Theoretical Computer Science 412 (2011) 4482–4492

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Approximating some network design problems with node costs

Guy Kortsarz [a,*], Zeev Nutov [b]

[a] Rutgers University, Camden, United States
[b] The Open University of Israel, Israel

ARTICLE INFO

Article history:
Received 9 September 2009
Received in revised form 20 December 2010
Accepted 6 April 2011
Communicated by P. Spirakis

Keywords:
Network design
Node costs
Multicommodity buy at bulk
Approximation algorithm

ABSTRACT

We study several multi-criteria undirected network design problems with node costs and lengths. All these problems are related to the Multicommodity Buy at Bulk (MBB) problem in which we are given a graph $G = (V, E)$, demands $\{d_{st} : s, t \in V\}$, and a family $\{c_v : v \in V\}$ of subadditive cost functions. For every $s, t \in V$ we seek to send d_{st} flow units from s to t, so that $\sum_v c_v(f_v)$ is minimized, where f_v is the total amount of flow through v. It is shown in Andrews and Zhang (2002) [2] that with a loss of $2-\varepsilon$ in the ratio, we may assume that each st-flow is *unsplittable*, namely, uses only one path. In the Multicommodity Cost–Distance (MCD) problem we are also given lengths $\{\ell(v) : v \in V\}$, and seek a subgraph H of G that minimizes $c(H)+\sum_{s,t\in V} d_{st} \cdot \ell_H(s, t)$, where $\ell_H(s, t)$ is the minimum ℓ-length of an st-path in H. The approximability of these two problems is equivalent up to a factor $2 - \varepsilon$ [2]. We give an $O(\log^3 n)$-approximation algorithm for both problems for the case of the demands polynomial in n. The previously best known approximation ratio for these problems was $O(\log^4 n)$ (Chekuri et al., 2006, 2007) [5,6].

We also consider the Maximum Covering Tree (MaxCT) problem which is closely related to MBB: given a graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, profits $\{p(v) : v \in V\}$, and a bound C, find a subtree T of G with $c(T) \leq C$ and $p(T)$ maximum. The best known approximation algorithm for MaxCT (Moss and Rabani, 2001) [18] computes a tree T with $c(T) \leq 2C$ and $p(T) = \Omega(\text{opt}/\log n)$. We provide the first nontrivial lower bound on approximation by proving that the problem admits no better than $\Omega(1/(\log\log n))$ approximation assuming NP $\not\subseteq$ Quasi(P). This holds true even if the solution is allowed to violate the budget by a constant ρ, as was done in [18] with $\rho = 2$. Our result disproves a conjecture of [18].

Another problem related to MBB is the Shallow Light Steiner Tree (SLST) problem, in which we are given a graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, lengths $\{\ell(v) : v \in V\}$, a set $U \subseteq V$ of terminals, and a bound L. The goal is to find a subtree T of G containing U with $\text{diam}_\ell(T) \leq L$ and $c(T)$ minimum. We give an algorithm that computes a tree T with $c(T) = O(\log^2 n) \cdot \text{opt}$ and $\text{diam}_\ell(T) = O(\log n) \cdot L$. Previously, a polylogarithmic bicriteria approximation was known only for the case of edge costs and edge lengths.

1. Introduction

Network design problems require finding a minimum cost (sub-)network that satisfies prescribed properties, often connectivity requirements. The most fundamental problems are the ones with 0, 1 connectivity requirements. Classic examples are: Shortest Path, Min-Cost Spanning Tree, Min-Cost Steiner Tree/Forest, Traveling Salesperson, and others. Examples of problems with high connectivity requirements are: Min-Cost k-Flow, Min-Cost k-Edge/Node-Connected Spanning Subgraph, Steiner Network, and others. All these problems also have practical importance in applications.

* Corresponding author.
E-mail addresses: guyk@camden.rutgers.edu, guyk@crab.rutgers.edu (G. Kortsarz), nutov@openu.ac.il (Z. Nutov).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.04.013

Two main types of costs are considered in the literature: the edge costs and the node costs. We consider the latter, which is usually more general than the edge cost variants; indeed, for most undirected network design problems there is a very simple reduction that transforms edge costs to node costs, but the inverse is, in general, not true. The study of network design problems with node costs is already well motivated and established from both theoretical as well as practical considerations [15,12,18,5,6,19]. For example, in telecommunication networks, expensive equipment such as routers and switches are located at the nodes of the underlying network, and thus it is natural to model some of these problems by assigning costs on the nodes rather than to the edges.

For some previous work on undirected network-design problems with node costs see the work of Klein and Ravi [15], Guha et al. [12], Moss and Rabani [18], and Chekuri et al. [5,6]. We mostly focus on resolving some open problems posed in these papers.

1.1. Problems considered

Given a *length function* ℓ on edges/nodes of a graph H, let $\ell_H(s,t)$ denote the ℓ-distance between s, t in H, that is, the minimum ℓ-length of an st-path in H (including the lengths of the endpoints). Let $\mathrm{diam}_\ell(H) = \max_{s,t\in V(H)} \ell_H(s,t)$ be the *ℓ-diameter of H*, that is the maximum ℓ-distance between two nodes in H. We consider the following two related problems on undirected graphs.

Multicommodity Buy at Bulk (MBB)
Instance: A graph $G = (V, E)$, a family $\{c_v : v \in V\}$ of sub-additive monotone non-decreasing cost functions, a set D of pairs from V, and positive demands $\{d_{st} : \{s,t\} \in D\}$.
Objective: Find a set $\{P_{st} : \{s,t\} \in D\}$ of st-paths so that $\sum_{v\in V} c_v(f_v)$ is minimized, where $f_v = \sum\{d_{st} : \{s,t\} \in D, v \in P_{st}\}$.

Multicommodity Cost--Distance (MCD)
Instance: A graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, lengths $\{\ell(v) : v \in V\}$, a set D of pairs from V, and positive integral demands $\{d_{st} : \{s,t\} \in D\}$.
Objective: Find a subgraph H of G that minimizes

$$w(H, D) = c(H) + \sum_{\{s,t\}\in D} d_{st} \cdot \ell_H(s,t). \tag{1}$$

As linear functions are subadditive, MCD is a special case of MBB. The following statement shows that up to a factor arbitrarily close to 2, MCD and MBB are equivalent w.r.t. approximation.

Proposition 1.1 (*[2]*). *If there exists a ρ-approximation algorithm for* MCD *then there exists a $(2\rho+\varepsilon)$-approximation algorithm for* MBB *for any $\varepsilon > 0$.*

We consider two other fundamental problems closely related to MBB (see an explanation below):

Maximum Covering Tree (MaxCT)
Instance: A graph $G = (V, E)$, costs $\{c(v) : v \in E\}$, profits $\{p(v) : v \in V\}$, and a bound C.
Objective: Find a subtree T of G with $c(T) \le C$ and $p(T)$ maximum.

