Greedy Heuristics and an Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem

Günther R. Raidl
Institute of Computer Graphics and Algorithms
Vienna University of Technology
1040 Vienna, Austria
raidl@ads.tuwien.ac.at

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
St. Cloud, MN 56301 USA
julstrom@eeyore.stcloudstate.edu

ABSTRACT

Given a connected, weighted, undirected graph G and a bound D, the bounded-diameter minimum spanning tree problem seeks a spanning tree on G of lowest weight in which no path between two vertices contains more than D edges. This problem is NP-hard for $4 \leq D < n - 1$, where n is the number of vertices in G. An existing greedy heuristic for the problem, called OTTC, is based on Prim's algorithm. OTTC usually yields poor results on instances in which the triangle inequality approximately holds; it always uses the lowest-weight edges that it can, but such edges do not in general connect the interior nodes of low-weight bounded-diameter trees. A new randomized greedy heuristic builds a bounded-diameter spanning tree from its center vertex or vertices. It chooses each next vertex at random but attaches the vertex with the lowest-weight eligible edge. This algorithm is faster than OTTC and yields substantially better solutions on Euclidean instances. An evolutionary algorithm encodes spanning trees as lists of their edges, augmented with their center vertices. It applies operators that maintain the diameter bound and always generate valid offspring trees. These operators are efficient, so the algorithm scales well to larger problem instances. On 25 Euclidean instances of up to 1 000 vertices, the EA improved substantially on solutions found by the randomized greedy heuristic.

Keywords

Bounded-diameter spanning tree, edge-list encoding, greedy heuristics, randomized heuristics, local improvement.

1. INTRODUCTION

In a tree, the *eccentricity* of a vertex v is the maximum number of edges in the tree from v to any other vertex. The *diameter* of a tree is the maximum eccentricity of its vertices, thus the maximum number of edges on any path in the tree. The *center* of a tree is the single vertex (if the tree's diameter is even) or the two connected vertices (if the diameter is odd) of minimum eccentricity [4, p. 113] [9]. Given a connected, undirected graph $G = (V, E)$ on $n = |V|$ vertices and an integer bound $D \geq 2$, a *bounded-diameter spanning tree* (BDST) is a spanning tree $T \subseteq E$ on G whose diameter does not exceed D. Figure 1 shows BDSTs on $n = 19$ vertices of diameter $D = 4$ and $D = 5$ and their centers.

If weights $w(e) \geq 0$ are associated with each edge $e \in E$, a *bounded-diameter minimum spanning tree* (BDMST) is a BDST of minimum weight $w(T) = \sum_{e \in T} w(e)$. Such trees are also called *diameter-constrained minimum spanning trees* and *shallow-light spanning trees.*

The search for a BDMST finds applications in such areas as telecommunications network and linear lightwave network design [2], distributed system design when considering mutual exclusion [17], and bit-compression for information retrieval [3]. A special case occurs when $D = 4$ and a root vertex is specified. This is called the *2-hop problem* [5]

The BDMST problem is NP-hard for $4 \leq D < n - 1$ [7, p. 206], and no exact algorithm is known that identifies optimum solutions on large problem instances. Kortsarz and Peleg [12] have shown that, unless $P = NP$, no polynomial-time algorithm can be guaranteed to find a tree whose weight is within $\log n$ of optimum. Thus we turn to heuristics to seek good approximate solutions.

Kortsarz and Peleg [11, 12] described an algorithm that identifies a BDST whose weight exceeds the optimum by a factor that is $O(D \log n)$. The algorithm combines a greedy

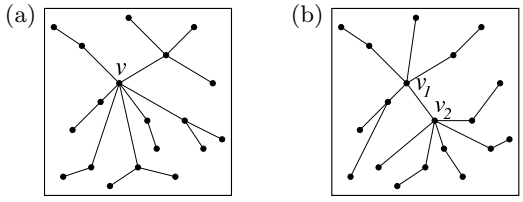

Figure 1: (a) A tree of even diameter $D = 4$ on $n = 19$ vertices; the vertex v is its center, and (b) a tree of odd diameter $D = 5$ on the same vertices; v_1 and v_2 together form its center.

SAC 2003, Melbourne, Florida, USA.

heuristic and exhaustive search. Unfortunately, its time is $O(n^{2D})$, so it is suitable only for small problem instances.

