Distributed Approximation of Minimum Routing Cost Trees

Alexandra Hochuli[1], Stephan Holzer[2,*], and Roger Wattenhofer[1]

[1] ETH Zurich, 8092 Zurich, Switzerland
{hochulia,wattenhofer}@ethz.ch
[2] MIT, Cambridge, MA 02139, USA
holzer@mit.edu

Abstract. We study the NP-hard problem of approximating a Minimum Routing Cost Spanning Tree in the message passing model with limited bandwidth (CONGEST model). In this problem one tries to find a spanning tree of a graph G over n nodes that minimizes the sum of distances between all pairs of nodes. In the considered model every node can transmit a different (but short) message to each of its neighbors in each synchronous round. We provide a randomized $(2+\varepsilon)$-approximation with runtime $\mathcal{O}(D + \frac{\log n}{\varepsilon})$ for unweighted graphs. Here, D is the diameter of G. This improves over both, the (expected) approximation factor $\mathcal{O}(\log n)$ and the runtime $\mathcal{O}(D \log^2 n)$ stated in [13].

Due to stating our results in a very general way, we also derive an (optimal) runtime of $\mathcal{O}(D)$ when considering $\mathcal{O}(\log n)$-approximations as in [13]. In addition we derive a deterministic 2-approximation.

1 Introduction

A major goal in network design is to minimize the cost of communication between any two vertices in a network while maintaining only a substructure of the network. Despite the fact that a tree is the sparsest substructure of a network it can be surprisingly close to the optimal solution. Every network contains a tree whose total cost of communication between all pairs of nodes is only a factor two worse than the communication cost when all edges in the graph are allowed to be used!

The problem of finding trees that provide a low routing cost is studied since the early days of computing in the 1960s [18] and is known to be NP-hard [12] on weighted and unweighted graphs[1]. These days networks of computers and electric devices are omnipresent and trees offer easy and fast implementations for applications. In addition, trees serve as the basis for control structures as well

* Corresponding author. Part of this work was done at ETH Zurich. At MIT the author was supported by the following grants: AFOSR Contract Number FA9550-13-1-0042, NSF Award 0939370-CCF, NSF Award CCF-1217506, NSF Award number CCF-AF-0937274.

[1] Even for seemingly simpler versions than those which we study the problem remains NP-hard [23].

as for information gathering/aggregation and information dissemination. This explains why routing trees are computed and used by wide spread protocols such as the IEEE 802.1D standard [3]. When bridging [19] is used in Local Area Networks (LAN) and Personal Area Networks (PAN), a spanning tree is computed to define the (overlay) network topology. Finding such a tree with low routing cost is crucial. As [3] demonstrates, current implementations do not perform well under the aspect of optimizing the routing costs and there is the need to find better and faster solutions. The nature of this problem and growth of wired and wireless networks calls for fast and good distributed implementations.

In this paper we present new approaches for distributed approximation of a Minimum Routing Cost Spanning Tree (MRCT) while extending previous work for approximation of those. By doing so we improve both, the round complexity and the approximation factor of the best known (randomized) result in a distributed setting for unweighted graphs. Our main contribution is an algorithm that computes a $\left(2 - \frac{2}{n} + \min\left\{\frac{\log n}{D}, \alpha(n,D)\right\}\right)$-approximation in time $\mathcal{O}\left(D + \frac{\log n}{\alpha(n,D)}\right)$ w.h.p.[2]. Previously, the best known distributed approximation for MRCT [13] (on weighted graphs) achieved an (expected) approximation-ratio of $\mathcal{O}(\log n)$ using randomness. The bound on the runtime of the algorithm of [13] is $\mathcal{O}(n \log^2 n)$ in the worst case – even when the network is fully connected (a clique). For unweighted graphs, the authors of [13] specify this runtime to be $\mathcal{O}(D \log^2 n)$. The distributed algorithms we present in this paper are for unweighted graphs as well[3] and compared to the (expected) approximation-ratio $\mathcal{O}(\log n)$ of [13] we essentially obtain a (guaranteed) approximation-ratio $2+\varepsilon$ in time $\mathcal{O}(D + \frac{\log n}{\varepsilon})$ w.h.p.. This follows from choosing $\alpha(n,D) = \varepsilon$ for an arbitrary small $\varepsilon > 0$. When choosing $\alpha(n,D) = \log n$, we obtain the same approximation ratio as in [13] in time $\mathcal{O}(D)$. To be general, we leave the choice of $\alpha(n,D)$ to the reader depending on the application.

Besides this randomized solution we present a deterministic algorithm running in linear time $\mathcal{O}(n)$ achieving an approximation-ratio of 2.

2 Model and Basic Definitions

Our network is represented by an undirected graph $G = (V, E)$. Nodes V correspond to processors, computers or routers. Two nodes are connected by an edge from set E if they can communicate directly with each other. We denote the number of nodes of a graph by n, and the number of its edges by m. Furthermore we assume that each node has a unique ID in the range of $\{1, \ldots, 2^{\mathcal{O}(\log n)}\}$, i.e. each node can be represented by $\mathcal{O}(\log n)$ bits. Nodes initially have no knowledge of the graph G, other than their immediate neighborhood.

We consider a synchronous communication model, where every node can send B bits of information over all its adjacent edges in one synchronous round of

[2] A more precise statement can be found in Theorem 3. This Theorem also considers a generalized version of MRCT.

[3] They extend to graphs with certain realistic weight-functions.

communication. We also consider a modified model, where time is partitioned into synchronized slots, but a message might receive a delay when traversing an edge. This delay might not be uniform but fixed for each edge. In principle it is allowed that in each round a node can send different messages of size B to each of its neighbors and likewise receive different messages from each of its neighbors. Typically we use $B = \mathcal{O}(\log n)$ bits, which allows us to send a constant number of node or edge IDs per message. Since communication cost usually dominates the cost of local computation, local computation is considered to be negligible. For $B = \mathcal{O}(\log n)$ this message passing model is known as CONGEST model [15]. We are interested in the number of rounds that a distributed algorithm needs to solve some problem. This is the time complexity of the algorithm.