Shallow--Light Steiner Tree (SLST)
Instance: A graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, lengths $\{\ell(v) : v \in V\}$, a set $U \subseteq V$ of terminals, and a bound L.
Objective: Find a subtree T of G containing U with $\mathrm{diam}_\ell(T) \le L$ and $c(T)$ minimum.

Each one of the above problems has an "edge version", where the costs/lengths are given on the edges. As was mentioned, the edge version admits an easy approximation ratio preserving reduction to the node version.

1.2. Relations between the problems

The following problem was defined in [13]:
k-Buy at Bulk Steiner Tree (k-BBST)
Instance: A graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, length $\{\ell(v) : v \in V\}$, a set $U \subseteq V$ of terminals, a root $r \in U$, a diameter bound L, a cost bound C, and an integer k.
Objective: Find a subtree H of G containing r and at least $k-1$ additional terminals, of diameter L and cost at most C.

This problem is related to MBB as follows (this holds for both edge and node costs). Let T be a feasible k-BBST solution for an instance at hand. Thus $c(T) \le C$ and T has diameter L.

Theorem 1.2 (*[13]*). *There exist two universal constants c_1, c_2 and a polynomial time algorithm that given an instance of k-*BBST *finds a a tree H containing at least $k/8$ terminals with cost at most $c_1 \log^3 n \cdot C$ and diameter at most $c_2 \cdot \log n \cdot L$.*

For completeness we describe some of the techniques used in the algorithm of [13].

The solution uses iterative merging. It maintains a collection of clusters $\mathcal{C}_i$. Cluster $\mathcal{C}_i$ contains rooted trees with number of terminals between 2^i and 2^{i+1} (at the beginning $\mathcal{C}_0$ contains all terminals). Every root of a tree of every $\mathcal{C}_i$ is a terminal. Trees are merged, but every merger is of two trees of the same cluster. This is used in the analysis as every vertex v goes through at most $O(\log n)$ merges.

As we shall see later, the algorithm deletes terminals (it does not delete non-terminals). Let W be the remaining terminals. We define *large* clusters as clusters than contain at least $|W|/(2\log n)$ terminals. Obviously at least one large cluster exists.

Call a terminal *a true terminal* if it belongs to the optimum tree solution. Let $\mathcal{C}_i$ be an arbitrary large cluster. The pairing lemma [20] states that if the number of terminals in the optimum tree is even, there is a way to pair the terminals to pairs $\{u, v\}$, so that in the optimum tree, the unique paths between the $\{u, v\}$ pairs are edge disjoint.

Clearly, in any pairing of the roots, every pair will be have pairwise length at most L. Because the paths are edge disjoint the total *sum of costs* of pairing paths is at most C. If there are "many" true terminals among the roots, by averaging, there exists a low length and cost path. Let such a path be called an *appropriate path*.

The algorithm operates as follows: if there is an appropriate path, add this path to the solution, recompute the clusters, and iterate. If there is no appropriate path, the algorithm deletes all roots of trees in $\mathcal{C}_i$.

It is shown in [13] that if no appropriate path exists, there are very "few" true terminals among the removed roots. To show this we use the property that the cluster is large.

In fact, using a (somewhat complex) calculation it is shown that terminal deletion never causes the number of terminals to drop below $k/8$. In [13], mergers continue until a tree with at least $k/8$ terminals is found. As the number of terminals never drops below $k/8$, it is clear that at the end, a tree containing at least $k/8$ terminals *must* be found. By the above considerations, it will have low diameter and low cost.

In [6] the algorithm of Theorem 1.2 was used to design a *combinatorial* $O(\log^4 n)$-approximation algorithm for MBB. An LP-based algorithm with the same ratio, is given in the same paper, which is however much slower since it repeatedly solves large linear programs. It thus seems important to try and get a better bicriteria approximation for k-BBST than the one in Theorem 1.2. An improved bicriteria algorithm for k-BBST would imply a better combinatorial algorithm for MBB. Specifically, if for k-BBST the cost returned is $O(\rho_1)\cdot C$ and the diameter is $O(\rho_2 \cdot L)$ then the approximation ratio derived for MBB is $O(\log n)\cdot\rho_1 + O(\log^3 n)\cdot\rho_2$.

Since we are facing difficulties in improving the results for k-BBST we study MaxCT and SLST which are relaxed variants of k-BBST. The hope is that it may shed light on k-BBST, and thus also on MBB. Also, MaxCT and SLST are interesting in their own right.

Comparing MaxCT **and** k-BBST**:** MaxCT with unit costs and cost bound k is seeking a tree with k terminals and maximizing the profit. However, the big difference between k-BBST and MaxCT is that there are no length constraints in MaxCT and that the primal–dual techniques of [18] do not seem suitable to handle length constraints.

Comparing SLST **and** k-BBST**:** SLST is a particular case of k-BBST where $k = n$. In general, problems that seek a tree containing all terminals are usually easier than the corresponding problems that seek a tree with at least k terminals.

In summary, one may hope that techniques for MaxCT that (in the case of uniform costs) are able to find a tree with k terminals and low cost, but may not be suited to handle distances constraints, could somehow be combined with techniques that do produce a tree that is both shallow and light, but work only for the case $k = |U|$.

1.3. Related work

We survey some results on relevant network design problems with node costs. Klein and Ravi [16] showed that the Node-Weighted Steiner Tree problem is Set-Cover hard, thus it admits no $o(\log n)$ approximation unless $P = NP$ [21]. They also obtained a matching approximation ratio using a greedy merging algorithm. Their method was generalized in [19] to handle problems with connectivity requirements higher than 1. Guha et al. [12] showed an $O(\log n)$ integrality gap of a natural LP-relaxation for the problem. The MBB problem is motivated by economies of scale that arise in a number of applications, especially in telecommunication. The problem is studied as the fixed charge network flow problem in operations research; see [3,10,11,23], and a survey in [17]. The first approximation algorithm for the problem is by Salman et al. [22]. For the multi-commodity version MBB the first non-trivial result is due to Charikar and Karagiazova [4] who obtained an $O(\log |D| \exp(O(\sqrt{\log n \log\log n})))$-approximation, where $|D|$ is the sum of the demands. In [5] an $O(\log^4 n)$-approximation algorithm is given for the edge costs case, and further generalized to the node costs case in [6]. See [1] for an $\Omega(\log^{1/2-\varepsilon} n)$-hardness result.

The MaxCT problem was introduced in [12] motivated by efficient recovery from power outage. In [12] a pseudo approximation algorithm is presented that returns a subtree T with $c(T) \le 2C$ and $p(T) = \Omega(P/\log^2 n)$, where P is the maximum profit under budget cost C. This was improved in [18] to produce a tree T with $c(T) \le 2C$ and $p(T) = \Omega(P/\log n)$. For a related minimization problem when one seeks to find a minimum cost tree T with $p(T) \ge P$ [18] gives an $O(\ln n)$-approximation algorithm.

