SIAM J. COMPUT.
Vol. 29, No. 3, pp. 761–778

A POLYNOMIAL-TIME APPROXIMATION SCHEME FOR MINIMUM ROUTING COST SPANNING TREES*

BANG YE WU†, GIUSEPPE LANCIA‡, VINEET BAFNA§, KUN-MAO CHAO¶, R. RAVI‖, AND CHUAN YI TANG**

Abstract. Given an undirected graph with nonnegative costs on the edges, the routing cost of any of its spanning trees is the sum over all pairs of vertices of the cost of the path between the pair in the tree. Finding a spanning tree of minimum routing cost is NP-hard, even when the costs obey the triangle inequality. We show that the general case is in fact reducible to the metric case and present a polynomial-time approximation scheme valid for both versions of the problem. In particular, we show how to build a spanning tree of an n-vertex weighted graph with routing cost at most $(1+\epsilon)$ of the minimum in time $O(n^{O(\frac{1}{\epsilon})})$. Besides the obvious connection to network design, trees with small routing cost also find application in the construction of good multiple sequence alignments in computational biology.

The communication cost spanning tree problem is a generalization of the minimum routing cost tree problem where the routing costs of different pairs are weighted by different requirement amounts. We observe that a randomized $O(\log n \log\log n)$-approximation for this problem follows directly from a recent result of Bartal, where n is the number of nodes in a metric graph. This also yields the same approximation for the generalized sum-of-pairs alignment problem in computational biology.

Key words. approximation algorithms, network design, spanning trees, computational biology

AMS subject classifications. 68W25, 68M10, 05C05, 92B02

PII. S009753979732253X

1. Introduction.

Consider the following problem in network design: given an undirected graph with nonnegative delays on the edges, the goal is to find a spanning tree such that the average delay of communicating between any pair using the tree is minimized. The delay between a pair of vertices is the sum of the delays of the edges in the path between them in the tree. Minimizing the average delay is equivalent to minimizing the total delay between all pairs of vertices in the tree.

In general, when the cost on an edge represents a price for routing messages between its endpoints (such as the delay), we define the *routing cost* for a pair of vertices in a given spanning tree as the sum of the costs of the edges in the unique tree path between them. The routing cost of the tree itself is the sum over all pairs of vertices of the routing cost for the pair in this tree.

*Received by the editors June 9, 1997; accepted for publication (in revised form) November 25, 1998; published electronically December 7, 1999.

http://www.siam.org/journals/sicomp/29-3/32253.html

†Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C. (bangye@ms16.hinet.net).

‡Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy (lancia@dei.unipd.it).

§Celera Genomics, 45 West Gude Drive, Rockville, MD 20850 (Vineet.Bafna@celera.com).

¶Department of Computer Science and Information Management, Providence University, Shalu, Taiwan, R.O.C. (kmchao@csim.pu.edu.tw). The research of this author was supported in part by NSC grant NSC86-2213-E-126-002.

‖GSIA, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213 (ravi@cmu.edu). The research of this author was supported by NSF Career grant CCR-9625297.

**Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C. (cytang@cs.nthu.edu.tw). The research of this author was supported by NSC grant NSC86-2213-E-007-008.

Finding a spanning tree of minimum routing cost in a general weighted undirected graph is known to be NP-hard [11]. In this paper we show that finding a minimum routing cost tree in a general weighted graph G is equivalent to solving the same problem on a complete graph in which the edge weights are the shortest path lengths in G. This result implies that the minimum routing tree problem with metric inputs is also NP-hard.

Wong [22] studied the minimum routing cost tree problem and presented a 2-approximation algorithm even without the metric requirement. We give a better result for the metric case, which, by the above remark, applies to the general case as well.

THEOREM 1.1. *There is a polynomial-time approximation scheme (PTAS) for finding the minimum routing cost tree of a weighted undirected graph. In particular, on an n-vertex graph, we can find a $(1+\epsilon)$-approximate solution in time $O(n^{2\lceil \frac{2}{\epsilon} \rceil - 2})$.*

Our result is derived by approximating a minimum routing cost tree by a restricted class of trees that we call *k-stars*. For any fixed size k, a k-star is a tree in which at most k vertices have degree greater than one. For a given accuracy parameter ϵ, we consider all $\lceil \frac{2}{\epsilon} - 1 \rceil$-stars and output the one with the minimum routing cost. To argue the performance guarantee, we show how a minimum routing cost tree can be converted into a k-star without much degradation in its routing cost (no more than a factor of $1 + \frac{2}{k+1}$). We also prove that for any fixed k, the minimum k-star can be determined in polynomial time. Hence, by finding the $\lceil \frac{2}{\epsilon} - 1 \rceil$-star with the minimum routing cost, we get a $(1 + \epsilon)$-approximate solution.

There is an important difference between our PTAS for the routing cost tree problem and Wong's 2-approximation: While we show an approximation bound to the best tree's routing cost, Wong's proof shows that his trees have routing cost at most twice the value of the sum of pairwise distances between nodes in the input graph. This stronger connection is exploited by Gusfield [9] in an application to multiple alignments in computational biology (described later).

1.1. Optimum communication spanning trees. Hu [10] formulated a general version of the routing cost spanning tree problem that he called optimum communication spanning trees. In this problem, in addition to the costs on edges, a requirement value r_{ij} is specified for every pair of vertices i, j. The communication cost between a pair in a given spanning tree is the cost of the path between them in the tree multiplied by their requirement r_{ij}. The communication cost of the tree is the sum of all the pairwise communication costs. Thus the routing cost is a special case of the communication cost when all the requirement values are one.[1] In [10], Hu derives weak conditions under which the optimum routing cost tree is a star. In this paper, we demonstrate that simple generalizations of stars are indeed sufficient to guarantee any desirable accuracy in approximating optimal routing trees.

By using a recent result of Bartal [3] on approximating metrics probabilistically by tree metrics, we notice the following result.

THEOREM 1.2. *There is an $O(\log^2 n)$-approximation algorithm for the communication spanning tree problem on an n-node metric.*

Recent improvements to Bartal's original result in [4, 6] also lead to an improvement of the performance guarantee in Theorem 1.2 to $O(\log n \log\log n)$.

The result in Theorem 1.2 is actually stronger in the same sense as Wong [22].

[1] Hu uses the term "optimum distance spanning trees" to denote trees with minimum routing cost.

Given (symmetric) requirement values r_{ij} and metric distances d_{ij} between node pairs i, j, our approximate solution has communication cost at most $O(\log^2 n)$ times $\sum_{i,j} r_{ij} d_{ij}$. As in [9], we exploit this connection in the application to computational biology.

An overview of the remainder of the paper is as follows. In section 2 we describe the application of minimum routing cost trees to alignment problems in computational biology. In section 3 we give some basic definitions. In section 4, we show how the general case of the problem can be reduced to the metric one. Section 5 describes how k-stars provide good approximations to the optimum routing cost trees in metrics. In section 6, we discuss a polynomial algorithm for finding minimum cost k-stars in a graph. Finally, in section 7 we describe an algorithm for approximating optimum communication spanning trees.

2. An application to computational biology.

2.1. Multiple sequence alignments. Multiple sequence alignments are important tools for highlighting patterns common to a set of genetic sequences in computational biology. A multiple alignment of a set of n strings involves inserting gaps in the strings and arranging their characters into columns with n rows, one from each string. The order of characters along a row corresponding to string s_i is the same as that in s_i, with possibly some blanks inserted. The following is an example of an alignment of three strings `ATTCGAC`, `TTCCGTC`, and `ATCGTC`.

```
A  T  T  -  C  G  A  -  C
-  T  T  C  C  G  -  T  C
A  -  T  -  C  G  -  T  C
```

The intent of identifying common patterns is represented by attempting as much as possible to place the same character in every column.