Abdalla, Deo, and Gupta [1, 6] described two heuristics for the BDMST problem, without approximation guarantees. One iteratively refines an unconstrained minimum spanning tree. This heuristic is computationally expensive and does not always find a feasible solution (even on a complete graph), particularly when D is small. The second greedily constructs a low-weight BDST, in imitation of Prim's algorithm for unconstrained minimum spanning trees [14], though its greediness leads it away from the best trees, as Section 2 below describes.

A new, more effective greedy heuristic also imitates Prim but begins at the tree's center and chooses each vertex to connect to the spanning tree at random. Section 3 describes this heuristic.

An evolutionary algorithm (EA) for the BDMST problem encodes candidate spanning trees as lists of their edges, with their center vertex or vertices specified. Crossover imitates Prim's algorithm, using parental edges unless none are feasible. Four different mutation operators take advantage of the structure imposed on trees by the diameter constraint.

The greedy heuristic of Abdalla, Deo, and Gupta, the randomized greedy heuristic, and the evolutionary algorithm were compared on 25 instances of the BDMST problem of 50 to 1 000 vertices with diameter bounds from five to 25. The randomized algorithm returned significantly lower-weight trees than did the completely greedy heuristic, and the EA identified the best trees.

The following sections of the paper describe the greedy heuristic of Abdalla, Deo, and Gupta and how it fails; the randomized greedy heuristic; the EA's coding and operators; the EA's structure and parameters; and the comparison of the three algorithms.

2. A COMPLETELY GREEDY HEURISTIC

Prim's algorithm [14] begins at an arbitrary start vertex and builds an unconstrained minimum spanning tree by repeatedly appending the lowest-weight edge that connects a vertex in the tree with one not in the tree. The diameter of the spanning tree at each step can always be known; modifying Prim's algorithm to accommodate the diameter bound yields greedy heuristics for the BDMST problem on complete graphs. One such has been described by Abdalla, Deo, and Gupta [1]. They called it *One Time Tree Construction* (OTTC).

Beginning at a specified start vertex, OTTC repeatedly extends the growing tree with the lowest-weight edge between a tree vertex and a non-tree vertex whose inclusion does not violate the diameter bound. The algorithm keeps track of the path lengths between and the eccentricities of the vertices in the tree. Appending each new edge in general changes several of these values, so that updating the algorithm's data structures requires, in the worst case, time that is $O(n^2)$. This step is repeated $n-1$ times, so the algorithm's time is $O(n^3)$. The quality of the tree the algorithm identifies depends heavily on the start vertex. To identify a low-weight BDST, the algorithm should be run starting from each vertex in the target graph in turn. The time of the entire process is then $O(n^4)$. On massively parallel hardware, Deo and Abdalla [6] obtained bounded-diameter spanning trees on complete, randomly created graphs on up to 1 000 vertices.

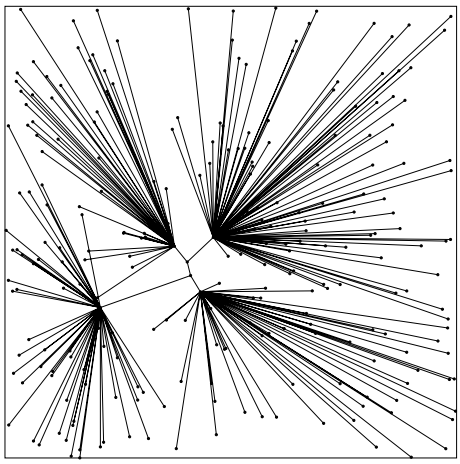

Figure 2: The best BDST of diameter $D = 5$ the OTTC heuristic finds for an Euclidean instance with $n = 250$ vertices.

2.1 Greediness Misleads OTTC

When OTTC is applied to problem instances whose vertices are points in Euclidean space and whose edge weights are the distances between the points, the heuristic in general yields spanning trees whose weights are much larger than minimum. This is particularly true when D is small compared to n, as Figure 2 illustrates.

The figure shows a spanning tree, identified by OTTC, of diameter $D = 5$ on $n = 250$ points in the unit square. Short edges connect only a few vertices near the center of this tree. The remaining vertices connect via longer edges to this core, forming a star-like structure. Let the *backbone* of a tree be the subgraph induced by all the tree's non-leaf vertices. In a low-weight BDST, the backbone is longer, so that leaves can connect to it via shorter edges.