1.4. Our results

The previously best known ratio for MCD/MBB was $O(\log^4 n)$ both for edge costs [5] and node costs [6], and this holds also for polynomial demands. We improve this result by using, among other things, a better LP-relaxation for the problem.

Theorem 1.3. MCD/MBB *with polynomial demands admits an* $O(\log^3 n)$*-approximation algorithm.*

Our next result is for the MaxCT problem. In [18] it is conjectured that MaxCT admits an $O(1)$ approximation algorithm (which would have been quite helpful for dealing with k-BBST). We disprove this conjecture.

Theorem 1.4. *Unless* NP $\subseteq$ Quasi-P, MaxCT *admits no* $o(\log\log n)$*-approximation algorithm. This is so even if one is allowed to use a budget of* $\rho \cdot B$ *for a universal constant* $\rho \geq 1$ *(as was done in* [18] *with* $\rho = 2$*).*

Our last result is for the SLST problem. For SLST with *edge costs* and *edge lengths*, the algorithm of [16] computes a tree T with $c(T) = O(\log n) \cdot \mathsf{opt}$ and $\mathsf{diam}_\ell(T) = O(\log n) \cdot L$. We consider the more general case of node costs and node lengths.

Theorem 1.5. SLST *with node costs and lengths admits an approximation algorithm that computes a tree* T *with* $c(T) = O(\log^2 n) \cdot \mathsf{opt}$ *and* $\mathsf{diam}_\ell(T) = O(\log n) \cdot L$.

Theorems 1.3–1.5 are proved in Sections 2–4, respectively.

2. Improved algorithm for MBB

In this section we prove Theorem 1.3. We give an $O(\log^2 n \cdot \log N)$-approximation algorithm for MCD with running time polynomial in M, where M is the sum of the demands plus n. If M is polynomial in n, the running time is polynomial in n, and the approximation ratio is $O(\log^3 n)$. We may assume (by duplicating nodes) that all demands are 1. Let M be the new number of terminals and observe that M is polynomial in n. Then our problem is:
Instance: A graph $G = (V, E)$, costs $\{c(v) : v \in V\}$, lengths $\{\ell(v) : v \in V\}$, and a set D of node pairs.
Objective: Find a subgraph H of G minimizing $w(H, D) = c(H) + \sum_{\{s,t\} \in D} \ell_H(s, t)$.

For the latter problem, we give an $O(\log^2 n \cdot \log |D|)$-approximation algorithm.

2.1. Greedy algorithms and junction trees

We use a result about the performance of a *Greedy Algorithm* for the following type of problems:

Covering Problem
Instance: A groundset Π and functions ν, w on 2^Π with $\nu(\Pi) = 0$.
Objective: Find $\mathcal{P} \subseteq \Pi$ with $\nu(\mathcal{P}) = \nu(\Pi)$ and with $w(\mathcal{P})$ minimized.

Let $\rho > 1$ and let opt be the optimal solution cost for the Covering Problem. The ρ*-Greedy Algorithm* starts with $\mathcal{P} = \emptyset$ and iteratively adds subsets of $\Pi - \mathcal{P}$ to $\mathcal{P}$ one after the other using the following rule. As long as $\nu(\mathcal{P}) > \nu(\Pi)$ it adds to $\mathcal{P}$ a set $\mathcal{R} \subseteq \Pi - \mathcal{P}$ so that

$$\sigma_{\mathcal{P}}(\mathcal{R}) = \frac{w(\mathcal{R})}{\nu(\mathcal{P}) - \nu(\mathcal{P} + \mathcal{R})} \leq \frac{\rho \cdot \mathsf{opt}}{\nu(\mathcal{P}) - \nu(\Pi)}. \tag{2}$$

The following known statement follows by a standard set-cover analysis, c.f., [15].

Theorem 2.1. *If* ν *is decreasing and* w *is increasing and subadditive, then the* ρ*-Greedy Algorithm computes a solution* $\mathcal{P}$ *with* $w(\mathcal{P}) \leq \rho \cdot [\ln(\nu(\emptyset) - \nu(\Pi)) + 1] \cdot \mathsf{opt}$.

In our setting, Π is the family of all st-paths, $\{s, t\} \in D$. For a set $\mathcal{R} \subseteq \Pi$ of paths connecting a set R of pairs in D, let $\nu(\mathcal{R}) = |D| - |R|$ be the number of pairs in D not connected by paths in $\mathcal{R}$, and let $w(\mathcal{R}) = c(\mathcal{R}) + \sum_{\{s,t\} \in R} \ell(P_{st})$, where $c(\mathcal{R})$ denotes the cost of the union of the paths in $\mathcal{R}$, and P_{st} is the shortest st-path in $\mathcal{R}$. Note that $\nu(\Pi) = 0$ and $\nu(\emptyset) = |D|$. We will show how to find such $\mathcal{R}$ satisfying (2) with $\rho = O(\log^2 n)$. W.l.o.g., we may consider the case $\mathcal{P} = \emptyset$. Otherwise, we consider the residual instance obtained by excluding from D all pairs connected by $\mathcal{P}$ and setting $\mathcal{P} = \emptyset$; it is easy to see that if $\mathcal{R}$ satisfies (2) for the residual instance, then this is also so for the original instance. Assuming $\mathcal{P} = \emptyset$, (2) can be rewritten as:

$$\sigma(\mathcal{R}) = \frac{c(\mathcal{R})}{|R|} + \frac{\sum_{\{s,t\} \in R} \ell(P_{st})}{|R|} \leq \frac{\rho \cdot \mathsf{opt}}{|D|}. \tag{3}$$

The quantity $\sigma(\mathcal{R})$ in (3) is the *density* of $\mathcal{R}$; it is a sum of "cost-part" $c(\mathcal{R})/|R|$ and the remaining "length-part". The following key statement from [5] shows that with $O(\log n)$ loss in the length part of the density, we may restrict ourselves to very specific $\mathcal{R}$, as given in the following definition; in [5] it is stated for edge-costs, but the generalization to node-costs is immediate.

Definition 2.1. A tree T with a designated node r is a *junction tree* for a subset $R \subseteq D$ of node pairs in T if the unique paths in T between the pairs in R all contain r.

Lemma 2.2 (*[5], The junction tree lemma*). *Let H^* be an optimal solution to an* MCD *instance with* $\{0,1\}$ *demands. Let* $C = c(H^*)$ *and let* $L = \sum_{\{s,t\}\in D} \ell_{H^*}(s,t)$. *Then there exists a junction tree T for a subset $R \subseteq D$ of pairs, so that* $\mathrm{diam}_\ell(T) = O(\log n)\cdot L/|D|$ *and* $c(T)/|R| = O(C/|D|)$.

If we could find a pair T, R as in Lemma 2.2 in polynomial time, then we would obtain an $O(\log|D|\cdot\log n)$-approximation algorithm, by Theorem 2.1. In [5] it is shown how to find such a pair that satisfies (3) with $\rho = O(\log^3 n)$. We will show how to find such a pair with $\rho = O(\log^2 n)$.

