SIAM J. DISCRETE MATH.
Vol. 9, No. 2, pp. 178–200, May 1996

003

SPANNING TREES—SHORT OR SMALL*

R. RAVI[†], R. SUNDARAM[‡], M. V. MARATHE[§], D. J. ROSENKRANTZ[¶], AND S. S. RAVI[||]

Abstract. We study the problem of finding small trees. Classical network design problems are considered with the additional constraint that only a specified number k of nodes are required to be connected in the solution. A prototypical example is the kMST problem in which we require a tree of minimum weight spanning at least k nodes in an edge-weighted graph. We show that the kMST problem is NP-hard even for points in the Euclidean plane. We provide approximation algorithms with performance ratio $2\sqrt{k}$ for the general edge-weighted case and $O(k^{1/4})$ for the case of points in the plane. Polynomial-time exact solutions are also presented for the class of treewidth-bounded graphs, which includes trees, series-parallel graphs, and bounded bandwidth graphs, and for points on the boundary of a convex region in the Euclidean plane.

We also investigate the problem of finding short trees and, more generally, that of finding networks with minimum diameter. A simple technique is used to provide a polynomial-time solution for finding k-trees of minimum diameter. We identify easy and hard problems arising in finding short networks using a framework due to T. C. Hu.

Key words. approximation algorithm, network design, spanning tree

AMS subject classifications. 05C05, 68Q25, 68R10

1. Introduction.

1.1. Motivation: Small trees. The oil reconnaissance boats are back from their final trip off the coast of Norway and present you with a detailed map of the seas surrounding the coastline. Marked in this map are locations that are believed to have a good chance of containing oil under the sea bed. Your company has a limited number of oil rigs that it is willing to invest in the effort. Your problem is to position these oil rigs at marked places so that the cost of laying down pipelines between these rigs is minimized. The problem at hand can be modeled as follows: given a graph with nonnegative edge weights and a specified number k, find a tree of minimum weight spanning at least k nodes. Note that a solution to the problem will be a tree spanning exactly k nodes. We call this problem the *k-minimum spanning tree* (or the kMST) problem. Moreover, the kMST problem is at the heart of several other optimization

* Received by the editors April 5, 1994; accepted for publication (in revised form) April 5, 1995. A preliminary version of this paper [31] appeared in *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 546–555.

† Center for Discrete Mathematics and Theoretical Computer Science, Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08854 (ravi@cs.princeton.edu). The research of this author was performed at Brown University and supported by an IBM Graduate Fellowship, National Science Foundation Presidential Young Investigator award CCR-9157620, and Defense Advanced Research Projects Agency contract N00014-91-J-4052/order 8225.

‡ Department of Computer Science, Massachussetts Institue of Technology, Cambridge MA 02139 (koods@theory.lcs.mit.edu). The research of this author was supported by Defense Advanced Research Projects Agency contract N0014-92-J-1799 and National Science Foundation grant 92-12184 CCR.

§ Los Alamos National Laboratory, P.O. Box 1663, MS M986, Los Alamos, NM 87545 (madhav@c3.lanl.gov). The research of this author was performed at the Department of Computer Science, State University of New York at Albany and supported by National Science Foundation grant CCR 89-03319.

¶ Department of Computer Science, State University of New York at Albany, Albany, NY 12222 (djr@cs.albany.edu). The research of this author was supported by National Science Foundation grant CCR-90-06396.

|| Department of Computer Science, State University of New York at Albany, Albany, NY 12222 (ravi@cs.albany.edu). The research of this author was supported by National Science Foundation grant CCR-89-05296.

problems, such as the latency problem [9] and the prize-collecting traveling salesperson problem [1], and hence is of independent interest. In this paper, we study such classical network-design problems as the MST problem with the additional constraint that only a specified number of nodes need to be incorporated into the network. Unlike the MST problem, which admits a polynomial-time solution [25], [28], the kMST problem is considerably harder to solve. In Theorem 2.1 we prove that the kMST problem is NP-complete. This result was independently obtained by Lozovanu and Zelikovsky [26]. The kMST problem remains NP-complete even when all the edge weights are drawn from the set $\{1, 2, 3\}$ (i.e., the graph is complete and every edge takes one of three different weights). It is not hard to show a polynomial-time solution for the case of two distinct weights. The problem remains NP-hard even for the class of planar graphs as well as for points in the plane.

1.2. Approximation algorithms. A ρ-approximation algorithm for a minimization problem is one that delivers a solution of value at most ρ times the minimum. Consider a generalization of the kMST problem, the k-Steiner tree problem: given an edge-weighted graph, an integer k, and a subset of at least k vertices specified as terminals, find a minimum-weight tree spanning at least k terminals. We can apply approximation results for the kMST problem to this problem by considering the auxiliary complete graph on the terminals with edges weighted by shortest-path distances. A ρ-approximation for the kMST problem on the auxiliary graph yields a 2ρ-approximation for the k-Steiner tree problem. Therefore we focus on approximations for the kMST problem. We provide the first approximation algorithm for this problem. In Theorem 3.1 we present a polynomial-time algorithm $2\sqrt{k}$-approximation algorithm for the kMST problem. The algorithm is based on a combination of a greedy technique that constructs trees using edges of small cost and a shortest-path heuristic that merges trees when the number of trees to be merged is small. The analysis of the performance ratio is based on a solution–decomposition technique [4], [14], [24], [29], [30] that uses the structure of an optimal solution to derive a bound on the cost of the solution constructed by the approximation algorithm.

Theorem 3.1 provides a $4\sqrt{k}$-approximation algorithm for the k-Steiner tree problem as well. Moreover, we construct an example that demonstrates the performance guarantee of the approximation algorithm is tight to within a constant factor.

We derive a better approximation algorithm for the case of points in the Euclidean plane. In Theorem 4.1 we show that there is a polynomial-time algorithm that, given n points in the Euclidean plane and a positive integer $k \leq n$, constructs a tree spanning at least k of these points such that the total length of the tree is at most $O(k^{1/4})$ times that of a minimum-length tree spanning any k of the points.

As before, we construct an example showing that the performance ratio of the algorithm in Theorem 4.1 is tight. Our proof of Theorem 4.1 also yields as a corollary an approximation algorithm for the rectilinear kMST problem.

1.3. Polynomially solvable special cases. Since the kMST problem is NP-complete even for the class of planar graphs, we focus on special classes of graphs and provide exact solutions that run in polynomial time. Robertson and Seymour in their seminal series of papers [32] introduced and developed the notion of treewidth. Many hard problems have exact solutions when attention is restricted to the class of treewidth-bounded graphs and much work has been done in this area, especially by Bodlaender [11]. Independently, Bern, Lawler, and Wong [8] introduced the notion of decomposable graphs. Later, it was shown [5] that the class of decomposable graphs and the class of treewidth-bounded graphs are one and the same. A class of decomposable graphs is defined using a finite number of primitive graphs and a finite collection of binary composition rules. Examples of decomposable graphs include trees, series-parallel graphs, and bounded-bandwidth graphs. We use a dynamic programming technique to show that for any class of decomposable graphs (or treewidth-bounded graphs), there is an $O(nk^2)$-time algorithm for solving the kMST problem. A polynomial-time algorithm for trees was also independently obtained by Lozovanu and Zelikovsky [26].

Though the kMST problem is hard for arbitrary configurations of points in the plane, we show in §5.2 that there is a polynomial-time algorithm for solving the kMST problem for the case of points in the Euclidean plane that lie on the boundary of a convex region. We also provide a faster algorithm to find the optimal kMST when all the points lie on a circle. The proof of the above facts uses a monotonicity property of an optimal tree along with a degree constraint on an optimal solution. This allows us to apply dynamic programming to find the exact solution. Several researchers in computational geometry have presented exact algorithms for choosing k points that minimize other objectives such as diameter, perimeter, area, and volume [3], [16]–[18].

1.4. Short trees. Keeping the longest path in a network small is often an important consideration in network design. We investigate the problem of finding networks with small diameter. Recall that the diameter of a tree is the maximum distance (path length) between any pair of nodes in the tree. The problem of finding a minimum-diameter spanning tree of an edge-weighted graph was shown to be polynomially solvable by Camerini, Galbiati, and Maffioli [13] when the edge weights are nonnegative. They also show that the problem becomes NP-hard when negative weights are allowed. Camerini and Galbiati [12] proposed polynomial-time algorithms for a bounded-path tree problem on graphs with nonnegative edge weights. Their result can be used to show that the minimum-diameter spanning tree problem as well as its natural generalization to Steiner trees can be solved in polynomial time. We use a similar technique to show that the following *minimum-diameter k-tree* problem is polynomially solvable: given a graph with nonnegative edge weights, find a tree of minimum diameter spanning at least k nodes.

We investigate easy and hard results in finding short networks. For this, we use a framework due to T. C. Hu [22]. In this framework, we are given a graph with nonnegative distance values d_{ij} and nonnegative requirement values r_{ij} between every pair of nodes i and j in the graph. The communication cost of a spanning tree is defined to be the sum over all pairs of nodes i, j of the product of the distance between i and j in the tree under d and the requirement r_{ij}. The objective is to find a spanning tree with minimum communication cost. Hu considered the case when all the d values are one and showed that a Gomory–Hu cut tree [21] using the r values as capacities is an optimal solution. Hu also considered the case when all the r values are one and derived sufficient conditions under which the optimal tree is a star. The general version of the latter problem is NP-hard [2], [13], [23].

We define the diameter cost of a spanning tree to be the maximum cost over all pairs of nodes i, j of the distance between i and j in the tree multiplied by r_{ij}. In Table 1, we present current results in this framework. All r_{ij} and d_{ij} values are assumed to be nonnegative. The first two rows of the table examine the cases when either of the two parameters is uniform-valued. The last two rows illustrate that the two problems become NP-complete when both parameters are two-valued.