The multiple sequence alignment problem has typically been formalized as an optimization problem in which some explicit objective function is minimized or maximized. One of the most popular objective functions for multiple alignment generalizes ideas from optimally aligning two sequences. The pairwise-alignment problem [21] can be phrased as that of finding a minimum mutation path between two sequences. Formally, given costs for inserting or deleting a character and for substituting one character of the alphabet for another, the problem is to find a minimum-cost mutation path from one sequence to the other. The cost of this path is the *edit distance* between them. An optimal alignment of two sequences of length l can be computed effectively by dynamic programming [14, 21] in $O(l^2)$ time. The generalization to multiple sequences leads to the sum-of-pairs objective.

The *sum-of-pairs* (SP) objective for multiple alignment is to minimize the sum, over all pairs of sequences, of the pairwise distance between them *in the alignment* (where the distance of two sequences in an alignment with l columns is obtained by adding up the costs of the pairs of characters appearing at positions $1, \ldots, l$).

Pioneering work of Sankoff and Kruskal [17] and Sankoff, Morel, and Cedergren [18] led to an exponential-time dynamic programming solution to the SP-alignment problem. A straightforward implementation requires time proportional to $2^n l^n$ for a problem with n sequences each of length at most l. Considering that in typical real-life instances l can be a few hundred, the basic dynamic programming approach turns out to be infeasible for all but very small problems. Carrillo and Lipman [5] have introduced some bounding criteria which reduce the time and space requirements of

dynamic programming and make solvable problems for $n \leq 6$ and $l \leq 200$. However, constructing *optimal* alignments is bound to be computationally expensive, since the problem has been shown to be NP-complete (Wang and Jiang, [20]). Despite these very expensive solution methods, the SP-objective is implemented in several popularly available multiple alignment packages such as MACAW [19] and MSA [13].

2.2. Approximation algorithms via routing cost trees. The first approximation algorithm for the SP-alignment problem was by Gusfield [9]. It had a performance ratio of $2 - \frac{2}{n}$ where n is the number of sequences aligned. This was slightly improved to $2 - \frac{3}{n}$ by Pevzner [15]. The best-known approximation algorithm for this problem is due to Bafna, Lawler, and Pevzner [2], which achieves a ratio of $2 - \frac{r}{n}$ for any fixed value of r. The running time is exponential in r. Notice that this is not a PTAS for the problem, and no polynomial-time approximation scheme is known yet for the SP-alignment problem.

Gusfield's approximation algorithm for the SP-alignment problem is based on the 2-approximation for minimum routing cost trees due to Wong [22]. Gusfield's algorithm uses a folklore approach to multiple alignment guided by a tree, due to Feng and Doolittle [8]: Given a spanning tree on the complete graph on the sequences to be aligned, the multiple alignment guided by the tree is built recursively as follows. First, remove a leaf sequence l in the tree attached to sequence v by a tree edge (l, v), and align the remaining sequences recursively. Then, reinsert the leaf sequence into the alignment guided by an optimal pairwise alignment between the pair l and v. If this optimal pairwise alignment introduces a gap in v, insert the same gap in the recursively computed alignment for the tree without the leaf. Since the cost of aligning a blank to a blank is assumed to be zero, the resulting alignment has the property that for every pair related by a tree edge, the cost of the induced pairwise alignment equals their edit distance. By the triangle inequality on edit-distances, the SP-cost of the alignment derived from this spanning tree can be upper-bounded by the routing cost of the tree.

Wong's 2-approximation algorithm considers the shortest path tree rooted at every vertex in turn, and picks the one with minimum routing cost. For graphs with metric distances obeying the triangle inequality, every shortest path tree is isomorphic to a star. Furthermore, in this case, Wong's analysis shows that the best star has routing cost at most twice the total cost of the graph itself. The cost of the graph in this case is the sum of pairwise edit distances between sequences, which is a lower bound on the SP-cost. Thus, Gusfield observed that a multiple alignment derived from the best center-star gives a 2-approximation for the SP-alignment problem.

2.3. Tree-driven SP-alignment. Despite the popularity of the SP-objective, most of the currently available methods for finding alignments use a *progressive* approach of incrementally building the alignment adding sequences one at a time with no performance guarantee on the SP-cost. The Feng–Doolittle procedure can be viewed as one such procedure. The advantage of such approaches is their low running time, but the shortcoming is that the order in which the sequences are merged into the alignment determines its cost.

In trying to define a middle ground between the SP-objective and the more practical progressive methods, we introduce the tree-driven SP-alignment method: apply the Feng–Doolittle procedure to the *best possible* spanning tree in the complete graph on the sequences. By our reasoning above, the tree that gives the best upper bound on the SP-cost of the alignment is the one with the minimum routing cost. Thus,

our PTAS for routing cost trees may be useful in finding good trees for applying any progressive alignment method such as the Feng–Doolittle procedure.

2.4. Generalized SP-alignments. A simple generalization of the SP objective for multiple alignments is to weight the different sequence pairs in the alignment differently in the objective function. Given a priority value r_{ij} for the pair i, j of sequences, the *generalized sum-of-pairs* objective for multiple alignment is to minimize the sum, over all pairs of sequences, of the pairwise distance between them in the alignment multiplied by the priority value of the pair. This allows one to increase the priority of aligning some pairs while down-weighting others, using other information (such as evolutionary) to decide on the priorities. An extreme case of assigning priorities is the *threshold* objective.

In an evolutionary context, a multiple alignment is used to reconstruct the blocks or motifs in a single ancestral sequence from which the given sequences have evolved. However, if the evolutionary events of the ancestral sequence occur randomly at a certain rate over the course of time, and independently at each location (character) of the string, after a sufficiently long time, the mutated sequence appears essentially like a random sequence compared to the initial ancestral sequence. If we postulate a threshold time beyond which this happens, this translates roughly to a threshold edit distance between the pair of sequences. The threshold objective sets r_{ij} to be one for all pairs of input sequences whose edit-distance is less than this threshold and zero for other pairs which are more distant. In this way we try to capture the most information about closely related pairs in the objective function by setting an appropriate threshold.

In the same vein as Gusfield [9], Theorem 1.2 can be used to approximate the generalized SP objective within an $O(\log^2 n)$ factor on inputs with n sequences. Let d_{ij} denote the edit distance between sequences i and j. The theorem guarantees a tree whose communication cost using the r_{ij} values given by the priority function is at most $O(\log^2 n)$ times $\sum_{i,j} r_{ij} d_{ij}$, which is a lower bound on the generalized SP value of any alignment. The Feng–Doolittle procedure guarantees that the generalized SP value of the resulting alignment is at most the communication cost of the tree which in turn is at most $O(\log^2 n)$ times the generalized SP value of any alignment.

3. Definitions. Throughout the paper we will be referring to a given weighted, connected, undirected graph $G = (V, E, w)$, where we assume $V = \{1, \ldots, n\}$ and w is a nonnegative edge weight function, not necessarily metric. For a subset $S \subseteq V$, by $\mathcal{P}(S)$ we denote the set of all unordered pairs of elements of S.

DEFINITION 3.1. *Let $G = (V, E, w)$ and $i, j \in V$. Let $S = (V_S, E_S, w)$ be a subgraph of G. By $SP(S, i, j)$ we denote a shortest path from i to j on S. When S is a tree, $SP(S, i, j)$ denotes the unique path between i and j.*

DEFINITION 3.2. *Let S be a subgraph of G and $i, j \in V$. The weight of S is denoted by $w(S) = \sum_{e \in E_S} w(e)$. The distance of i and j in S is denoted by $d_S(i, j) := w(SP(S, i, j))$. We define $d_G(i, S) = \min_{j \in V_S} d_G(i, j)$. If T is a tree and $S \subset T$, we denote the value $w(SP(T, i, j) \cap S)$ by $w_S(T, i, j)$.*

DEFINITION 3.3. *Let S be a subgraph of G. The* routing cost *of S is defined as $C(S) = \sum_{(i,j) \in \mathcal{P}(V_S)} d_S(i, j)$.*

DEFINITION 3.4. *Given a graph $G = (V, E, w)$, the* minimum routing cost spanning tree problem *(*MRCT*) is to find a spanning tree $\widehat{T}_G$ of G such that $C(\widehat{T}_G)$ is minimum.*

DEFINITION 3.5. *A* metric graph $G = (V, E, w)$ *is a complete graph in which* $w(i,j) \geq 0$ *and* $w(i,j) + w(j,k) \geq w(i,k)$ *for all* $i \neq j \neq k \in V$.