Theorem 2.3. *There exists a polynomial time algorithm that given an instance of* MCD *with* $\{0,1\}$ *demands computes a set $\mathcal{R}$ of paths connecting a subset $R \subseteq D$ of pairs satisfying* (3) *with* $\rho = O(\log^2 n)$.

Let U' be all nodes that belong to at least one pair.

Motivated by Lemma 2.2, the following LP was used in [5,6]. Guess the common node r of the paths in $\mathcal{R}$ of the junction tree T. Relax the integrality constraints by allowing "fractional" nodes and paths. For $v \in V$, x_v is the "fraction of v" taken into the solution. For $s, t \in D$ it is clear that connecting s to t by flow 1 is equivalent to sending a unit of flow from s to r and then sending a unit of flow from r to t (this follows as all flow paths go via r). Reverse the flow direction for t, and assume that a unit of flow is sent from t to r. Since the cost of sending a unit of flow from t to r equals the cost of sending a unit of flow from r to t, we can define an equivalent instance with only one sink. Each vertex s that belongs to U' has to send some y_s flow units to r. Also, its mate t has to send an equal flow of $y_t = y_s$ from t to r. This gives a flow instance, with all sinks equal to r. We add the restriction $y_s = y_t$ for every $\{s,t\} \in D$ into the LP. For $u \in U'$ let $\Pi(u)$ be the set of all ur-paths in Π, and thus $\Pi = \cup_{u\in U'}\Pi(u)$. For $P \in \Pi$, f_P is the amount of flow through P. Dividing all variables by $|R|$ (note that this does not affect the objective cost), gives the following LP:

$$
\begin{aligned}
\text{(LP1)}\quad & \min \sum_{v\in V} c(v)\cdot x_v + \sum_{P\in\Pi} \ell(P)\cdot f_P \\
\text{s.t.}\quad & \sum_{u\in U'} y_u = 2 \\
& y_s - y_t = 0 \quad \{s,t\}\in D \\
& \sum_{v\in P\in\Pi(u)} f_P \le x_v \quad v\in V, u\in U' \\
& \sum_{P\in\Pi(u)} f_P \ge y_u \quad u\in U' \\
& x_v, f_P, y_u \ge 0 \quad v\in V, P\in\Pi, u\in U'.
\end{aligned}
$$

For the first inequality, note that for a pair s, t $y_s = y_t$ and this value is counted twice in the sum. For simplicity, for the rest of the paper we replace the 2 above by 1 as this only changes the approximation ratio by a small constant.

2.2. The LP used

Let $A\cdot\log n\cdot L/|D|$ be the bound on the lengths of the paths in $\mathcal{R}$ guaranteed by Lemma 2.2, where A is some universal constant. We use almost the same LP as (LP1), except that we seek to minimize the cost only, and restrict ourselves to paths of length at most $A\cdot\log n\cdot L/|D|$, which better reflects the statement in Lemma 2.2. Let $\tilde{\Pi} = \{P\in\Pi : \ell(P)\le A\cdot\log n\cdot L/|D|\}$ and let $\tilde{\Pi}(u)$ be the paths in $\tilde{\Pi}$ that start at u, for $u\in U'$. Again recall that y_u is the flow delivered from u to r. The LP we use is:

$$
\begin{aligned}
\text{(LP2)}\quad & \min \sum_{v\in V} c(v)\cdot x_v \\
\text{s.t.}\quad & \sum_{u\in U'} y_u = 1 \\
& y_s - y_t = 0 \quad \{s,t\}\in D \\
& \sum_{\{P\in\tilde{\Pi}(u)\mid v\in P\}} f_P \le x_v \quad v\in V, u\in U' \\
& \sum_{P\in\tilde{\Pi}(u)} f_P \ge y_u \quad u\in U' \\
& x_v, f_P, y_u \ge 0 \quad v\in V, P\in\tilde{\Pi}, u\in D.
\end{aligned}
$$

Lemma 2.4. *For any* $\varepsilon > 0$, *a solution for* (LP2) *of value* $\le (1+\varepsilon)\mathrm{opt}$ *can be found in time polynomial in n.*

Proof. The proof is similar to the proof of [8, Lemma 3.5]. Although the number of variables in (LP2) might be exponential, any basic feasible solution to (LP2) has $O(N^2)$ non-zero variables; recall that N is the new number of nodes after the reduction to demands 1 and that N is polynomial in n. An approximate solution for (LP2) is derived from an approximate solution to its dual LP, see the proof of [8, Lemma 3.3]; if we had a polynomial time separation oracle for the dual LP, we could compute an optimal solution to (LP2) (the non-zero entries) in polynomial time. In the dual LP, the only problematic constraints are the ones that correspond to the variables f_P, which appear in the last two constraint types in (LP2), that can be rewritten as follows:

$$\sum_{P\in\tilde{\Pi}(u)} f_P - y_u \geq 0 \quad u \in U'$$

$$x_v - \sum_{\{P\in\tilde{\Pi}(u)|v\in P\}} f_P \geq 0 \quad v \in V, u \in U'.$$

Let w_u and z_{uv} be the corresponding dual variables. The corresponding dual constraints are:

$$w_u \leq \sum_{v\in P} z_{vu} \quad P \in \tilde{\Pi}(u).$$

To separate these constraints for a specific node $u \in U$, it would be sufficient to find the lightest path with respect to the weights z_{uv} among the paths in $\tilde{\Pi}(u)$, namely, among the ur-paths whose length with respect to ℓ is at most $A \cdot \log n \cdot L/|D|$. This is called the Restricted Shortest Path problem (with node costs), which is NP-hard but admits an FPTAS [14]. Therefore we get an approximate separation oracle for these constraints, which for any $\varepsilon > 0$ checks whether there exists a path $P \in \tilde{\Pi}(u)$ so that $w_u \leq \sum_{v\in P} z_{vu}/(1+\varepsilon)$, in time polynomial in n and $1/\varepsilon$. The statement now follows by an argument identical to the one in the proof of [8, Lemma 3.5]. □

By Lemma 2.2 there exists a solution to (LP2) of cost $O(C/|D|)$. Indeed, let $T, R, \mathcal{R}$ be as in Lemma 2.2; in particular, $c(T)/|R| = O(C/|D|)$. For $u \in T$ let P_u be the unique ur-path in T. Define a feasible solution for (LP2) as follows: $x_v = 1/|R|$ for every $v \in T$, $y_u = f_{P_u} = 1/|R|$ for every u that belongs to some pair in R, and x_u, y_u, f_P are zero otherwise. It easy to see that this solution is feasible for (LP2), and its cost is $c(T)/|R| = O(C/|D|)$.