To be more formal, we are interested in evaluating a function $g : \mathbb{G}_n \to S$, where $\mathbb{G}_n$ is the set of all graphs over n vertices and S is e.g. $\{0,1\}$, $\mathbb{N}$ or $\mathbb{G}_n$, and define distributed round complexity as follows:

Definition 1 (Distributed round complexity). *Let $\mathcal{A}$ be the set of distributed deterministic algorithms that evaluate a function g on the underlying graph G over n nodes (representing the network). Denote by $R^{dc}(A(G))$ the distributed round complexity (indicated by dc) representing the number of rounds that an algorithm $A \in \mathcal{A}$ needs in order to compute $g(G)$. We define $R^{dc}(g) = \min_{A \in \mathcal{A}} \max_{G \in \mathbb{G}_n} R^{dc}(A(G))$ to be the smallest amount of rounds/time slots any algorithm needs in order to compute g.*

We denote by $R^{dc-rand}_{\varepsilon}(g)$ the randomized round complexity of g when the algorithms have access to randomness and compute the desired output with an error probability smaller than ε. By w.h.p. (with high probability) we denote a success probability larger than $1 - 1/n$.

The unweighted shortest path in G between two nodes u and v is a path with minimum number of edges among all (u,v)-paths. Denote by $d_G(u,v)$ the unweighted distance between two nodes u and v in G which is the length of an unweighted shortest (u,v)-path in G. We also say u and v are $d_G(u,v)$ hops apart. By $\omega_G : E \to \mathbb{N}$ we denote a graph's weight function and by $\omega_G(e)$ the weight of an edge in G. By $\omega_G(u,v) := \min_{\{P|P \text{ is } (u,v)\text{-path in } G\}} \sum_{e \text{ is edge in } P} \omega_G(e)$ we define the weighted distance between two nodes u and v, that is the weight of a shortest weighted path in a graph G connecting u and v[4].

The time-bounds of our algorithms as well as those of previous algorithms depend on the diameter of a graph. We also use the eccentricity of a node.

Definition 2 (Eccentricity, diameter). *The* weighted eccentricity $ecc_{\omega_G}(u)$ *in G of a node u is the largest weighted distance to any other node in the graph, that is $ecc_{\omega_G}(u) := \max_{v \in V} \omega_G(u,v)$. The* weighted diameter $D_\omega(G) := \max_{u \in V} ecc_{\omega_G}(u) := \max_{u,v \in V} \omega_G(u,v)$ *of a graph G is the maximum weighted distance between any two nodes of the graph. The* unweighted diameter *(or hop*

[4] Note that in the context of MRCT, ω often corresponds to the cost of an edge. In the literature the routing cost between any node u and v in a given spanning tree T of G is usually denoted by $c_T(u,v)$, while in generalized versions of MRCT, the weight of an edge can be different from the cost. In this paper we use $\omega_T(u,v) = c_T(u,v)$.

diameter) $D_h(G) := \max_{u,v \in V} \min_{\{P | P \text{ is } (u,v)\text{-path}\}} |P|$ *of a graph* G *is the maximum number of hops between any two nodes of the graph. Here* $|P|$ *indicates the number of edges on path* P.

We often write D_ω and D_h instead of $D_\omega(G)$ and $D_h(G)$ when we refer to the diameter of a graph G in context. Observe that $D_h = D_\omega$ for unweighted graphs.

Finally, we define the problems that we study.

Definition 3 (S-Minimum Routing Cost Tree (S-MRCT)). *Let* S *be a subset of the vertices* V *in* G*. The* S*-routing cost of a subgraph* H *is defined as* $RC_S(H) := \sum_{u,v \in S} \omega_H(u,v)$ *and denotes the routing cost of* H *with respect to* S*. An* S*-MRCT is a subgraph* T *of* G *that is a tree, contains all nodes* S *and has minimum* S*-routing cost* $RC_S(T)$ *among all spanning trees of* T.

This is a generalization of the MRCT problem [22]. According to this definition V-MRCT (i.e. $S = V$) and MRCT of [22] are equivalent. Therefore all results are valid for the classical MRCT problem when choosing $S := V$.

In this paper we consider approximation algorithms for these problems. Given an optimization problem P, denote by OPT the cost of the optimal solution for P and by SOL_A the cost of the solution of an algorithm A for P. We say A is ρ-approximative for P if $OPT \leq SOL_A \leq \rho \cdot OPT$ for any input.

Fact 1 *The eccentricity of any node is a good approximation of the diameter. For any node* $u \in V$ *we know that* $ecc_{\omega_G}(u) \leq D_\omega(G) \leq 2 \cdot ecc_{\omega_G}(u)$.

3 Our Results

In Section 8 we prove the following two theorems.

Theorem 2. *In the CONGEST model, the deterministic algorithm proposed in Section 8 needs time* $\mathcal{O}(|S| + D_\omega)$ *to compute a* $(2 - 2/|S|)$*-approximation for* S*-MRCT when using either uniform weights for all edges or a weight function* $\omega(e)$ *that reflects the delay/edge traversal time of edge* e.

Theorem 3. *Let* $\alpha(n, D_\omega)$ *be some function in* n *and* D_ω*. The randomized algorithm proposed in Section 8 computes w.h.p. a* $\left(2 - \frac{2}{|S|} + \min\left\{\frac{\log n}{D_\omega}, \alpha(n, D_\omega)\right\}\right)$*-approximation for* S*-MRCT in the CONGEST model in time* $\mathcal{O}\left(D_\omega + \frac{\log n}{\alpha(n, D_\omega)}\right)$ *when using either uniform weights for all edges or a weight function* $\omega(e)$ *that reflects the delay/edge traversal time of edge* e.

We emphasize that the analysis of [20] yields a 2-approximation when compared to the routing cost in the original graph[5] and that we modify this analysis.

[5] Note that most other approximation algorithms are with respect to the routing cost of a minimal routing cost tree of the graph. In the full version of this paper [8] we provide an example that shows that sometimes even no subgraph with $o(n^2)$ edges exists that yields better approximations to the routing cost in the original graph than the trees presented here. From this we conclude that algorithms that compare their result only to the routing cost of the minimum routing cost tree do not always yield better results than those presented here.