1.5. Short small trees. We consider the k-tree versions of the minimum-communication-cost and minimum-diameter-cost spanning tree problems and show in

TABLE 1
Results on minimum-communication-cost spanning trees and minimum-diameter-cost spanning trees.

r_{ij}	d_{ij}	Communication cost	Diameter cost
Arbitrary	$\{a\}$	Cut-tree [22]	Open
$\{a\}$	Arbitrary	NP-complete [23]	Poly-time [13]
$\{a,b\}$	$\{0,c\}$	Cut-tree variant (this paper, [22])	Poly-time (this paper)
$\{a,4a\}$	$\{c,5c\}$	NP-complete [23]	NP-complete (this paper)

Theorem 6.6 that the minimum-communication k-tree problem and the minimum-diameter k-tree problem are both hard to approximate within any factor even when all the d_{ij} values are one and the r_{ij} values are nonnegative.

In the next section, we present the NP-completeness results. Section 3 contains the $2\sqrt{k}$ approximation for the kMST problem. In §4 we present the stronger result for the case of points in the plane. In §5 we address polynomially solvable cases of the problem. In §6 we prove our results on short trees. We close with a discussion of directions for future research.

2. NP-completeness results.

THEOREM 2.1. *The (decision version of the) kMST problem is NP-complete.*

Proof. It is easy to see that the kMST problem is in NP. In this section we show that the kMST problem is NP-hard by reducing the Steiner tree problem to it. The Steiner tree problem is known to be NP-hard [19]. As an instance of the Steiner tree problem we are given an undirected graph G, a set of terminals R (which is a subset of the vertex set of G), and a positive integer M, and the question is whether there exists a tree spanning R and containing at most M edges. We transform this input to an instance G', k, M of the kMST problem as follows: We let $X = |V(G)| - |R|$ and connect each terminal of G to a distinct path of X new vertices, the path consisting of zero-weighted edges. We assign weight one to the already existing edges of G and set the weight between all other pairs of vertices to ∞ (a very large number). This is the graph G' (see Fig. 1). We set k to be $|R| \cdot (X+1)$. And now we ask if G' has a tree spanning k vertices of weight at most M. If there exists a Steiner tree in G spanning the set R and containing at most M edges, then it is easy to construct a kMST of weight at most M in G'. Conversely, by our choice of k and X, any kMST in G' must contain at least one node from the path corresponding to each terminal in R. Hence any kMST can be used to derive a Steiner tree for R in G. This completes the reduction. Extensions of hardness to the case of planar graphs and points in the plane follow in a similar way from the hardness of the Steiner tree problem in these restricted cases. Given a planar embedding of G we can create an embedded version of G' since only paths are added.

The NP-hardness holds even when all the edge costs are from the set $\{1, 2, 3\}$. The reduction for this case is similar to the above. Without loss of generality we assume that in the given instance of the Steiner tree problem, G is connected and $M \leq |V| - 1$. We let $X = |V(G)| - |R|$ as before and connect each terminal of G to a distinct set of X vertices by edges of weight one. We set the original edges of G to have weight two and all other edges to have weight three. We choose $k = |R| \cdot X + M + 1$ and the bound on the cost of the kMST to be $|R| \cdot X + 2M$. If there exists a Steiner tree in G spanning the set R and containing at most M edges, then it is easy to construct a kMST of weight at most $|R| \cdot X + 2M$ in G'. This is done by connecting all the newly added vertices to the Steiner tree using the weight-one edges and then picking up more vertices (note that the graph is connected and $M \leq |V| - 1$) using the weight-two edges until there are $|R| \cdot X + M + 1$ vertices. If there exists a kMST of weight at most $|R| \cdot X + 2M$ in G', then observe that the kMST cannot contain an edge of weight three because it has exactly $k - 1 = |R| \cdot X + M$ edges; and if it contained an edge of weight three, then it would have to contain at least $|R| \cdot X + 1$ edges of weight one but there are only $|R| \cdot X$ edges of weight one in G'. Further, the kMST must span R, and since it has at most M edges of weight two, there must exist a Steiner tree in G spanning R and containing at most M edges. □

When there are only two distinct edge costs, i.e., the graph is complete and every edge has one of two possible weights, the kMST problem can be solved in polynomial time. The basic idea is the following: Let w_1 and w_2 denote the two edge weights, where $w_1 < w_2$. Construct an edge subgraph G_1 of G containing all the edges of weight w_1. Choose a minimum number, say r, of the connected components of G_1 to obtain a total of k nodes (dropping some nodes if necessary). Construct a spanning tree for each chosen component, and connect the trees into a single tree by adding exactly $r - 1$ edges of weight w_2. It is straightforward to verify that the resulting solution is optimal.

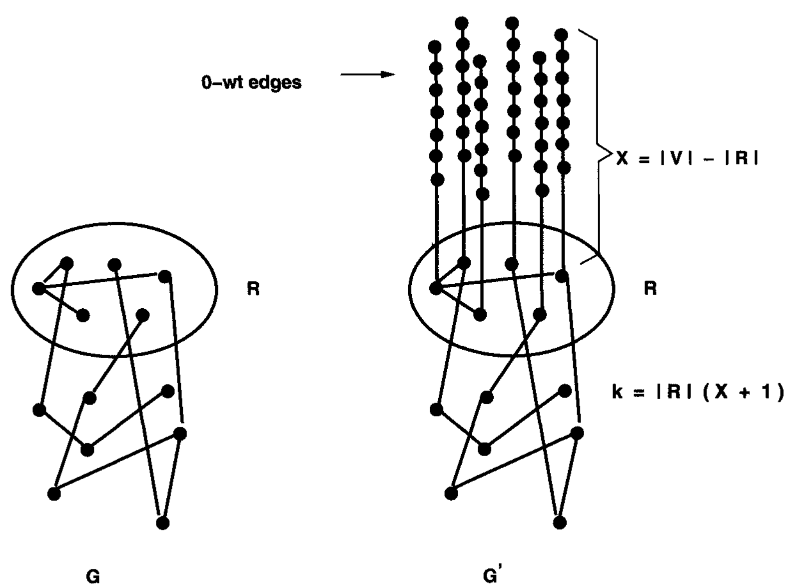

FIG. 1. *The basic NP-hardness reduction from Steiner tree to kMST.*

3. The approximation algorithm for the general case.

THEOREM 3.1. *There is a polynomial-time algorithm that, given an undirected graph G on n nodes with nonnegative weights on its edges and a positive integer $k \leq n$, constructs a tree spanning at least k nodes of weight at most $2\sqrt{k}$ times that of a minimum-weight tree spanning any k nodes.*

In this section, we present the proof of the above theorem. As input, we are given an undirected graph G with nonnegative edge weights and an integer k.

3.1. The algorithm and its running time. It is useful to think of the algorithm as running in two distinct phases: a merge phase and a collect phase.

During the merge phase, the algorithm maintains a set of clusters and a spanning tree on the vertex set of each cluster. Initially each vertex forms a singleton cluster. At each step of the merge phase, we choose an edge of minimum cost among all edges that are between two clusters and merge them by using the edge to connect their spanning trees.

Define the size of a cluster to be the number of vertices that it contains. During the course of the merge phase, the clusters grow in size. The collect phase is entered only when

(i) there exists a set of at most $\sqrt{k}$ clusters containing at least k vertices among themselves, and

(ii) no cluster has size k or more.

In the collect phase, we consider each cluster in turn as the root and perform a shortest-path computation between clusters using the weights on intercluster edges. We determine for each cluster C, the shortest distance d_C such that, within distance d_C from C, there exist at most $\sqrt{k}$ clusters whose sizes sum to at least k. Note that by the first precondition for starting the collect phase, the distance d_C is well defined. We choose the cluster C with the minimum value of d_C and connect it using shortest paths of length at most d_C to each of these $\sqrt{k}$ clusters. We prune edges from some of these shortest paths to output a tree of clusters whose sizes sum to k. We may do this since any cluster has less than k nodes at the start of this phase by the second precondition.

The merge phase of the algorithm continues to run until both the preconditions of the collect phase are satisfied. Beginning with the step of the merge phase after which both preconditions of the collect phase are satisfied, at each subsequent step, the algorithm forks off an execution of the collect phase for the current configuration of clusters. The merge phase continues to run until a cluster of size k or more is formed. Next, the merge phase prunes the edges of the spanning tree of the cluster whose size is between k and $2k$ so as to obtain a spanning tree of size exactly k. At this point, the merge phase terminates and outputs the spanning tree of the cluster of size k. Each forked execution of the collect phase outputs a spanning tree of size between k and $2k$ as well. The algorithm finally outputs the tree of least weight among all these trees. The algorithm is given as follows.

ALGORITHM MERGE-COLLECT

1. Initialize each vertex to be in singleton-connected components and the set of edges chosen by the algorithm to be ϕ. Initialize the iteration count $i = 1$.
2. Repeat until there exists a cluster whose size is between k and $2k$.
   (a) Let $VS_i = \{C_1 \ldots C_l\}$ denote the set of connected components at the start of this iteration. Assume that the components are numbered in nonincreasing order of their size.
   (b) Form an auxiliary graph $G(VS_i, E')$ where the edge (C_i, C_j) between two components is the minimum-cost edge in E whose endpoints belong to C_i and C_j, respectively.
   (c) Choose a minimum-cost edge (C_i, C_j) in $G(VS_i, E')$ and merge the corresponding clusters C_i and C_j.
   (d) $VS_{i+1} = VS_i - \{C_i\} - \{C_j\} \cup \{C_i \cup C_j\}$.
   Remark: This corresponds to one iteration of merge phase.
   (e) Let $j^* = \min\{j : \sum_{i=1}^{j} |C_i| \geq k\}$.
   (f) If $j^* \leq \sqrt{k}$, then $SOL_i = \text{Collect}(G(VS, E'))$.
   (g) $i = i + 1$.

