimum spanning tree is searched none of whose vertices has a degree greater than $d \geq 2$. This problem has applications in the design of telecommunication networks and integrated circuits.

Due to the hardness of the degree-constrained MST problem, it is addressed by heuristic methods including evolutionary algorithms (EAs). In contrast to several previous evolutionary approaches, we present here a new, straight-forward technique involving a simple edge-set representation in combination with specialized initialization, crossover, and mutation operators. In this way, only feasible candidate solutions are always produced, and a high level of locality is guaranteed. Nevertheless, the computational effort of crossover and mutation is low. The proposed EA has further been improved by including heuristics based on the general idea to prefer low-cost edges over others. Empirical comparisons with other optimization methods on several hard graphs indicate a superior behavior of the new approach. Besides the usually higher quality of final solutions, the new EA is faster especially for problems involving large graphs.

The following section describes the degree-constrained MST problem. A brief summary of tree representations used in previous EAs for MST related problems is given in Sect. 3. The new edge-set representation together with the appropriate initialization, crossover, and mutation operators is presented in Sect. 4. Section 5 describes how edge-cost based heuristics can be incorporated into the EA. Several implementation details of the overall EA can be found in Sect. 6, and an empirical comparison to other optimization techniques is given in Sect. 7. Some concluding remarks are made in Sect. 8.

2 The Degree-Constrained Minimum Spanning Tree Problem

A spanning tree of an undirected, complete graph $G(V, E)$ is a cycle-free subgraph $T(V, E_{\mathrm{T}})$, $E_{\mathrm{T}} \in E$, such that all vertices in V are connected. Note that a spanning tree always consists of $|V| - 1$ edges, and a complete graph G has $|V|^{|V|-2}$ spanning trees [4]. When numerical costs $c_{i,j} \geq 0$ are associated with each edge $(i, j) \in E$, $i, j \in V$, a minimum spanning tree (MST) is a spanning tree with minimum total edge cost

$$C = \sum_{(i,j)\in E_{\mathrm{T}}} c_{i,j} \,.$$

In the degree-constrained MST problem, we consider the additional constraint that the degree $deg(i)$ of every vertex $i \in V$, i.e. the number of edges adjacent to every vertex, must be less than or equal to a given upper bound d. Thus we seek from all spanning trees fulfilling the degree constraint (d-STs) one that minimizes the total edge cost (a d-MST).

A 2-MST is a Hamiltonian path of minimum length. Finding such a path is related to the familiar traveling salesman problem and is $\mathcal{NP}$-hard [8]. Often the vertices on which we

seek MSTs are points in the plane, and edge costs are the Euclidean distances between these points. For this case, Monma and Suri [17] showed that there always exists a MST with degree no more than five. Although Papadimitriou and Vazirani [20] proved that finding a d-MST in the Euclidean plane is $\mathcal{NP}$-hard when $d = 3$, and conjectured that it remains $\mathcal{NP}$-hard when $d = 4$, Euclidean problems are relatively simple to solve. Exact branch-and-bound techniques as described by Narula and Ho [18] and Krishnamoorthy et al. [13] can find optimal solutions even for large problem instances including several hundred vertices in reasonable computing times. Furthermore, there exist effective polynomial-time heuristics for finding d-MSTs in the plane [7, 25].

In the more general case, the costs associated with the graph's edges are arbitrary and need not satisfy the triangle inequality. In this case, a minimum spanning tree may have degree up to $|V| - 1$. Finding a d-MST in a graph with such a high-degree MST is usually a hard task. Exact approaches are too time-consuming, and many existing heuristics are either not applicable or less efficient.

Narula and Ho [18] proposed a simple but relatively effective heuristic: They modified Prim's algorithm so that at each step it includes the cheapest eligible edge—one connecting a vertex currently in the (partial) spanning tree with one not yet connected—that does not violate the degree constraint. We refer to this heuristic as d-Prim.

3 Representations of Trees in EAs

Another appealing approach is to use evolutionary algorithms for identifying low-cost d-STs. A crucial decision is how to represent potential solutions in an EA. Several approaches can be found in the literature for representing spanning trees of graphs.

Piggott and Suraweera [21] use a bit string of size $|E|$ to represent a solution. Even when they ensure that each bit string always contains exactly $|V| - 1$ set bits, there is only a small chance that such a bit string actually represents a valid spanning tree. In addition, each bit string needs $O(|E|) = O(|V|^2)$ memory for a complete graph G and also the computational effort to go back and forth between the encoding and the tree is $O(|V|^2)$.