4 Related Work

Minimum Routing Cost Trees are also known as uniform Minimum Communication Cost Spanning Trees [16,17] and shortest Total Path Length Spanning Trees [21]. Furthermore the MRCT problem is a special case of the Optimal Network Problem, first studied in the 1960s by [18] and later by [6]. In [20] Wong presented heuristics and approximations to the Optimal Network Problem with a restriction that makes the problem similar to the MRCT problem and obtained a 2-approximation. In [12] it is shown that this restricted version, which Wong studied on unweighted graphs, is NP-hard as well. It seems that earlier the authors of [11] formulated a similar problem under the name "Optimum communication spanning tree" where in addition to costs on edges, we are given a requirement-value $r_{u,v}$ for each pair of vertices that needs to be taken into account when computing the routing cost. In this setting one wants to find a tree T such that $\sum_{u,v \in V} r_{u,v} d_T(u,v)$ is minimized. In [22] it is argued that for metric graphs, the results by [1,2,4] yield a $\mathcal{O}(\log n \log\log n)$-approximation to this problem. Using a result presented in [7], this can be improved to be an $\mathcal{O}(\log n)$-approximation. In [13] it is shown how to implement this result in a distributed setting. They state their result depending on the shortest path diameter $D_{sp}(G) := \max_{u,v \in V}\{|P| \; |P$ is a shortest weighted (u,v)-path$\}$ of a graph. This diameter represents the maximum number of hops of any shortest weighted path between any two nodes of the graph. The authors of [13] obtain a randomized approximation of the MRCT with expected approximation-ratio $\mathcal{O}(\log n)$ in time $\mathcal{O}\left(D_{sp} \cdot \log^2(n)\right)$. Observe that this might be only a $\mathcal{O}(n \log^2 n)$-approximation even in a graph with $D_h = 1$ and $D_{sp} = n-1$, such as a clique where all edges have weight n except $n-1$ edges of weight 1 forming a line as a subgraph.[6] In our distributed setting we know that it is hard to approximate an MRCT due to Theorem 4.

Theorem 4 (Version of Theorem 5.1. of [5]). *For any polynomial function $\alpha(n)$, numbers p, $B \geq 1$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, there exists a constant $\varepsilon > 0$ such that in the CONGEST model any distributed $\alpha(n)$-approximation algorithm for the MRCT problem whose error probability is smaller than ε requires $\Omega\left(\left(\frac{n}{pB}\right)^{\frac{1}{2}-\frac{1}{2(2p+1)}}\right)$ time on some $\Theta(n)$-vertex graph of diameter $2p+2$.*

For certain realistic weight-functions our randomized algorithm breaks this $\Omega(\sqrt{n}+D)$-time lower bound. This is no contradiction, as the construction of [5] heavily relies on being able to choose highly different weights, which might not always appear in practice: in current LAN/PAN networks, weights (delays) usually differ only by a small factor. In case the weights are indeed the delay-times, the runtime of our algorithm just depends on the maximal delay that occurs between any two nodes in the network. Observe that also the runtime of the algorithm of [13] stated for arbitrary weight functions does not contradict this

[6] According to [22] it is NP-hard to find an MRCT in a clique.

approximation lower bound. The algorithm's runtime depends on the shortest path diameter D_{sp}, which is $\Theta(\sqrt{n}+D)$ in the worst case graphs provided in [5]. Finally we want to point out that for weighted graphs it might be possible to combine the recent result of [14] with the techniques developed in this paper. This might improve over the approximation factor of [13] for weighted graphs while getting a better runtime in some cases.

Related work in the non-distributed setting includes [22], where a PTAS to find the MRCT of a weighted undirected graph is presented. It is shown how to compute a $(1 + 2/(k + 1))$-approximation for any $k \geq 1$ in time $\mathcal{O}\left(n^{2k}\right)$. Details on the limits of transferring this PTAS into our distributed setting can be found in the full version of this paper [8]. In [8] we also summarize further related work in other models (non-distributed and parallel) that deal with the MRCT problem as well as the related problem of computing low stretch spanning trees.

5 Trees That 2-Approximate the Routing Cost

The main structure we need in this section are shortest path trees:

Definition 4 (Shortest path tree). *A shortest path tree (SP-tree) rooted in a node v, is a tree that connects any node u to the root v by a shortest path in G. In unweighted graphs, this is simply a breadth first-search tree.*

Previously it was known due to Wong [20], Theorem 3, that there is an SP-tree, which 2-approximates the routing cost of an MRCT. We restate this result by using an insight stated in Wong's analysis such that this tree not only 2-approximates the routing cost $RC_V(T)$ of an MRCT T of G (which is a V-MRCT) as Wong stated it, but even yields a 2-approximation of the routing cost $RC_V(G)$ when using shortest paths in the network G itself. Thus, on average the distances between two pairs in the tree are only a factor 2 worse than the distances in G.

The algorithm that corresponds to Wong's analysis computes and evaluates n SP-trees, one for each node in V. We show, that for the S-MRCT problem it is sufficient to consider only those shortest path trees rooted in nodes of S. At the same time, a slightly more careful analysis yields a slightly improved approximation factor of $2 - 2/|S|$, which is of interest for small sets S. Before we start, we define a useful measure for the analysis.

Definition 5 (Single source routing cost). *By $SSRC_S\left(v\right) := \sum_{u\in S} \omega_G\left(v, u\right)$ we denote the sum of the single source routing costs from node v to every other node in S by using edges in G.*

Note that for simplicity we defined an SP-tree to contain all nodes of V. However, one could also consider the subtree where all leaves are nodes in S. The measures RC_S and $SSRC_S$ would not change, as any additional edges are never used by any shortest paths and thus do not contribute to the S-routing cost of the tree. Such a tree can easily be obtained from the tree we compute.

Theorem 5. *Let $|S|$ be at least 2. In weighted graphs, the SP-tree T_v rooted in a node v with minimal single source routing cost $SSRC_S(v) = \min_{u \in S} SSRC_S(u)$ over all SP-trees rooted in nodes of S is a $(2 - 2/|S|)$-approximation to the S-routing cost $RC_S(G)$ in G.*

Corollary 1. *In weighted graphs, an SP-tree with minimum routing cost over all SP-trees rooted in nodes of S is a $(2 - 2/|S|)$-approximation to an S-MRCT.*

The proof of this theorem uses and modifies the ideas of the proof of Theorem 3 in [20]. The following proof is an adapted version of this proof.

Proof. Let v be the node for which the SP-tree T_v has minimal single source routing cost with respect to S among all SP-trees, that is $v := arg \min_{v \in V} SSRC_S\left(v\right)$.