3. Prune the edges of the cluster whose size is between k and $2k$ to obtain a tree with exactly k vertices. Denote the tree obtained by $MSOL$.
4. The output of the heuristic is the minimum valued tree among $MSOL$ and all the SOL_is.

PROCEDURE COLLECT($G(V, E)$)

1. For each cluster vertex C do
   (a) With the cluster C as the root, form a shortest path tree.
   (b) Let d_C be the minimum distance such that there is a set of at most $\sqrt{k}$ clusters within a distance of d_C from C containing at least k vertices.
   (c) Choose these clusters and join them to the root cluster by using the edges in the shortest path tree computed in Step 1(a).
   (d) Prune the edges of the tree to obtain a tree having exactly k nodes.
2. Output the tree corresponding to the choice of the root cluster C that minimizes d_C.

It is easy to see that there are at most $O(n)$ steps in the merge phase and hence at most this many instances of the collect phase to be run. Using Dijkstra's algorithm [15] in each collect phase, the whole algorithm runs in time $O(n^2(m + n \log n))$ where m and n denote the number of edges and nodes in the input graph, respectively. The running time of the collect phase dominates the running time of the merge phase.

3.2. The performance guarantee. Consider an optimal kMST of weight OPT. During the merge phase, nodes of this tree may merge with other nodes in clusters. We focus our attention on the number of edges of the optimal kMST that are *exposed*, i.e., remain as intercluster edges. We show that at any step in which a large number of edges of the kMST are exposed, every edge in the spanning tree of each cluster has small weight.

LEMMA 3.2. *If at the beginning of a step of the merge phase, an optimal kMST has at least x exposed edges (intercluster edges), then each edge in the spanning tree of any cluster at the end of the step has weight at most $\frac{OPT}{x}$.*

Proof. Since the edges are chosen in nondecreasing order of cost, it is clear that each edge in the spanning tree of any cluster at the end of the step has weight at most that of any intercluster edge. Since an optimal kMST has at least x exposed edges, one of these edges has weight at most $\frac{OPT}{x}$. Hence each edge in the spanning tree of any cluster at the end of the step has weight at most $\frac{OPT}{x}$. □

We now prove the performance guarantee in Theorem 3.1. The above lemma is useful as long as the number of exposed edges is high. Applying the lemma with $x = \sqrt{k}$ shows that every edge in the spanning tree of each cluster has weight at most $\frac{OPT}{\sqrt{k}}$. Consider the scenario when the merge phase runs to completion to produce a tree with at least k nodes even before the number of exposed edges falls below $\sqrt{k}$. In this case, since the resulting tree has at most k nodes, the cost of the tree is at most $\frac{OPT}{\sqrt{k}} \cdot k \leq 2\sqrt{k} \cdot OPT$.

Otherwise, the number of exposed edges falls below $\sqrt{k}$ before the merge phase runs to completion. However, in this case, note that both preconditions for the start of the collect phase will have been satisfied. Hence the algorithm must have forked off a run of the collect phase. We show that the tree output by this run has low weight. Consider a shortest-path computation of the collect phase rooted at a cluster containing a node of the optimal kMST. Then clearly, within a distance at most OPT, we find at most $\sqrt{k}$ clusters whose sizes sum to at least k. Since the number of exposed edges is less than $\sqrt{k}$, the clusters containing nodes of the optimal tree form such a collection. Since there are at most $\sqrt{k}$ clusters to connect to, the weight of these connections is at most $\sqrt{k} \cdot OPT$. To complete the analysis we need to upper-bound the weight of the spanning trees within each of the clusters retained in the output solution. This is not hard since all edges in these clusters have weight at most $\frac{OPT}{\sqrt{k}}$ by Lemma 3.2. Since the size of the output tree is at most k (as a result of the pruning), the total weight of all the edges retained within these clusters is at most $\sqrt{k} \cdot OPT$. By summing the weight of these intracluster edges and the intercluster connections we show that the output tree has cost at most $2\sqrt{k} \cdot OPT$. This proves the performance ratio of $2\sqrt{k}$ claimed in Theorem 3.1.

The example in Fig. 2 shows that the performance ratio of the algorithm is $\Omega(\sqrt{k})$. The optimal kMST is the horizontal path, each edge of which has weight zero or $\frac{OPT}{\sqrt{k}}$. The horizontal path has $\sqrt{k}$ edges of weight $\frac{OPT}{\sqrt{k}}$ each. All zero-weight edges will be chosen first in the merge phase. The merge phase running to completion will extend each of the zero-weight upward-directed paths to include $\Omega(k)$ edges each of weight $\frac{OPT}{4\sqrt{k}}$ resulting in a tree of weight $\Omega(OPT \cdot \sqrt{k})$. The collect phases may output trees consisting of all the $(\sqrt{k}+1)$-sized clusters at the bottom of the figure, each of weight $\Omega(OPT \cdot \sqrt{k})$.

4. An approximation algorithm for points on the plane.

THEOREM 4.1. *There is a polynomial-time algorithm that, given n points in the Euclidean plane and a positive integer $k \leq n$, constructs a tree spanning at least k of these points such that the total length of the tree is at most $O(k^{1/4})$ times that of a minimum-length tree spanning any k of the points.*

In this section, we present a heuristic for the kMST problem for points on the plane and a proof of its performance guarantee. Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the given set of points. For any pair of points s_i and s_j, let $d(i,j)$ denote the Euclidean distance between s_i and s_j.

4.1. The heuristic.

I. **For** each distinct pair of points s_i, s_j in S **do**

(1) Construct the circle C with diameter $\delta = \sqrt{3}d(i,j)$ centered at the midpoint of the line segment $\langle s_i, s_j \rangle$.

(2) Let S_C be the subset of S contained in C. If S_C contains fewer than k points, skip to the next iteration of the loop (i.e., try the next pair of points). Otherwise, do the following.

(3) Let Q be the square of side δ circumscribing C.

(4) Divide Q into k square cells each with side $= \delta/\sqrt{k}$.

(5) Sort the cells by the number of points from S_C they contain and choose the minimum number of cells so that the chosen cells together contain at least k points. If necessary, arbitrarily discard points from the last chosen cell so that the total number of points in all the cells is equal to k.

(6) Construct a minimum spanning tree for the k chosen points. (For the rectilinear case, construct a rectilinear minimum spanning tree for the k chosen points.)

(7) The solution value for the pair $\langle s_i, s_j \rangle$ is the length of this MST.

II. **Output** the smallest solution value found.

It is easy to see that the above heuristic runs in polynomial time. In the next section, we show that the heuristic provides a performance guarantee of $O(k^{1/4})$. We

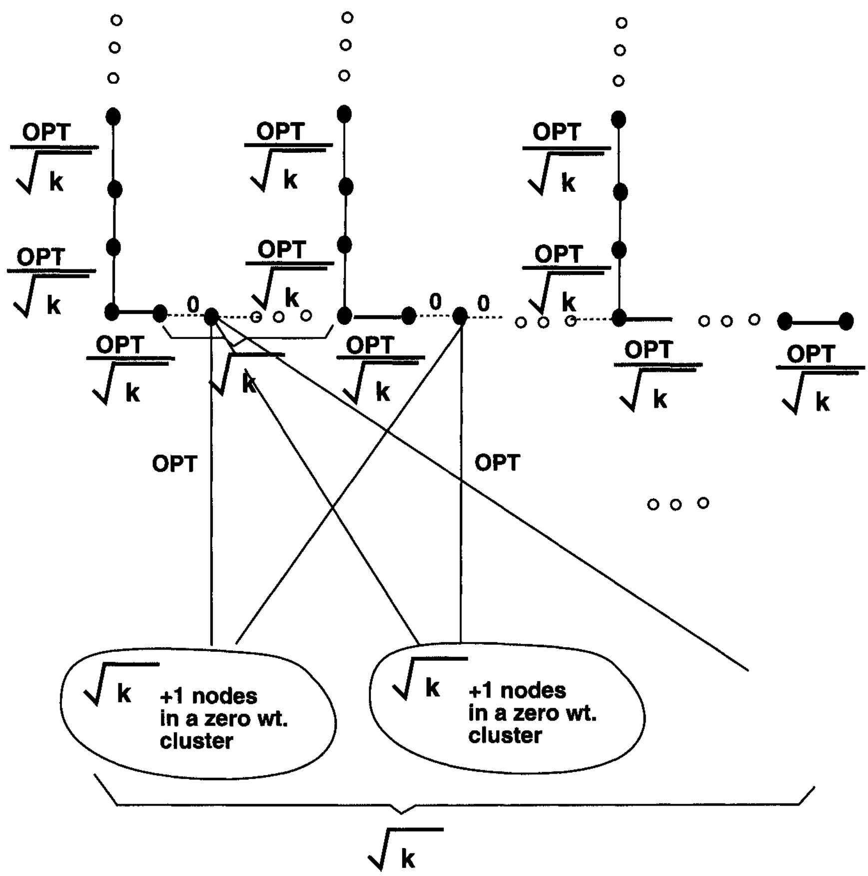

FIG. 2. *Example of a graph in which the algorithm in Theorem* 3.1 *outputs a tree of weight* $\Omega(OPT \cdot \sqrt{k})$.

begin with some lemmas.