Palmer and Kershenbaum [19] and Krishnamoorthy et al. [13] describe a predecessor encoding in which a root vertex is designated, and for each other vertex, the immediate predecessor on the path to this root is stored. Although only $O(|V|)$ memory is needed, this encoding does also represent subgraphs that are not feasible trees. Chu et al. [5] presented a variant of predecessor encoding for the d-MST problem. Infeasible solutions are repaired if possible or penalized otherwise. Another encoding that allows also non-trees to be represented is the determinant factorization encoding described by Abuali et al. [1].

A deceptively appealing coding of spanning trees for evolutionary search is based on Prüfer's proof of Cayley's Formula, which identifies the number of distinct spanning trees on a complete graph as $|V|^{|V|-2}$ [4], [6, pp. 103–104]. The proof establishes a one-to-one correspondence between spanning trees and strings of length $|V| - 2$ over an alphabet of $|V|$ symbols by describing algorithms that derive a tree from its string and vice versa [6, pp. 104–106].

Despite its elegance, the Prüfer coding is problematic in EAs due to its weak locality. As several observers have pointed out, a symbol's meaning depends on all its predecessors [12, 13, 19, 24]. Patterns of symbols do not represent consistent substructures of spanning trees, so that crossover may generate offsprings whose trees do not resemble the trees of their parents, and the mutation of even one symbol may change the represented tree radically.

Nonetheless, several researchers use the Prüfer coding to represent candidate solutions in EAs for spanning tree problems. Zhou and Gen [26] presented a Prüfer-coding based EA for the d-MST problem. This algorithm applies conventional genetic operators: uniform crossover and mutation by randomly modifying a symbol. Solutions that violate the degree constraint are repaired. Two other variants of genetic algorithms that employ Prüfer coding for solving the d-MST problem are presented by Krishnamoorthy et al. [13]. The better variant (F-EA) simply discards infeasible spanning trees violating the degree constraint and uses standard single point crossover.

Kim and Gen [10] extended the Prüfer coding in an EA for another network design problem. A Prüfer string represents a spanning tree of service centers, and a second string indicates clusters of users the centers serve. Gargano et al. [9] extended the Prüfer coding with permutations in an EA for the time-dependent minimum spanning tree problem, in which edge costs depend on when the edges are included in the spanning tree; the Prüfer string represents a spanning tree, and the permutation the order in which vertices, and thus edges, are added to the tree.

Li and Bouchebaba [15] presented an EA for the optimal communication spanning tree problem. This approach represents trees in a direct way via their adjacency lists. Specialized operators such as path crossover and path mutation are used to generate always feasible new solutions. Although this approach works well for the optimal communication spanning tree problem, it seems difficult to adapt the genetic operators for the d-MST problem. Depending on the implementation, the proposed recombination and mutation operators seem to have a computational effort of at least $O(|V|^2)$.

Knowles and Corne [12] described another EA for the d-MST problem. In their algorithm, chromosomes are sequences of integer values that influence the order in which d-Prim (see Sect. 2) connects vertices to the growing spanning tree. Experiments on large, misleading graphs indicated the superiority of this approach over several other heuristics including simulated annealing and a dual simplex heuristic proposed by Boldon et al. [3]. We refer to this EA as K-EA.

Another efficient, indirect encoding of spanning trees was

originally proposed by Palmer and Kershenbaum [19]. Raidl and Julstrom [23] improved this technique and adapted it to the d-MST problem. In this weight-coded approach, a feasible spanning tree is represented by a string of numerical weights associated with the vertices V. During decoding, these weights temporarily bias the graph's edge costs $c_{i,j}$, and d-Prim, applied to the biased costs, identifies the feasible spanning tree a chromosome represents. In an experimental comparison, this weight-coded EA outperformed Knowles and Corne's approach. We refer to this EA as W-EA.

Although K-EA and W-EA work well, a disadvantage of both is the high computational effort. Both approaches use d-Prim for decoding a solution, and the computational complexity of d-Prim is $O(|V|^2 \log |V|)$. Therefore, the running time increases dramatically for larger problem instances.

4 The Edge-Set Representation

Following the general rules of designing EAs [2, 16] and considering the experiences with previous approaches, we stated the following properties as our primary design goals for a new, efficient EA for the d-MST problem:

(1) Only feasible d-STs should be generated by the EA, and all possible d-STs should be representable.

(2) A 1:1 correspondence between chromosomes and d-STs seems to be desirable, i.e. each d-ST should be represented by a unique chromosome and vice versa.

(3) The decoding of a chromosome, the crossover operator, and the mutation operator should be computationally efficient in order to be able to practically apply the EA also to large problem instances.

(4) The encoding together with the crossover and mutation operators has to provide a high level of locality, i.e. a d-ST generated by the crossover operator should inherit most edges from its parents, and mutation should change only few edges.