DEFINITION 3.6. *The* metric closure *of* G *is the complete weighted graph* $\bar{G} = (V, \mathcal{P}(V), \delta)$, *where* $\delta(i,j) := d_G(i,j)$ *for all* $(i,j) \in \mathcal{P}(V)$. *Note that* $\bar{G}$ *is a metric graph.*

DEFINITION 3.7. *Given a metric graph* G, *the* metric minimum routing cost spanning tree problem *(*ΔMRCT*) is to find a spanning tree* T *of* G *such that* $C(T)$ *is minimum.*

4. A reduction from the general to the metric case. Let $G = (V, E, w)$ and $\bar{G} = (V, \mathcal{P}(V), \delta)$ be its metric closure. In this section, we present an algorithm which can transfer a spanning tree of $\bar{G}$ into a spanning tree of G without increasing cost. This implies that we can solve the MRCT problem on G by solving the same problem on $\bar{G}$. An edge (a,b) in $\bar{G}$ is termed a *bad edge* if $(a,b) \notin E$ or $w(a,b) > \delta(a,b)$. For any bad edge $e = (a,b)$, there must exist a path $P \neq e$ such that $w(P) = \delta(a,b)$. Given any spanning tree T of $\bar{G}$, the algorithm iteratively replaces bad edges (if any) in T with edges from the path defining the weight of the edge until there are no more bad edges in the tree. Since the resulting tree Y has no bad edge, it can be thought of as a spanning tree of G with the same cost. It will be shown later that the iteration will be executed at most $O(n^2)$ times and the cost is never increased while replacing the bad edges. The algorithm listed below details how to obtain Y from T.

Algorithm Remove_bad
Input: a spanning tree T of $\bar{G}$
Output : a spanning tree Y of G (i.e., without any bad edge) such that $C(Y) \leq C(T)$.

```
Compute all-pairs shortest paths of G.
while there exists a bad edge in T                                  (1)
    Pick a bad edge (a,b). Root T at a.
    /* assume SP(G,a,b) = (a,x,...,b) and y is the father of x in T */
    if b is not an ancestor of x then
        Y1 = T ∪ (x,b) − (a,b)
        Y2 = Y1 ∪ (a,x) − (x,y)
    else
        Y1 = T ∪ (a,x) − (a,b)
        Y2 = Y1 ∪ (b,x) − (x,y)
    endif
    if C(Y1) < C(Y2) then
        Y = Y1
    else
        Y = Y2
    endif
    T = Y                                                           (2)
endwhile
```

We assume that the shortest paths obtained in the beginning of the algorithm have the following property: If the obtained shortest path between a and b is $(a,x) \cup P$, then P is the obtained shortest path between x and b. Note that since x is on the shortest a-b path, $\delta(a,b) = \delta(a,x) + \delta(x,b)$.

PROPOSITION 4.1. *The loop* (1) *is executed at most* $O(n^2)$ *times.*

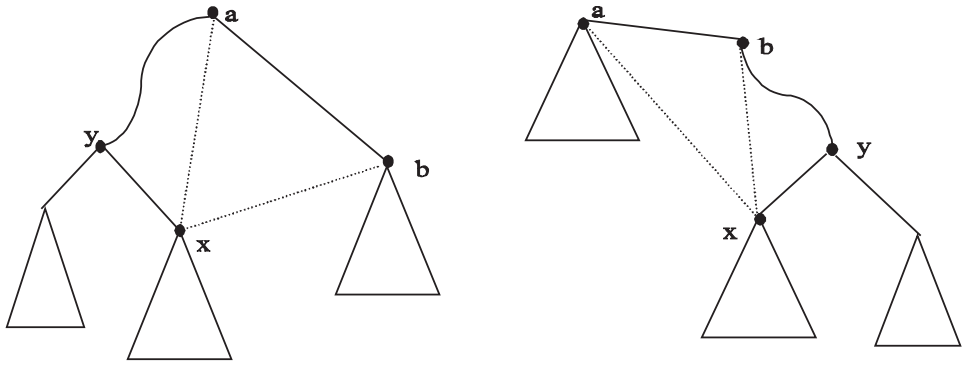

FIG. 4.1. *Remove bad edge* (a, b). *Case* 1 *(left) and Case* 2 *(right).*

Proof. For each bad edge $e = (a, b)$, let $l(e)$ be the number of edges in $SP(G, a, b)$ and $f(T) = \sum_{\text{bad } e} l(e)$. Since $l(e) \leq n - 1$, $f(T) < n^2$. Since $l(x, b) < l(a, b)$ and (a, x) is not a bad edge, it is easy to check that $f(T)$ decreases by at least 1 at each loop iteration. □

PROPOSITION 4.2. *Before instruction* (2) *is executed,* $C(Y) \leq C(T)$.

Proof. For any node v, define $S_v = \{u | v \text{ is an ancestor of } u \text{ on } T\} \cup \{v\}$. Also, let $C(T, S_1, S_2) = \sum_{i \in S_1, j \in S_2} d_T(i, j)$.

Case 1. (see Figure 4.1.) $x \in S_a - S_b$. If $C(Y_1) \leq C(T)$, the result follows. Otherwise, let $S_1 = S_a - S_b$ and $S_2 = S_a - S_b - S_x$. Since the distance between any two vertices both in S_1 (or both in S_b) does not change, we have

$$\begin{aligned}
& C(T) < C(Y_1) \\
\Rightarrow\ & C(T, S_1, S_b) < C(Y_1, S_1, S_b) \\
\Rightarrow\ & |S_b| C(T, a, S_1) + |S_1||S_b|\delta(a, b) < |S_b| C(T, x, S_1) + |S_1||S_b|\delta(x, b) \\
\Rightarrow\ & C(T, a, S_1) + |S_1|\delta(a, b) < C(T, x, S_1) + |S_1|\delta(x, b) \\
\Rightarrow\ & C(T, a, S_1) < C(T, x, S_1) - |S_1|\delta(a, x).
\end{aligned}$$

The last inequality follows from the property of the shortest path lengths alluded to earlier.

Also,

$$C(Y_2) - C(T) = (C(Y_2, S_2, S_x) - C(T, S_2, S_x)) + (C(Y_2, S_b, S_1) - C(T, S_b, S_1)).$$

Since $d_{Y_2}(i, j) \leq d_T(i, j)$ for $i \in S_b$ and $j \in S_1$, the second term is not positive, and

$$\begin{aligned}
& C(Y_2) - C(T) \\
\leq\ & C(Y_2, S_2, S_x) - C(T, S_2, S_x) \\
=\ & |S_x| C(T, a, S_2) + |S_2||S_x|\delta(a, x) - |S_x| C(T, x, S_2) \\
=\ & |S_x|((C(T, a, S_1) - C(T, a, S_x)) + \ |S_2|\delta(a, x) - (C(T, x, S_1) - C(T, x, S_x))) \\
=\ & |S_x|((C(T, a, S_1) - C(T, x, S_1)) + \ |S_2|\delta(a, x) + (C(T, x, S_x) - C(T, a, S_x))) \\
<\ & |S_x|\,(-|S_1|\delta(a, x) + |S_2|\delta(a, x)) \\
\leq\ & 0.
\end{aligned}$$

Case 2. $x \in S_b$. The case is identical to Case 1 if we reroot the tree at b and follow the analysis in Case 1 exchanging the roles of a and b. □

As a direct consequence of Propositions 4.1 and 4.2 we obtain the following lemma.