4.2. The performance guarantee.

LEMMA 4.2. *Let S denote a set of points on the plane, with diameter Δ. Let a and b be two points in S such that $d(a,b) = \Delta$. Then the circle with diameter $\sqrt{3}\Delta$ centered at the midpoint of the line segment $\langle a, b\rangle$ contains S.*

Proof. Suppose there exists a point $p \in S$ not contained within the circle of diameter $\sqrt{3}\Delta$ centered at the midpoint of the line segment $\langle a, b\rangle$. If p lies on the perpendicular bisector of the line segment $\langle a, b\rangle$, then it is clear that $d(a,p) = d(b,p) > \Delta$, else p is closer to one of a and b than the other. Say p is closer to a; then it is easy to see that $d(b,p) > \Delta$. Thus, if there exists a point outside the circle, then it contradicts the fact that the diameter of the set S is Δ. Hence S must be contained within the circle. □

Lower bounds on an optimal kMST. The following lemma is used to establish a lower bound on OPT.

LEMMA 4.3. *Consider a square grid on the plane with the side of each cell being σ. Then the length of an MST for any set of t points, where each point is from a distinct cell, is $\Omega(t\sigma)$.*

Proof. Pick a point from the set and discard all points in the eight cells neighboring the cell containing the chosen point. Doing this repeatedly we choose a subcollection of $t/9 = \Omega(t)$ points such that the distance between any pair of points in the subcollection is at least σ. The lemma then follows from the observation that the minimum length of a tree spanning $\Omega(t)$ points that are pairwise σ-distant is $\Omega(t\sigma)$. $\square$

Let P^* denote the set of points in an optimal solution to the problem instance. Let Δ denote the *diameter* of P^* (i.e., the maximum distance between a pair of points in P^*) and OPT denote the length of an MST for P^*. Consider an iteration in which the circle constructed by the heuristic is defined by two points a and b in P^* such that $d(a,b) = \Delta$. Let g be the number of square cells used by the heuristic in selecting k points in this iteration. To establish the performance guarantee of the heuristic, we show that the length of the MST constructed by the heuristic during this iteration is within a factor $O(k^{1/4})$ of OPT.

It is easy to see that $OPT \geq \Delta$ because Δ is the diameter of P^*.

Since the heuristic uses a minimum number (g) of square cells in selecting k points, the points in P^* must occur in g or more square cells. Note that the side of each square cell is $\frac{\sqrt{3}\Delta}{\sqrt{k}}$. This gives us the following corollary to Lemma 4.3.

COROLLARY 4.4.

$$OPT = \Omega(\frac{g\Delta}{\sqrt{k}}).$$

Upper bound on the cost of the heuristic. We now prove an upper bound on the cost of the spanning tree returned by the heuristic. For this, we need the following lemma.

LEMMA 4.5. *The length of a minimum spanning tree for any set of q points in a square with side σ is length $O(\sigma\sqrt{q})$.*

Proof. Paste a square grid over the square where each subcell in the grid has side $\frac{\sigma}{\sqrt{q}}$. Connect each point to a closest vertex in the grid. Consider the tree consisting of one vertical line, all the horizontal lines in the grid connected to the vertical line, and the vertical lines connecting each point to its nearest horizontal line (see Fig. 3). It is clear that the grid lines in the tree have total length $O(\sigma\sqrt{q})$ and the lines connecting the points to the grid have total length $q \cdot O(\frac{\sigma}{\sqrt{q}}) = O(\sigma\sqrt{q})$. This is a Steiner tree. But, it is a simple matter to observe that a spanning tree of at most twice the length can be obtained by shortcutting the Steiner tree. $\square$

LEMMA 4.6. *The length of the spanning tree constructed by the heuristic is $O(\sqrt{g}\Delta)$.*

Proof. Let Q_i denote the set of points in the ith cell chosen by the heuristic, $1 \leq i \leq g$. Thus $\sum_{i=1}^{g} |Q_i| = k$. Consider the following two-stage procedure for constructing a spanning tree for the points in $\bigcup_{i=1}^{g} Q_i$.

Stage I. Construct a minimum spanning tree for the points in Q_i, $1 \leq i \leq g$. Note that the points in Q_i are within a square of side $\sqrt{3}\Delta/\sqrt{k}$. Using Lemma 4.5, the length of an MST for Q_i is $O(\frac{\Delta}{\sqrt{k}}\sqrt{|Q_i|})$. Thus, the total length of all the minimum spanning trees constructed in this stage is $O(\frac{\Delta}{\sqrt{k}}\sum_{i=1}^{g}\sqrt{|Q_i|}) = O(\sqrt{g}\ \Delta)$ by the Cauchy–Schwartz inequality.

Stage II. Connect the g spanning trees constructed in Stage I into a single spanning tree as follows. Choose a point arbitrarily from each Q_i ($1 \leq i \leq g$), and construct an MST for the g chosen points. Note that these g points are within a square of side $\sqrt{3}\ \Delta$. Thus, by Lemma 4.5, the length of the MST constructed in this stage is $O(\sqrt{g}\ \Delta)$ as well.

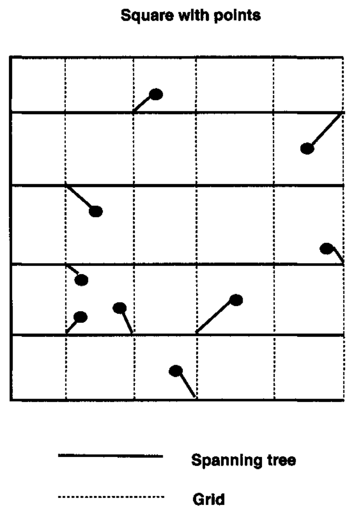

FIG. 3. *A spanning tree of length $O(\sigma\sqrt{q})$ on any q points in a square of side σ.*

Thus, the total length of the spanning tree constructed by the two-stage procedure is $O(\sqrt{g}\,\Delta)$. □

The final analysis. We are now ready to complete the proof of the performance bound. As argued above, $OPT = \Omega(\Delta)$, and from Corollary 4.4, $OPT = \Omega(\frac{g\Delta}{\sqrt{k}})$. Thus $OPT = \Omega(\max\{\Delta, \frac{g\Delta}{\sqrt{k}}\})$. Also from Lemma 4.6, the length of the spanning tree produced by the heuristic is $O(\sqrt{g}\ \Delta)$. Therefore, the performance ratio is $O(\min\{\sqrt{g}, \sqrt{k/g}\}) = O(k^{1/4})$ as claimed.

The example in Fig. 4 shows that the performance ratio of the heuristic is $\Omega(k^{1/4})$. The big square has side σ. Each cell of the square grid has side $\frac{\sigma}{\sqrt{k}}$. There are $\sqrt{k}$ points clustered closely together in each cell along the diagonal of the big square. In each of the $\sqrt{k}$ cells distributed uniformly throughout the big square there are $\sqrt{k}$ uniformly distributed points. The heuristic may pick up the points in the uniformly distributed cells, forming a tree of length $\Omega(\sigma \cdot k^{1/4})$, while the tree spanning the points along the diagonal has length $O(\sigma)$.

Observe that both our lower bounds on an optimal solution and the upper bound on the spanning tree obtained also apply to the case of constructing a rectilinear kMST. Hence it follows that the above approximation algorithm delivers a performance guarantee of $O(k^{1/4})$ for the rectilinear kMST problem too. This proves the following corollary.

COROLLARY 4.7. *There is a polynomial-time algorithm that, given n points in the plane and a positive integer $k \leq n$, constructs a rectilinear tree spanning at least k of these points such that the total length of the tree is at most $O(k^{1/4})$ times that of a minimum-length rectilinear tree spanning any k of the points.*

5. Polynomially solvable special cases.

5.1. kMST for treewidth-bounded (or decomposable) graphs. In this section, we give the details of our polynomial-time algorithm for the class of treewidth-

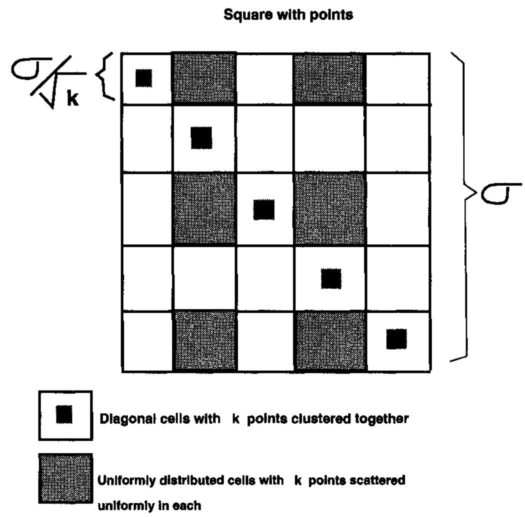

FIG. 4. *Example of a configuration of points on the plane in which the heuristic outputs a tree of length* $\Omega(OPT \cdot \sqrt{k})$.

bounded graphs. As mentioned earlier Arnborg et al. [5] showed that the class of treewidth-bounded graphs is the same as the class of decomposable graphs defined by Bern, Lawler, and Wong [8]. We use the characterization of Bern, Lawler, and Wong to explain our algorithm.

THEOREM 5.1. *For any class of decomposable graphs, there is an* $O(nk^2)$*-time algorithm for solving the* k*MST problem.*

In this section, we prove the above theorem. A class of *decomposable graphs* Γ is inductively defined as follows [8].

1. The number of primitive graphs in Γ is finite.
2. Each graph in Γ has an ordered set of special nodes called *terminals*. The number of terminals in each graph is bounded by a constant.
3. There is a finite collection of binary composition rules that operate only at terminals, either by identifying two terminals or adding an edge between terminals. A composition rule also determines the terminals of the resulting graph, which must be a subset of the terminals of the two graphs being composed.

Examples of decomposable graphs include trees, series-parallel graphs, and bounded-bandwidth graphs [8].