(5) It should be easy to incorporate problem specific heuristics or local improvement operators. Many EAs for combinatorial optimization problems benefit from such a hybridization in terms of shorter running times or better final solutions.

The following straight-forward *edge-set representation* in combination with specialized initialization, crossover, and mutation operators fulfills all these criteria.

A candidate solution of the EA, i.e. a d-ST, is directly represented by the set E_{T} of edges forming the tree. In an efficient implementation, a hash-table storing each pair of vertices connected by an edge ($(i,j) \in E_{\mathrm{T}}$) can be used for this purpose. In this way, the insertion or deletion of an edge and the test whether a given edge is contained in the tree or not can be performed with expected constant effort, and traversing all edges needs $O(|V|)$ time.

4.1 Initialization

In order to create only feasible solutions for the initial population of the EA, an algorithm that creates valid random d-STs is needed. Figure 1 shows the pseudo-code of a procedure derived from Kruskal's MST algorithm [14]. First, E_{T} is initialized to the empty set, and a random ordering of all edges in E is determined. One edge after the other is then checked in the predetermined order for an eventual inclusion in the spanning tree. Edge (i,j) is included in E_{T} if it does not violate the degree constraint in vertices i and j, and these vertices are not yet connected via other edges in E_{T}. The procedure terminates when a complete spanning tree has been built, i.e. when the number of edges in E_{T} is one less than the number of vertices. Note that this procedure will always produce a valid d-ST since only edges that do not introduce a cycle or degree violation are included.

```
procedure initialize;
begin
  E_T ← ∅;
  for all edges (i, j) ∈ E in random order do
    if deg(i) < d and deg(j) < d
        and not(connected(i, j, E_T)) then
      E_T ← E_T ∪ {(i, j)};
      if |E_T| = |V| − 1 then
        return E_T;
end;
```

Figure 1: *Creating an initial, random d-ST.*

For an efficient implementation of the test whether two vertices are already connected via some edges or not, a union-find data structure should be used [11]. In this way, the test can be performed with constant expected time effort, and the whole procedure runs in $O(|E|) = O(|V|^2)$ time.

Assuming we have a feasible parental population, new solutions are generated by means of the following *edge crossover* and *edge insertion mutation* operators.

4.2 Edge Crossover

The development of the crossover operator was substantially guided by the idea to produce a new d-ST by inheriting as many edges as possible from two parental d-STs. Figure 2 shows an illustrative example, and a detailed pseudo-code of the procedure is given in Fig. 3.

In a first step, E_{T} is initialized to the set of edges contained in both parents E_{T}^1 and E_{T}^2. These edges will always be included in the new solution. In a second step, all edges contained either in E_{T}^1 or E_{T}^2 (but not in both) are checked for inclusion. As in the initialization procedure, these edges are processed one after the other in a random order and only edges that do not introduce a degree violation or a cycle are included in E_{T}. If a complete d-ST with $|V| - 1$ edges can be constructed in this way, the procedure returns this solution and terminates. Unfortunately, this may not always be the case due to the degree constraint, see Fig. 2. Such a partial spanning tree must be completed by including also edges not contained in the parents.