2.3. Proof of Theorem 2.3

We now proceed similarly to [5,6]. We may assume that $\max\{1/y_u : u \in U'\}$ is polynomial in n, see [6]. Partition U' into $O(\log n)$ sets $U_j = \{u \in U : 1/2^{j+1} \leq y_u \leq 1/2^j\}$. There is some U_j that delivers $\Omega(1/\ln n)$ flow units to r. Focus on that U_j. Clearly, $|U_j| = \Theta(2^j)/\log n$. Setting $x'_v = \min\{\Theta(2^j)\cdot x_v, 1\}$ for all $v \in V$ and $f'_P = \min\{\Theta(2^j)\cdot f_P, 1\}$ for all $P \in \Pi$, gives a feasible solution for the following LP that requires from every node in U_j to deliver a flow unit to r.

$$\text{(LP3)} \quad \min \sum_{v\in V} c(v)\cdot x'_v + \sum_{P\in\Pi} \ell(P)\cdot f'_P$$

$$\text{s.t.} \quad \sum_{\{P\in\Pi(u)|v\in P\}} f'_P \leq x'_v \quad v \in V, u \in U_j$$

$$\sum_{P\in\Pi(u)} f'_P \geq 1 \quad u \in U_j$$

$$x'_v, f'_P \geq 0 \quad v \in V, P \in \Pi.$$

We bound the cost of the above solution x', f' for (LP3). Since $\sum_{v\in V} c(v)x_v = O(C/|D|)$, we have

$$\sum_{v\in V} c(v)x'_v = O(2^j)\cdot C/|D|.$$

We later see that, since $|U_j| = \Theta(2^j/\log n)$, an extra $\log n$ factor is invoked in the cost-density part of our solution; if, e.g., $|U_j| = 2^j$ would hold, this $\log n$ factor would have been saved. Our main point is that the length-part of the density does not depend on the size of U_j. We show this as follows. All paths used in (LP2) are of length $O(\log n \cdot L/|D|)$. First, assure that $\sum_{P\in\tilde{\Pi}_u} f'_P$ is not too large. For any $u \in U_j$ the fractional values of $\{f'_P : P \in \Pi_u\}$ only affect u, namely, if $u \neq u'$ then $\tilde{\Pi}_u \cap \tilde{\Pi}_{u'} = \emptyset$. Therefore, if $\sum_{P\in\tilde{\Pi}_u} f_P >> 1$, we may show that the sum is at most 3/2 as follows. If a single path carries at least 1/2 unit of flow then (scaling values by only 2) this path can be used as the solution for u. A collection of paths delivering a flow of value at least one from u to r is *minimal* if deleting any path from the collection brings the flow from u to r to be less than 1. Clearly, $\tilde{\Pi}_u$ can be made a minimal collection $\tilde{\Pi}'_u$, and we claim that $\tilde{\Pi}'_u$ delivers at most 3/2 units of flow to r: removing any path P from $\tilde{\Pi}'_u$ results in a flow of value less than one. Since P carries at most 1/2 a unit of flow, $\tilde{\Pi}'_u$ carries at most 3/2 flow units. Hence the contribution of a single node u to the fractional length-part is

$$O(\log n \cdot L/|D|) \sum_{P\in\tilde{\Pi}_u} f'_P = O(\log n \cdot L/|D|).$$

Over all terminals, the contribution is $O(|U_j| \cdot \log n \cdot L/|D|)$. Now, use the main theorem of [6]:

Theorem 2.5 ([6]). *There exists a polynomial time algorithm that finds an integral solution to* (LP3) *of cost* $O(\log n)$ *times the optimal fractional cost of* (LP3).

Hence we can find in polynomial time a tree T containing r and U_j with $c(T) = O(\log n \cdot 2^j \cdot C/|D|)$ and $\sum_{u \in U_j} \ell_T(u, r) = O(|U_j| \cdot \log^2 n \cdot L/|D|)$.

Note that if the tree contains i terminals then it contains $i/2$ pairs. This is due to the constraint $y_s = y_t$. Since the tree spans $\Theta(2^j/\log n)$ pairs, its cost-part density is $O(\log^2 n) \cdot C/|D|$. Clearly, the length-part density is $O(\log^2 n) \cdot L/|D|$. This finishes the proof of Theorem 2.3, and thus also the proof of Theorem 1.3 is complete.

Remark. Here we used the following trick. In the junction tree lemma, the length density is $O(\log n)$ times larger than the cost density. But by the above, we were able to add $O(\log^2 n)$ to the cost density, but only $O(\log n)$ to the length density, making $\rho = O(\log^2 n)$ in both quantities.

3. A lower bound for MaxCT

Here we prove the following statement that implies Theorem 1.4.

Theorem 3.1. *Unless* NP $\subseteq$ DTIME$(n^{O(\log\log n \cdot \ln 2c \cdot \exp(8c))})$, MaxCT *admits no better than a* c*-approximation algorithm, even if one is allowed to use budget* $\rho \cdot B$ *for a universal constant* $\rho \geq 1$.

Remark. The size of the instance produced is $s = n^{O(\log\log n \ln 2c \cdot \exp(8c))}$ and thus $c = \Theta(\log\log s)$. Therefore, it is not possible to get a stronger hardness than $\log\log n$ unless we get a better gap in terms of c.

We first show the result assuming the solution does not violate the budget. The extra details for the case the solution may violate the budget by a factor of ρ are given in Section 3.4.

3.1. The Max-Coverage problem

For a graph $H = (V, E)$ and $X \subseteq V$ let $\Gamma_H(X)$ denote the set of *neighbors* of X in H. The following is a description of the Max-Coverage problem in terms of bipartite graphs.

Max-Coverage
Instance: A bipartite graph $H = (A + B, E)$ and an integer k.
Objective: Find $A' \subseteq A$ with $|A'| = k$ so that $\Gamma_H(A')$ is maximum.

Here A is the collection of *sets* and B is the set of *elements*, and we say that A' covers B' if $B' \subseteq \Gamma_H(A')$. The following statement is very similar to [7, Proposition 5.2].

Claim 3.2. *There exists an* $n^{O(\log\log n)}$ *time reduction from any instance* I *of size* n *of an arbitrary NP-complete language to a* Max-Coverage *instance* H, k *that contains* $M = O(n^{\log\log n})$ *sets and the same number of elements, such that the following holds.*

- *If* I *is a YES-instance then there exists* $A' \subseteq A$ *with* $|A'| = k$ *so that* $\Gamma_H(A') = B$.
- *If* I *is a NO-instance then every* $A' \subseteq A$ *of size* $|A'| \leq 4c \cdot k$ *covers at most* $1 - e^{-8c}$ *fraction of the elements, for any* $1 < c < \ln |B|$.

Proof. In the Set-Cover problem we are given a bipartite graph H as in Max-Coverage, and the goal is to find a min-size $A' \subseteq A$ that covers all B. Feige [7] proved that there is a reduction from any NP-complete language I of size n to a Set-Cover instance of size $n^{O(\log logn)}$ with a parameter k so that:

- If I is a YES-instance, then there exists $A' \subseteq A$ of size $|A'| = k$ that covers all B.
- If I is a NO-instance, then for every universal constant ε, any subset $A' \subseteq A$ of size at most $(1 - \varepsilon) \ln k$ does not cover all B.