Let Γ be any class of decomposable graphs. The kMST problem for Γ can be solved optimally in polynomial time using dynamic programming. Following [8], it is assumed that a given graph G is accompanied by a parse tree specifying how G is constructed using the rules and that the size of the parse tree is linear in the number of nodes of G.

Consider a fixed class of decomposable graphs Γ. Suppose that G is a graph in Γ. Let π be a partition of a nonempty subset of the terminals of G. We define the following set of costs for G.

$Cost_i^{\pi}(G)$ = Minimum total cost of any forest containing a tree for each block of π, such that the terminal nodes occurring in each tree are exactly the members of the corresponding block of π, no pair of trees is connected, and the total number of edges in the forest is i $(1 \leq i < k)$.

$Cost_{k-1}^{\emptyset}(G)$ = Minimum cost of a tree within G containing $k-1$ edges and *no* terminal nodes of G.

For any of the above costs, if there is no forest satisfying the required conditions, the value of *Cost* is defined to be ∞.

Note that because Γ is fixed, the number of cost values associated with any graph in the parse tree for G is $O(k)$. We now show how the cost values can be computed in a bottom–up manner, given the parse tree for G.

To begin with, since Γ is fixed, the number of primitive graphs is finite. For a primitive graph, each cost value can be computed in constant time, since the number of forests to be examined is fixed. Now consider computing the cost values for a graph G constructed from subgraphs G_1 and G_2, where the cost values for G_1 and G_2 have already been computed.

Let Π_{G_1}, Π_{G_2}, and Π_G be the set of partitions of a subset of the terminals of G_1, G_2, and G, respectively. Let A be the set of edges added to G_1 and G_2 by the composition rule R used in constructing G from G_1 and G_2. Corresponding to rule R, there is a partial function $f_R : \Pi_{G_1} \times \Pi_{G_2} \times 2^A \rightarrow \Pi_G$, such that a forest corresponding to partition π_1 in Π_{G_1}, a forest corresponding to partition π_2 in Π_{G_2}, and a subset $B \subseteq A$ combine to form a forest corresponding to partition $f_R(\pi_1, \pi_2, B)$ of G. Furthermore, if the forest corresponding to π_1 contains i edges and the forest corresponding to π_2 contains j edges, then the combined forest in G contains $i+j+|B|$ edges.

Similarly, there is a partial function $g_R : \Pi_{G_1} \times 2^A \rightarrow \Pi_G$, such that a forest corresponding to partition π_1 in Π_{G_1} and a subset $B \subseteq A$ combine to form a forest corresponding to partition $g_R(\pi_1, B)$ of G. If the forest corresponding to π_1 contains i edges, then the combined forest in G contains $i + |B|$ edges. There is also a similar partial function $h_R : \Pi_{G_2} \times 2^A \rightarrow \Pi_G$. Finally, there is a partial function $j_R : 2^A \rightarrow \Pi_G$.

Using functions f_R, g_R, h_R, and j_R, cost values for G can be computed from the set of cost values for G_1 and G_2. For instance, suppose that $f_R(\pi_1, \pi_2, B) = \pi$. Then a contributor to computing $Cost_i^{\pi}(G)$ is $Cost_t^{\pi_1}(G_1) + Cost_{i-t-|B|}^{\pi_2}(G_2) + w(B)$, for each t such that $1 \leq t \leq i - |B| - 1$. Here $w(B)$ is the total cost of all edges in B. The value of $Cost_i^{\pi}(G)$ is the minimum value among its contributors.

When all the cost values for the entire graph G have been computed, the cost of an optimal kMST is equal to $\min_{\pi \in \pi_G}\{Cost_{k-1}^{\pi}(G)\}$, where the forest corresponding to π consists of a single tree.

We now analyze the running time of the algorithm. For each graph occurring in the parse tree, there are $O(k)$ cost values to be computed. Each of the cost values can be computed in $O(k)$ time. As in [8], we assume that the size of the given parse tree for G is $O(n)$. Then the dynamic-programming algorithm takes time $O(nk^2)$. This completes the proof of Theorem 5.1.

It is also easy to see that a straightforward extension of the above algorithm works for the weighted case, when the edges of noninfinite weight form a decomposable graph.

5.2. kMST for points on the boundary of a convex region.

THEOREM 5.2. *There is a polynomial-time algorithm for solving the kMST problem for the case of points in the Euclidean plane that lie on the boundary of a convex region.*

We now restrict our attention to the case where we are given n points that lie on the boundary of a convex region and show that the kMST on these points can be computed in polynomial time using dynamic programming. We also provide a faster algorithm if the points are constrained to lie on the boundary of a circle.

LEMMA 5.3. *Any optimal kMST for a set of points in the plane is non-self-intersecting.*

Proof. Suppose an optimal kMST is self-intersecting; then let $\langle a, b\rangle$ and $\langle c, d\rangle$ be the intersecting line segments. On removing the edges $\langle a, b\rangle$ and $\langle c, d\rangle$ from the kMST we get three connected components; hence some two vertices, one from $\{a, b\}$ and one from $\{c, d\}$, must be in the same connected component. Say a and d are in the same connected component; then since in any convex quadrilateral the sum of two opposite sides is less than the sum of the two diagonals, replacing $\langle a, b\rangle$ and $\langle c, d\rangle$ by $\langle a, c\rangle$ and $\langle b, d\rangle$ we still get a tree spanning k nodes but with less weight. This contradicts the fact that the kMST with which we started out was optimal. Hence any optimal kMST on a set of points in the plane must be non-self-intersecting. □

LEMMA 5.4. *Given n points on the boundary of a convex polygon, no vertex in an optimal kMST of these points has degree greater than 4.*

Proof. Suppose there is a vertex v in an optimal kMST with degree greater than 4. Let $v_1, v_2, \ldots, v_d, d \geq 5$, be its neighbors in the optimal kMST as shown in Fig. 5. Using the well-known fact that any convex polygon lies entirely on one side of a supporting line, we have that $\angle v_1 v v_d \leq 180°$. By the pigeon-hole principle, there is an i such that $\angle v_i v v_{i+1} \leq 180°/(d-1) < 60°, 1 \leq i \leq d-1$, since d is at least 5. Thus in $\triangle v_i v v_{i+1}$, $\angle v_i v v_{i+1}$ is not the largest angle and $v_i v_{i+1}$ is not the largest side. Therefore replacing the larger of vv_i and vv_{i+1} in the optimal kMST with $v_i v_{i+1}$ we obtain a tree with lesser weight, contradicting the assumption that the kMST was optimal. This completes the proof. □

We now characterize the structure of an optimal solution in the following decomposition lemma and use it to define the subproblems that we need to solve recursively using dynamic programming. The next lemma intuitively points out that an optimal solution for the kMST problem for the whole polygon can be constructed from optimal solutions for smaller polygons obtained by triangulating the original polygon.

LEMMA 5.5 (decomposition lemma). *Let $v_0, v_1, \ldots, v_{n-1}$ be the vertices of a convex polygon in, say, clockwise order. Let v_i be a vertex of degree d_i in an optimal kMST. Note that $1 \leq d_i \leq 4$.*

If $d_i \geq 2$ let the removal of v_i from the optimal kMST produce connected components $C_1, C_2, \ldots, C_{d_i}$ (see Fig. 6). Let $|C_i|$ denote the number of vertices in component C_i. Then there exists a partition of $v_{i+1}, v_{i+2}, \ldots, v_{i-1}$ (indices taken mod n*), into d_i contiguous subsegments $S_1, S_2, \ldots, S_{d_i}$ such that $\forall j, 1 \leq j \leq d_i$, the optimal kMST induced on $S_j \bigcup \{v_i\}$ is an optimal $(|C_j| + 1)MST$ on $S_j \bigcup \{v_i\}$ among all such trees in which the degree of v_i is one.*

If $d_i = 1$, let v_j be v_i's neighbor in the optimal kMST. Let v_j be adjacent to d_{j1} vertices in $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ and d_{j2} vertices in $v_{j+1}, v_{j+2}, \ldots, v_{i-1}$. Let the optimal kMST contain $|C_1|$ vertices from the set $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ and $|C_2|$ vertices from the set $v_{j+1}, v_{j+2}, \ldots, v_{i-1}$. Then the optimal kMST induced on $v_{i+1}, v_{i+2}, \ldots, v_j$ is an optimal $(|C_1|+1)MST$ on $v_{i+1}, v_{i+2}, \ldots, v_j$ with degree of $v_j = d_{j1}$ and the optimal

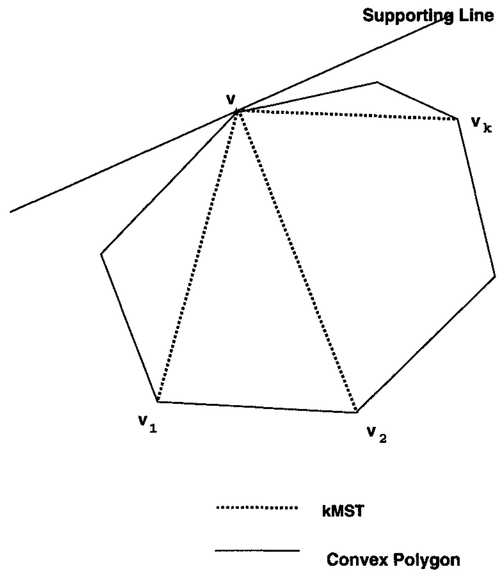

FIG. 5. *Points on a convex polygon.*

kMST induced on $v_j, v_{j+1}, \ldots, v_{i-1}$ is an optimal $(|C_2|+1)MST$ on $v_j, v_{j+1}, \ldots, v_{i-1}$ among all such trees with degree of $v_j = d_{j2}$.