For this purpose we could proceed by checking all re-

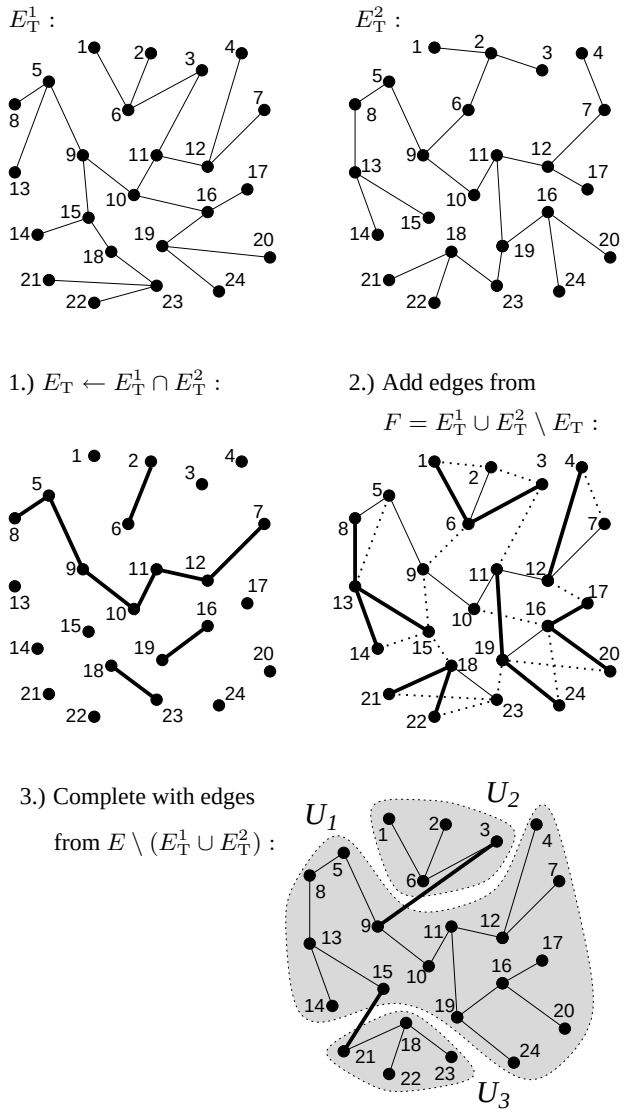

Figure 2: *An example for edge crossover ($d = 3$).*

maining edges $E \setminus (E_{\mathrm{T}}^1 \cup E_{\mathrm{T}}^2)$ for inclusion as before, but this would lead to a computational effort of $O(|V|^2)$. In order to complete a partial solution in a more efficient way, all unconnected components are determined in a first step, i.e. V is partitioned into disjointed sets U_k containing vertices only connected to each other. Then, these components are connected to the final d-ST by repeatedly choosing two random vertices with a degree smaller than d in two unconnected components and including an edge between these vertices in E_{T}. In this way, the completion step and therefore the whole edge crossover can be implemented with a computational effort of only $O(|V|)$.

4.3 Edge Insertion Mutation

The edge insertion mutation is based on the principle of inserting a randomly chosen new edge and deleting another edge lying on the cycle introduced by the insertion, see Fig. 4.

procedure edge-crossover$(E_{\mathrm{T}}^1, E_{\mathrm{T}}^2)$;
begin
  $E_{\mathrm{T}} \leftarrow E_{\mathrm{T}}^1 \cap E_{\mathrm{T}}^2$;
  $F \leftarrow E_{\mathrm{T}}^1 \cup E_{\mathrm{T}}^2 \setminus E_{\mathrm{T}}$;
  for all edges $(i, j) \in F$ in random order **do**
    if $deg(i) < d$ **and** $deg(j) < d$
        and not(connected(i, j, E_{T})) **then**
      $E_{\mathrm{T}} \leftarrow E_{\mathrm{T}} \cup \{(i, j)\}$;
      if $|E_{\mathrm{T}}| = |V| - 1$ **then**
        return E_{T};
  (* determine all unconnected components U_k: *)
  $U \leftarrow \{U_k\}$ with $\forall i, j \in V, i \neq j$:
    $i \in U_k \wedge$ connected$(i, j, E_{\mathrm{T}}) \longrightarrow j \in U_k$,
    $i \in U_k \wedge$ **not**(connected$(i, j, E_{\mathrm{T}})) \longrightarrow j \notin U_k$,
    $\bigcup_k U_k = V$;
  (* connect components randomly: *)
  for all $U_k \in U \setminus \{U_1\}$ in random order **do**
    choose $i \in U_1 \mid deg(i) < d$ randomly;
    choose $j \in U_k \mid deg(j) < d$ randomly;
    $E_{\mathrm{T}} \leftarrow E_{\mathrm{T}} \cup \{(i, j)\}$;
    $U_1 \leftarrow U_1 \cup U_k$;
  return E_{T};
end;

Figure 3: *Edge crossover.*

The detailed procedure in Fig. 5 starts by randomly choosing two different vertices i and j to be connected with a new edge. Care must be taken that not both vertices have degree d, since inserting such an edge would lead to an unavoidable degree constraint violation in the final tree. Therefore, we restrict the choice of j to vertices of degree less than d. In the next step, we determine the set L of edges lying on the path from vertex i to vertex j in the original tree. This can be done in $O(|V|)$ time by temporarily building an adjacency list representation of the tree and performing a depth-first search. Together with edge (i, j) the edges in L describe the cycle that would arise when inserting (i, j) into E_{T}. Therefore, an edge $(a, b) \in L$ must be chosen for deletion. In case vertex i already has degree d, the edge adjacent to i must be selected to avoid a degree constraint violation; otherwise, we make a random choice. Finally, edge (i, j) is inserted into the solution while (a, b) is deleted from it. The computational effort of the whole procedure is again $O(|V|)$.

