Greedy Heuristics for the Bounded Diameter Minimum Spanning Tree Problem

BRYANT A. JULSTROM
St. Cloud State University

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 minimum weight among the trees in which no path between two vertices contains more than D edges. In Prim's algorithm, the diameter of the growing spanning tree can always be known, so it is a good starting point from which to develop greedy heuristics for the bounded diameter problem. Abdalla, Deo, and Gupta described such an algorithm. It imitates Prim's algorithm but avoids edges whose inclusion in the spanning tree would violate the diameter bound. Running the algorithm from one start vertex requires time that is $O(n^3)$.

A modification of this approach uses the start vertex as the center of the spanning tree (if D is even) or as one of the two center vertices (if D is odd). This yields a simpler algorithm whose time is $O(n^2)$. A further modification chooses each next vertex at random rather than greedily, though it still connects each vertex to the growing tree with the lowest-weight feasible edge. On Euclidean problem instances with small diameter bounds, the randomized heuristic is superior to the two fully greedy algorithms, though its advantage fades as the diameter bound grows. On instances whose edge weights have been chosen at random, the fully greedy algorithms outperform the randomized heuristic.

Categories and Subject Descriptors: G.2.2 [**Graph Theory**]: Graph Algorithms; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Bounded-diameter minimum spanning tree problem, diameter, greedy heuristics, spanning trees

ACM Reference Format:

Julstrom. B. A. 2009. Greedy heuristics for the bounded diameter minimum spanning tree problem. ACM J. Exp. Algor. 14, Article 1.1 (February 2009), 14 pages DOI 10.1145/1498698.1498699 http://doi.acm.org/10.1145/1498698.1498699

1. INTRODUCTION

The diameter of a tree is the number of edges on a longest path within it. Given a connected, weighted, undirected graph G and a bound D, the bounded

Author's address: B. A. Julstrom, Department of Computer Science, St. Cloud State University, St. Cloud, MN 56301 USA; email: julstrom@stcloudstate.edu.

diameter minimum spanning tree (BDMST) problem seeks a spanning tree on G with diameter no more than D and minimum weight among all such trees. A greedy heuristic for the problem, due to Abdalla, Deo, and Gupta [2000] [Deo and Abdalla 2000], imitates Prim's algorithm [1957]. It begins at an arbitrary vertex and builds a low-weight bounded diameter spanning tree (BDST) by repeatedly appending to the tree the lowest-weight edge whose inclusion does not violate the diameter bound.

I propose two faster greedy heuristics that employ similar strategies but use the start vertex as the center of the tree (if D is even) or as one of two vertices of the center (if D is odd). One heuristic always extends the growing tree with the most economical vertex and edge that do not place the vertex too far from the center. The second chooses the start vertex and each subsequent vertex at random rather than greedily, though each connection to the growing tree is still the most economical given the chosen vertex.

These three algorithms are compared on 960 instances of the BDMST tree problem, half Euclidean and half whose edge weights have been chosen at random. On the Euclidean instances with small diameter bounds, the randomized heuristic is decisively superior to the two fully greedy algorithms, but this ranking is reversed when the diameter bounds are significant fractions of the diameters of unconstrained minimum spanning trees. On the random-weight instances, the two fully greedy algorithms outperform the randomized heuristic.

The following sections of this article describe the BDMST; describe the greedy heuristic of Abdalla, Deo, and Gupta [2000]; present the center-based heuristic; present the randomized version of the center-based heuristic; and compare the three heuristics on the test problem instances.

2. THE PROBLEM

In an undirected graph, the eccentricity of a vertex v is the maximum of the minimum numbers of edges on the paths from v to every other vertex in the graph. The diameter of a graph is the maximum eccentricity of its vertices, and the center of a graph is the induced subgraph whose vertices have minimum eccentricity.

In a tree, the path between each pair of vertices is unique, so the eccentricity of a vertex v is simply the maximum number of edges in the tree from v to any other vertex. The diameter of a tree is the maximum number of edges on any path between two vertices, and the center of a tree is a single vertex, if the tree's diameter is even, or two adjacent vertices, if the diameter is odd. Figure 1 shows two trees, one of even diameter and one of odd diameter, and their centers.

Given a connected, undirected graph G on n vertices and a bound D between 2 and $n - 1$, a BDST is a spanning tree on G with diameter no more than D. When real-valued weights are associated with G's edges, the weight of a spanning tree on it is the sum of the weights of the tree's edges. Given G and D, the search for a BDST of minimum weight on G is the BDMST problem.

This problem is NP-hard for $4 \leq D < n - 1$ [Garey and Johnson 1979] and thus a suitable target for heuristics. A special case of the problem, related to the heuristics of Sections 4 and 5, is the k-hop problem, which fixes a root vertex

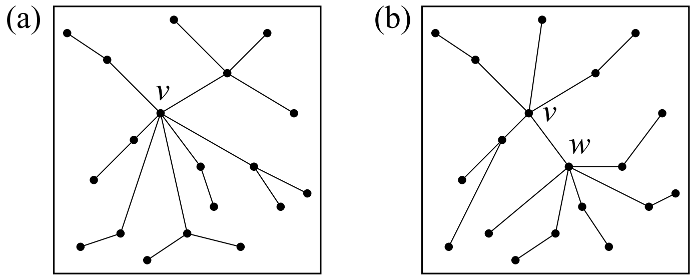

Fig. 1. (a) A tree of even diameter on $n = 19$ vertices; the vertex v is its center, and (b) a tree of odd diameter on the same vertices; v and w together form its center.

and requires that no path from the root to any vertex contain more than k edges [Althaus et al. 2005]. A narrower special case sets $k = 2$ and is called the 2-hop problem [Dahl 1997; Sörensen and Janssens 2002].