LEMMA 4.3. *Given a spanning tree T of $\bar{G}$, the algorithm Remove_bad constructs a spanning tree Y of G with $C(Y) \leq C(T)$ in $O(n^3)$ time.*

The above lemma implies that $C(\widehat{T}_G) \leq C(\widehat{T}_{\bar{G}})$. Since, for any edge, the weight on the original graph is no less than the weight on the metric closure, it is easy to see that $C(\widehat{T}_G) \geq C(\widehat{T}_{\bar{G}})$. Therefore, we have the following corollary.

COROLLARY 4.4. $C(\widehat{T}_G) = C(\widehat{T}_{\bar{G}})$.

COROLLARY 4.5. *If there is a $(1+\varepsilon)$-approximation algorithm for* ΔMRCT *with time complexity $O(f(n))$, then there is a $(1+\varepsilon)$-approximation algorithm for* MRCT *with time complexity $O(f(n)+n^3)$.*

Proof. Let G be the input graph for a MRCT problem. We can construct $\bar{G}$ in time $O(n^3)$ (see, e.g., [7]). If there is a $(1+\varepsilon)$-approximation algorithm for the ΔMRCT problem, we can compute in time $O(f(n))$ a spanning tree T_1 of $\bar{G}$ such that $C(T_1) \leq (1+\varepsilon)C(\widehat{T}_{\bar{G}})$. Using Algorithm Remove_bad, we can then construct a spanning tree T_2 of G such that $C(T_2) \leq C(T_1) \leq (1+\varepsilon)C(\widehat{T}_{\bar{G}}) = (1+\varepsilon)C(\widehat{T}_G)$. The overall time complexity is then $O(f(n)+n^3)$. ∎

5. A PTAS for the ΔMRCT problem.

5.1. Overview. As described in the previous section, the fact that the costs w may not obey the triangle inequality is irrelevant, since we can simply replace these costs by their metric closure. Therefore, in this and the following sections we may assume that $G = (V, E, w)$ is a metric graph. We remind the reader that $n = |V|$. Also, for a subgraph G' of G, we use $V(G')$ to denote the vertex set of G'.

To establish the performance guarantee, we use k-stars, i.e., trees with no more than k internal nodes. In section 6 we show that for any constant k, the minimum routing cost k-star can be determined in polynomial (in n) time. In order to show that a k-star achieves a $(1+\epsilon)$ approximation, we show that, for any tree T and constant $\delta \leq 1/2$:

1. It is possible to determine a δ-separator (a particular subtree of T to be defined later), and the separator can be cut into several δ–paths such that the total number of cut nodes and leaves of the separator is at most $\lceil \frac{2}{\delta} \rceil - 3$ (Lemma 5.9).
2. Using the separator, T can be converted into a $(\lceil \frac{2}{\delta} \rceil - 3)$-star $X(T)$, whose internal nodes are just those cut nodes and leaves. The routing cost of $X(T)$ satisfies $C(X(T)) \leq (1 + \frac{\delta}{1-\delta})C(T)$ (Lemma 5.13).

By using $T = \widehat{T}_G$, $\delta = \frac{\epsilon}{1+\epsilon}$ and finding the best $(\lceil \frac{2}{\delta} \rceil - 3)$-star K, we obtain $C(K) \leq C(X(\widehat{T}_G)) \leq (1+\frac{\delta}{1-\delta})C(\widehat{T}_G) = (1+\epsilon)C(\widehat{T}_G)$, i.e., the desired approximation.

5.2. The δ-spine of a tree.

DEFINITION 5.1. *Let T be a spanning tree of G and S be a connected subgraph of T. A* branch *of S is a connected component of $T \setminus S$. Let $\delta \leq 1/2$ be a positive number. If $|V(B)| \leq \delta n$ for every branch B of S, then S is a* δ-separator *of T. A δ-separator S is* minimal *if any proper subgraph of S is not a δ-separator of T.*

Intuitively, a δ-separator is like a "center" of the tree. Starting from any node, there are sufficiently many nodes which cannot be reached without touching the separator. To illustrate the concept of separator, we examine the simplest case for $\delta = 1/2$. For any tree T, there always exists a 1/2-separator which contains only one vertex. That is, we can always cut a tree at a node c such that each branch contains at most

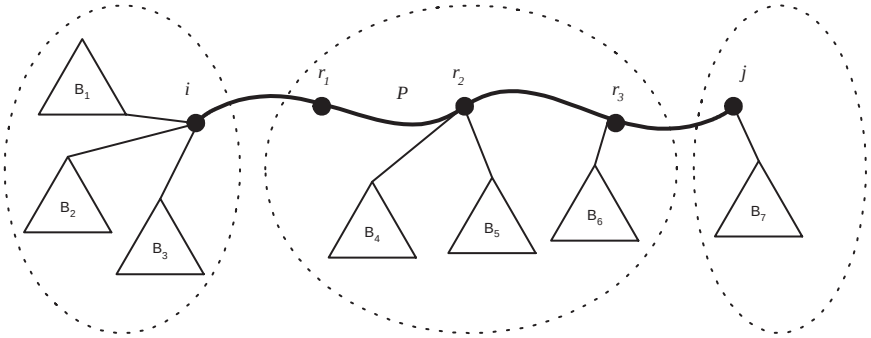

FIG. 5.1. *$B_1, \ldots, B_7$ are branches of P. $VB(T,P,i) = \{i\} \cup V(B_1) \cup V(B_2) \cup V(B_3)$. P^c is the number of vertices in $\{r_1, r_2, r_3\} \cup V(B_4) \cup V(B_5) \cup V(B_6)$.*

half of the nodes. Such a node is usually called the *centroid* of the tree in the literature. Note that this also shows the existence of a minimal δ-separator for any $\delta \leq 0.5$.

If we construct a star X centered at the centroid c, the routing cost will be at most twice that of T. This can be easily shown as follows. First, if i and j are two nodes not in the same branch, $d_T(i,j) = d_T(i,c) + d_T(j,c)$. Consider the total distance of all *ordered* pairs of nodes on T. This value is exactly $2C(T)$ by the definition. For any node i, since each branch contains no more than half of the nodes, the term $d_T(i,c)$ will be counted in the total distance at least n times, $n/2$ times for i to others, and $n/2$ times for others to i. Hence, we have $2C(T) \geq n \sum_i d_T(i,c)$. Since $C(X) = (n-1) \sum_i d_G(i,c)$, it follows that $C(X) \leq 2C(T)$. The idea in this paper can be thought as a generalization of the above method. However, the proof is much more involved.

DEFINITION 5.2. *Let T be a spanning tree of G and S be a connected subgraph of T. For any vertex i in S, $VB(T,S,i)$ denotes the set of vertex i and the vertices in the branches connected to i.*

DEFINITION 5.3. *Let $P = SP(T,i,j)$ in which $|VB(T,P,i)| \geq |VB(T,P,j)|$. We define $P^a = |VB(T,P,i)|$, $P^b = |VB(T,P,j)|$, and $P^c = n - |VB(T,P,i)| - |VB(T,P,j)|$. Assume $P = (i, r_1, r_2, \ldots, r_h, j)$. Define $Q(P) = \sum_{1 \leq x \leq h} |VB(T,P,r_x)| \times d_T(r_x, i)$.*

The above notations are defined to simplify the expressions. P^a and P^b are the numbers of vertices that are hanging off the two endpoints of the path. Note that we always assume $P^a \geq P^b$. In the case that P contains only one edge, $P^c = 0$. The notations are illustrated in Figure 5.1.