A good heuristic for the BDMST problem will not prefer lower-weight edges as it builds the backbone. OTTC always uses the lowest-weight edge available, and so is misled. This observation holds for almost all BDMST problem instances in which the triangle inequality is approximately satisfied, thus for most applications of the problem.

3. A RANDOMIZED GREEDY HEURISTIC

A new *randomized greedy heuristic* for the BDMST problem on complete graphs avoids the pitfall the previous section described. Rather than always extending the tree with the nearest unconnected vertex, it chooses each next vertex at random and connects this vertex to the tree via the lowest-weight edge that maintains the diameter constraint.

The algorithm also differs from OTTC in that it begins by fixing the tree's center. The start vertex v_0 is chosen at random. If D is even, v_0 is the center. If D is odd, another

```
T ← ∅;
v0 ← a random vertex from V;
U ← V − {v0};
C ← {v0};
depth[v0] ← 0;
if D is odd then
  v1 ← a random vertex from U;
  T ← {(v0, v1)};
  U ← U − {v1};
  C ← C ∪ {v1};
  depth[v1] ← 0;
while U ≠ ∅ do
  v ← a random vertex from U;
  u ← vertex from C with smallest w((u, v));
  T ← T ∪ {(u, v)};
  U ← U − {v};
  depth[v] ← depth[u] + 1;
  if depth[v] < ⌊D/2⌋ then
    C ← C ∪ {v};
return T.
```

Figure 3: A randomized greedy heuristic that builds a low-weight BDST with diameter bound D on a complete graph with vertices V.

vertex v_1 is chosen at random and $\{v_0, v_1\}$ is the center; the edge joining them is the first in the tree. Instead of maintaining the eccentricities of and path lengths between vertices, the randomized heuristic stores the *depth* of each connected vertex: the number of edges on the path from it to the center. This value is set when a vertex joins the tree and does not subsequently change. No vertex may have a depth greater than $\lfloor D/2 \rfloor$; otherwise the diameter constraint is violated or v_0 (v_1) is displaced from the center.

Figure 3 presents a sketch of the randomized heuristic. In it, U is the set of unconnected vertices and C is the set of tree-vertices of depth less than $\lfloor D/2 \rfloor$, thus the set of vertices to which new edges may be connected. In each iteration, the algorithm randomly picks a vertex v from U and determines its nearest vertex u in C. The edge (u, v) is added to the tree. The vertex v is removed from U and, if its depth is less than $\lfloor D/2 \rfloor$, added to C. The depth of a new vertex is always one more than the depth of the vertex to which it connects.

Identifying the vertex $u \in C$ that is nearest to v requires time that is $O(|C|) = O(n)$. This operation is repeated $n-1$ or $n-2$ times, so the time of the algorithm is $O(n^2)$, a factor of n less than that of OTTC. Running the randomized greedy heuristic n times and taking the best solution, as with OTTC, is then $O(n^3)$. Tests in Section 6 demonstrate that the randomized greedy heuristic yields substantially better solutions on Euclidean problem instances than does OTTC.

4. AN EVOLUTIONARY APPROACH

Evolutionary algorithms have proven effective on several hard spanning tree problems, and much work was been done on representing spanning trees for evolutionary search (*e.g.* [13, 15, 18]). Recent studies have indicated the general usefulness of representing spanning trees directly as lists of their edges and applying operators that always yield feasible trees [8, 10, 16]. We adopt the edge-list coding here, but augment each chromosome with the center vertex or vertices of the spanning tree it represents.

```
F1 ← edges appearing in both parents;
F2 ← edges appearing in only one parent;
T ← ∅;

// Determine the center:
if D is even then
  v0 ← the first parent's v0;
  U ← V − {v0};
  C ← {v0};
  depth[v0] ← 0;
else (D is odd)
  (v0, v1) ← two random vertices from the union
             of the parents' centers;
  T ← {(v0, v1)};
  U ← V − {v0, v1};
  C ← {v0, v1};
  depth[v0] ← 0;
  depth[v1] ← 0;
A1 ← all edges from F1 incident to the center;
A2 ← all edges from F2 incident to the center;

// Add other nodes iteratively:
while U ≠ ∅ do
  if A1 ≠ ∅ then
    pick an edge (u, v) ∈ A1 at random;
    A1 ← A1 − {(u, v)};
  else if A2 ≠ ∅ then
    pick an edge (u, v) ∈ A2 at random;
    A2 ← A2 − {(u, v)};
  else
    pick u ∈ C at random;
    pick v ∈ U at random;
  if v ∈ U then
    T ← T ∪ {(u, v)};
    U ← U − {v};
    depth[v] ← depth[u] + 1;
    if depth[v] < ⌊D/2⌋ then
      A1 ← A1 ∪ all edges from F1 incident to v;
      A2 ← A2 ∪ all edges from F2 incident to v;
      C ← C ∪ {v};

return T.
```