The BDMST problem has been addressed in a variety of ways. Achuthan et al. [1994] presented three branch-and-bound algorithms for it and solved instances with up to 100 vertices. Gouveia and Magnanti [2003] described a network flow model that solved instances with up to 100 vertices and 1,000 edges, and dos Santos et al. [2004] extended the methods of Achuthan et al. [1994]. Gruber and Raidl [2005a] described a 0–1 integer linear programming model for the problem.

Researchers have also described heuristics for the BDMST problem. In addition to the greedy algorithm that the next section describes, Abdalla, Deo, and Gupta [2000] presented an iterative refinement algorithm that develops a low-weight BDST from an unconstrained minimum spanning tree. Raidl and Julstrom [2003] developed the greedy heuristic that Section 5 below describes and an evolutionary algorithm that encoded valid bounded diameter trees directly as sets of their edges. Julstrom and Raidl [2003] compared the edge-set-coded evolutionary algorithm with another that represented BDSTs as permutations of their vertices.

Gruber and Raidl [2005b] applied variable neighborhood search to the problem, examining four different neighbor operators, and Gruber, van Hemert, and Raidl [2006] compared neighbor operators for the problem in variable neighborhood search, evolutionary, and ant colony optimization algorithms. Kopinitsch [2006] also described an ant colony algorithm for the problem.

Singh and Gupta [2007] extended greedy constructive heuristics with a local search step that reassesses previous vertex connections after appending each new vertex and made these heuristics faster by maintaining lists of vertices adjacent to each vertex in ascending order of their weights. They also presented an evolutionary algorithm for the problem that encoded candidate trees as permutations of the vertices; to these permutations they applied uniform order-based crossover [Davis 1991] and mutation by swapping, both of two random positions and of the first position and a random position. Gruber and Raidl [2008] applied a heuristic cut technique in a branch-and-cut algorithm applied to an integer linear programming formulation of the problem, and Raidl and Gruber

[2008] presented a hybrid heuristic that combined Lagrangean relaxation with variable neighborhood descent search.

We consider here three constructive, greedy heuristics. Prim's algorithm [1957] 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 yet in the tree. The diameter of the tree at each step can always be known, thus Prim is an appropriate starting point in the development of constructive, greedy heuristics for the BDMST problem.

3. A GREEDY HEURISTIC

Abdalla, Deo, and Gupta [2000] and Deo and Abdalla [2000] described a Prim-based heuristic for the BDMST problem. It begins at an arbitrary vertex and builds a tree by repeatedly including the edge of lowest weight that joins a vertex not yet in the tree to one in the tree without violating the diameter bound. Its authors call this heuristic one-time tree construction (OTTC); the following description is based on theirs.

Call a vertex in the developing spanning tree eligible if appending an edge to it will not violate the diameter bound, and call an edge eligible if it joins an eligible vertex to one not yet in the spanning tree. As in the usual implementation of Prim's algorithm, two one-dimensional arrays hold information about nontree vertices' possible connections to the tree. For each vertex u not yet in the spanning tree, `near[`u`]` is the eligible tree vertex nearest u, and `wnear[`u`]` is the weight of the (eligible) edge connecting u and `near[`u`]`.

Two more arrays hold path lengths and vertices' eccentricities. A tree's diameter is the maximum eccentricity of its vertices, and the heuristic honors the diameter bound by ensuring that no vertex in the tree has eccentricity greater than D. For each vertex u, `ecc[`u`]` is the current eccentricity of u if it is in the spanning tree, -1 otherwise. Joining a vertex to the tree changes the eccentricities of some of its vertices; for each pair of vertices u and v, `dist[`u`][`v`]` is the number of edges on the path connecting u and v if both are in the spanning tree, -1 otherwise. Note that for each vertex u, `ecc[`u`]` is the maximum of the row `dist[`u`][·]`.

All four arrays are updated after the addition of each edge to the spanning tree, thus $n - 1$ times. As Abdalla, Deo, and Gupta [2000] point out, updating `near[·]` and `wnear[·]` requires time that, in the worst case, is $O(n^2)$, so the time of the algorithm is $O(n^3)$. Moreover, the quality of the tree the algorithm identifies depends on the start vertex. To identify a BDMST of low weight, the algorithm should be run starting from each vertex in the target graph in turn. The time of this entire process is $O(n^4)$.