LEMMA 5.4. *Let S be a minimal δ-separator of T. If i is a leaf of S, then $|VB(T,S,i)| > \delta n$.*

Proof. If S contains only one vertex, the result is trivial since $|VB(T,S,i)| = n$. Otherwise, if $|VB(T,S,i)| \leq \delta n$, deleting i from S we still get a δ-separator. This is a contradiction to S being minimal. □

DEFINITION 5.5. *Let $1 \leq k \leq n$. A* k-star *is a spanning tree of G which has no more than k internal nodes. The set of all k-stars is denoted by $k^*(G)$. T is a* minimum k-star *if $T \in k^*(G)$ and $C(T) \leq C(Y)$ for all $Y \in k^*(G)$.*

We now turn to the notions of δ-path and δ-spine. Informally, a δ-path is a path such that not too many nodes (at most $\delta n/2$) are hanging off its internal nodes. A δ-spine is a set of edge-disjoint δ-paths, whose union is a minimal δ-separator. That is, a δ-spine is obtained by cutting the minimal δ-separator into δ-paths. In the case

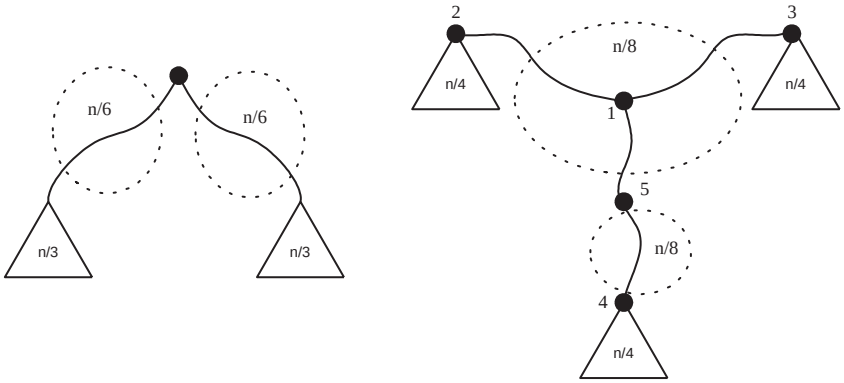

FIG. 5.2. *Trees with maximum value for the size of the minimum cut and leaf set.*

that the minimal δ-separator contains just one node, the only δ-spine is the empty set.

DEFINITION 5.6. *Given a spanning tree T of G, and $0 < \delta \leq 0.5$, a δ-path of T is a path P such that $P^c \leq \delta n/2$.*

DEFINITION 5.7. *Let $0 < \delta \leq 0.5$. A* δ-spine $Y = \{P_1, P_2, ..., P_h\}$ *of T is a set of pairwise edge-disjoint δ-paths in T such that $S = \bigcup_{1 \leq i \leq h} P_i$ is a minimal δ-separator of T. Furthermore, for any pair of distinct paths P_i and P_j in the spine, we require that either they do not intersect or, if they do, the intersection point is an endpoint of both paths.*

DEFINITION 5.8. *Let Y be a δ-spine of a tree T. $CAL(Y)$ (which stands for the* cut and leaf set *of Y) is the set of the endpoints of the paths in Y. In the case that Y is empty, the CAL set contains only one node which is the δ-separator of T. Formally $CAL(Y) = \{u | \exists P \in Y, v \in T : P = SP(T, u, v)\}$ if Y is not empty, and otherwise $CAL(Y) = \{u | u$ is the minimal δ-separator $\}$.*

Two trees achieving the maximum value for the size of the minimum CAL set for $\delta = 1/3$ ($|CAL(Y)| = 3$) and $\delta = 1/4$ ($|CAL(Y)| = 5$) are depicted in Figure 5.2. Next, we show that for any tree, there always exists a (1/3)-spine Y_1 such that $|CAL(Y_1)| \leq 3$ and a (1/4)-spine Y_2 such that $|CAL(Y_2)| \leq 5$.

LEMMA 5.9. *For any constant $0 < \delta \leq 0.5$, and spanning tree T of G, there exists a δ-spine Y of T such that $|CAL(Y)| \leq \lceil 2/\delta \rceil - 3$.*

Proof. Let S be a minimal δ-separator of T. S is a tree. Let U_1 be the set of leaves in S, U_2 be the set of vertices which have more than two neighbors in S, and $U = U_1 \cup U_2$. Let $h = |U_1|$. Clearly, $|U| \leq 2h - 2$. Let Y_1 be the set of paths obtained by cutting S at all the vertices in U_2. For example, for the tree on the right side of Figure 5.2, $U_1 = \{2, 3, 4\}$; $U_2 = \{1\}$; Y_1 contains $SP(T, 1, 2)$, $SP(T, 1, 3)$, and $SP(T, 1, 4)$. For any $P \in Y_1$, if $P^c > \delta n/2$ then P is called a *heavy* path. It is easy to check that Y_1 satisfies the requirements of a δ-spine except that there may exist some heavy paths. Suppose P is not a δ-path. We can break it up into δ-paths by the following process. First find the longest prefix of P starting at one of its endpoints and ending at some internal vertex, i, say, in the path, that determines a δ-path. Now we break P at vertex i. Then we repeat the breaking process on the remaining suffix of P starting at i stripping off the next δ-path and so on. In this way P can be cut into δ-paths by breaking it in at most $\lceil 2P^c / (\delta n) \rceil - 1$ vertices. Since there are at

least δn nodes hung at each leaf,

$$\sum_{P\in Y_1} P^c < n - h\delta n.$$

Assume U_3 to be the minimal vertex set for cutting the heavy paths to result in a δ-spine Y of T. We have

$$|U_3| \le \lceil 2\,(n - h\delta n)\,/\,(\delta n)\rceil - 1 = \lceil 2/\delta\rceil - 2h - 1.$$

So, $|CAL(Y)| = |U| + |U_3| \le \lceil 2/\delta\rceil - 3$. ∎

5.3. Lower bound.

DEFINITION 5.10. *The* routing load *of an edge e in T is the number $e^a e^b$ of pairs in T connected by a path containing e.*

The following lemma is immediate.

LEMMA 5.11. *For any spanning tree T of G, $C(T) = \sum_{e\in T} e^a e^b w(e)$.*

LEMMA 5.12. *Let Y be a δ-spine of a spanning tree T of G and $S = \bigcup_{P\in Y} P$ be a minimal δ-separator of T. Then*

$$C(T) \ge (1-\delta)n \sum_{v\in V} d_T(v,S) + \sum_{P\in Y} \left(P^b(P^a + P^c)w(P) + (P^a - P^b)Q(P)\right).$$

Proof. Since $e^a \ge (1-\delta)n$ for any edge $e \in T \setminus S$, we have

$$\begin{aligned} C(T) &= \sum_{e\in T} e^a e^b w(e) \\ &\ge \sum_{e\in T\setminus S} (1-\delta)n e^b w(e) + \sum_{e\in S} e^a e^b w(e) \\ &\ge (1-\delta)n \sum_{v\in V} d_T(v,S) + \sum_{P\in Y}\sum_{e\in P} e^a e^b w(e). \end{aligned}$$

Now we simplify the second term. Assume $P = (r_0, r_1, r_2, \ldots, r_h)$ in which $|VB(T,P,r_0)| \ge |VB(T,P,r_h)|$. Let $|VB(T,P,r_i)| = n_i$ for $1 \le i \le h-1$ and $e_i = (r_{i-1}, r_i)$ for $1 \le i \le h$.

$$\begin{aligned} &\sum_{e\in P} e^a e^b w(e) \\ &= \sum_{i=1}^{h} \left(P^a + P^c - \sum_{j=i}^{h-1} n_j\right)\left(P^b + \sum_{j=i}^{h-1} n_j\right) w(e_i) \\ &\ge \sum_{i=1}^{h} P^b\,(P^a + P^c)\,w(e_i) + (P^a - P^b)\sum_{i=1}^{h}\sum_{j=i}^{h-1} n_j w(e_i) \\ &\quad + \sum_{i=1}^{h} \left(\sum_{j=i}^{h-1} n_j\right)\left(P^c - \sum_{j=i}^{h-1} n_j\right) w(e_i) \\ &\ge P^b(P^a + P^c)w(P) + (P^a - P^b)\sum_{j=1}^{h-1} n_j \left(\sum_{i=1}^{j} w(e_i)\right) \\ &= P^b\,(P^a + P^c)\,w(P) + (P^a - P^b)Q(P). \end{aligned}$$

This completes the proof. ∎

5.4. From trees to stars.

LEMMA 5.13. *For any constant* $0 < \delta \leq 0.5$, *there exists a spanning tree* $X \in (\lceil 2/\delta \rceil - 3)^*(G)$ *such that* $C(X) \leq \frac{1}{1-\delta} C(\widehat{T}_G)$.