Proof. If $d_i \geq 2$, then it is easy to see that a partition of $v_{i+1}, v_{i+2}, \ldots, v_{i-1}$ into contiguous subsegments $S_1, S_2, \ldots, S_{d_i}$ exists such that $\forall j, 1 \leq j \leq d_i, C_j \subset S_j$, because the optimal kMST is non-self-intersecting by Lemma 5.3. Further, the optimal kMST induced on $S_j \bigcup \{v_i\}$ must be an optimal $(|C_j|+1)$MST on $S_j \bigcup \{v_i\}$ with degree of $v_i = 1$, for otherwise we could replace it getting a lighter kMST. The proof of the case when $d_i = 1$ is equally straightforward and is omitted. □

Thus the subproblems we consider are specified by the following four parameters: a size s, a vertex v_i, the degree d_i of v_i, and a contiguous subsegment $v_{k1}, v_{k1+1}, \ldots, v_{k2}$ such that $i \notin [k1 \ldots k2]$. A solution to such a subproblem denoted by $SOLN(s;\, v_i;\, d_i;\, v_{k1}, v_{k1+1}, \ldots, v_{k2})$ is the weight of an optimal sMST on $\{v_i, v_{k1}, v_{k1+1}, \ldots, v_{k2}\}$ in which v_i has degree d_i. Using the decomposition lemma above, we can write a simple recurrence relation for $SOLN(s; v_i; d_i; v_{k1}, v_{k1+1}, \ldots, v_{k2})$ as

$SOLN(s; v_i; d_i; v_{k1}, v_{k1+1}, \ldots, v_{k2}) =$

$$\begin{cases} \infty : \text{ if } d_i = 0 \text{ or } s < d_i + 1 \text{ or } ((k2 - k1 + 1) \bmod n) + 1 < s, \\ \min\limits_{k'_0 = k1 < k'_1 \cdots < k'_{d_i} = k2} \quad \min\limits_{s_1 \cdots + s_{d_i} = s + d_i - 1, s_j \geq 1} \quad \Sigma_{1 \leq j \leq d_i} SOLN(s_j; v_i; 1; v_{k'_{j-1}}, \ldots, v_{k'_j}) \\ \qquad : \text{ if } d_i \geq 2, \\ \min\limits_{j_0 = k1 \leq j_1 \leq j_2 = k2} \{w(v_i v_{j_1}) + \min_{0 \leq d_1 + d_2 \leq 3} \min_{s_1 + s_2 = s} \\ (SOLN(s_1; v_{j_1}; d_1; v_{j_0}, \ldots, v_{j_1 - 1}) + SOLN(s_2; v_{j_1}; d_2; v_{j_1+1}, \ldots, v_{j_2}))\}) \,:\, \text{if } d_i = 1. \end{cases}$$

Here $w(v_i v_j)$ is the cost of the edge (v_i, v_j). The optimal kMST is expressed as

$$\min_{1 \leq i \leq n} \min_{1 \leq d \leq 4} SOLN(k; v_i; d; v_{i+1}, v_{i+2}, \ldots, v_{i-1}).$$

Note that we have $O(kn^3)$ subproblems and each subproblem requires looking up the solution to at most $O(k^3n^3)$ smaller subproblems. This yields a running

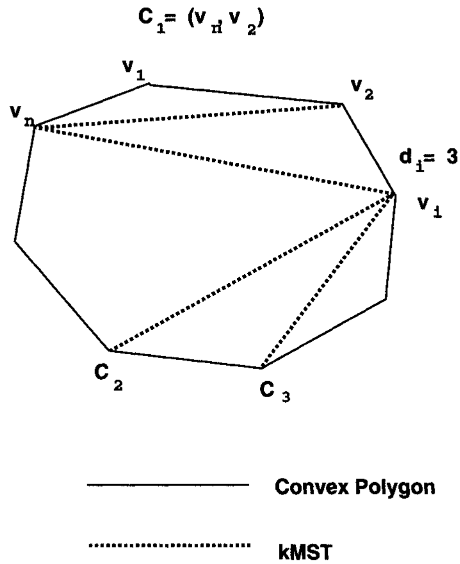

FIG. 6. *Decomposition.*

time of $O(k^4n^6)$. When $k = \Omega(\sqrt{n})$, this running time can be further improved by organizing the computation of the recurrences for the smaller subproblems better. Each subproblem specified by s, v_i, d_i, and the interval $v_{k1}, \ldots, v_{k2}$ can be solved by first computing a partition of the interval into at most four subintervals (exactly four when $d_i = 4$). For the first subinterval, we compute the best tree with $j-1$ nodes from this subinterval and containing v_i so that it has degree one in this tree, for $1 \leq j \leq s$. This computation takes $O(nk)$ time since there are at most $s \leq k$ trees to be computed, and for each j there are at most n nodes with which v_i shares the single edge in the best tree. Next, we include the next subinterval and compute for $1 \leq j' \leq s$ the best tree on $j'-1$ nodes containing v_i and nodes from these two subintervals, where v_i have degree two with one edge to a node in the first and one edge to a node in the second subinterval. This set of trees can also be computed in $O(nk)$ time given the set of trees for the first subinterval as follows: First, compute the best tree on j nodes for $1 \leq i \leq s$ containing v_i and nodes only in the second subinterval, where v_i has exactly one edge to a node in this subinterval, in $O(nk)$ time as before. Using these values and the analogous set of values for the first subinterval, the best j' trees for the first two subintervals can be obtained in $O(k^2) = O(nk)$ time since each of the $s \leq k$ trees requires looking up at most s different pairs of trees, one from each subinterval. This method can be extended to compute the solution for the whole set of four subintervals in $O(nk)$ time. Since there are $O(n^3)$ ways to partition a given interval into four subintervals, the recurrence for this subproblem can be solved in $O(kn^4)$ time. So the total time to solve one subproblem is $O(kn^4)$ time. Since there are a total of $O(kn^3)$ subproblems, the total running time of the algorithm is $O(k^2n^7)$.

We now provide a faster algorithm to find the optimal kMST in the case when all n points lie on a circle. We assume that no two points are diametrically opposite.

LEMMA 5.6. *Given n points $v_1, v_2, \ldots, v_n$ on a circle, no vertex in an optimal kMST has degree more than 2.*

Proof. Suppose point v_p in an optimal kMST has degree greater than 2. Then consider the diameter passing through v_p. At least two neighbors of v_p lie on one side of this diameter. Let these neighbors be v_q and v_r, where v_q is closer to v_p than v_r. Then since $\angle v_p v_q v_r$ is obtuse, we replace $v_p v_r$ by $v_q v_r$ to get a smaller tree. □

Lemma 5.6 implies that if the points lie on a circle, then every optimal kMST is a path. Moreover, if the path "zig-zags," then we replace the crossing edge with a smaller edge. Thus we have the following lemma.

LEMMA 5.7. *Given n points $v_1, v_2, \ldots, v_n$ on a circle, let a minimum length k-path on these points be $v_{i_1}, \ldots, v_{i_p}$. Then the line segment joining v_{i_1} and v_{i_p} along with the k-path forms a convex k-gon.*

Proof. By Lemma 5.6 the minimum-length k-path is also the minimum-length kMST. Suppose the line segment joining v_{i_1} and v_{i_p} along with the minimum k-path does not form a convex k-gon. Then there exists a zig-zag in the path as shown in Fig. 7. Say the center of the circle lies to the right of the edge $\langle a, b\rangle$; then we replace $\langle a, b\rangle$ by the edge $\langle b, c\rangle$ to get a smaller kMST, which contradicts the fact that the k-path with which we started was optimal. □

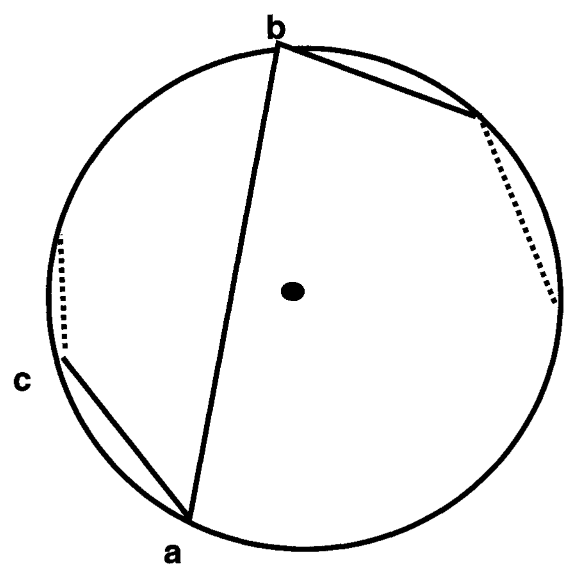

FIG. 7. *Illustration of Lemma* 5.7.

Lemmas 5.6 and 5.7 lead to a straightforward dynamic-programming algorithm to compute an optimal kMST for points on a circle: for each point on the circle compute the minimum-length i-path $(1 \leq i \leq k)$, which lies completely on one side of the diameter passing through the point; then combine these solutions to find the optimal kMST. It is easy to see this algorithm takes $O(k^2 n)$ time.

6. Short trees and short small trees.

6.1. Short trees. In this subsection, we prove our results on short trees. First, we address the minimum-diameter k-tree problem: given a graph with nonnegative edge weights, find a tree of minimum diameter spanning at least k nodes.