4. STARTING AT THE CENTER

Handler [1978] observed that, in a tree of diameter D, no vertex is more than $D/2$ edges from the tree's center. A faster Prim-based heuristic grows a BDST not from a vertex that may fall anywhere in the resulting structure but from the tree's center. This algorithm does not need to bound each vertex's eccentricity; it suffices to bound each vertex's depth—the number of edges on the path from

```
if $D$ is even
    depth[$v_o$] $\leftarrow 0$ // $v_o$ is the center.
    initialize the arrays near[$\cdot$] and wnear[$\cdot$]
    $T \leftarrow \emptyset$
else // $D$ is odd
    $v_1 \leftarrow$ vertex nearest $v_o$ // $\{v_o, v_1\}$ is the center. (*)
    depth[$v_o$] $\leftarrow 0$
    depth[$v_1$] $\leftarrow 0$
    $T \leftarrow \{(v_o, v_1)\}$
    initialize the arrays near[$\cdot$] and wnear[$\cdot$]

while $|T| < n - 1$
    $v \leftarrow$ vertex not in $T$ with smallest wnear[$v$] (*)
    $T \leftarrow T \cup \{(v,$ near[$v$]$)\}$
    depth[$v$]$\leftarrow 1+$ depth[near[$v$]]
    if depth[$v$]$< D/2$
        for each vertex $u \notin T$
            if d[$u$][$v$] $<$ wnear[$u$]
                near[$u$] $\leftarrow v$
                wnear[$u$] $\leftarrow$ d[$u$][$v$]

return $T$
```

Fig. 2. A sketch of the center-based heuristic CBTC for the bounded diameter minimum spanning tree problem. T is the spanning tree, D is the diameter bound, d[·][·] is the array of weights in the target graph, and v_o is the start vertex. The statements marked with (*) are modified in Section 5.

the tree's center to the vertex—and a tree vertex is eligible if its depth is less than $\lfloor D/2 \rfloor$. Call this algorithm center-based tree construction (CBTC).

CBTC maintains three arrays that hold information about the vertices. As in Prim's algorithm and OTTC, near[u] identifies the eligible tree vertex nearest each nontree vertex u, and wnear[u] is the weight of the edge between u and near[u]. The array depth[v] holds the depth of each tree vertex v.

When the diameter bound D is even, the center of a tree consists of a single vertex, and CBTC uses the start vertex as the center of the spanning tree. The center's depth is zero. The heuristic repeatedly extends the tree with the edge of smallest weight between a vertex v not in the tree and a tree vertex near[v] whose depth is less than $D/2$. The new vertex v is one edge farther from the center, so its depth is depth[near[v]]$+1$.

When D is odd, the center of the tree consists of two vertices, one of which is the start vertex. As the second, CBTC chooses a vertex nearest the start vertex. Both are assigned depth zero. As in the even case, the heuristic repeatedly extends the tree with the edge of smallest weight that joins a vertex not in the tree with a tree vertex whose depth is less than $\lfloor D/2 \rfloor$. Note that $n - 2$ edges are identified in this way rather than the $n - 1$ in the even case; the first edge joins the two vertices of the center. Figure 2 summarizes the center-based heuristic.

The arrays and the spanning tree itself represent the state of the computation. As in Prim's algorithm and OTTC, they are updated when each new edge joins the tree. In `depth[·]`, this requires only the single assignment of the depth of the new vertex; each vertex's depth is fixed when the vertex joins the tree. Appending an edge does not change the depth of any vertex already in the tree.

When the depth of a new vertex v is less than $\lfloor D/2 \rfloor$, it is eligible: later vertices may connect to it. In this case, the heuristic checks the vertices not yet in the tree to see if v is nearer to each one, a linear-time step. This operation may be performed up to $n - 1$ or $n - 2$ times, so the entire algorithm's time is $O(n^2)$. Running CBTC from each vertex in turn results in a process whose time is $O(n^3)$. Also, while the center-based heuristic still requires the edge weights, it does not need the $n \times n$ array of path lengths `dist[·][·]` of the OTTC algorithm, a significant saving of storage.

5. A RANDOMIZED ALGORITHM

As the tests in Section 6 demonstrate, both OTTC and CBTC are, when applied to Euclidean instances of the BDMST problem with small diameter bounds, too greedy. In both, each new vertex is always the one nearest to an eligible vertex in the tree, so they connect interior vertices with short edges and then must use longer edges to connect the remaining vertices. This difficulty can be mitigated by choosing each next vertex not greedily but at random, as first described by Raidl and Julstrom [2003].

Modify the center-based heuristic CBTC by choosing the start vertex and all subsequent vertices at random from those not yet in the spanning tree. The connection of each new vertex v to the tree remains greedy; it always uses the lowest-weight edge that connects v to a vertex in the tree whose depth is less than $\lfloor D/2 \rfloor$. Call the resulting algorithm randomized center-based tree construction (RTC). Figure 2 summarizes RTC when v_o is chosen at random and the two statements labeled with (*) are changed to assign random unconnected vertices to v_1 and v.

Choosing each next vertex v does not require scanning the array `wnear[·]` and thus takes only constant time, but updating `near[·]` and `wnear[·]` remains $O(n)$. Thus the time of RTC, like that of CBTC, is $O(n^2)$. Running the randomized heuristic n times and reporting the best solution is thus $O(n^3)$.

6. COMPARISONS

The OTTC algorithm of Abdalla, Deo, and Gupta [2000], the center-based heuristic CBTC, and the randomized center-based heuristic RTC were compared on nearly 1,000 instances of the BDMST problem, half Euclidean and half with random edge weights, with a variety of diameter bounds. This section describes the instances and the algorithms' performances on them and presents observations based on those results.