Proof. Let $T = \widehat{T}_G = (V, E, w)$ and $n = |V|$. Also, let $Y = \{P_i | 1 \leq i \leq h\}$ be a δ-spine of T in which $|CAL(Y)| \leq \lceil 2/\delta \rceil - 3$. Note that the set of all the edges in Y form a δ-separator S. Assume $P_i = SP(T, u_i, v_i)$ and $|VB(T, P_i, u_i)| \geq |VB(T, P_i, v_i)|$.

We construct a spanning tree whose internal nodes are exactly the CAL set of the δ-spine we just identified. We connect these nodes by short-cutting paths along the spine to include a set of acyclic edges with the same skeletal structure as the spine. All vertices in subtrees hanging off the CAL nodes of the spine are connected directly to their closest node in the spine. Along a δ-path in the spine, all the internal nodes and nodes in subtrees hanging off internal nodes are connected to one of the two endpoints of this path (note that both are in the CAL set of the spine) in such a way as to minimize the resulting routing cost. This is the spanning tree used to argue the upper bound on the routing cost in the proof.

More formally, construct a subgraph $R \subset G$ with vertex set $CAL(Y)$ and edge set $E_r = \{(u_i, v_i) | 1 \leq i \leq h\}$. Trivially, R is a tree. Let $f(i)$ be an indicator variable such that if $\left(P_i^a - P_i^b\right) P_i^c w(P_i) - n\left(2Q(P_i) - P_i^c w(P_i)\right) \geq 0$ then $f(i) = 1$, else $f(i) = 0$. The indicator variable $f(i)$ determines the endpoint of P_i to which all the internal nodes and nodes hanging off such internal nodes will be directly connected. We construct a spanning tree X of G where the edge set E_x is determined by the following rules:

1. $R \subset X$.
2. If $q \in VB(T, S, r)$, then $(q, r) \in E_x$, for any $r \in \{u_i, v_i | 1 \leq i \leq h\}$.
3. For the vertex set $V_i = V - VB(T, P_i, u_i) - VB(T, P_i, v_i)$, if $f(i) = 1$, then $\{(q, u_i) | q \in V_i\} \subset E_x$, else $\{(q, v_i) | q \in V_i\} \subset E_x$. That is, the vertices in V_i are either all connected to u_i or all connected to v_i.

It is easy to see that $X \in (\lceil 2/\delta \rceil - 3)^*(G)$. Let's consider the cost of X.

$$\begin{aligned} C(X) &= \sum_{e \in E_x} e^a e^b w(e) \\ &= \sum_{e \in E_r} e^a e^b w(e) + (n-1) \sum_{e \in E_x - E_r} w(e). \end{aligned}$$

First, for any $e = (u_i, v_i) \in E_r$,

$$\begin{aligned} e^a e^b w(e) &\leq \left(P_i^a + f(i) P_i^c\right) \left(P_i^b + (1 - f(i)) P_i^c\right) w(P_i) \\ &= P_i^a P_i^b w(P_i) + \left(f(i) P_i^b + (1 - f(i)) P_i^a\right) P_i^c w(P_i). \end{aligned}$$

Recall that for subset of edges $S \subset T$, $w_S(T, i, j)$ stands for $w(SP(T, i, j) \cap S)$. Second, by the triangle inequality,

$$\begin{aligned} \sum_{e \in E_x - E_r} w(e) &\leq \sum_{v \in V} d_T(v, S) + \sum_{i=1}^{h} \sum_{v \in V_i} \left(f(i) w_S(T, v, u_i) + (1 - f(i)) w_S(T, v, v_i)\right) \\ &= \sum_{v \in V} d_T(v, S) + \sum_{i=1}^{h} \left(f(i) Q(P_i) + (1 - f(i)) \left(P_i^c w(P_i) - Q(P_i)\right)\right). \end{aligned}$$

Thus,

$$C(X) \le \sum_{i=1}^{h} P_i^a P_i^b w(P_i) + n \sum_{v \in V} d_T(v,S) + \sum_{i=1}^{h} \min\{P_i^b P_i^c w(P_i) + nQ(P_i), P_i^a P_i^c w(P_i) + n(P_i^c w(P_i) - Q(P_i))\}.$$

Since the minimum of two numbers is not larger than their weighted mean, we have

$$\begin{aligned}&\min\{P_i^b P_i^c w(P_i) + nQ(P_i), P_i^a P_i^c w(P_i) + n\left(P_i^c w(P_i) - Q(P_i)\right)\} \\ &\le \left(P_i^b P_i^c w(P_i) + nQ(P_i)\right) \frac{P_i^a}{P_i^a + P_i^b} + \left(P_i^a P_i^c w(P_i) + n\left(P_i^c w(P_i) - Q(P_i)\right)\right) \frac{P_i^b}{P_i^a + P_i^b}.\end{aligned}$$

Then,

$$\begin{aligned}C(X) &\le \sum_{i=1}^{h} P_i^a P_i^b w(P_i) + n \sum_{v \in V} d_T(v,S) + \sum_{i=1}^{h} \frac{\left(2P_i^a P_i^b P_i^c + nP_i^b P_i^c\right) w(P_i)}{P_i^a + P_i^b} \\ &\quad + \sum_{i=1}^{h} \frac{(P_i^a - P_i^b) nQ(P_i)}{P_i^a + P_i^b} \\ &= n \sum_{v \in V} d_T(v,S) + \sum_{i=1}^{h} \frac{w(P_i)}{P_i^a + P_i^b} \left(\left(P_i^a P_i^b + P_i^b P_i^c\right) n + P_i^a P_i^b P_i^c\right) \\ &\quad + \sum_{i=1}^{h} \frac{(P_i^a - P_i^b) nQ(P_i)}{P_i^a + P_i^b}.\end{aligned}$$

The simplification in the last inequality uses the observation that for any i, we have $P_i^a + P_i^b + P_i^c = n$. By Lemma 5.12,

$$C(X) \le C(T) \max_{1 \le i \le h} \left\{ \frac{1}{1-\delta}, \frac{n}{P_i^a + P_i^b} + \frac{P_i^a P_i^c}{(P_i^a + P_i^b)(P_i^a + P_i^c)} \right\}.$$

Since $P_i^c \le \delta n/2$,

$$\begin{aligned}&\frac{n}{P_i^a + P_i^b} + \frac{P_i^a P_i^c}{(P_i^a + P_i^b)(P_i^a + P_i^c)} \\ &\le \frac{n}{P_i^a + P_i^b} + \frac{P_i^c}{P_i^a + P_i^b} \\ &= \frac{n + P_i^c}{n - P_i^c} \le \frac{2+\delta}{2-\delta} \le \frac{1}{1-\delta}.\end{aligned}$$

This completes the proof. □

In the following section we will show that it is possible to determine the minimum k-star of a graph in polynomial time. In fact, we have the following lemma.

LEMMA 5.14. *The minimum k-star of a graph G can be constructed in time $O(n^{2k})$.*

The proof is delayed to the next section. The following theorem establishes the time-complexity of our PTAS.

THEOREM 5.15. *There exists a PTAS for the* ΔMRCT *problem, which can find a* $(1+\varepsilon)$*-approximation solution in* $O(n^\rho)$ *time complexity where* $\rho = 2\lceil 2/\varepsilon \rceil - 2$.

Proof. By Lemma 5.13, there exists a spanning tree $X \in (\lceil 2/\delta \rceil - 3)^*(G)$ such that $C(X) \leq \frac{1}{1-\delta} C(\widehat{T}_G)$. For finding a $(1+\varepsilon)$-approximation solution, we set $1/\delta = (1/\varepsilon) + 1$ and find a minimum k-star with $k = \lceil 2/\delta \rceil - 3 = \lceil 2/\varepsilon \rceil - 1$. The time complexity is $O(n^\rho)$ where $\rho = 2\lceil 2/\varepsilon \rceil - 2$ from Lemma 5.14. ∎

The result in Theorem 1.1 is immediately derived from Theorem 5.15 and Corollary 4.5.