Let M be the size of both A and B in the above construction (it can be assumed without loss of generality that $|A| = |B|$). Assume for the sake of contradiction that the part concerning the NO-instance in Claim 3.2 does not hold. We then show that there exists a set set $A' \subset A$ of size at most $\ln k/2$ that covers all B contradicting [7]. Apply the assumed algorithm (that follows from the negation of the NO-instance in the claim; the part about a yes instance always holds in the reduction of Feige [7]) in iterations. Each iteration adds $4c \cdot k$ to the solution size and covers a fraction of least $\left(1 - e^{-8c}\right)$ of the uncovered elements. Thus, the number of covered elements is now $M - M\left(1 - (e)^{-8c}\right) = M \cdot e^{-8c}$. Thus the decline of the uncovered elements is by a fraction of e^{-8c} in every iteration. Thus, ℓ that satisfies $\left(e^{-8c}\right)^{\ell} = 1/M$ is an upper bound on the number of iterations needed to reduce the number of uncovered elements to 0. Set $\ell = \ln M/8 \cdot c$. Raising $e^{-8 \cdot c}$ to power ℓ gives

$$\left(\frac{1}{e}\right)^{lnM} = \frac{1}{M}.$$

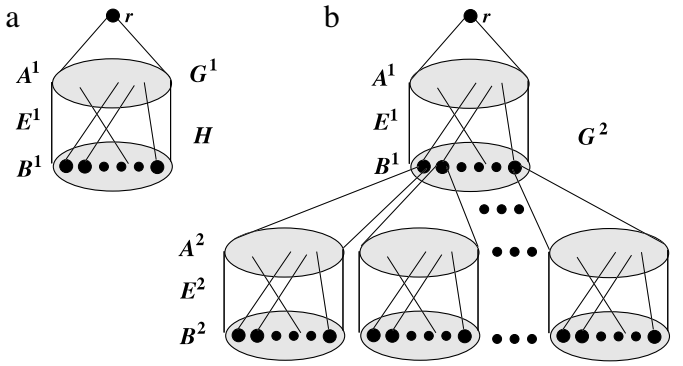

Fig. 1. (a) The graph G^1. (b) The graph G^2; if instead of copies of G^1 we "attach" to nodes in B_1 roots of the copies of G^{i-1}, then we obtain G^i.

As at every iteration $4c \cdot k$ sets are added to the solution, the total number of sets in the solution is at most

$$\ell \cdot 4 \cdot c \cdot k = \frac{\ln M}{2} k.$$

This contradicts the result of Feige [7], for $\epsilon = 1/2$. □

3.2. The reduction

Given a bipartite graph $H = (A + B, E)$ that obeys Claim 3.2 (hence has size $|I|^{O(\log\log|I|)}$) with I the NPC instance) define a sequence of graphs $G^1, G^2, \ldots$ by induction (see Fig. 1). For simplicity, we denote the number of sets and elements in a set-cover instance by n. To obtain G^1, take H, add a root r, and connect r to every node in A. Let $A_1 = A$ and $B_1 = B$. To obtain G^i from G^{i-1}, $i \geq 2$, take G^1 and $|B|$ copies of G^{i-1}, each corresponding to a node in B_1, and for every copy identify its root with the node corresponding to it in B_1. As the construction resembles a tree, we borrow some terms from the terminology of trees. A copy of H has *level* i if its A sets have distance $2i-1$ to the root r. The copies of H at level i are ordered arbitrarily. A typical copy of H at level i is denoted by $H_{ip} = (A_{ip}, B_{ip}, E_{ip})$ with i the level of the copy and p the *index* of the copy. This means that the A_{ip} sets are at distance $2i-1$ from the root and that p is the index of the copy inside level i. Let $A^i = \bigcup_p A_{ip}$ and $B^i = \bigcup_p B_{ip}$.

Definition 3.1 (*The Height of the Reduction*)**.** For any constant c, let $h = h(c) = 4e^{8c} \cdot \ln(2c)$.

Definition 3.2. We say that a node $v \in B_{ij}$ is an ancestor of H_{qp}, $q > i$, if any path between r and an H_{qp} node must go via v. In addition, such H_{qp} are called descendants of v and are also called descendants of H_{ij}.

The *terminals* of G^h are $\bigcup_j B_{hj}$, namely, the elements of the last level, and each of them has profit 1. Other nodes have profit 0. The cost of every node in A_{ij} is $1/|B|^{i-1}$ (so the nodes in $A_1 = A_{11}$ have cost 1), and the cost of any other node is 0. The cost bound is $C = h \cdot k$.

Fact 3.3. *If we start with an NPC instance I, the size (and the construction time) of the obtained instance of* MaxCT *is* $n^{O(h)} = |I|^{h \cdot O(\log\log|I|)}$.

3.3. Analysis

While increasing the level by 1, the number of Max-Coverage instances grows up by $|B|$ but the node costs go down by $|B|$. Hence the total cost of every level i is $|A|$, and the total cost of G is $h \cdot |A|$. We may assume that any solution T to the obtained MaxCT instance contains r. Otherwise, we may add the shortest path from r to T; the cost added is negligible in our context.

Lemma 3.4 (*The YES-instance*)**.** *If I is a YES-instance then the obtained* MaxCT *instance admits a feasible subtree T of cost $h \cdot k$.*

Proof. Consider the graph T induced in G by r and all the copies of $A' \cup B$ so that $|A'| = \mathsf{opt}$ and A' covers B. This graph is clearly connected and contains all terminals as each A' copy covers its copy of B. The cost of all copies of A' at any level i is k. Summing over all levels gives total cost $c(T) = h \cdot k = C$, as claimed. □

Lemma 3.5 (*The NO-instance*)**.** *If I is a No-instance then in the obtained* MaxCT *instance any feasible subtree T of cost $h \cdot k$ contains at most $1/c$ fraction of the terminals.*

In the rest of this section we prove Lemma 3.5. Fix a feasible solution T for MaxCT.

Definition 3.3. Level i in G is *cheap* (w.r.t. T) if $c(T \cap A^i) \leq 2k$ and is *expensive* otherwise. A copy A_{ij} of A on a cheap level i is called:

- *Heavy:* if $c(T \cap A_{ij}) \geq 4 \cdot c \cdot k/|B|^{i-1}$ (namely, if $|T \cap A_{ij}| \geq 4 \cdot c \cdot k$).
- *Lost:* if $A_{ij} \cap T = \emptyset$ (because one of its ancestors $v \in B_{pq}$ is not covered by $T \cap A_{pq}$).
- *Active:* if A_{ij} is not heavy nor lost.