6.1 The Problem Instances

The test BDMST instances were based on 240 graphs, 120 Euclidean and an equal number with edge weights chosen at random. The Euclidean graphs consisted of points randomly placed in the unit square, 30 graphs each of $n = 100$,

Table I. Summary of the Trials of OTTC, CBTC, and RTC on the Euclidean Instances

Instances			OTTC			CBTC			RTC		
		D	$\overline{X}$	s	$\overline{\text{time}}$	$\overline{X}$	s	$\overline{\text{time}}$	$\overline{X}$	s	$\overline{\text{time}}$
n	100	5	29.38	1.71	0.07	26.48	1.51	0.01	15.30	0.78	<0.01
$\overline{\ell}$	6.84	10	18.43	1.86	0.07	15.59	1.28	0.01	9.38	1.80	0.01
$\overline{d}$	42.67	15	12.84	1.47	0.07	10.95	1.00	0.01	9.25	0.28	0.01
min d	29	25	8.06	0.59	0.07	7.69	0.34	0.02	9.19	0.27	0.01
n	250	10	58.20	5.38	1.18	49.07	2.99	0.16	16.89	0.33	0.16
$\overline{\ell}$	10.62	15	41.59	4.03	1.22	36.44	3.15	0.19	15.30	0.23	0.25
$\overline{d}$	79.60	20	32.55	3.86	1.26	26.17	2.36	0.22	15.05	0.23	0.27
min d	48	40	14.43	2.11	1.36	12.51	0.83	0.33	14.98	0.21	0.27
n	500	15	106.87	5.03	12.69	94.38	5.56	1.63	22.26	0.37	4.47
$\overline{\ell}$	14.75	30	58.82	7.10	14.65	46.51	4.13	2.80	21.54	0.33	5.30
$\overline{d}$	123.90	45	32.23	5.66	16.66	25.63	2.18	4.25	21.43	0.33	5.43
min d	91	60	20.33	3.46	17.64	17.72	0.92	5.41	21.45	0.34	5.29
n	1,000	20	217.71	9.48	150.06	195.96	7.97	13.18	31.15	0.24	52.28
$\overline{\ell}$	20.84	40	124.21	17.33	167.91	99.57	7.86	22.61	30.85	0.22	56.17
$\overline{d}$	202.13	60	69.83	12.20	183.21	50.97	5.24	33.28	30.84	0.24	56.82
min d	152	100	28.95	4.14	189.33	23.41	0.78	55.34	30.84	0.24	55.51

250, 500, and 1,000 points. In each set of graphs, 15 can be found in Beasley's [Beasley 1990] OR-library,[1] where they are listed as instances of the Euclidean Steiner problem, and 15 more were randomly generated. In each set, the points are the vertices of complete graphs whose edge weights are the Euclidean distances between the points.

Four more sets of 30 complete graphs also consisted of $n = 100$, 250, 500, and 1,000 vertices. The edge weights of these graphs were chosen at random on the interval [0.01, 0.99].

Four diameter bounds D were chosen for each set of 30 graphs, yielding 960 BDMST instances. The largest value of D was always less than the smallest diameter of an unconstrained minimum spanning tree in the set of graphs. For example, on the Euclidean graphs with $n = 100$ points, the smallest diameter of an unconstrained MST was 29; the four diameter bounds were $D = 5$, 10, 15, and 25. On the random-weight graphs with $n = 100$ vertices, the smallest diameter of a MST was 17; the four diameter bounds for these graphs were $D = 5, 7, 10$, and 15. Note that on average, unconstrained MSTs on the random-weight graphs have smaller diameters than MSTs on the Euclidean graphs.

The leftmost columns of Tables I and III list the number n of vertices and the mean length $\overline{\ell}$, the mean diameter $\overline{d}$, and the minimum diameter min d of unconstrained MSTs on the sets of Euclidean and random-weight graphs, respectively. The tables' second columns list the diameter bounds D of the BDMST instances based on the graphs.

6.2 The Trials

The three heuristics were implemented in C++ and executed on a Pentium 4, running at 2.53 GHz with 1 GByte of memory under Red Hat Linux 9.0. On each BDMST instance, OTTC and CBTC were run starting at each vertex in

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

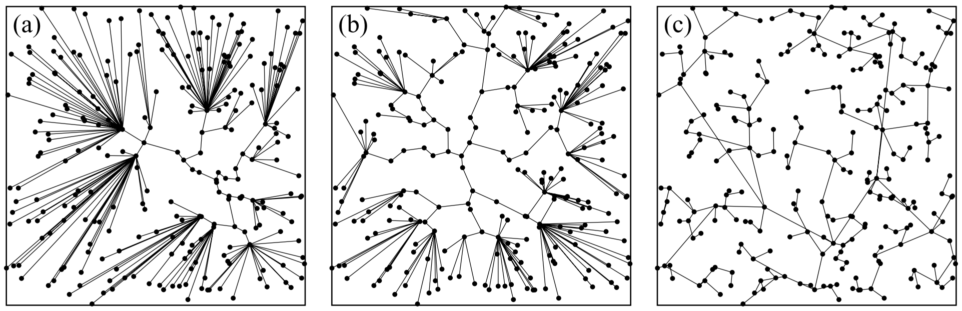

Fig. 3. On the second Euclidean instance with $n = 250$ vertices and diameter bound $D = 15$: (a) The lowest-weight spanning tree found by the OTTC algorithm; its weight is 52.38. (b) The lowest-weight tree found by the center-based heuristic CBTC; it has weight 32.44. (c) The lowest-weight tree found by the randomized greedy heuristic RTC; it has weight 15.08.

turn, as Sections 3 and 4 described, and RTC was run n independent times with random start vertices. Each algorithm reported the smallest weight of a BDST that it found on each instance.