Figure 4: The recombination operator.

The fitness of a chromosome is the total weight of its tree, which can be found, by scanning the edge-list, in time that is $O(n)$. The diameter bound renders the initialization and variation operators of the prior EAs inapplicable or ineffective. The following sections describe operators appropriate to the BDMST problem on complete graphs.

4.1 Recombination

An EA's recombination operator should provide strong heritability. Here, this means that the tree produced by recombining two parent trees should consist mostly of parental edges. It is also beneficial to favor edges that are common to both parents [16].

Figure 4 presents a sketch of a recombination operator, based on the randomized greedy heuristic, that satisfies

these goals. The operator selects a center vertex or vertices at random from the parental centers. Like the heuristic, it maintains a set U of unconnected vertices and a set C of tree vertices of depth less than $\lfloor D/2 \rfloor$. Edges may be attached to the vertices in C. The operator also maintains two sets A_1 and A_2 of edges incident to vertices in C that may be used to extend the tree. A_1 contains edges found in both parents, and A_2 contains edges found in just one.

To extend a partial tree, the operator chooses an edge at random from A_1 or—if A_1 is empty—from A_2. If A_2 is also empty, it creates an edge joining a random vertex in C and a random vertex in U. Only in this last case does recombination introduce non-parental edges into an offspring tree.

Temporary adjacency lists representing the parent trees allow quick identification of the parental edges adjacent to a vertex and thus the quick updating of A_1 and A_2. The recombination operator can run in time that is $O(n)$.

EA's for the degree-constrained minimum spanning tree problem and the traveling salesman problem have been more effective when their recombination operators probabilistically favored low-weight edges [16]. Here, however, such bias was not helpful because, as described in Section 2.1, a low-weight tree's backbone does not in general consist of low-weight edges.

4.2 Mutation Operators

The EA applies four mutation operators, which complement each other to search the space of BDSTs. All four always generate valid trees. The last two improve a solution locally and significantly speed up the search for low-weight trees.

4.2.1 Edge-delete mutation

This operator removes a random edge from the parent tree. (When D is odd, this edge cannot be the one that connects the two center vertices.) Using the parent's center, it then builds a new BDST as in recombination, using parental edges when possible and new edges when necessary. The operator runs in time that is $O(n)$.

4.2.2 Center-move mutation

Edge-delete mutation preserves the parent tree's center; this operator changes it. It chooses at random a vertex v_a adjacent to the parent's center. If D is even, v_a becomes the center of the offspring tree; if D is odd, v_a and one of the parental center vertices form the offspring's center. The operator builds the offspring from this center as in edge-delete mutation. The time of center-move mutation is also $O(n)$.

4.2.3 Greedy-edge-replace mutation

This operator is a greedy version of edge-delete mutation. It too begins by removing a random edge from the parent tree (except for the center edge when D is odd). This disconnects a subtree rooted at a vertex r. Let S be the set of vertices in this subtree, including r. The operator reconnects the tree with the lowest-weight edge (u, r) with $u \in V - S$ that maintains the diameter bound.

The operator begins with a depth-first search that identifies the depth of each vertex in the parent tree. It also identifies the height h of the disconnected subtree. An edge (u, r) may be used to reconnect the tree if $\mathit{depth}[u] + h < \lfloor D/2 \rfloor$. The operator's time is $O(n)$.