6. Finding the best *k*-star. In this section we describe an algorithm for finding the minimum routing cost k-star in G for a given value of k. As mentioned before, given an accuracy parameter $\epsilon > 0$, we apply this algorithm for $k = \lceil \frac{2}{\epsilon} - 1 \rceil$ and return the minimum routing cost k-star as a $(1+\epsilon)$-approximate solution.

For a given k, to find the best k-star, we consider all possible subsets S of vertices of size k, and for each such choice, find the best k-star where the remaining vertices have degree one.

6.1. A polynomial-time method. First, we verify that the overall complexity of this step is polynomially bounded for any fixed k. Any k-star can be described by a triple $(S, \tau, \mathcal{L})$, where $S = \{v_1, \ldots, v_k\} \subseteq V$ is the set of k distinguished vertices which may have degree more than one, τ is a spanning tree topology on S, and $\mathcal{L} = (L_1, \ldots, L_k)$, where $L_i \subseteq V \setminus S$ is the set of vertices connected to vertex $v_i \in S$.

Let $l = (l_1, \ldots, l_k)$ be a nonnegative k-vector[2] such that $\sum_{i=1}^k l_i = n - k$. We say that a k-star $(S, \tau, \mathcal{L})$ has the configuration (S, τ, l) if $l_i = |L_i|$ for all $1 \leq i \leq k$. For a fixed k, the total number of configurations is $O(n^{2k-1})$ since there are $\binom{n}{k}$ choices for S, k^{k-2} possible tree topologies on k vertices, and $\binom{n-1}{k-1}$ possible such k-vectors. (To see this, observe that every such vector can be put in correspondence with picking $k-1$ among $n-1$ linearly ordered elements and using the cardinalities of the segments between consecutively picked segments as the components of the vector.) Note that any two k-stars with the same configuration have the same routing load on their corresponding edges. We define $\alpha(S, \tau, l)$ to be the minimum routing cost k-star with configuration (S, τ, l).

Note that any vertex v in $V \setminus S$ that is connected to a node $s \in S$ contributes a term of $w(v, s)$ multiplied by its routing load of $n-1$. Since all these routing loads are the same, the best way of connecting the vertices in $V \setminus S$ to nodes in S is obtained by finding a minimum-cost way of matching up the nodes of $V \setminus S$ to those in S which obey the degree constraints on the nodes of S imposed by the configuration, where the costs are the distances w. This problem can be solved in polynomial time for a given configuration (by a straightforward reduction to an instance of minimum-cost perfect matching). The above minimum-cost perfect matching problem, also called the *assignment* problem, has been well studied and several efficient algorithms can be found in [1]. For instance, by using an $O(n^3)$ algorithm for the assignment problem, the overall complexity would be $O(n^{2k+2})$ for finding the best k-star.

6.2. A faster method. We now show how the minimum k-stars for the different configurations can be computed more efficiently by carefully ordering the matching problems for the configurations and exploiting the common structure of two consecutive problems. In particular, we show how we can obtain the optimal solution of any configuration in this order by performing a single augmentation on the optimal

[2]For any $r \in Z^+$, an r-vector is an integer vector with r components.

solution of the previous configuration. Thus, we show (Lemma 6.2) how to compute $\alpha(S, \tau, l)$ for a given configuration in time $O(nk)$.

Let W_{ab} be the set of all nonnegative a-vectors whose entries add up to a constant b. In $W_{ab} \times W_{ab}$, we introduce the relation $\sim$ as $l \sim l'$ if there exist $1 \leq s, t \leq a$ such that

$$l'_i = \begin{cases} l_i - 1 & \text{if } i = s, \\ l_i + 1 & \text{if } i = t, \\ l_i & \text{otherwise.} \end{cases}$$

For a pair l and l' such as the above, we say that l' *is obtained from* l *by* s *and* t.

Let $r = |W_{ab}| = \binom{a+b-1}{a-1}$. The following proposition shows that the elements of W_{ab} can be linearly ordered as $l^1, \ldots, l^r$ so that $l^{i+1} \sim l^i$ for all $1 \leq i \leq r-1$.

PROPOSITION 6.1. *For all positive integers* a, b, *there exists a permutation* $\pi^{a,b}$ *of* W_{ab} *such that* $\pi^{a,b}_1$ *is the lexicographic minimum,* $\pi^{a,b}_r$ *is the lexicographic maximum, and* $\pi^{a,b}_{i+1} \sim \pi^{a,b}_i$ *for all* $i = 1, \ldots, r-1$.

Proof. By induction. The claim is clearly true when $a = 1$ for any b. Assume the claim is true for all b when $a = m - 1$. For $a = m$ construct the ordering as follows: first, the elements for which $l_1 = 0$, ordered by applying $\pi^{a-1,b}$ to $(l_2, \ldots, l_a)$; then the elements for which $l_1 = 1$, ordered according to *decreasing* $\pi^{a-1,b-1}$. In general each block for which $l_1 = h$ is ordered by applying $\pi^{a-1,b-h}$ to $(l_2, \ldots, l_a)$, forward or backward according to the parity of h. Note that $\pi^{a,b}_{i+1} \sim \pi^{a,b}_i$ within one block. Furthermore, at block boundaries the part $(l_2, \ldots, l_a)$ is either a lexicographic minimum or maximum so that it is feasible to increase by one l_1. Finally, it is obvious that the first and the last of the constructed ordering are the lexicographic minimum and maximum respectively. □

According to Proposition 6.1 we can order the elements of $W_{k,(n-k)}$ as $l^1, \ldots, l^r$, where $r = \binom{n-1}{k-1}$. Note that $l^1 = (0, \ldots, 0, n-k)$ and $l^r = (n-k, 0, \ldots, 0)$. In the remainder of this section, we shall prove the following lemma.

LEMMA 6.2. $\alpha(S, \tau, l^{i+1})$ *can be computed from* $\alpha(S, \tau, l^i)$ *in* $O(nk)$ *time.*

Proof. We shall show that $\alpha(S, \tau, l^{i+1})$ can be found from $\alpha(S, \tau, l^i)$ by means of a shortest path computation. A similar argument is used in [1, Exercise 10.20]; for solving a minimum cost flow problem given the solution of another minimum cost flow problem which differs by only one unit capacity arc.

For convenience, let us rename the vertices so that $S = \{1, \ldots, k\}$. Let $l^i = (|L_1|, \ldots, |L_k|)$ and $(S, \tau, \mathcal{L}) = \alpha(S, \tau, l^i)$. Let us define an auxiliary weighted digraph $D(\mathcal{L}) = (V, A, \delta)$ in which the arc set is $A = \{(u, v) | u \in V \setminus S, v \in S\} \cup \{(u, v) | u \in S, v \in L_u\}$ and $\delta(u, v) = w(u, v)$ if $u \notin S$, and $\delta(u, v) = -w(u, v)$ if $u \in S$. For a node in S, the weight on an outgoing arc reflects the cost reduction for removing a leaf from its neighbors, and the weight on an incoming arc reflects the increase in cost for connecting a leaf to the node.

It is immediately seen that any cycle (not necessary simple) in the graph describes a way of changing $(S, \tau, \mathcal{L})$ into another k-star with the same configuration, and the difference in cost between the new and the old k-stars is given by $(n-1)$ times the length of the cycle. Because $(S, \tau, \mathcal{L})$ is optimal for its configuration, there is no negative length cycle in $D(\mathcal{L})$.