The remaining columns of Tables I and III summarize these trials' results on the Euclidean and random-weight instances. The tables aggregate the results for each set of 30 instances; that is, for each problem size and diameter bound. The runs of each algorithm on a set of instances constitute a sample of size 30 from the population of the algorithm's results on random instances like those in the set. Thus, the tables report the mean $\overline{X}$ and standard deviation s of each algorithm's results as well as its mean clock time in seconds.

The three algorithms' performances differ decisively and consistently both between the Euclidean and random-weight instances and, for the Euclidean instances, as the diameter bound D varies.

6.3 Results on the Euclidean Instances

On the Euclidean BDMST instances, the three algorithm's relative performances vary with the tightness of the diameter bound D. For all the numbers of points, when D is small, CBTC identifies BDSTs that are slightly shorter than those OTTC finds, but RTC's trees are much shorter than those of OTTC and CBTC, and its advantage grows with the number of points.

This is reasonable. Let the backbone of a tree be the subgraph induced by its interior vertices; that is, the tree without its leaves and the edges leading to them. By always choosing the shortest legal edge to the nearest unconnected vertex, OTTC and CBTC build backbones of short edges; the remaining points connect to these backbones via longer edges, so OTTC and CBTC build longer trees than necessary. Since RTC's vertex choices are not greedy, it can build backbones that span clusters of vertices, and vertices can connect to such backbones via edges that are generally shorter. Figure 3 illustrates this phenomenon. It shows the lowest-weight trees found by OTTC, CBTC, and RTC on the second instance with $n = 250$ points and diameter bound $D = 15$. Note that in RTC's tree, the backbone contains longer edges that join separated groups of points.

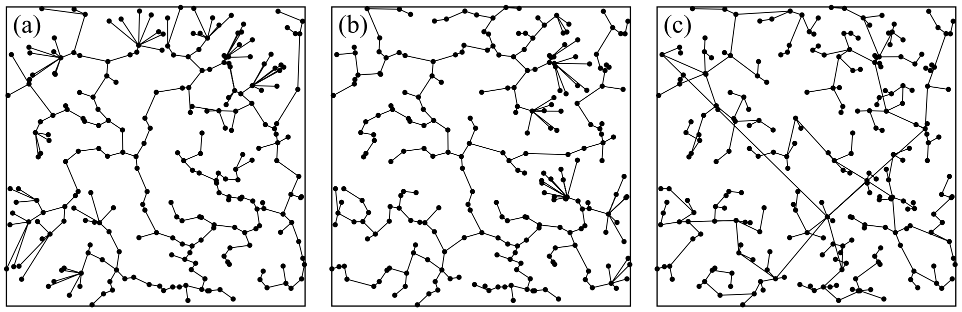

Fig. 4. On the second Euclidean instance with $n = 250$ vertices and diameter bound $D = 40$: (a) The lowest-weight spanning tree found by the OTTC algorithm; its weight is 52.38. (b) The lowest-weight tree found by the center-based heuristic CBTC; it has weight 12.67. (c) The lowest-weight tree found by the randomized greedy heuristic RTC; it has weight 14.84.

With larger diameter bounds, the differences in the three algorithms' results diminish, to the particular advantage of CBTC. When D is approximately half of the mean diameter of unconstrained MSTs, CBTC identifies the shortest bounded-diameter trees, followed by OTTC and then by RTC, whose performance is now worst.

These results also make sense. When D is nearer to the diameter of an unconstrained MST, a short bounded diameter tree resembles that MST. The Prim-like greediness of OTTC and CBTC identifies a tree that is similar to a MST and thus short, but RTC, which chooses its next vertices at random, is less likely to make good choices. Figure 4 illustrates this observation, again using the second Euclidean instance with $n = 250$ points, whose MST has diameter 95, but with $D = 40$. The trees that OTTC and CBTC identify resemble a MST and are shorter. RTC's tree again includes a few longer edges, now to its disadvantage.

Statistical tests confirm these observations. The application of a heuristic to a set of BDMST instances yields a random sample of size 30 from the population of the heuristic's results on all possible instances like those in the set. We compare two heuristics, then, with an approximate t-test for the difference of the means of independent samples from populations whose variances may differ [King and Julstrom 1982]. The null hypothesis states that the two means are equal; the alternative, that they differ.

Table II lists the values of such t statistics and their degrees of freedom for the comparisons of OTTC and CBTC, OTTC and RTC, and CBTC and RTC on the 16 sets of Euclidean BDMST instances. Almost all the differences are significant at at least the 5% level, indicating that the means—the average performances of the two algorithms—differ. In particular, we see again that CBTC always outperforms OTTC, that RTC outperforms both OTTC and CBTC when D is small, and that CBTC is best when D is larger.