The cost of connecting a node $u \neq v$ to all other nodes in S using edges in T_v is upper bounded by $(|S| - 2) \cdot \omega_G\left(v, u\right) + SSRC_S\left(v\right)$. This essentially describes the cost of connecting u to each other node by a path via the root v and using edges in T_v. Therefore the total routing cost $RC_S\left(T_v\right)$ for S using the network T_v can be bounded by

$$RC_S\left(T_v\right) \leq SSRC_S\left(v\right) + \sum_{v \neq u \in S} \left(\left(|S| - 2\right) \cdot \omega_G\left(v, u\right) + SSRC_S\left(v\right)\right).$$

As $|S| \geq 2$, this can be further transformed and bounded to be

$$\begin{aligned} &= |S| \cdot SSRC_S\left(v\right) + \left(|S| - 2\right) \sum_{u \in S} \omega_G\left(v, u\right) \\ &= |S| \cdot SSRC_S\left(v\right) + \left(|S| - 2\right) \cdot SSRC_S\left(v\right) \\ &= \left(2 - 2/|S|\right) \cdot |S| \cdot SSRC_S\left(v\right) \\ &\leq \left(2 - 2/|S|\right) \cdot \sum_{u \in S} SSRC_S\left(u\right). \end{aligned}$$

Where the last bound follows, as $SSRC_S\left(v\right)$ is minimal among all $SSRC(u)$ for $u \in S$. Since $\sum_{u \in V} SSRC_S\left(u\right)$ is the same as $RC_S\left(G\right)$, we obtain that $RC_S\left(T_v\right) \leq 2RC_S\left(G\right)$. □

6 Considering few Randomly Chosen SP-Trees Is Almost as Good

We show that when investigating a small subset of all SP-trees chosen uniformly at random, with high probability one of these trees is a good approximation as well.

Lemma 1. *Let $\beta(n, D)$ be a positive function in n and D and define $\gamma := \left\lceil \frac{2-2/|S|}{\beta(n,D)} \right\rceil + 1$. Assume $S \subseteq V$ is of size at least $\gamma \ln n$. Let S' in turn be a subset of S chosen uniformly at random among all subsets of S of size $\gamma \ln n$. Let $v \in S'$ be a node such that $SSRC_S(v) = \min_{u \in S'} SSRC_S(u)$. Then $RC_S(T_v) \leq (2 - 2/|S| + \beta(n, D))RC_S(G)$.*

Proof. For simplicity, without loss of generality we assume that $|S|$ is a multiple of γ. Denote by $v_1, \ldots, v_{|S|}$ the nodes in S such that $SSRC_S(v_1) \leq SSRC_S(v_2) \leq \cdots \leq SSRC_S(v_{|S|})$. That is they are ordered corresponding to their single source routing costs. We say a node v is good, if the corresponding SP-tree T_v is among the $1/\gamma$-fraction of the SP-trees with lowest single source routing cost[7] . Therefore v is good if $SSRC_S(v) \leq SSRC_S(v_{|S|/\gamma})$ with respect to the above order of the trees.

First we prove that w.h.p. set S' contains a good node. Second we prove, that the corresponding SP-tree yields the desired approximation ratio.

1) Probability analysis: We know that $Pr_{v \in S}[v \text{ is good}] = 1/\gamma$. Furthermore each node $v \in S$ is included in set S' independent of the other nodes. Therefore we can conclude that the probability that at least one of the nodes v in S' is good is $1 - \left(1 - \frac{1}{\gamma}\right)^{|S'|} = 1 - \left(1 - \frac{1}{\gamma}\right)^{\gamma \ln n} > 1 - 1/n$ and thus high.

2) Approximation-ratio analysis: Let v_i be a good node. As in the proof of Theorem 5 we know that $RC_S(T_{v_i}) \leq (2 - 2/|S|) \cdot |S| \cdot SSRC_S(v_i)$..As $RC_S(G) = \sum_{u \in S} SSRC_S(u)$ and v_i is good, we can conclude that $SSRC_S(v_i) \leq \frac{1}{(1-1/\gamma)\cdot|S|} \cdot RC_S(G)$ as there are at most $(1-1/\gamma)|S|$ nodes v_j with $SSRC_S(v_j) \geq SSRC_S(v_i)$. Equality is approached in the worst case, where $j := |S|/\gamma$ and $SSRC_S(v_j) = 0$ for each $j < i$ and $SSRC_S(v_i) = SSRC_S(v_j)$ for all $j \geq i$.

Combined with Bound (6) it follows that $RC_S(T_{v_i}) \leq \frac{2-2/|S|}{1-1/\gamma} \cdot RC_S(G)$. Due to the choice of γ we conclude the statement of the Lemma.

7 How to Compute the Routing Cost of Many SP-Trees in Parallel

In Theorem 5 (and Lemma 1) we demonstrated that an SP-tree T_v with minimum single source routing cost yields a 2-approximation for $RC_S(G)$. The single source routing cost of a tree can be computed by computing distances between the root of a tree and nodes in S. However, instead of finding an SP-tree with smallest single source routing cost the literature usually considers finding an SP-tree with smallest routing cost. This is done e.g. in [20]. The reason for this is that the bound in the proof of Lemma 5 is not sharp when using the single source routing cost. To see this, we recall that while obtaining the bound, one approximates the distance between two nodes in the tree by adding up their distance to the root. Thus the bound considers the single source routing cost of an SP-tree. Compared to this, the routing cost takes the actual distance of the two nodes in an SP-tree into account. An explicit example for a graph that contains a node u such that $RC_S(T_u) < RC_S(T_v)$, where T_v has minimum single source routing cost is given in the full version of this paper [8]. Like in [20] we focus on this more powerful version of finding a tree of small routing cost.

[7] Due to the choice of $\gamma := \left\lceil \frac{2-2/|S|}{\beta(n,D)} \right\rceil + 1$ a good tree is among the $n\beta(n, D)$ cheapest trees.

Lemma 2. *Let $S := \{v_1, \ldots, v_{|S|}\}$ be a subset*[8] *$S \subseteq V$ of all nodes of a graph. Then we can compute the values $RC_S(T_{v_1}), \ldots, RC_S(T_{v_{|S|}})$ in time $\mathcal{O}(D_\omega + |S|)$ when using either uniform weights for all edges or a weight function implied by the delay/edge traversal time.*

The proof of this lemma can be found at the end of this section. First, we describe our algorithm that is used to prove this lemma. In Part 1 of this algorithm we start by computing SP-trees T_v for each $v \in S$. A pseudocode for this algorithm can be found as Algorithm 7.1. Part 2 deals with computing the routing cost of a single tree and is described later in this section.