Because the total budget bound is $C = h \cdot k$ we get:

Fact 3.6. *At most half of the levels are expensive. In a cheap level i, at most a fraction of* $1/(2c)$ *(and a total of* $|B|^{i-1}/(2c)$*) of the A copies are heavy.*

When we say that an α fraction of the copies are active in level i, this refers to the fraction compared to the initial number of copies at level i, namely, this means that $\alpha|B|^{i-1}$ copies are active. We now want to estimate how many copies go from active to lost going down the "tree". We cannot estimate the loss caused by expensive levels, hence we ignore the effect of expensive levels; namely, we will assume that in each expensive level the elements of every non-lost copy may be fully covered and fully belong to T. We also assume that if A_{ij} is heavy then B_{ij} belongs to T.

Assume for the sake of contradiction that T contains at least $1/c$ fraction of the terminals.

Claim 3.7. *The fraction of active copies at every cheap level i is at least* $1/(2c)$.

Proof. If there is a cheap level i so that less than $1/(2c)$ of its copies are active, then since there are at most a $1/(2c)$ fraction of heavy copies (see Fact 3.6), in total there are less than $2 \cdot |B|^{i-1}/(2c) = |B|^{i-1}/c$ non-lost copies. Since by symmetry every H_{ij} copy has the same number of descendant terminals, less than a $1/c$ fraction of the terminals belong to T; a contradiction. □

Claim 3.8. *If i is a cheap level then the number of non-lost copies in level* $i+1$ *is* at most $(1 - e^{-8c}/2)$ *times the fraction of non-lost copies in level i.*

Proof. There are at least $1/(2c)$ active copies in level i (see Claim 3.7). We now show that some of their children in level $i+1$ become lost. Fix an active copy A_{ij}. As A_{ij} is not heavy, $|A_{ij} \cap T| \leq 4c \cdot k$. Thus we have a situation such as in set cover. The number of sets taken is at most $4c \cdot k$. Claim 3.2, tells us that in a set cover situation, when only $4c \cdot k$ sets are selected, the fraction of elements covered is at most $(1-e^{-8c})$. Thus, the number of uncovered elements is at least $n-n(1-e^{-8c}) = n \cdot e^{-8c}$, or a fraction of e^{-8c} of the elements in B_{ij}. Each uncovered element leads to a loss of its unique child A-copy in level $i+1$. Thus the fraction of children copies of active copies lost is at least e^{-8c}.

We compute the fraction of lost copies among *all* non-lost copies (namely, among active and heavy copies). There may be the same number of heavy and active copies. Thus the fraction copies that become lost among all non-lost copies is at least $e^{-8c}/2$. The fraction of active copies among all copies is thus now at most $(1 - e^{-8c}/2)$ because for every single set cover instance the number of non-lost terminals is at most $n - n \cdot e^{-8c}/2 = n(1 - e^{-8c}/2)$. □

Claim 3.9. *There exists a level in which the fraction of active A-copies is strictly less than* $1/(2c)$.

Proof. Recall that $h = 4 \cdot e^{8c} \ln(2c)$. Consider the accumulative affect of going down over all cheap levels. The number of cheap levels is at least $h/2$. Say that at the beginning all copies are active. After going over at $h/2$ (or more) cheap levels we get that the number of of active copies is at most

$$\left(1 - \frac{1}{2e^{8c}}\right)^{h/2} < \frac{1}{2c}. \quad \square$$

The last claim contradicts Claim 3.7. This finishes the proof of Lemma 3.5, and thus the proof of Theorem 3.1 is also complete.

3.4. The details needed to get a bicriteria lower bound

We now show that the same hardness holds even if the NO-instance is allowed to violate the budget by a constant ρ.

Here we define $h = 4 \cdot e^{8c\rho} \ln(2c)$.

There is no change in the analysis of the YES-instance. A level is called cheap if the cost used in this level is at most $2\rho \cdot k$. An A_{ij} in a cheap level is called heavy if $|T \cap A_{ij}| > 4c\rho \cdot k$. The budget is still $h \cdot k$.

The number of cheap levels is even larger than before. The number of expensive copies at a cheap level is still at most $1/(2c)$ fraction. This is because we have defined a copy as heavy if $|T \cap A_{ij}| \geq 4c\rho \cdot k$, and a cheap level has cost at most $2\rho \cdot k$.

Note that now we may assume that a light copy contains at most $4c\rho \cdot k$ sets. Thus the fraction of remaining active copies goes down by at least a fraction of $(1 - e^{-8\rho \cdot c \cdot \mathrm{opt}}/2)$ at every cheap level. However, since we defined $h = 4 \cdot e^{8c\rho} \cdot \ln 2c$ the affect of the above is canceled by the larger h.

4. Algorithm for SLST

As was mentioned, for SLST with edge costs/lengths the algorithm of [16] computes a tree T with $c(T) = O(\log n) \cdot \mathsf{opt}$ and $\mathsf{diam}_\ell(T) = O(\log n) \cdot L$. This is done as follows. Let $C = \mathsf{opt}$. Maintain a collection of node-disjoint trees rooted by terminals. At the beginning every terminal roots a tree of size 1. Then iterate as follows. Let S be the set of roots.

For $s, t \in S$ let $c(s, t)$ be the minimum cost of an st-path among st-paths of length at most L. Although computing $c(s, t)$ is an NP-hard problem, for any $\varepsilon > 0$ we can approximate it using the FPTAS of [14], which computes a path P_{st} with $\ell(P_{st}) \leq L$ and $c(P_{st}) \leq (1 + \varepsilon)c(s, t)$. Now, construct an auxiliary complete graph on S with the costs of every edge st being $c(P_{st})$. As all terminals belong to the solution, by the pairing Lemma [20] there exists a matching on the roots of total cost at most $(1 + \varepsilon)C$, with at most one tree root unmatched (indeed the pairing lemma states that the roots can be paired to produce edge disjoint paths in the optimum tree, and so the sum of paths costs is at most $(1 + \varepsilon)C$). Compute the above perfect matching. Replace every edge st in the matching by the corresponding path P_{st} in G and merge the corresponding two trees together, making one of s, t the root of the merged tree. At every iteration, the cost invested is at most $(1+\varepsilon)C$, the radius of each tree is increased by at most L, and the number of trees is roughly halved. The latter implies that the number of iterations is $O(\log n)$, and the $(O(\log n), O(\log n))$ bicriteria approximation follows.

Before we present the algorithm, we discuss the difference between edge and node costs and describe the difficulties in getting $O((\log n), O(\log n))$ approximation.

In the edge case, first the above "metric completion" is computed and a minimum matching is found. The cost of the matching is at most the optimum cost. Clearly, it is not possible to get a pairing lemma with node disjoint paths; to see this, just consider a large star.

In the vertex case, a tree can be decomposed into vertex disjoint spiders.

Definition 4.1. A tree is a *spider* if it has at most one node of degree ≥ 3. A *spider decomposition* $\mathcal{D}$ of a tree T rooted at r is a collection of node-disjoint spiders, so that each of them is a rooted subtree of T, and so that the leaf sets of the spiders in $\mathcal{D}$ partition the leaf set of T.