Finally, the average clock times reported in Table I confirm the observations that accompanied the three algorithms' descriptions. OTTC's time grows markedly more quickly than that of CBTC and RTC. Among the latter, CBTC is faster than RTC, and both take longer when D is larger. In this last case, the

Table II. t-tests for the Trials on the Euclidean Instances

		OTTC vs. CBTC		OTTC vs. RTC		CBTC vs. RTC	
n	D	t	df	t	df	t	df
100	5	6.963	57	41.032	41	36.030	43
100	10	6.889	51	19.151	58	15.400	52
100	15	5.823	51	13.140	31	8.966	34
100	25	2.976	46	−9.539	41	−18.923	55
250	10	8.125	45	41.978	29	58.593	30
250	15	5.515	55	35.673	29	36.661	29
250	20	7.724	48	24.788	29	25.686	30
250	40	4.638	38	−1.421	30	−15.802	33
500	15	9.124	57	91.885	29	70.890	29
500	30	8.209	47	28.728	29	33.010	29
500	45	5.960	37	10.434	29	10.434	30
500	60	3.993	33	−1.764	30	−20.830	37
1,000	20	9.619	56	107.754	29	113.211	29
1,000	40	7.092	40	29.504	29	47.869	29
1,000	60	7.780	39	17.501	29	21.019	29
1,000	100	7.203	31	−2.496	29	−49.867	34

Table III. Summary of the Trials on the Instances with Random Edge Weights, as in Table I

			OTTC			CBTC			RTC		
Instances		D	$\overline{X}$	s	$\overline{\text{time}}$	$\overline{X}$	s	$\overline{\text{time}}$	$\overline{X}$	s	$\overline{\text{time}}$
n	100	5	5.36	0.42	0.06	5.35	0.33	0.01	6.97	0.36	<0.01
$\overline{\ell}$	2.15	7	3.37	0.21	0.06	3.34	0.19	0.01	4.87	0.28	0.01
$\overline{d}$	23.4	10	2.59	0.12	0.06	2.56	0.13	0.01	4.10	0.23	0.01
min d	17	15	2.27	0.11	0.06	2.26	0.11	0.02	3.84	0.23	0.01
n	250	5	10.17	0.48	1.18	10.14	0.43	0.17	12.90	0.54	0.05
$\overline{\ell}$	3.68	10	4.49	0.15	1.27	4.45	0.16	0.27	6.67	0.22	0.18
$\overline{d}$	37.3	15	4.00	0.14	1.32	3.99	0.13	0.38	6.14	0.25	0.25
min d	24	20	3.81	0.12	1.32	3.80	0.12	0.38	6.80	0.25	0.25
n	500	10	7.30	0.13	16.51	7.25	0.11	4.09	10.01	0.21	3.23
$\overline{\ell}$	6.15	15	6.63	0.08	17.48	6.61	0.08	5.84	9.23	0.16	4.78
$\overline{d}$	48.9	20	6.38	0.07	18.56	6.36	0.06	6.78	9.06	0.15	5.22
min d	37	30	6.22	0.06	19.25	6.21	0.06	8.14	9.04	0.15	5.24
n	1,000	10	12.74	0.12	171.19	12.71	0.10	33.66	16.17	0.18	28.76
$\overline{\ell}$	11.18	20	11.51	0.05	183.49	11.50	0.05	59.37	14.72	0.18	53.72
$\overline{d}$	66.8	30	11.30	0.04	199.20	11.30	0.04	70.76	14.66	0.18	55.76
min d	53	50	11.20	0.04	199.96	11.19	0.03	82.79	14.65	0.17	56.07

number of eligible vertices at each step is larger, so it takes longer to search for the best attachment of a new vertex to the existing tree.

6.4 Results on the Random-Weight Instances

Table III shows that the observations just made about the three algorithms' performances on the Euclidean instances are almost entirely reversed when OTTC, CBTC, and RTC are applied to the random-weight instances. Again, the trees CBTC identifies have on average lower weights than those OTTC finds, though the differences are negligible. More importantly, on all the sets of instances, regardless of n or D, the performance of RTC is always decisively worse than that of both OTTC and RTC. Its trees always have markedly higher

Table IV. t-tests for the Trials on the Random Instances

		OTTC vs. CBTC		OTTC vs. RTC		CBTC vs. RTC	
n	D	t	df	t	df	t	df
100	5	0.103	55	−15.941	57	−18.169	58
100	7	0.580	57	−23.474	54	−24.766	51
100	10	0.929	58	−31.881	44	−31.927	46
100	15	0.352	58	−33.729	42	−33.944	42
250	5	0.255	57	−20.696	57	−21.900	55
250	10	0.999	58	−44.843	51	−44.699	53
250	15	0.287	58	−40.907	46	−41.792	44
250	20	0.323	58	−59.057	42	−59.254	42
500	10	1.608	56	−60.099	48	−63.768	44
500	15	0.968	58	−79.608	43	−80.221	43
500	20	1.188	57	−88.679	41	−91.539	38
500	30	0.645	58	−95.607	38	−95.946	38
1,000	10	1.052	56	−86.842	51	−92.035	45
1,000	20	0.775	58	−94.114	33	−94.407	33
1,000	30	0.000	58	−99.807	32	−99.807	32
1,000	50	1.095	54	−108.201	32	−109.781	31

weight that those of the other two heuristics, and its disadvantage grows with increasing problem size.

The explanation for these results lies in the structure—or lack thereof—of the random-weight instances, in which there are, in general, a few low-weight edges at every vertex. The methodically greedy OTTC and CBTC find and use these edges, while RTC, picking each next vertex at random, often does not, particularly early in its execution when the tree it is building contains only a few vertices. Furthermore, since every vertex is likely to share low-weight edges with at least a few others, there is no requirement that a low-weight tree's backbone join "distant" groups of vertices. The lack of Euclidean structure in the random-weight instances helps OTTC and CBTC and hinders RTC.