We start by noting that for the weight functions we consider an SP-tree is just a Breath First Search tree (BFS-tree). This part is essentially the same as in the S-SP algorithm of [10] extended to edge-weights derived from the delays to send a message. We also store some additional data that is used later in Algorithm 7.2 to compute routing costs but was not needed for the S-SP computation in [10]. In Algorithm 7.2, for each node $v \in S$ an SP-tree T_v is constructed using what we call delayed breadth first search (DBFS). By DBFS we think of a breadth first search, where traversing edge (u, u') takes $\omega_G(u, u')$ time slots. In the end each node u in the graph knows $\omega_G(u, v)$. In addition each node u knows for each $v \in S$ its parent in the corresponding tree T_v. Furthermore node u knows at what time the DBFS, that computed T_v, sent its message to u via u's parent. During Algorithm 7.2, these timestamps are used to compute the routing cost of all these trees in time $\mathcal{O}(|S| + D_\omega)$.

Remark 1. Compared to Algorithm S-SP presented in [10] we added Lines 2, 6 and 26 in Algorithm 7.1 and extended the algorithm to certain delay functions as mentioned above (the proof in [10] can be naturally extended to those.) By doing so, we can store in $\tau[v]$ the time when a message of the computation of tree T_v was received the first time (via edge *parent_in_*T_v). In the end, $\omega_u[v]$ stores the distance $\omega_G(v, u)$ to v and *parent_in_*T_v indicates the first edge of a (u, v)-path witnessing this.

Despite its similarity to algorithm S-SP in [10], we describe Algorithm 7.1 in more detail for completeness. For the simplicity of the writeup, we refer to u not only as a node, we use u to refer to u's ID as well. Each node u stores $\delta(u)$ sets L_i, one for each of the $\delta(u)$ neighbors $u_1, \ldots, u_{\delta(u)}$ of u, and the sets L and L_{delay} to keep track of which messages were received, transmitted or need to be delayed. At the beginning, if $u \in S$, all these sets contain just u, else they are empty (Lines 1–7). Set L_{delay} is always initialized to be empty. Furthermore u maintains an array ω_u that eventually stores at position v (indicated by $\omega_u[v]$) the distance $\omega_G(u, v)$ to node v. Initially $\omega_u[v]$ is set to infinity for all v and is updated as soon as the distance is known (Line 27). In each node u, array τ stores at position v the time when a message of the computation of tree T_v was received the first time in u. At any time, set L contains all node IDs corresponding to

[8] Note that S used here can be e.g. S as in Section 5 or the smaller set S' as in Section 6.

Algorithm 7.1. Computing $SSRC_S(v)$ for each $v \in S$ Part 1 (executed by node u)

```
1:  $L := \emptyset$; $\omega_u := \{0, 0, \ldots, 0\}$; $L_{delay} := \emptyset$;
2:  $\tau := \{\infty, \infty, \ldots, \infty\}$ // **new**
3:  if $u \in S$ then
4:      $L := \{u\}$;
5:      $\omega_u(u) := 0$;
6:      $\tau(u) := 0$; // **new**
7:  end if
8:  $L_1, \ldots, L_{\delta(u)} := L$;
9:  if $u$ equals 1 then
10:     compute $D'_\omega := ecc(u)$; //** According to Fact 1, $D_\omega$ is smaller than $2 \cdot D'_\omega$.
11:     broadcast $D'_\omega$;
12: else
13:     wait until $D'_\omega$ was received;
14: end if
15: //** Compute $S$ shortest path trees
16: for $t = 1, \ldots, |S| + 2 \cdot D'_\omega$ do
17:     for $i = 1, \ldots, \delta(u)$ do
18:         $(l_i, \omega_i) := \begin{cases} \perp & : \text{if } L_i \setminus \cap L_{delay} = \emptyset \\ \arg\min \{v \in L_i \setminus L_{delay} \mid \tau[v] + \omega_G(u,v) \geq t\} & : \text{else} \end{cases}$
19:     end for
20:     within one time slot:
          if $l_1 \neq \perp$ then send $(l_1, \omega_u[l_1] + \omega_G(u, u_1))$ to neighbor $u_1$;
          receive $(r_1, \omega_1)$ from $u_1$;
          ⋮
          if $l_{\delta(u)} \neq \perp$ then send $\left(l_{\delta(u)}, \omega_u[l_{\delta(u)}] + \omega_G\left(u, l_{\delta(u)}\right)\right)$ to neighbor $u_{\delta(u)}$;
          receive $\left(r_{\delta(u)}, \omega_{\delta(u)}\right)$ from $u_{\delta(u)}$;
21:     $R := \{r_i | r_i < l_i \text{ and } i \in 1 \ldots \delta(u)\} \setminus L$
22:     $s := \begin{cases} \infty & \text{if } L_{delay} = \emptyset \\ \min(L_{delay}) & \text{else} \end{cases}$
23:     if $s \leq \min(R)$ and $s < \infty$ then
24:         $L_{delay} := L_{delay} \setminus \{s\}$;
25:     end if
26:     for $i = 1, \ldots, \delta(u)$ do
27:         if $r_i < l_i$ then
28:             //** $T_{l_i}$'s message is delayed due to $T_{r_i}$.
29:             if $r_i \notin L$ then
30:                 $\tau[r_i] := t$; // **new**
31:                 $\omega_u[r_i] = \omega_i$;
32:                 $L := L \cup \{r_i\}, L_1 := L_1 \cup \{r_i\}, L_2 := L_2 \cup \{r_i\}$,
                    $\ldots L_{i-1} := L_{i-1} \cup \{r_i\}, L_{i+1} := L_{i+1} \cup \{r_i\}, \ldots L_{\delta(u)} := L_{\delta(u)} \cup \{r_i\}$;
33:                 if $\min(R) < r_i$ or $s < r_i$ then
34:                     $L_{delay} = L_{delay} \cup \{r_i\}$
35:                 end if
36:                 $parent\_in\_T_{r_i}$ := neighbor $i$;
37:             end if
38:         else
39:             $L_i := L_i \setminus \{l_i\}$; //** $T_{l_i}$'s message was successfully sent to neighbor $i$.
40:         end if
41:     end for
42: end for
```

the tree computations (where each node with a stored ID is the root initiating the computation of such a tree) that already reached u until now. The set L_{delay} contains all root IDs that reached v until time t but are marked to be delayed before forwarded. This ensures that we indeed compute BFS-trees.