4.2.4 Subtree-optimize mutation

This operator chooses a vertex r at depth $\lfloor D/2 \rfloor - 1$ at random in the parent tree and optimally rearranges the subtree rooted at r. Let S be the set of vertices in this subtree and p the predecessor of r on its path to the center. The operator tries each vertex v in S as the root of the subtree, connecting the vertices in $S - \{v\}$ to v. The weight of the subtree with root v is

$$w_S(v) = w((p, v)) + \sum_{u \in S - \{v\}} w((v, u)) \,,$$

and the subtree that minimizes this weight becomes part of the offspring BDST.

Finding the weight of a subtree requires time that is $O(|S|)$ and there are $|S|$ subtrees to examine, so the time of subtree-optimize mutation is $O(n + |S|^2)$. Since the number of vertices in a subtree is usually small, this operator is not significantly less efficient than the other three mutations.

5. THE EA FRAMEWORK

The representation and operators that the last section described were implemented in a conventional steady-state evolutionary algorithm. The EA applies the randomized greedy heuristic to generate candidate solutions for its initial population, so it starts with a diverse collection of relatively good solutions. It selects parents in tournaments with replacement. Recombination generates some offspring, but every offspring is mutated with one of the four mutation operators. Each offspring replaces the worst solution in the population, except that duplicates are discarded.

For the tests the next section describes, the EA's parameters were set according to experience gained from preliminary tests. In particular, its population contained 400 chromosomes, the size of its selection tournaments was three, and it applied recombination with a probability of 60%. It applied edge-delete and center-move mutation with probabilities of 20%, and greedy-edge-replace and subtree-optimize mutation with probabilities of 30%. The EA terminated when the population's best solution did not improve over 100 000 new chromosomes.

6. TESTS

The OTTC heuristic, the randomized greedy heuristic (RGH), and the evolutionary algorithm (EA) were compared on 25 Euclidean instances of the BDMST problem, five instances each of $n = 50, 100, 250, 500$, and 1 000 vertices. The instances are found in Beasley's OR-library[1], listed as instances of the Euclidean Steiner problem. The library contains fifteen instances of each size; we used the first five from each group. The instances consist of points in the unit square. We consider the points to be the vertices of complete graphs whose edge weights are the Euclidean distances between the points. When $n = 50$, the diameter bound D is set to 5; when $n = 100$, $D = 10$; when $n = 250$, $D = 15$; when $n = 500$, $D = 20$; and when $n = 1\,000$, $D = 25$.

[1] http://mscmga.ms.ic.ac.uk/info.html

Table 1: Results of OTTC, the randomized greedy heuristic (RGH), and the evolutionary algorithm (EA) on 25 Euclidean instances of the BDMST problem.

Instance			OTTC			RGH			EA				
n	D	nr.	best	mean	stddev	best	mean	stddev	best	mean	stddev	iterations	time [s]
50	5	1	13.84	21.18	4.82	9.34	12.82	2.48	7.60	7.93	0.22	33 947	8.6
50	5	2	14.20	18.15	2.75	8.98	11.56	1.56	7.68	7.87	0.14	36 403	9.2
50	5	3	12.53	18.06	2.79	8.76	11.54	1.90	7.24	7.51	0.15	27 919	7.1
50	5	4	11.04	15.81	2.78	7.47	10.57	1.66	6.59	6.75	0.15	31 382	7.9
50	5	5	13.04	17.77	2.32	8.79	10.91	1.61	7.32	7.49	0.09	349 24	8.9
100	10	1	18.79	29.01	6.53	9.35	10.77	0.81	8.00	8.30	0.12	189 026	105.0
100	10	2	17.69	25.24	4.54	9.41	10.80	0.81	8.10	8.41	0.16	205 891	114.6
100	10	3	20.16	24.25	4.35	9.75	11.25	0.90	8.22	8.61	0.19	176 043	97.0
100	10	4	17.64	26.40	6.27	9.55	11.03	0.89	8.27	8.57	0.17	163 142	90.3
100	10	5	16.63	28.63	5.65	9.78	11.36	1.06	8.48	8.72	0.15	164 651	90.5
250	15	1	42.09	68.50	14.54	15.14	16.51	0.69	12.93	13.36	0.19	471 803	809.0
250	15	2	52.38	71.21	10.67	15.20	16.33	0.67	12.86	13.25	0.20	466 047	796.7
250	15	3	43.40	65.38	13.63	15.08	16.19	0.56	12.69	13.06	0.20	464 618	796.8
250	15	4	46.76	72.45	10.52	15.49	16.77	0.62	13.22	13.65	0.19	442 446	758.3
250	15	5	39.54	64.63	13.68	15.42	16.53	0.58	13.02	13.40	0.19	497 450	856.9
500	20	1	90.67	147.49	33.85	21.72	22.86	0.51	18.33	18.77	0.29	527 659	2 140.0
500	20	2	85.34	141.09	29.96	21.46	22.52	0.46	18.17	18.60	0.19	652 009	2 672.2
500	20	3	69.37	148.40	32.66	21.51	22.78	0.50	18.33	18.76	0.28	504 315	2 050.6
500	20	4	89.34	144.11	30.81	21.82	22.85	0.47	18.32	18.74	0.18	654 871	2 658.5
500	20	5	85.35	145.69	38.78	21.37	22.52	0.51	17.80	18.40	0.28	648 148	2 673.7
1 000	25	1	184.49	320.80	74.86	30.97	32.19	0.41	26.13	26.72	0.31	501 047	4 535.3
1 000	25	2	189.50	321.37	69.94	30.90	32.05	0.42	26.14	26.58	0.27	464 397	4 191.6
1 000	25	3	184.24	294.56	68.95	30.69	31.77	0.42	25.47	26.21	0.29	554 601	5 018.4
1 000	25	4	192.30	312.01	73.06	30.93	32.18	0.43	26.13	26.65	0.22	508 038	4 589.1
1 000	25	5	193.02	310.08	56.18	30.85	31.93	0.42	25.91	26.29	0.27	647 818	5 911.8