5 Inclusion of Edge-Cost Based Heuristics

Often, an EA can be improved by incorporating problem specific heuristics or local optimization techniques [2, 16]. For the proposed EA employing the edge-set representation, we consider the heuristic assumption that usually an inclusion of edges with small costs $c_{i,j}$ should be prefered over an inclusion of more expensive edges. This basic idea is incorporated into the procedures for initialization, crossover, and mutation in the following ways.

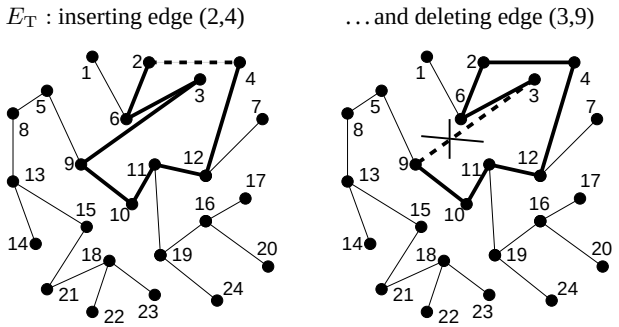

Figure 4: *An example for the edge insertion mutation with $d = 3$. The short-circuited path is L={(2,6), (3,6), (3,9), (9,10), (10,11), (11,12), (4,12)}.*

procedure edge-insertion-mutation(E_{T});
begin
  (* choose edge (i, j) for insertion: *)
  choose $i \in V$ randomly;
  choose $j \in V \setminus \{i\} \mid deg(j) < d \wedge (i, j) \notin E_{\mathrm{T}}$ randomly;
  (* choose edge (a, b) for deletion: *)
  $L \leftarrow \{(k, l) \in E_{\mathrm{T}} \mid (k, l)$ lies on path from i to $j\}$;
  if $deg(i) = d$ **then**
    $(a, b) \leftarrow (a, b) \in L \mid a = i \vee b = i$;
  else
    choose $(a, b) \in L$ randomly;
  $E_{\mathrm{T}} \leftarrow E_{\mathrm{T}} \cup \{(i, j)\} \setminus \{(a, b)\}$;
  return E_{T};
end;

Figure 5: *Edge insertion mutation.*

Heuristic initialization. Instead of processing all edges in a completely random order, we sort all edges according to increasing costs. The first candidate solution of the initial population is then created using this processing order. To ensure diversity, all following initial solutions are created with less heuristic bias by randomly permuting the cheapest k edges from the initial, sorted edge processing order. In this process, the number k of shuffled edges is increased with the time, i.e. the i-th initial solution of a population of size P ($i = 1, \ldots, P$) is created with

$$k = \alpha\,(i-1)\,|V|\,/\,P\,.$$

α is considered a strategy parameter which controls the average heuristic bias, respectively the diversity, of the initial population.

Heuristic edge crossover. During crossover, the order of checking edges from $F = E_{\mathrm{T}}^1 \cup E_{\mathrm{T}}^2 \setminus (E_{\mathrm{T}}^1 \cap E_{\mathrm{T}}^2)$ for inclusion into the offspring E_{T} is crucial. We use an edge-order that gives low-cost edges higher probabilities to be processed before more expensive edges and, therefore, to be included in E_{T}: As long as F is not empty, the next edge is determined via a binary tournament selection on the edges in F, i.e. a group of two edges is chosen randomly and the cheaper edge is selected. This edge is then removed from F and checked for inclusion into E_{T}.

Heuristic edge insertion mutation. The mutation operator is hybridized by giving low-cost edges higher probabilities for being selected for the insertion into E_{T}. For this purpose, a tournament selection could be performed on the set of all allowed edges, but since this set is huge, a large group size depending on $|V|$ would be necessary for the tournament selection to favor the cheapest edges in an appropriate way. The following different strategy turned out to be more effective: We choose the i-th cheapest edge $e_i \in E$ with

$$i = \lfloor |\mathcal{N}(0, \beta\,|V|)| \rfloor \bmod (|V|(|V|-1)/2) + 1\,.$$

$\mathcal{N}(0, \beta\,|V|)$ is a normally distributed random number with mean 0 and standard deviation $\beta\,|V|$. By determining the absolute value, rounding, calculating the modulus, and finally adding 1, we obtain a positive random integer in $[1, |V|(|V|-1)/2] = [1, |E|]$ with a distribution that favors low-cost edges depending on the strategy parameter β. Smaller values for β will lead to a stronger bias towards cheaper edges. Unfortunately, there is the possibility that the chosen edge is already contained in the solution E_{T} or the edge connects two vertices having already reached the maximum degree. Since the probability of this case is only small, we repeatedly determine new edges in the described way until a feasible edge is found.