Set L_i contains all IDs of L except those that could be forwarded successfully to neighbor u_i in the past. We say an ID l_i is forwarded successfully to neighbor u_i, if u_i is not sending a smaller ID r_i to u at the same time.

To compute the trees in Algorithm 7.1, the unique node with ID 1 computes D'_ω and thus a 2-approximation to the distance-diameter D_ω. This value is subsequently broadcast to the network (Lines 8–12). Then the computation of the $|S|$ trees starts and runs for $|S| + 2D'_\omega$ time steps. Lines 14–17 make sure that at any time the smallest ID, that is not marked to be delayed and was not already forwarded successfully to neighbor u_i is sent to u_i together with the length of the shortest (v, u_i)-path that contains u. In Line 18 we define the set R of all IDs that are received successfully in this time slot for the first time. This set is then used to decide whether to remove an ID s from L_{delay} in Lines 20 and 21, since all IDs that cause a delay to s are transmitted successfully by now. ID s is computed in Line 20. ID s is the smallest element of L_{delay} and is removed from L_{delay} if no other ID smaller than s was received successfully for the first time in this timeslot.

If a node ID r_i was received successful for the first time (verified in Lines 23 and 25), we update $\tau[r_i]$ and $\omega_u[r_i]$, add r_i to the according lists (Lines 28–30) and remember who u's parent is in T_{r_i} (Line 31). In case the ID v was received the first time from several neighbors, the algorithm as we stated it chooses the edge with lowest index i. On the other hand if we did not successfully receive a message from neighbor u_i but sent successfully a message to neighbor u_i, the transmitted ID is removed from L_i (Line 33).

Lemma 3. *Algorithm 7.1 computes an SP-tree T_v for each $v \in S$ in time $\mathcal{O}(|S| + D_\omega)$.*

Proof. This is essentially Theorem 6.1. in [9] stated for Algorithm 7.1 instead of Algorithm S-SP of [9]. Those parts of the two algorithms which contribute to the runtime and correctness are equivalent.

Now Part 2 of our algorithm calculates the routing cost of each tree T_v in parallel in time $\mathcal{O}(D_\omega + |S|)$. A pseudocode of this algorithm is stated in Algorithm 7.2.

To compute the routing cost of a tree, we look at each edge e in each tree T_v and compute the number of (v, w)-paths in T_v that contain the edge e, for $v, w \in S$. The sum of these numbers for each edge in a tree is the tree's routing cost. Given a tree T, for each edge e in T, the edge partitions the tree into two trees (when e was removed). To be more precise, denote by w_e, w'_e the two vertices to which e is incident. Edge e partitions the vertices of T into two subsets, which we call Z_e^1 and Z_e^2 defined by:

$$Z_e^1(T) := \{w \in S | e \text{ is contained in the unique } (w_e, w)\text{-path in } T\}$$
$$Z_e^2(T) := \{w \in S | e \text{ is contained in the unique } (w'_e, w)\text{-path in } T\}$$

We observe that edge e occurs in all $|Z_e^2(T)|$ paths from any node $v \in Z_e^1(T)$ to any node $w \in Z_e^2(T)$. Note that the total number of paths in which e occurs is $|Z_e^1(T)| \cdot |Z_e^2(T)|$. This fact is later used to compute $RC_S(T)$.

Algorithm 7.2. Computing $RC_S(T_v)$ for each $v \in S$ alternative Part 2 (executed by node u)

```
1: $rc_S := \{\infty, \ldots, \infty\}$; //** is updated during the runtime of the algorithm.
2: if $u \in S$ then
3:     $z := \{1, \ldots, 1\}$; //** is updated during the runtime of the algorithm.
4: else
5:     $z := \{0, \ldots, 0\}$;
6: end if
7: for $t = 1, \ldots, |S| + 2D'_\omega$ do
8:     within one time slot:
         For each $v \in L$ such that $t = |S| + 2 \cdot D'_\omega - \tau[v]$ send $(v, rc_S[v], z[v])$ to
         $parent\_in\_T_v$;
         receive $(v_1, r_1, z_1)$ from neighbor $u_1$; //** $r_1$ equals $rc_S(T_{v_1}, u_1)$,
                                                 //** $z_1$ equals $Z^1_{(u,u_1)}(T_{v_1})$
         receive $(v_2, r_2, z_2)$ from neighbor $u_2$; //** $r_2$ equals $rc_S(T_{v_2}, u_2)$,
                                                 //** $z_2$ equals $Z^1_{(u,u_2)}(T_{v_2})$
         ⋮
         receive $(v_{\delta(u)}, r_{\delta(u)}, z_{\delta(u)})$ from $u_{\delta(u)}$; //** $r_{\delta(u)}$ equals $rc_S(T_{v_{\delta(u)}}, u_{\delta(u)})$,
                                                 //** $z_{\delta(u)}$ equals $Z^1_{(u,u_{\delta(u)})}(T_{v_{\delta(u)}})$
9:     for $i = 1, \ldots, \delta(u)$ do
10:        if $v_i \neq \perp$ then
11:            $rc_S[v_i] := rc_S[v_i] + r_i + 2\omega_G(u, v) \cdot z_i \cdot (|S| - z_i)$;
12:            $z[v] := z[v] + z_i$;
13:        end if
14:    end for
15: end for
16: //** Now $rc_S[u]$ equals $RC_S(T_u)$ in case that $u \in S$. Else it is $\infty$ and was never
    modified.
```

Lemma 4. *For a tree T, the routing cost $RC_S(T)$ can be restated as $RC_S(T) = 2 \cdot \sum_{e \in T} |Z_e^1(T)| \cdot |Z_e^2(T)| \cdot \omega_G(e)$.*

The proof of this lemma can be found in the full version of this paper [8].