On each instance, OTTC was run n times, starting from each vertex in turn. RGH was also run n times, with random start vertices. The EA was run 50 times on each instance. Table 1 summarizes the results of these trials. For each instance, the table lists n, D, and the instance's number. For OTTC and RGH, it lists the weight of the best BDST found, the average weight, and the standard deviation of the weights. For the EA, it lists the weight of the best tree found in 50 trials, the mean of the trial's smallest weights, and the mean number of iterations and mean CPU-time (in seconds on a Pentium-III/800 Mhz PC) to a trial's best tree.

The results are consistent across all the instances, and differences in the mean weights of the three algorithms' best trees are significant at the 1% level. RGH identified substantially better solutions with much smaller standard deviations than did OTTC. The BDSTs OTTC identified were up to five times longer than those RGH found. OTTC's greediness in its selection of backbone vertices accounts for its poor performance, as we observed in Section 2.1.

The EA's solutions are on average never worse than those of RGH, since the EA applies RGH to build its initial population. The mean weights of the EA's trees were 10% to 15% (on average 13%) smaller than the weights of the best trees RGH found on the same instances. The CPU-times and numbers of iterations show that the EA scales well to larger instances. The mean time per iteration increases approximately linearly with the number of vertices.

Figure 5 shows the lowest-weight trees the three algorithms found on the second instance with $n = 250$ and $D = 15$. They clearly illustrate the advantage of RGH over OTTC and the superiority of the EA to both.

7. CONCLUSION

Given a connected, weighted, undirected graph G and a bound D, the bounded-diameter minimum spanning tree problem seeks a spanning tree on G of lowest weight in which no path between two vertices contains more than D edges. A Prim-inspired greedy heuristic for the problem identifies high-weight, though valid, trees; the backbone of a low-weight tree of bounded diameter does not in general consist of the low-weight edges the heuristic chooses.

A randomized greedy heuristic connects vertices to the tree in random order, but each with the valid edge of lowest weight. This simpler and more efficient heuristic identifies trees of much lower weight on Euclidean problem instances than does the completely greedy heuristic.