Again, t-tests confirm the qualitative observations just made. For the random-weight problem instances, Table IV lists the values of t statistics and their degrees of freedom for each comparison of two algorithms on the 16 sets of Euclidean BDMST instances, as in Table II. The t statistics for the mean results of OTTC and CBTC are never significant, indicating that on random-weight BDMST instances such as these, the performances of these two heuristics are statistically indistinguishable. The statistics for the other two comparisons are always significant, at at least the 1% level, confirming that both OTTC and CBTC outperform RTC.

Finally, the times of the three heuristics on the random-weight instances are similar to their times on the Euclidean instances and conform to the earlier observations of their big-O behavior. Also, on the larger problems, all the algorithms take longer when the diameter bound D is larger.

6.5 The Effect of Non-Euclidean Weights

We have seen that on the Euclidean BDMST instances with small diameter bounds, RTC performs better than CBTC, which is in turn better than OTTC, while on the random-weight instances with similar values of n and D, CBTC

Table V. Mean Performances of OTTC, CBTC, and RTC on a Sequence of Instances Built from the First Euclidean Instances with $n = 250$ and 500

	$n = 250$, $D = 10$				$n = 500$, $D = 15$		
b	OTTC	CBTC	RTC	b	OTTC	CBTC	RTC
0.00	41.95	41.95	17.17	0.00	106.07	92.25	22.41
0.05	52.96	48.74	22.85	0.05	76.96	88.44	32.47
0.10	54.29	43.78	26.67	0.10	96.32	69.95	38.86
0.25	48.61	42.19	34.80	0.25	78.27	67.77	53.46
0.50	48.69	48.69	44.31	0.50	79.35	76.46	67.75
1.00	60.83	57.52	55.20	1.00	90.05	88.40	85.61
2.50	76.15	70.63	79.32	2.50	113.50	108.36	119.32
5.00	89.65	89.65	103.07	5.00	131.30	128.97	153.75
10.00	108.83	108.83	131.29	10.00	166.19	168.97	206.30
25.00	158.48	152.85	204.12	25.00	227.33	227.33	308.54

D is set to 10 and 15, respectively, but the instances have edge weights equal to those in the Euclidean instances plus random values on the interval $[0, b]$.

returns trees of slightly lower weight than OTTC and markedly lower weight than RTC. On the Euclidean instances, RTC builds backbones that connect distant sets of points, while on the random-weight instances, OTTC and CBTC effectively apply their deterministic greediness.

How much, then, must an instance with small D depart from a Euclidean structure for the advantage to pass from RTC to OTTC and CBTC? More generally, how "random" must such an instance's weights be for the latter algorithms to be the more effective?

We address these questions by applying the three heuristics to two sequences of related BDMST instances. The graphs underlying these instances were constructed from the first Euclidean instances with $n = 250$ and 500 points by adding to their (Euclidean) edge weights random values on the interval $[0, b]$ for 10 values of b from zero to 25.00. To complete BDMST instances, diameter bounds were set to $D = 10$ and $D = 15$, respectively. Table V summarizes the results of the algorithms' trials on the instances.

When $b = 0.0$, the graphs are unchanged, so the first row of the graph presents the heuristics' results on the original instances, on which RTC's trees are decisively best. RTC retains its advantage for values of b up to 1.00, but when b is larger, CBTC returns the lowest-weight trees.

The graphs on which the original BDMST instances are based consist of points in the unit square with edge weights equal to the Euclidean distances between the points; the largest possible weight in such a graph is $\sqrt{2}$. RTC returns the lowest-weight trees as long as the random perturbations imposed on the edge weights are on average no greater than the original weights themselves. When the perturbations on average exceed the Euclidean edge weights, the advantage shifts to OTTC and, especially, CBTC.

7. CONCLUSION

Given a connected, weighted, undirected graph G and a bound D, the BDMST problem seeks a spanning tree on G of lowest weight among the trees in which no path between two vertices contains more than D edges. Three greedy heuristics for this problem are based on Prim's algorithm.

Abdalla, Deo, and Gupta's [2000] OTTC grows a BDST from a start vertex by repeatedly appending the lowest-weight edge that joins a vertex in the tree with one not yet in it without violating the diameter bound. The algorithm maintains the diameter bound by limiting the eccentricities of the vertices in the tree. Appending a new vertex in general modifies these eccentricities and may render vertices ineligible. Updating the algorithm's data structures requires time that is $O(n^2)$, so the algorithm's time is $O(n^3)$, and repeating it starting at each vertex in turn is $O(n^4)$.

The center-based heuristic CBTC applies the same Prim-based strategy but uses the start vertex as the center of the spanning tree (if D is even) or as one of two vertices in the center (if D is odd). No vertex can be more than $\lfloor D/2 \rfloor$ edges from the center, and the depth—thus the eligibility—of a vertex is fixed when it joins the tree. Updating this algorithm's data structures requires only linear time in the worst case (constant time when a new vertex's depth is $\lfloor D/2 \rfloor$), so the algorithm's time is $O(n^2)$ and starting at each vertex in turn is $O(n^3)$.