To formulate the definition of $RC_S(T)$ in this way helps us to argue that we can compute $RC_S(T)$ recursively in a bottom-up fashion for any T. To do so, we consider trees to be oriented such that we use the notion of child/parent.

Definition 6 (Subtree, partial routing cost). *Given a tree T, for each node u in an oriented tree T, we define $T|_u$ to be the subtree of T rooted in u containing all descendants of u in T. Denote by V_v the vertices in $T|_v$. Given* *node u, denote by $rc_S(T,u)$ the part of the routing cost $RC_S(T)$ that is due to the edges in $T|_u$. We define $rc_S(T,u)$ in a recusive way. In case that $T|_u$ consists of only one node, $T|_u$ contains no edges that could contribute to $rc_S(T,u)$ and we set $rc_S(T,u) := 0$. In case that $T|_u$ contains more than one node, we denote the children of u in T by $u_1, \ldots, u_{\delta(u)-1}$ and define $rc_S(T,u) := \sum_{i=1}^{\delta(u)-1} rc_S(T,u_i) + 2 \cdot \sum_{i=1}^{\delta(u)-1} \omega_G(u,u_i) \cdot |Z^1_{(u,u_i)}(T)| \cdot |Z^2_{(u,u_i)}(T)|$.*

Note that $rc_S(T,u)$ is a measure with respect to the routing cost in T and thus different from $RC_S(T|_u)$. Besides $RC_S(T|_u)$ being undefined when $T|_u$ does not contain all nodes in S, $RC_S(T|_u)$ would take only routing cost within $T|_u$ into account.

We now formally prove that $rc_S(T,u)$ essentially describes the contribution of edges in subtree $T|_u$ to the total routing cost and conclude:

Lemma 5. *Let T be a tree rooted in node r. Then $RC_S(T) = rc_S(T,r)$.*

The proof of this lemma can be found in the full version of this paper [8].

Using this insight we are able to compute $RC_S(T_v)$ for all $v \in S$ in parallel recursively in a bottom-up fashion. This is by computing $rc_S(T_v,u)$ for each u based on aggregating $rc_S(T_v,u_j)$ for each of u's children. For each $v \in S$ these computations of $RC_S(T_v)$ run in parallel. A schedule on how to do these bottom-up computations in time $\mathcal{O}(|S| + D_\omega)$ is provided by using the inverted entries of τ.

In more detail each node u computes for each $v \in S$ the costs $rc_S(T_v,u)$ (stored in $rc_S[v]$) of its subtree of T_v as well as the number of nodes in $T_v|_u$ (stored in $z[v]$ and sends this information to its parent in T_v. When we computed T_v in Algorithm 7.1, we connected u via edge *parent_in_*T_v to T_v at time $\tau[v]$. To avoid congestion we send information from u to its parent in T_v only at time $t = |S| + 2D'_\omega - \tau[v]$ (Line 7). Note that this schedule differs from the one that is implied by the computation of the trees in the sense that now only edges in the tree are used, while more edges were scheduled while building the trees. The edges used now in time slot $t = |S| + 2D'_\omega - \tau[v]$ are a subset of those scheduled at time $t = |S| + 2D'_\omega - \tau[v]$ while constructing the trees, such that there is no congestion from this modification.

At the same time as u sends, u receives messages from its neighbors. E.g. neighbor u_i might send $rc_S(T_{v'},u_i)$ and $Z^1_{(u,u_i)}(T_{v'})$ for another node v'. In Lines $8 - 11$ node u updates its memory depending on the received values. In the end the node with ID 1 computes $v := arg\min_{v \in V} RC_S(T_v)$ via aggregation using T_1. Node 1 informs the network that tree T_v is a 2-approximation to an S-MRCT.

Theorem 6. *The algorithm presented in this section computes all $|S|$ values $RC_S(T_v)$ for each node $v \in S$ in time $\mathcal{O}(|S| + D_\omega)$.*

Proof. **Runtime:** The construction of the $|S|$ trees in Algorithm 7.1 takes at most $\mathcal{O}(|S| + D_\omega)$ rounds as stated in Lemma 3. To forward/compute the costs from the leaves to the roots $v \in S$ in Algorithm 7.2 takes $|S| + 2D'_\omega$ since we just use the schedule τ of this length computed in Algorithm 7.1. Thus the total time used is $\mathcal{O}(|S| + D_\omega)$.

Correctness: We consider time slot $|S| + 2D'_\omega - \tau[v]$. If u is a leaf of T_v, it sends $(v, 0, 1)$ to its parent in T_v in case $u \in S$, else it sends $(v, 0, 0)$, which is correct. In case u is not a leaf, each child u_i has sent $rc_S(T_v, u_i)$ (stored in r_i) as well as $Z^1_{(u,u_i)}(T_v)$ (stored in z_i) to u at an earlier point in time. This is true as time-stamp $\tau[v]$ stored in u_i is always larger than time-stamp $\tau[v]$ stored in u, as u_i is a child of u. Each time u received some of these values from its children in T_v, it updated its memory according to Lemma 5 (Lines $8-11$ of Algorithm 7.2), leading to sending the correct values $rc_S(T_v, u)$ and $Z^1_{(parent\_in\_T_v,u)}(T_v)$ to its parent in T_v at time $|S|+2D'_\omega-\tau[v]$. Thus in any case u sends the correct values.

We conclude that each node $v \in S$ has computed $rc_S(T_v, v) = RC_S(T_v)$ after Algorithm 7.2 has finished.

8 Proofs of Main Results

We put the tools of the previous sections together and prove the Theorems of Section 1.

Proof. (of Theorem 2). First, Algorithms 7.1 and 7.2 are used to compute $RC_S(v)$ for each $v \in S$. For each such node v, the value $RC_S(v)$ is stored in node v itself. A leader node (e.g. with lowest ID, which can be found in time $\mathcal{O}(D_\omega)$) computes $u := arg \min_{v \in V} RC_S(v)$ via aggregation using T_l, where l is the leader node. As stated in Theorem 5 the tree T_u is a $(2 - 2/|S|)$-approximation of a S-MRCT. The leader node informs the network that tree T_u is a $(2 - 2/|S|)$-approximation to an S-MRCT. The runtime follows from Lemma 2 and the fact, that to determine u by aggregating the corresponding minimum and to broadcast u can be done in time $\mathcal{O}(D_\omega)$.