An evolutionary algorithm encodes spanning trees as lists of their edges, augmented with their center vertices, and applies operators that maintain the diameter bound; their offspring always represent valid trees. The recombination operator imitates Prim's algorithm and uses parental edges whenever possible. Of four mutation operators, two perform local improvements. All the variation operators can be implemented to require only linear or near-linear time, so the EA scales well to larger problem instances. On instances of

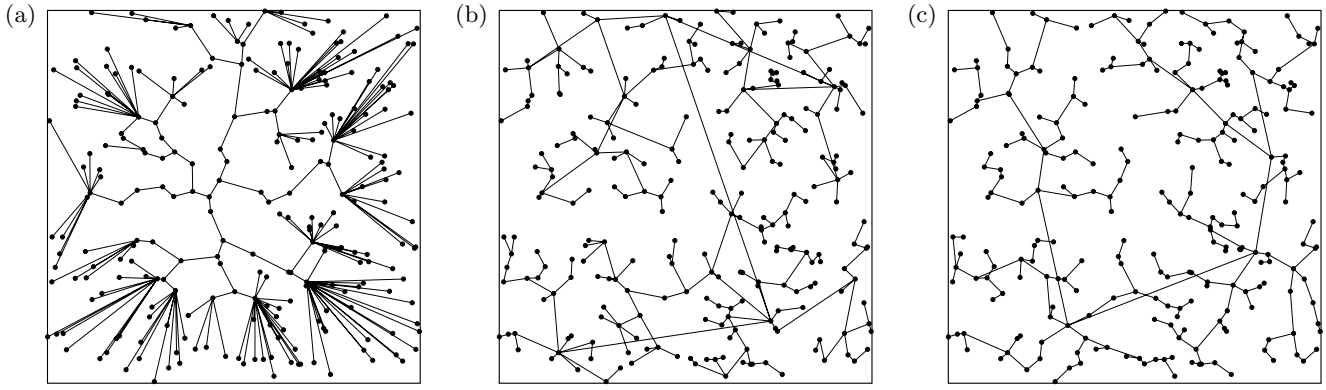

Figure 5: The lowest-weight BDSTs found by the three algorithms on the second problem instance with $n = 250$ and $D = 15$: (a) OTTC, weight: 52.38; (b) randomized greedy heuristic, weight: 15.20; and (c) evolutionary algorithm, weight: 12.86.

up to 1 000 points, the EA identified lower-weight trees than did either greedy heuristic. In particular, it always improved significantly the solutions provided for its initial population by the randomized greedy heuristic.

Acknowledgments

This work is supported by the Austrian Science Fund (FWF) under grant P13602–INF.

8. REFERENCES

[1] A. Abdalla, N. Deo, and P. Gupta. Random-tree diameter and the diameter constrained MST. *Congressus Numerantium*, 144:161–182, 2000.

[2] K. Bala, K. Petropoulos, and T. E. Stern. Multicasting in a linear lightwave network. In *IEEE INFOCOM'93*, pages 1350–1358, 1993.

[3] A. Bookstein and S. T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16(4):110–118, 1996.

[4] G. Chartrand and O. Oellermann. *Applied and Algorithmic Graph Theory*. McGraw-Hill, New York, 1993.

[5] G. Dahl. The 2-hop spanning tree problem. Technical Report 250, University of Oslo, 1997.

[6] N. Deo and A. Abdalla. Computing a diameter-constrained minimum spanning tree in parallel. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, *Algorithms and Complexity*, number 1767 in LNCS, pages 17–31. Springer, Berlin, 2000.

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

[8] J. Gottlieb, B. A. Julstrom, F. Rothlauf, and G. R. Raidl. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In L. Spector, E. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 343–350. Morgan Kaufmann, 2000.

[9] C. Jordan. Sur les assemblages des lignes. *J. Reine Angew. Math.*, 70:185–190, 1869.

[10] B. A. Julstrom. Encoding rectilinear Steiner trees as lists of edges. In G. Lamont, J. Carroll, H. Haddad, D. Morton, G. Papadopoulos, R. Sincovec, and A. Yfantis, editors, *Proceedings of the 16th ACM Symposium on Applied Computing*, pages 356–360. ACM Press, 2001.

[11] G. Kortsarz and D. Peleg. Approximating shallow-light trees. In *Proceedings of the 8th Symposium on Discrete Algorithms*, pages 103–110, 1997.

[12] G. Kortsarz and D. Peleg. Approximating the weight of shallow Steiner trees. *Discrete Applied Mathematics*, 93:265–285, 1999.

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

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

[15] G. R. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In C. Fonseca, J.-H. Kim, and A. Smith, editors, *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 104–111. IEEE Press, 2000.

[16] G. R. Raidl and B. A. Julstrom. Edge-sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 2003. To appear.

[17] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

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