In a proper implementation, all edges of E are sorted according to their costs only once at the beginning of a run, since this edge-order is needed during each initialization and mutation. In this way, the proposed hybrid operators can be efficiently implemented, i.e. especially crossover and mutation run as before in linear time.

6 The Evolutionary Algorithm

The new edge-set representation and its operators were implemented in an otherwise conventional steady-state EA. Strategy parameters were tuned by performing preliminary test runs. In the tests the next section describes, the following configuration was used. The EA's population holds $P = 100$ solutions. Offsprings are generated by selecting parents via binary tournaments and applying crossover with probability $p_c = 0.8$ and additionally also mutation with probability $p_m = 0.8$. A new offspring replaces the population's current worst solution, with one exception: To preserve diversity, the EA discards any offspring that is identical to a solution already contained in the population. The parameters α and β, which control the heuristic bias during initialization and mutation respectively, were both set to 1.5.

7 Empirical Comparisons

The proposed approach was tested on problem instance sets of different characteristics. As already indicated by Knowles

Table 1: *Average results (quality gains over d-Prim [18] in %) on hard problem instances of two Prüfer-coded EAs (F-EA and P-EA) [13], problem space search (PSS) [13], simulated annealing (SA) [13], branch-and-bound terminated after 10min CPU-time (B&B) [13], the weight-coded EA (W-EA) [23], and the new EA employing the edge-set representation (S-EA).*

Problem	$\|V\|$	d	F-EA	P-EA	PSS	SA	B&B	W-EA			S-EA		
			avg.	avg.	avg.	avg.	avg.	avg.	best	time	avg.	best	time
SHRD150	15	3	13.66	15.07	16.62	14.93	**18.03**	14.20	16.76	3.1s	**18.03**	18.03	1.2s
		4	10.83	0.39	12.99	11.61	14.76	11.42	11.42	3.0s	**15.35**	15.35	1.2s
		5	4.00	-1.07	**9.60**	9.07	**9.60**	3.53	9.33	2.9s	**9.60**	9.60	1.2s
SHRD200	20	3	11.32	5.38	10.91	10.43	10.91	12.29	12.68	5.1s	**12.43**	12.60	1.4s
		4	6.82	0.80	7.05	5.57	7.05	8.50	8.75	4.9s	**8.78**	8.86	1.5s
		5	6.28	1.46	7.30	7.74	7.30	7.96	8.03	4.8s	**8.44**	8.47	1.4s
SHRD250	25	3	13.07	13.41	15.40	14.73	15.40	16.51	16.91	7.8s	**16.75**	17.34	1.6s
		4	4.84	1.59	6.79	5.56	6.79	6.83	7.37	7.4s	**7.69**	7.80	1.6s
		5	5.37	5.92	6.74	5.19	8.29	**9.01**	9.02	7.2s	**9.01**	9.02	1.6s
SHRD300	30	3	6.51	6.51	11.27	9.53	11.27	**12.50**	12.78	10.8s	12.17	12.78	1.9s
		4	7.30	3.79	10.58	8.45	10.58	**11.76**	11.92	10.4s	10.80	11.87	1.9s
		5	2.18	0.19	4.74	2.50	4.74	**5.77**	5.93	10.1s	4.79	5.74	1.9s
Total average:			*7.68*	*4.45*	*10.00*	*8.78*	*10.39*	*10.02*	*10.91*	*6.5s*	*11.15*	*11.46*	*1.5s*

and Corne [12] and Krishnamoorthy et al. [13], Euclidean and purely randomly generated instances are usually easy to solve. As Knowles and Corne's K-EA and the weight-coded approach W-EA [23], but in contrast so several Prüfer-coded EAs [13, 26], the new EA with edge-set encoding had no problem in identifying optimal or the best-known solutions for these easy problem instances within short time. To point out the differences of the considered EAs and other approaches, we present here results for particularly demanding problem instances with high-degree unconstrained spanning trees and low-cost edges that mislead simple heuristics.

Table 1 shows results for the "structured hard" (SHRD) problem instance set of [13]. The number of vertices ranges from 15 to 30, the maximum degree was set to 3, 4, and 5. The solution quality is measured by the relative difference between the final objective value C obtained by a specific approach and the objective value $C_{d-\mathrm{Prim}}$ of the solution found by the simple d-Prim heuristic[1] [18] in percent:

$$\textit{quality gain} = (C_{d-\mathrm{Prim}} - C)/C_{d-\mathrm{Prim}} \cdot 100\% \; .$$

In other words, we use d-Prim as a reference algorithm and calculate relative quality improvements for the other approaches; larger values indicate better results.