Similarly, if l^{i+1} is obtained from l^i by s and t, then any path from s to t in $D(\mathcal{L})$ changes $(S, \tau, \mathcal{L})$ into a k-star with configuration (S, τ, l^{i+1}). Conversely, any k-star with configuration (S, τ, l^{i+1}) can be obtained by a path from s to t and possibly some cycles. Since positive length cycles contribute positive cost and there is no

negative length cycle, it is clear that there is a path P from s to t, changing $\alpha(S, \tau, l^i)$ into $\alpha(S, \tau, l^{i+1})$, which is simple. To see that it must be a shortest s-t path, let P' be any path from s to t, which changes $\alpha(S, \tau, l^i)$ into a k-star K'. Since K' has the same configuration of $\alpha(S, \tau, l^{i+1})$, the difference between their costs is given by $(n-1)(\delta(P') - \delta(P))$. Therefore, we conclude that P must be the shortest path from s to t in $D(\mathcal{L})$. We now show how such a shortest path can be computed in $O(kn)$ time.

Consider any shortest path $(u_1, v_1, u_2, \ldots, v_{h-1}, u_h)$ between two nodes $u_i \in S$ and $v_i \in V \setminus S$ in $D(\mathcal{L})$. Take two consecutive edges (u_i, v_i) and (v_i, u_{i+1}) in the path. Since the path is shortest, v_i must be such as to minimize the sum of the two edge lengths. Recall that $\delta(u_i, v_i) = -w(u_i, v_i)$ and $\delta(v_i, u_{i+1}) = w(v_i, u_{i+1})$. Then, we have that the sum of the two edge lengths is $\min_{v_i \in L_i}\{w(v_i, u_{i+1}) - w(u_i, v_i)\}$. Therefore, to find the shortest path from s to t on $D(\mathcal{L})$, it is enough to construct a complete digraph $D'(\mathcal{L})$ with vertex set S and lengths δ', in which $\delta'(i, j) = \min_{v \in L_i}\{w(v, j) - w(i, v)\}$. It is easy to see that the length of the shortest path from s to t on $D'(\mathcal{L})$ is the same as the one on $D(\mathcal{L})$. Given the graph $D'(\mathcal{L})$, a shortest s-t path (and also the corresponding path on $D(\mathcal{L})$) can be found in $O(k^2)$ time. Finally, to construct $D'(\mathcal{L})$, for each vertex $i \in S$, we have to find $k-1$ minima (one for every other $j \in S$), each over a set of l_i elements. Adding up, the total time complexity is $(k-1)\sum_{i=1}^{k} l_i = (k-1)(n-k) = O(nk)$. □

We are now able to prove Lemma 5.14, i.e., that a minimum k-star can be found in time $O(n^{2k})$.

Proof. When S and τ are fixed, to find an optimum k-star we begin by $\alpha(S, \tau, l^1)$, which is readily obtained by setting $L_k = V \setminus S$. Then, using Lemma 6.2, we compute the optimal k-stars for configurations $l^2, \ldots, l^r$, and we report the best overall.

In general, a minimum routing cost k-star in G can be found in time $O(n^{2k})$, given by $\binom{n}{k}$ choices for S, k^{k-2} possible tree topologies, and for each fixed S and τ, $\binom{n-1}{k-1}$ configurations, of cost $O(nk)$ each. □

7. Optimal communication spanning trees. We begin with a few definitions following Bartal [3]. Let V be a set of n points and let M be a metric space defined over V. The distance between i and j in M is denoted by $d_M(i, j)$. A metric N over V dominates another metric M over V if for every pair $i, j \in V$, we have $d_N(i, j) \geq d_M(i, j)$.

DEFINITION 7.1. *A metric N over V α-approximates a metric M over V if it dominates M and for every $i, j \in V$, we have $d_N(i, j) \leq \alpha \cdot d_M(i, j)$.*

Define a tree (or additive) metric over V as a metric space corresponding to paths in a tree which contains all the points of V. Note that we allow the tree defining the additive metric to contain points other than those in V.

We are interested in tree metrics that approximate any given metric M. However, even for the simple metric induced by arranging the nodes of V in a cycle, if we restrict ourselves to approximating this by tree metrics, $\alpha = \Omega(|V|)$ [3, 16]. Hence we turn to the following notion.

DEFINITION 7.2. *Let M be a metric space over V. A set of metric spaces S over V α-probabilistically-approximates M, if every metric space in S dominates M and there exists a probability distribution over metric spaces $N \in S$ such that for every $i, j \in V$, $E(d_N(i, j)) \leq \alpha \cdot d_M(i, j)$.*

Bartal's main result is the following theorem.

THEOREM 7.3 (see [3]). *For any metric space on V, it can be $O(\log^2 |V|)$-probabilistically approximated by a set of tree metrics on V. Furthermore, the tree*

metrics and the distribution over them can be computed in polynomial time.

As has been observed earlier [12], it is not hard to transform the tree metrics in this theorem into spanning tree metrics, namely, those that do not contain any extra points other than those in V. We use the above theorem to approximate the given metric M by spanning tree metrics N. By using a spanning tree N randomly picked from this collection according to the given distribution as the solution, the expected value of its communication cost is $\sum_{ij} r_{ij} d_N(i,j) \leq O(\log^2 |V|) \sum_{ij} r_{ij} d_M(i,j)$ by linearity of expectation. By repeatedly picking a few trees and using the best one, this bound is achieved with high probability, giving the result in Theorem 1.2. As mentioned earlier, this bound has been improved and derandomized in [4, 6].

Acknowledgments. Thanks to Sampath Kannan for describing the relevance of the threshold objective for multiple alignments. We also thank Tao Jiang and Howard Karloff for suggesting the merger of two different works to obtain this joint paper. Finally, we thank the referees for their valuable comments.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows—Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[2] V. Bafna, E. L. Lawler, and P. Pevzner, *Approximation algorithms for multiple sequence alignment*, Proceedings of the 5th Combinatorial Pattern Matching Conference, Lecture Notes in Comput. Sci. 807, Springer, New York, 1994, pp. 43–53.

[3] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 184–193.

[4] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 161–168.

[5] H. Carrillo and D. Lipman, *The multiple sequence alignment problem in biology*, SIAM J. Appl. Math., 48 (1988), pp. 1073–1082.

[6] M. Charikar, C. Chekuri, A. Goel, and S. Guha, *Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median*, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 114–123.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1994.

[8] D. Feng and R. Doolittle, *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*, J. Molecular Evol., 25 (1987), pp. 351–360.

[9] D. Gusfield, *Efficient methods for multiple sequence alignment with guaranteed error bounds*, Bull. Math. Biology, 55 (1993), pp. 141–154.

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

[11] 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.

[12] G. Konjevod, R. Ravi, and F. S. Salman, *On Approximating Planar Metrics by Tree Metrics*, manuscript, 1997.

[13] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu, *A tool for multiple sequence alignment*, Proc. Nat. Acad. Sci. USA, 86 (1989), pp. 4412–4415.

[14] S. B. Needleman and C. D. Wunsch, *A general method applicable to search the similarities in the amino acid sequences of two proteins*, J. Molecular Biol., 48 (1970), pp. 443–453.

[15] P. A. Pevzner, *Multiple alignment, communication cost, and graph matching*, SIAM J. Appl. Math., 52 (1992), pp. 1763–1779.

[16] Y. Rabinovich and R. Raz, *Lower bounds on the distortion of embedding finite metric spaces in graphs*, Discrete Comput. Geom., 19 (1998), pp. 79–94.

[17] D. Sankoff and J. B. Kruskal, eds. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison–Wesley, Reading, MA, 1983.

[18] D. Sankoff, C. Morel, and R. J. Cedergren, *Evolution of the 5S Ribosomal RNA*, Nature New Biology, 245 (1973), pp. 232–234.

[19] G. D. Schuler, S. F. Altschul, and D. J. Lipman, *A workbench for multiple alignment construction and analysis*, Proteins Structure Function Genetics, 9 (1991), pp. 180–190.
[20] L. Wang and T. Jiang, *On the complexity of multiple sequence alignment*, J. Comput. Biol., 1 (1994), pp. 337–348.
[21] M. S. Waterman, *Introduction to Computational Biology*, Chapman & Hall, London, 1995.
[22] R. Wong, *Worst-case analysis of network design problem heuristics*, SIAM J. Alg. Discrete Methods, 1 (1980), pp. 51–63.