THEOREM 6.1. *There is a polynomial-time algorithm for the minimum-diameter k-tree problem on graphs with nonnegative edge weights.*

Recall that the diameter of a tree is the maximum distance (path length) between any pair of nodes in the tree. We introduce the notion of subdividing an edge in a weighted graph. A subdivision of an edge $e = (u, v)$ of weight w_e is the replacement of e by two edges $e_1 = (u, r)$ and $e_2 = (r, v)$ where r is a new node. The weights of e_1 and e_2 sum to w_e. Consider a minimum-diameter k-tree. Let x and y be the endpoints of a longest path in the tree. The weight of this path, D, is the diameter of the tree. Consider the midpoint of this path between x and y. If it falls in an edge, we subdivide the edge by adding a new vertex as specified above. The key observation is that there exist at least k vertices at a distance at most $D/2$ from this midpoint. This immediately motivates an algorithm for the case when the weights of all edges are integral and bounded by a polynomial in the number of nodes. In this case, all such potential midpoints lie in half-integral points along edges of which there are only a polynomial number. Corresponding to each candidate point, there is a smallest distance from this point within which there are at least k nodes. We choose the point with the least such distance and output the breadth-first search (bfs) tree rooted at this point appropriately truncated to contain only k nodes.

When the edge weights are arbitrary, the number of candidate midpoints are too many to check in this fashion. However, we use a graphical representation of the distance of any node from any point along a given edge to bound the search for candidate points. We think of an edge $e = (u, v)$ of weight w as a straight line between its endpoints of length w. For any node x in the graph, consider the shortest path from x to a point along the edge e at distance ℓ ($\leq w$) from u. The length of this path is the minimum of $\ell + d(x, u)$ and $w - \ell + d(v, x)$. We plot this distance of the node x as a function of ℓ. The resulting plot is a piecewise linear bitonic curve that we call the roof curve of x in e (see Fig. 8). For each edge e, we plot the roof curves of all the vertices of the graph in e. For any candidate point in e, the minimum diameter of a k-tree centered at this point can be determined by projecting a ray upward from this point in the plot and determining the least distance at which it intersects the roof curves of at least k distinct nodes. The best candidate point for a given edge is one with the minimum such distance. Such a point can be determined by a simple line-sweep algorithm on the plot. Determining the best midpoint over all edges gives the midpoint of the minimum-diameter k-tree. This proves Theorem 6.1.

The following lemma gives yet another way to implement the polynomial-time algorithm for finding a tree of minimum diameter spanning k nodes.

LEMMA 6.2. *Given two vertices in a graph, v_i and v_j, such that every other vertex is within distance d_i of v_i or d_j of v_j, it is possible to find two trees, one rooted at v_i and of depth at most d_i and one rooted at v_j of depth at most d_j, that partition the set of all vertices.*

Proof. Consider the shortest-path trees T_i and T_j rooted at v_i and v_j of depth d_i and d_j, respectively. Every vertex occurs in one tree or both trees. Consider a vertex v_p that occurs in both trees. If it is the case that $d_i - \text{depth}_{T_i}(v_p)$ is greater than $d_j - \text{depth}_{T_j}(v_p)$, then the same is true of all descendants of v_p in T_j. Hence we can remove v_p and all its descendants from T_j since we are guaranteed that all these vertices occur in T_i. Repeating this procedure bottom-up we get two trees satisfying the required conditions and partitioning the vertex set. $\square$

The above lemma motivates the following alternate algorithm for finding a minimum-diameter tree spanning at least k nodes. For each vertex v_i in the graph compute the shortest distance d_i such that there are k vertices within distance d_i of v_i. For each edge (v_i, v_j) compute the least $d_{ij}^i + d_{ij}^j$ such that there are k vertices within distance

d^i_{ij} of v_i or d^j_{ij} of v_j. Then compute the least of all the d_i's and $d^i_{ij} + d^j_{ij} + w(v_i, v_j)$'s, and this is the diameter of the k-tree with least diameter. It can be easily seen that the running time of the algorithm is $O(\min\{k^2, E\}E)$.

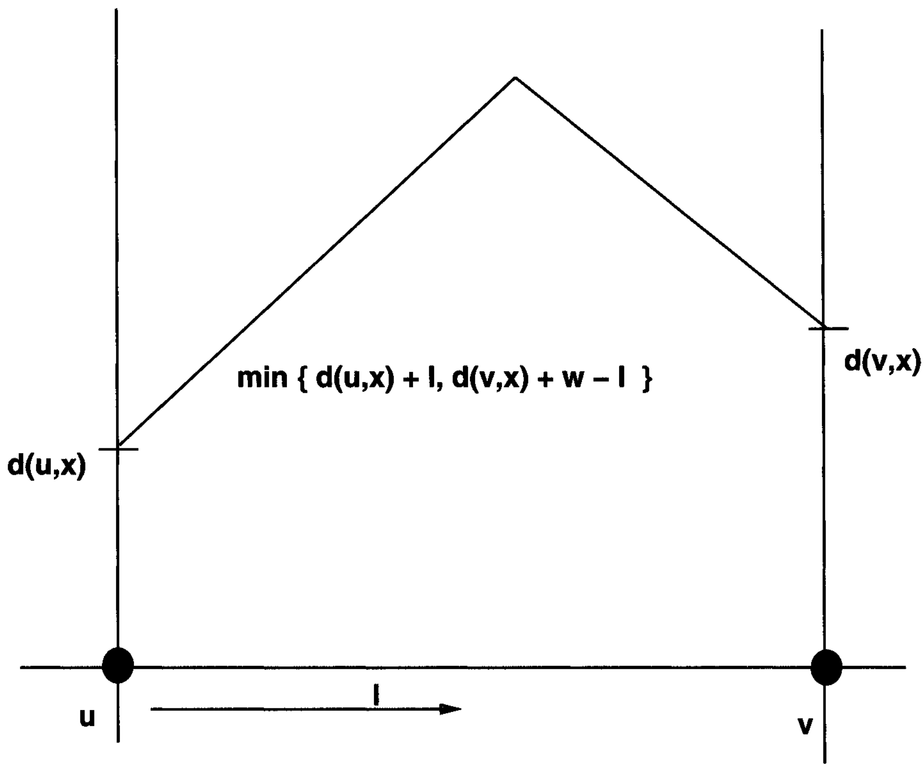

FIG. 8. *A roof curve of a node x in edge $e = (u, v)$.*

We now address the results in the third row of Table 1.

LEMMA 6.3. *If the r_{ij} values are drawn from the set $\{a, b\}$ and the d_{ij} values from $\{0, c\}$, then the minimum-communication-cost spanning tree can be computed in polynomial time.*

Proof. When the d_{ij} values are all uniform, Hu [22] observed that the Gomory–Hu cut tree with the r_{ij} values as capacities is a minimum-communication-cost tree. We can use this result to handle the case when zero-cost d_{ij} edges are allowed as well. We contract the connected components of the graph using zero-cost d_{ij} edges into supernodes. The requirement value r_{IJ} between two supernodes v_I and v_J is the sum of the requirement values r_{ij} such that $i \in v_I$ and $j \in v_J$. Now we find a Gomory–Hu cut tree between the supernodes using the r_{IJ} values as capacities. By choosing an arbitrary spanning tree of zero-d_{ij}-valued edges within each supernode and connecting them to the Gomory–Hu tree, we get a spanning tree of the whole graph. It is easy to verify that this is a minimum-communication-cost spanning tree in this case. □

LEMMA 6.4. *When all the d_{ij} values are uniform and there are at most two distinct r_{ij} values (say a and b), then the minimum-diameter-cost spanning tree can be computed in polynomial time.*

Proof. Let the higher of the two r_{ij} values be a. If the edges with requirement a

form a cyclic subgraph, then any spanning tree has diameter cost $2a$. In this case, any spanning star (a star is a rooted tree of depth 1) is an optimal solution. Otherwise, consider the forest of edges with requirement a. Determine a center for each tree in this forest. Consider the tree formed by connecting these centers in a star. The root of the star is a center of the tree of largest diameter in the forest. If the diameter cost of the resulting tree is less than $2a$, it is easy to see that this tree has optimum diameter cost. Otherwise any star tree on all the nodes has diameter cost $2a$ and is optimal. Note that we can extend this solution to allow zero-cost d_{ij} edges by using contractions as before. □

Now we address the results in the fourth row of Table 1.

LEMMA 6.5. *The minimum-diameter-cost spanning tree problem is NP-complete even when the r_{ij}'s and d_{ij}'s take on at most two distinct values.*

Proof. It is easy to see that the minimum-diameter-cost spanning tree problem is in NP. We now prove that it is NP-hard by using a reduction from an instance of 3SAT. Without loss of generality, we assume that all clauses in the given instance of 3SAT contain three distinct literals. We form a graph that contains a special node t (the "true" node), a node for each literal and each clause. We use two d_{ij} values, c and $5c$ where we assume $c \neq 0$. Each literal is connected to its negation with an edge of distance c. The true node is connected to every literal with an edge of distance c. Each clause is connected to the three literals that it contains with edges of distance c. All other edges in the graph have distance $5c$. Now we specify the requirements on the edges. We use requirement values from $\{a, 4a\}$, where $a \neq 0$. The requirement value of an edge between a literal and its negation is $4a$. The requirement value of all other edges is a (see Fig. 9). It is easy to check that there exists a spanning tree of this graph with diameter cost at most $4ac$ if and only if the 3SAT formula is satisfiable. □