The results for the two Prüfer-coded EA-variants F-EA and P-EA (they differ in the handling of infeasible solutions violating degree constraint, the crossover method, the replacement strategy, and the termination criterion), problem space search PSS (an EA with a similar basic idea as W-EA), simulated annealing SA, and branch-and-bound B&B are adopted from [13] and printed for reference purposes only.

[1]The first vertex is used as starting point. If two or more vertices can be concatenated to the partially built spanning tree via edges having the same costs, the vertex labeled with the smallest number is always prefered.

Note that branch-and-bound, which is in general an exact technique, was in these cases used as a heuristic: Since complete runs would have been too time-demanding, each run had been terminated after 10min CPU-time, and the best solution found so far was reported as final solution.

With the weight-coded EA and the new approach based on the edge-set representation (S-EA), 20 independent runs were performed for each problem instance and maximum degree $d \in \{3, 4, 5\}$. Each run terminated after 10,000 evaluations. Besides average gains, the gains of the best runs and average CPU-times in seconds are reported in Table 1.

For each problem instance the largest obtained average gains are printed bold. Nearly always, S-EA performed best. Only for the instances with 30 vertices, W-EA found solutions with slightly larger gains. Due to its computationally efficient variation operators, S-EA is particularly faster than W-EA.

Table 2 shows results for larger "misleading" problem instances M1 to M9 from Knowles and Corne [12]. In addition, we generated ourself even larger, equivalently structured problem instances M10 to M12 with up to 500 vertices (124,750 edges). The results included in the table for the dual-simplex heuristic from Boldon et al. [3] (DS) and Knowles and Corne's EA (K-EA) are adopted from [12] for reference purposes. Note that DS cannot compete with the other approaches; it gives in case of M3, M5, and M6 even worse results than d-Prim, which is indicated by the negative quality gains. While W-EA is superior to K-EA in most cases, S-EA consequently gives the best results. Note also the generally small differences between average and best gains of S-EA, which indicate high confidence in identifying high quality solutions. Furthermore, S-EA is in all cases substantially faster than W-EA (for M12 over a factor of 100!). For

Table 2: *Average results (quality gains over d-Prim [18] in %) on large, misleading problems with maximum degree $d = 5$ of the dual simplex heuristic (DS) [3], the EA of Knowles and Corne (K-EA) [12], the weight-coded EA (W-EA) [23], and the new EA employing the edge-set representation (S-EA).*

Problem	$\|V\|$	DS	K-EA		W-EA			S-EA		
		avg.	avg.	best	avg.	best	time	avg.	best	time
M1	50	24.14	27.59	36.09	42.76	43.45	25.7s	**43.59**	43.59	2.6s
M2	50	14.23	33.22	43.94	48.63	50.19	25.8s	**50.59**	50.59	2.6s
M3	50	-11.51	26.98	32.93	29.25	32.93	25.6s	**33.33**	33.34	2.6s
M4	100	17.81	28.89	34.13	39.67	40.84	99.1s	**42.21**	42.41	4.9s
M5	100	-14.03	31.25	39.48	47.22	48.68	99.2s	**49.50**	49.66	4.8s
M6	100	-16.92	26.51	29.85	46.04	47.84	98.4s	**49.02**	49.21	4.8s
M7	200	18.43	25.72	30.09	31.54	33.00	400.9s	**34.68**	34.82	10.0s
M8	200	9.74	25.43	32.08	42.64	43.85	401.8s	**44.73**	45.81	10.1s
M9	200	5.88	24.09	25.42	24.53	24.97	389.1s	**25.56**	25.61	10.3s
Total average:		*5.31*	*27.74*	*33.78*	*39.14*	*40.64*	*174.0s*	*41.47*	*41.67*	*5.9s*
M10	300	–	–	–	10.66	13.26	964.3s	**26.85**	27.16	15.9s
M11	400	–	–	–	22.01	24.34	1835.7s	**37.00**	38.00	22.5s
M12	500	–	–	–	14.98	16.25	3092.0s	**31.29**	34.14	29.7s

W-EA, the CPU-time increases dramatically when the problem is getting larger, since the computational complexity of the decoding is $O(|V|^2 \log |V|)$. In contrast, the measured CPU-times of S-EA confirm its linear time complexity.

8 Conclusions and Future Work

We presented a new evolutionary approach for the problem of identifying a degree-constrained minimum spanning tree of a complete, undirected graph. While the genotypic representation of a candidate solution is simply the set of all edges included in the spanning tree, specialized initialization, crossover, and mutation operators are used. Altogether, they fulfill the stated design goals for an effective EA. Most importantly, S-EA produces only feasible candidate solutions and provides substantially stronger locality than most previous approaches, especially the various Prüfer-coded EAs.