Proof. (of Theorem 3). First we select a subset $S' \subseteq S$ of the size stated in Lemma 1. Each node joins a set S'' with probability $c \cdot s/n$, where s is the (desired) size of S' stated in Lemma 1 and c a constant depending on a Chernoff bound used now. Using such a Chernoff Bound, w.h.p. S'' is of size $c \cdot s$ or some constant $c \geq 1$. Now all IDs of nodes in S'' are sent to the leader who selects and broadcasts a subset S' of the desired size among the IDs of S''.

From now on the algorithm works exactly as in the proof of Theorem 2, except that the algorithm is run on S' instead of S (it computes and aggregates each $RC_S(v)$ for $v \in S'$ instead of S). As stated in Lemma 1, a tree T_u is found that is a $(2 - 2/|S| + \beta(n, D))$-approximation of an S-MRCT. The leader node informs the network that tree T_u is a $(2 - 2/|S| + \beta(n, D))$-approximation to an S-MRCT. Choosing $\beta(n, D) := \min\left\{\frac{\log n}{D}, \alpha(n, D)\right\}$ yields the desired approximation ratio of $2-2/|S|+\min\left\{\frac{\log n}{D}, \alpha(n, D)\right\}$, as stated in the Theorem.

Runtime analysis: As $s = \left(\left\lceil \frac{2-2/|S|}{\beta(n,D)} \right\rceil + 1\right) \cdot \ln n$, selecting a set S'' and deriving S' can be done w.h.p. in time

$$\mathcal{O}(D+s) = \mathcal{O}\left(D + \left(\left\lceil \frac{2-2/|S|}{\beta(n,D)} \right\rceil + 1\right) \cdot \ln n\right) = \mathcal{O}\left(D + \frac{\log n}{\beta(n,D)}\right),$$

which is $\mathcal{O}\left(D + \frac{\log n}{\alpha(n,D)}\right)$ due to the choice of β. The same runtime follows from Lemma 2 for computing the single source routing costs for all $v \in S'$. Combined with the fact that the aggregation and broadcast of u can be done in time $\mathcal{O}(D)$, the stated result is obtained.

Acknowledgment. We would like to thank Benjamin Dissler and Mohsen Ghaffari for helpful discussions and insights.

References

1. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, FOCS 1996, Burlington, Vermont, USA, October 14-16, pp. 184–193 (1996)
2. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: Vitter, J.S. (ed.) Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC 1998, Dallas, Texas, USA, May 23-26, pp. 161–168 (1998)
3. Campos, R., Ricardo, M.: A fast algorithm for computing minimum routing cost spanning trees. Computer Networks 52(17), 3229–3247 (2008)
4. Charikar, M., Chekuri, C., Goel, A., Guha, S.: Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In: Vitter, J.S. (ed.) Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC 1998, Dallas, Texas, USA, May 23-26, pp. 114–123 (1998)
5. Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed verification and hardness of distributed approximation. SIAM Journal on Computing 41(5), 1235–1265 (2012)
6. Dionne, R., Florian, M.: Exact and approximate algorithms for optimal network design. Networks 9(1), 37–59 (1979)
7. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Larmore, L.L., Goemans, M.X. (eds.) Proceedings of the 35th Annual ACM Symposium on Theory of Computing, STOC 2003, San Diego, California, USA, June 9-11, pp. 448–455 (2003)
8. Hochuli, A., Holzer, S., Wattenhofer, R.: Distributed approximation of minimum routing cost trees. Computing Research Repository CoRR, abs/1406.1244 (2014), `http://arxiv.org/abs/1406.1244`
9. Holzer, S., Peleg, D., Roditty, L., Tal, E., Wattenhofer, R.: Optimal distributed all pairs shortest paths and applications (2014), `http://www.dcg.ethz.ch/~stholzer/APSP-full.pdf`, preliminary full version of two merged papers to be submitted to a journal). New versions available on request

10. Holzer, S., Wattenhofer, R.: Optimal distributed all pairs shortest paths and applications. In: Kowalski, D., Panconesi, A. (eds.) Proceedings of the 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2012, Funchal, Madeira, Portugal, July 16-18, pp. 355–364 (2012)
11. Hu, T.C.: Optimum communication spanning trees. SIAM Journal on Computing 3(3), 188–195 (1974)
12. Johnson, D.S., Lenstra, J.K., Rinnooy Kan, A.H.G.: The complexity of the network design problem. Networks 8(4), 279–285 (1978)
13. Khan, M., Kuhn, F., Malkhi, D., Pandurangan, G., Talwar, K.: Efficient distributed approximation algorithms via probabilistic tree embeddings. In: Bazzi, R.A., Patt-Shamir, B. (eds.) Proceedings of the 27th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Ontario, August 18-21, pp. 263–272 (2008)
14. Nanongkai, D.: Distributed approximation algorithms for weighted shortest paths. To appear in: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC 2014, New York, USA, May 31-June 3 (2014)
15. Peleg, D.: Distributed computing: a locality-sensitive approach. Society for Industrial and Applied Mathematics, Philadelphia (2000)
16. Peleg, D.: Low stretch spanning trees. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 68–80. Springer, Heidelberg (2002)
17. Reshef, E.: Approximating minimum communication cost spanning trees and related problems. Master's thesis, Weizmann Institute of Science, Rehovot, Israel (1999)
18. Scott, A.J.: The optimal network problem: Some computational procedures. Transportation Research 3(2), 201–210 (1969)
19. Wikipedia. Bridging (networking) (April 28, 2014), `http://en.wikipedia.org/wiki/Bridging_(networking)`
20. Wong, R.T.: Worst-case analysis of network design problem heuristics. SIAM Journal of Algebraic Discrete Methods 1(1), 51–63 (1980)
21. Wu, B.Y., Chao, K.-M., Tang, C.Y.: Approximation algorithms for the shortest total path length spanning tree problem. Discrete Applied Mathematics 105(1), 273–289 (2000)
22. Wu, B.Y., Lancia, G., Bafna, V., Chao, K.-M., Ravi, R., Tang, C.Y.: A polynomial-time approximation scheme for minimum routing cost spanning trees. SIAM Journal on Computing 29(3), 761–778 (1999)
23. Wu, B.Y.: A polynomial time approximation scheme for the two-source minimum routing cost spanning trees. Journal of Algorithms 44(2), 359–378 (2002)