The randomized center-based heuristic RTC modifies CBTC by choosing each next vertex not greedily but at random, though its connection to the growing spanning tree is still the most economical. RTC's time is the same as that of CBTC.

The three heuristics were compared on 960 instances of the BDMST, half Euclidean and half whose edge weights were chosen at random, with a variety of diameter bounds. The algorithms' relative performances varied greatly depending on the kind of instance they addressed and, on the Euclidean instances, on the diameter bound D.

On the Euclidean instances with small diameter bounds, CBTC identified slightly shorter trees than did OTTC, but the randomized heuristic RTC decisively outperformed them both. When the diameter bound was a significant fraction of the diameter of a MST, CBTC enjoyed a slight advantage over OTTC, and both returned significantly shorter trees than did RTC.

On the random-weight instances, CBTC in general found lower-weight trees than did OTTC, but the difference was small. Both consistently returned trees of much lower weights than did RTC.

Tests on BDMST instances contrived to have various degrees of "randomness" in their edge weights confirmed that when D is small, the structure of Euclidean instances hinders OTTC and CBTC and helps RTC; the lack of that structure helps OTTC and CBTC and hinders RTC.

REFERENCES

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

Achuthan, N. R., Caccetta, L., Caccetta, P., and Geelen, J. F. 1994. Computation methods for the diameter restricted minimum weight spanning tree problem. *Australasian Journal of Combinatorics 10*, 51–71.

Althaus, E., Funke, S., Har-Peled, S., Könemann, J., Ramos, E. A., and Skutella, M. 2005. Approximating k-hop minimum-spanning trees. *Operations Research Letters 33*, 2, 115–120.

Beasley, J. E. 1990. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society 41*, 1069–1072.

DAHL, G. 1997. The 2-hop spanning tree problem. Tech. rep. 250, University of Oslo.

DAVIS, L. 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

DEO, N. AND ABDALLA, A. 2000. Computing a diameter-constrained minimum spanning tree in parallel. In *Algorithms and Complexity*, G. Bongiovanni, G. Gambosi, and R. Petreschi, Eds. Number 1767 in LNCS. Springer-Verlag, Berlin, 17–31.

DOS SANTOS, A. C., LUCENA, A., AND REBEIRO, C. C. 2004. Solving diameter constrained minimum spanning tree problems in dense graphs. In *Experimental and Efficient Algorithms: Proceedings of the 3rd International Workshop, WEA 2004*, C. C. Rebeiro and S. L. Martins, Eds. Number 3059 in LNCS. Springer, Berlin, 458–467.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

GOUVEIA, L. AND MAGNANTI, T. L. 2003. Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. *Networks 41*, 159–173.

GRUBER, M. AND RAIDL, G. R. 2005a. A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem. In *Proceedings of the 2nd International Network Optimization Conference*. Vol. 1. 178–185. Lisbon.

GRUBER, M. AND RAIDL, G. R. 2005b. Variable neighborhood search for the bounded diameter minimum spanning tree problem. In *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, P. Hansen, N. Mladenovic, J. A. M. Pérez, B. M. Batista, and J. M. Moreno-Vega, Eds. Tenerife, Spain.

GRUBER, M. AND RAIDL, G. R. 2008. Heuristic cut separation in a branch&cut approach for the bounded diameter minimum spanning tree problem. In *Proceedings of the 2008 Symposium on Applications and the Internet*. Turku, Finland.

GRUBER, M., VAN HEMERT, J., AND RAIDL, G. R. 2006. Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, M. Keijzer et al., Eds. Vol. 2. ACM Press, New York, 1187–1194.

HANDLER, G. Y. 1978. Minimax location of a facility in an undirected graph. *Transportation Science 7*, 287–293.

JULSTROM, B. A. AND RAIDL, G. R. 2003. A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In *2003 Genetic and Evolutionary Computation Conference Workshop Program*, A. Barry, Ed. 2–7. Chicago.

KING, R. S. AND JULSTROM, B. 1982. *Applied Statistics Using the Computer*. Alfred Publishing Company, Sherman Oaks, CA.

KOPINITSCH, B. 2006. An ant colony optimisation algorithm for the bounded diameter minimum spanning tree problem. M.S. thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

PRIM, R. C. 1957. Shortest connection networks and some generalizations. *Bell System Technical Journal 36*, 1389–1401.

RAIDL, G. R. AND GRUBER, M. 2008. A Lagrangean relax-and-cut approach for the bounded diameter minimum spanning tree problem. In *International Conference on Numerical Analysis and Applied Mathematics 2008*. AIP Conference Proceedings, vol. 1048. American Institute of Physics, 446–449.

RAIDL, G. R. AND JULSTROM, B. A. 2003. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, G. Lamont, H. Haddad, G. A. Papadopoulos, B. Panda, J. Carroll, R. Stansifer, W. Jones, R. Menezes, and W. Shoaff, Eds. ACM Press, New York, 747–752.

SINGH, A. AND GUPTA, A. K. 2007. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Computing 11*, 10, 911–921.

SÖRENSEN, K. AND JANSSENS, G. K. 2002. Generating solutions for the 2-hop constrained spanning tree and the simple plant location problem in order of decreasing cost. In *Proceedings of the 16th Belgian Conference on Quantitative Methods for Decision Making*. Brussels.

Received June 2006; revised January 2009; accepted January 2009