Above that, the suggested operators are computationally efficient. The suggested initialization function has time-complexity $O(|V|^2)$, and via selection, crossover, and mutation, a new, meaningful candidate solution can always be created and evaluated in $O(|V|)$ time. This distinguishes S-EA from the also well working, but significantly slower weight-coded EA [23].

In addition, we have described how local heuristics can effectively be incorporated into the initialization, crossover, and mutation operators without increasing the time-complexities. The basic idea of these heuristics is simple and always the same: Low-cost edges are included into a candidate solution with higher probabilities than more expensive edges. These heuristics speed-up the EA essentially and usually lead also to significantly better final results.

We presented empirical results for several hard problem instances from the literature with up to 500 vertices. These results clearly indicate the superiority of S-EA over several other, also non-evolutionary approaches regarding the quality of final solutions and running times. High-quality solutions are typically obtained within few seconds.

It seems possible to adapt the proposed edge-set representation together with its operators also to other network optimization problems. The diameter-constrained MST problem, the capacitated MST problem, and the optimum communication spanning tree problem [8, 19] are examples which we currently investigate.

Acknowledgments

We thank Joshua Knowles and Andreas Ernst for providing us their test data sets. This work is supported by the Austrian Science Fund (FWF) under the grant P13602-INF.

Bibliography

[1] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld. Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. In L. J. Eshelman, ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 470–477. Morgan Kaufmann, 1995.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997.

[3] 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, 1995.

[4] A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, vol. 23, pp. 376–378, 1889.

[5] C.-H. Chu, G. Premkumar, C. Chou, and J. Sun. Dynamic degree constrained network design: A genetic algorithm approach. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds., *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 141–148. Morgan Kaufman, 1999.

[6] S. Even. *Algorithmic Combinatorics*. The Macmillan Company, New York, 1973.

[7] S. 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] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[9] M. L. Gargano, W. Edelson, and O. Koval. A genetic algorithm with feasible search space for minimal spanning trees with time-dependent edge costs. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, H. Iba, and R. L. Riolo, eds., *Genetic Programming 1998: Proceedings of the Third Annual Conference*, p. 495. Morgan Kaufmann, 1998.

[10] J. R. Kim and M. Gen. Genetic algorithm for solving bicriteria network topology design problem. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala, and W. Porto, eds., *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pp. 2272–2279. IEEE Press, 1999.

[11] J. H. Kingston. *Algorithms and Data Structures – Design, Correctness, Analysis*. Addison-Wesley, Singapore, 1990.

[12] J. Knowles and D. Corne. A new evolutionary approach to the degree constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, accepted 1999, in press.

[13] M. Krishnamoorthy, A. T. Ernst, and Y. M. Sharaiha. *Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree*. Tech. rep., CSIRO Mathematical and Information Sciences, Clayton, Australia, submitted to Journal of Heuristics, 1999.

[14] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematics Society*, vol. 7(1), pp. 48–50, 1956.

[15] Y. Li and Y. Bouchebaba. A new genetic algorithm for the optimal communication spanning tree problem. In C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, and M. Schoenauer, eds., *Proceedings of Artificial Evolution: Fifth European Conference*, to appear as LNCS. Springer, Berlin, 1999.

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

[17] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete and Computational Geometry*, vol. 8(3), pp. 265–293, 1992.

[18] S. C. Narula and C. A. Ho. Degree-constrained minimum spanning trees. *Computers and Operations Research*, vol. 7, pp. 239–249, 1980.

[19] C. C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. In D. Schaffer, H.-P. Schwefel, and D. B. Fogel, eds., *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 379–384. IEEE Press, 1994.

[20] C. H. Papadimitriou and U. V. Vazirani. On two geometric problems related to the traveling salesman problem. *Journal of Algorithms*, vol. 5, pp. 231–246, 1984.

[21] P. Piggott and F. Suraweera. Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem. In X. Yao, ed., *Progress in Evolutionary Computation*, LNAI 956, pp. 305–314. Springer, Berlin, 1995.

[22] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.

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

[24] F. Rothlauf and D. Goldberg. Tree network design with genetic algorithms – An investigation in the locality of the Pruefernumber encoding. In S. Brave and A. S. Wu, eds., *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pp. 238–243. Orlando, FL, 1999.

[25] M. Savelsbergh and T. Volgenant. Edge exchanges in the degree-constrained minimum spanning tree problem. *Computers and Operations Research*, vol. 12(4), pp. 341–348, 1985.

[26] G. Zhou and M. Gen. Approach to degree-constrained minimum spanning tree problem using genetic algorithm. *Engineering Design & Automation*, vol. 3(2), pp. 157–165, 1997.