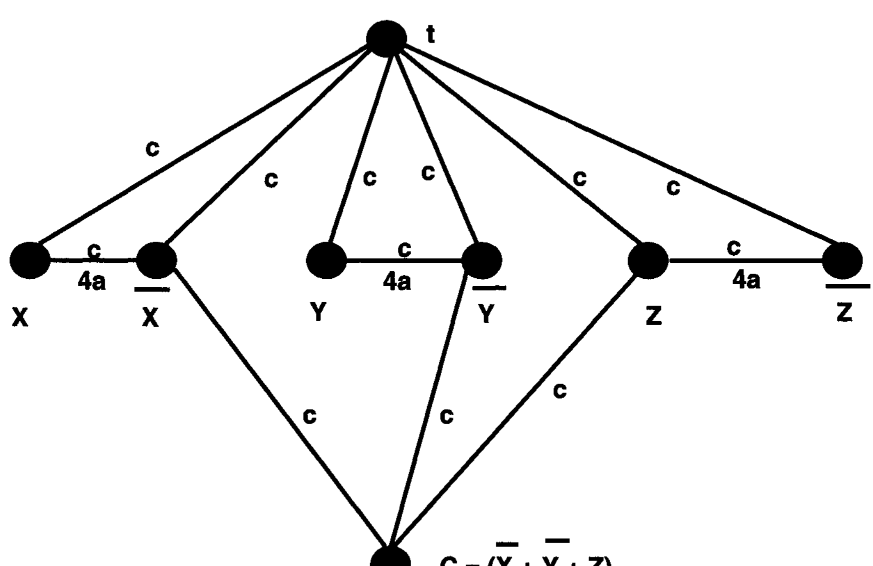

FIG. 9. *Reduction from an instance of 3SAT to the minimum-diameter-cost spanning tree problem.*

6.2. Short small trees.

THEOREM 6.6. *The minimum-communication k-tree problem and the minimum-diameter k-tree problem are both NP-hard to approximate within any factor even when all the d_{ij} values are one and the r_{ij} values are nonnegative.*

Proof. We prove the above theorem for the communication tree case. The proof of the other part is similar. Suppose there is a polynomial-time M-approximation algorithm for the minimum-communication-cost k-tree problem where all the d_{ij} values are one and all r_{ij} values are nonnegative. Then, we show that the k-independent set problem can be solved in polynomial time. The latter problem is well known to be NP-complete [19]. Given a graph G of the k-independent set problem, produce the following instance of the communication k-tree problem: $d_{ij} = 1$ for every pair of nodes i, j; assign r_{ij} = one if (i, j) is *not* an edge in G and $Mk(k-1)+1$ otherwise. If G has an independent set of size k, then we form a star on these k nodes (choosing an arbitrary node as the root). In the star, the distance between any pair of nodes is at most 2 and the r value for each pair is 1. Thus, the communication cost of an optimum solution is at most $k(k-1)$. The approximation algorithm will return a solution of cost at most $Mk(k-1)$. The nodes in this solution are independent in G by the choice of r_{ij} for nonedges $(i, j) \in G$. On the other hand, if there is no independent set of size k in G, the communication cost of any k-tree is greater than $Mk(k-1)$. □

7. Closing remarks.

7.1. Future research. A natural question is whether there are approximation algorithms for the kMST problem that provide better performance guarantees than those presented in this paper. In this direction, Garg and Hochbaum [20] gave an $O(\log k)$-approximation algorithm for the kMST problem for points on the plane using an extension of our lower-bounding technique in §4. Blum, Chalasani, and Vempala [10] recently improved upon this to obtain a constant-factor approximation for points on the plane. Also, Awerbuch, Azar, Blum, and Vempala [1] obtained an $O(\log^2 k)$-approximation algorithm for the kMST problem. An interesting observation in this regard is the following: any edge in an optimal kMST is a shortest path between its endpoints. This observation allows us to assume without loss of generality that the edge weights on the input graph obey the triangle inequality. Although we have been unable to exploit the triangle inequality property in our algorithms, it is possible that this remark holds the key to improving our results.

Table 1 is incomplete. It would be interesting to know the complexity of the minimum-diameter-cost spanning tree problem when the distance values are uniform. Note that any star tree on the nodes provides a 2-approximation to the minimum-diameter-cost spanning tree in this case. The above problem can be shown to be polynomial-time equivalent to the following tree reconstruction problem: given integral nonnegative distances d_{ij} for every pair of vertices i, j, does there exist a spanning tree on these nodes such that the distance between i and j in the tree is at most d_{ij}?

7.2. Maximum acyclic subgraph. In the course of our research we considered the k-forest problem: given an undirected graph is there a set of k nodes that induces an acyclic subgraph? The optimization version of this problem is the maximum acyclic subgraph problem. Since this problem is complementary to the minimum feedback vertex set problem [19], NP-completeness follows. While the feedback vertex set problem is 4-approximable [7], we show that the maximum acyclic subgraph problem is hard to approximate within a reasonable factor using an approximation-preserving transformation from the maximum independent set problem [6]. This same result was also derived in a more general form in [27].

THEOREM 7.1. *There is a constant $\epsilon > 0$ such that the maximum acyclic subgraph problem cannot be approximated within a factor $\Omega(n^\epsilon)$ unless $P = NP$.*

Proof. Note that any acyclic subgraph of size S contains a maximum independent set of size at least $S/2$ since acyclic subgraphs are bipartite and each partition is an independent set. Further, every independent set is also an acyclic subgraph. These two facts show that the existence of a ρ-approximation algorithm for the maximum acyclic subgraph problem implies the existence of a 2ρ-approximation algorithm for the maximum independent set problem. But by the result in [6] we know that there is a constant $\epsilon > 0$ such that the maximum independent set problem cannot be approximated within a factor $\Omega(n^\epsilon)$ unless $P = NP$. Hence, the same is true of the maximum acyclic subgraph problem. □

Acknowledgments. The authors thank Alex Zelikovsky and Naveen Garg for helpful conversations during the initial stages of the paper. They are grateful to Professor John Oomen for observing that our algorithm for points in the plane extends to the rectilinear case and to Professor Arie Tamir for his observations on §6. We thank the referees for detailed comments and suggestions that substantially improved the quality of presentation.

REFERENCES

[1] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, in Proc. 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 277–283.

[2] D. Adolphson and T. C. Hu, *Optimal linear ordering*, SIAM J. Appl. Math., 25 (1973), pp. 403–423.

[3] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, *Finding k points with minimum diameter and related problems*, J. Algorithms, 12 (1991), pp. 38–56.

[4] A. Agrawal, P. Klein, and R. Ravi, *When trees collide: an approximation algorithm for the generalized Steiner tree problem on networks*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, New Orleans, LA, 1991, pp. 134–144.

[5] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese, *An algebraic theory of graph reduction*, J. Assoc. Comput. Mach., 40 (1993), pp. 1134–1164.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, in Proc. 33rd IEEE Symposium on the Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 14–23.

[7] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth, *Approximation algorithms for the cycle-cover problem with applications to constraint satisfaction and Bayesian inference*, in Proc. Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 344–354.

[8] M. W. Bern, E. L. Lawler, and A. L. Wong, *Linear time computation of optimal subgraphs of decomposable graphs*, J. Algorithms, 8 (1987), pp. 216–235.

[9] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proc. 26th Annual ACM Symposium on Theory of Computing, Montreal, Canada, 1994, pp. 163–172.

[10] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, in Proc. 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 294–302.

[11] H. L. Bodlaender, *Dynamic programming on graphs of bounded treewidth*, in Proc. 15th International Colloquium on Automata, Languages and Programming, Springer-Verlag, Berlin, New York, NY, 1988, pp. 105–118.

[12] P. M. Camerini and G. Galbiati, *The bounded path problem*, SIAM J. Alg. Discrete Methods, 3 (1982), pp. 474–484.

[13] P. M. Camerini, G. Galbiati, and F. Maffioli, *Complexity of spanning tree problems: Part 1*, European J. Oper. Res., 5 (1980), pp. 346–352.

[14] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Report 338, GSIA, CMU, Pittsburgh, PA, 1976.

[15] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[16] D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, *Finding smallest polygons*, in Advances in Computing Research 1, JAI Press, Greenwich, CT, 1983, pp. 181–214.

[17] D. Eppstein, *New algorithms for minimum area k-gons*, in Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 83–88.

[18] D. Eppstein and J. Erickson, *Iterated nearest neighbors and finding minimal polytopes*, in Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, 1993, pp. 64–73.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[20] N. Garg and D. Hochbaum, *An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane*, in Proc. 26th ACM Symposium on Theory of Computing, Montreal, Canada, 1994, pp. 432–438.

[21] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, SIAM J. Appl. Math., 9 (1961), pp. 551–570.

[22] T. C. Hu, *Optimum communication spanning trees*, SIAM J. Comput., 3 (1974), pp. 188–195.

[23] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan, *The complexity of the network design problem*, Networks, 8 (1978), pp. 279–285.

[24] P. Klein and R. Ravi, *A nearly best-possible approximation for node-weighted Steiner trees*, in Proc. 3rd MPS conference on Integer Programming and Combinatorial Optimization, 1993, pp. 323–332.

[25] J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 7 (1956), pp. 48–50.

[26] D. Lozovanu and A. Zelikovsky, *Minimal and bounded tree problems*, Tezele Congresului XVIII al Academiei Romano-Americane, Kishniev, 1993, p. 25.

[27] C. Lund and M. Yannakakis, *On the hardness of the maximum subgraph problems*, in Proc. 20th International Colloquium on Automata, Languages and Programming, Lund, Sweden, 1993, Springer-Verlag, Berlin, New York, NY, pp. 40–51.

[28] R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.

[29] R. Ravi, *Steiner Trees and Beyond: Approximation Algorithms for Network Design*, Ph.D. Thesis, Tech. report TR-CS-93-41, Department of Computer Science, Brown University, Providence, RI, September 1993.

[30] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H.B. Hunt III, *Many birds with one stone: multi-objective approximation algorithms*, in Proc. 25th Annual ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 438–447.

[31] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, *Spanning trees short or small*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 546–555.

[32] N. Robertson and P. Seymour, *Graph minors IV: Tree-width and well-quasi-ordering*, J. Combin. Theory Ser. B, 48 (1990), pp. 227–254.