Lemma 4.1 (*[15]*). *Any tree T rooted at r admits a spider decomposition so that every spider has at least two leaves, or in the decomposition there is exactly one spider with one leaf and root r.*

To get an $(O(\log n), O(\log n))$ approximation, we need to find a cover of V by disjoint spiders, so that each spider has length at most L, and the sum of costs of the spider paths is within a constant from the optimum. The problem of finding such a spider collection is basically as general as the *dominating set* problem (see [9]). The dominating set is equivalent to set-cover with respect to approximation and so can not be approximated by $o(\ln n)$ unless $P = NP$ [21]. Thus an $(O(\log n), O(\log n))$ approximation seems hard to be derived. However, we now show that finding a *single* tree whose density is at most the density of the optimum spider, is very easy.

Guess the root r of the best density spider. Guess the number j of leaves in the star. After that, tree T' consists of the j lowest cost paths, from r to the best j terminals, among paths of distance at most L from r to terminals. Clearly, we can bound $c(T')$ by the sum of costs of its j paths (in many cases the cost of the tree is much less than that). In the optimum star, the paths are disjoint, except for the root, which means that the cost of the star is by itself the sum of lengths of vertex disjoint paths, except for r. The root r can be ignored as it contributes $jc(r)$ to both sum of paths costs in the spider and to the sum of path costs in T'. Since in T' the cost is bounded by the sum of the j shortest cost paths from r to the best terminals, while in the spider, the cost is a sum of *some* j paths from r to some terminals, the density of T' not larger than the density of the optimum spider.

It seems that the only way to find a reasonable spider decomposition is to iteratively find the cheapest spider at this moment, using a greedy algorithm.

Lemma 4.2. *For any $\varepsilon > 0$ there exists a polynomial algorithm that computes a collection of rooted trees of total cost $O(\log n) \cdot \mathsf{opt}$ containing all terminals, so that each tree has ℓ-radius at most L, and so that every tree, except of maybe one, contains at least two terminals.*

Proof. While there are at least two terminals not belonging to any tree, iteratively find a tree F with at least 2 terminals of radius L whose cost-density, (which is its cost over the number of terminals in it) is at most $(1 + \varepsilon)$ times the one of the optimum density spider. This is done as explained above. We stress that the density in question is only with respect to non-covered terminals (only terminals not covered by previous spiders are considered). Then we remove the covered terminals, and iterate. An analysis similar to the classical set-cover analysis shows that the total cost of the resulting decomposition is $O(\log n)$ times the cost of the minimum cost spider decomposition which is $O(\log n) \cdot \mathsf{opt}$. □

The other parts of the algorithm are similar to the algorithm of [16] for the edge cost case. Maintain a node-disjoint collection of rooted trees (the roots are terminals). Initialize every terminal as a root of a tree of size 1. Then iterate as follows. Let S be the root set. Given $\varepsilon > 0$, compute a family of trees as in Lemma 4.2, considering only the set S of roots as terminals, and for every spider, merge the trees of the roots it contains; the root of the new tree is set to be one of these roots. At every iteration, the cost invested is at most $(1 + \varepsilon)C \cdot O(\log n)$, the radius of each tree is increased by at most L, and the number of trees is roughly halved. The latter implies that the number of iterations is $O(\log n)$, and the $(O(\log^2 n), O(\log n))$ bicriteria approximation follows.

Acknowledgements

We thanks a referee for his/her insightful comments that significantly helped in improving the presentation of the paper. The first author was partially supported by NSF support grant award number 0829959.

References

[1] M. Andrews, Hardness of buy-at-bulk network design, in: FOCS, 2004, pp. 115–124.
[2] M. Andrews, L. Zhang, Approximation algorithms for access network design, Algorithmica 34 (2) (2002) 197–215.
[3] M. Balinski, Fixed cost transportation problems, Nav. Res. Log. Q. (1961) 58–76.
[4] M. Charikar, A. Karagiozova, On non-uniform multicommodity buy-at-bulk network design, in: STOC, 2005, pp. 176–182.
[5] C. Chekuri, M.T. Hajiaghayi, G. Kortsarz, M.R. Salavatipour, Approximation algorithms for non-uniform buy-at-bulk network design, in: FOCS, 2006, pp. 677–686.
[6] C. Chekuri, M.T. Hajiaghayi, G. Kortsarz, M.R. Salavatipour, Approximation algorithms for node-weighted buy-at-bulk network design, in: SODA, 2007, pp. 1265–1274.
[7] U. Feige, A threshold of ln n for approximating set cover, J. ACM 45 (1998) 634–652.
[8] M. Feldman, G. Kortsarz, Z. Nutov, Improved approximating algorithms for directed dteiner forest, in: SODA, 2009, pp. 922–931.
[9] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[10] G. Gallo, C. Sandi, C. Sodini, An algorithm for the min concave cost flow problem, European J. Oper. Res. 4 (1980) 248–255.
[11] M. Goldstein, B. Rothfrab, The one terminal telepak problem, Oper. Res. 19 (1971) 156–169.
[12] S. Guha, A. Moss, J.S. Naor, B. Schieber, Efficient recovery from power outage, in: STOC, 1999, pp. 574–582.
[13] M.T. Hajiaghayi, G. Kortsarz, M.R. Salavatipour, Approximating buy-at-bulk k-Steiner tree, in: APPROX, 2006, pp. 152–163.
[14] R. Hassin, Approximation schemes for the restricted shortest path problem, Math. Oper. Res. 17 (1) (1992) 36–42.
[15] P.N. Klein, R. Ravi, A nearly best-possible approximation algorithm for node-weighted steiner trees, J. Algorithms 19 (1) (1995) 104–115.
[16] M.V. Marathe, R. Ravi, R. Sundaram, S.S. Ravi, D.J. Rosenkrantz, H.B. Hunt III., Bicriteria network design problems, J. Algorithms 28 (1) (1998) 142–171.
[17] M. Minoux, Network synthesis and optimum network design problems: models, solution methods and applications, Networks 19 (1989) 313–360.
[18] A. Moss, Y. Rabani, Approximation algorithms for constrained node weighted Steiner tree problems, in: STOC, 2001, pp. 373–382.
[19] Z. Nutov, Approximating steiner networks with node weights, in: LATIN, 2008, pp. 411–422.
[20] R. Ravi, M. Marathe, S. Ravi, D. Rosenkrantz, H. H III., Many birds with one stone: multi-objective approximation algorithms, in: STOC, 1993, pp. 438–447.
[21] R. Raz, S. Safra, A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, in: STOC, 1997, pp. 475–484.
[22] F.S. Salman, J. Cheriyan, R. Ravi, S. Subramanian, Approximating the single-sink link-installation problem in network design, SIAM J. Optim. 11 (3) (2000) 595–610.
[23] W. Zangwill, Minimum concave flow in certain networks, Mgmt Sci. 14 (68) 429–450.