Approximating Buy-at-Bulk and Shallow-light k-Steiner trees*

MohammadTaghi Hajiaghayi † Guy Kortsarz‡ Mohammad R. Salavatipour§

Abstract

We study two related network design problems with two cost functions. In the buy-at-bulk k-Steiner tree problem we are given a graph $G(V, E)$ with a set of terminals $T \subseteq V$ including a particular vertex s called the root, and an integer $k \leq |T|$. There are two cost functions on the edges of G, a buy cost $b : E \longrightarrow \mathbb{R}^+$ and a distance cost $r : E \longrightarrow \mathbb{R}^+$. The goal is to find a subtree H of G rooted at s with at least k terminals so that the cost $\sum_{e\in H} b(e) + \sum_{t\in T-s} dist(t, s)$ is minimized, where $dist(t, s)$ is the distance from t to s in H with respect to the r cost. We present an $O(\log^4 n)$-approximation algorithm for the buy-at-bulk k-Steiner tree problem. The second and closely related one is bicriteria approximation algorithm for Shallow-light k-Steiner trees. In the shallow-light k-Steiner tree problem we are given a graph G with edge costs $b(e)$ and distance costs $r(e)$, and an integer k. Our goal is to find a minimum cost (under b-cost) k-Steiner tree such that the diameter under r-cost is at most some given bound D. We develop an $(O(\log n), O(\log^3 n))$-approximation algorithm for a relaxed version of Shallow-light k-Steiner tree where the solution has at least $\frac{k}{8}$ terminals. Using this we obtain an $(O(\log^2 n), O(\log^4 n))$-approximation algorithm for the shallow-light k-Steiner tree and an $O(\log^4 n)$-approximation algorithm for the buy-at-bulk k-Steiner tree problem. Our results are recently used to give the first polylogarithmic approximation algorithm for the non-uniform multicommodity buy-at-bulk problem [9].

1 Introduction

We study network design problems on graphs with two cost functions on the edges. These are the *buy-at-bulk* $k-$*Steiner tree* problem and the *shallow-light* $k-$*Steiner tree* problem. Suppose we are given an undirected graph $G(V, E)$ with a terminal set $T \subseteq V$, a specific vertex $s \in T$ called *the root*, and an integer $k \leq |T| \leq |V| = n$; vertices in $V - T$ are called Steiner nodes. A Steiner tree is a subgraph of G that is a tree and contains a required number of the terminal nodes (possibly all of them). In the buy-at-bulk $k-$Steiner tree problem, along with graph G, we also have two (non-related) cost functions on the edges of G: buy cost $b : E \longrightarrow \mathbb{R}^+$ and distance cost (sometimes also called rent cost) $r : E \longrightarrow \mathbb{R}^+$. We use the term *non-uniform* to denote that b and r are not related. All variants studied here are non-uniform. Our goal is to find a Steiner tree H spanning at least k vertices of T including the root (also known as k-Steiner tree) which minimizes the following:

*A preliminary version of this paper appeared in the Proceedings of 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2006, LNCS 4110, pp 153-163, 2006.

†Department of Computer Science, Carnegie Mellon University. E-mail: hajiagha@cs.cmu.edu. Supported in part by IPM under grant number CS1383-2-02.

‡Department of Computer Science, Rutgers University-Camden. E-mail: guyk@crab.rutgers.edu.

§Department of Computing Science, University of Alberta. E-mail: mreza@cs.ualberta.ca. Supported by NSERC grant No. G121210990, and a faculty start-up grant from University of Alberta.

$$\sum_{e \in H} b(e) + \sum_{t \in T-s} L(t,s), \tag{1}$$

where $L(t,s) = \sum_{e \in P(t,s)} r(e)$ with $P(t,s)$ being the unique path from t to s in H.

Buy-at-bulk network optimization problems with two cost functions, have been studied extensively, sometimes under different names such as cost-distance (where one function defines buy cost and another function defines length). These problems have practical importance. Consider for example the case that every edge can either be bought or rented (but not both). The chosen edges are required to provide bandwidth to satisfy some multicommodity demands. As we shall see later, this model is equivalent to the above buy-at-bulk model (in which every edge has both rent and buy costs). The rent or buy (but not both) version of the problem captures the practical setting where it is needed to decide whether to buy an edge and pay $b(e)$ once (typically $b(e)$ is relatively large) or to rent the edge e. If the user decides to rent the edge, then the pay is $r(e)$ per every commodity that goes via e. Buy-at-bulk problems have been studied through a long line of papers in the operation research and computer science communities after the problem was introduced by Salman et al. [28] (see e.g. [2, 8, 16, 17, 18, 19, 25, 27]).

The second problem we consider is a variant of the shallow-light network design problem. A graph $G(V,E)$ and a collection $T \subseteq V$ of terminals are given in addition to cost and length functions $b, r : E \longrightarrow \mathbb{R}^+$ and two bounds, a cost bound B and a length bound D. The cost of a spanning subtree $H(V,E')$ is $b(E') = \sum_{e \in E'} b(e)$. For a path P, $L(P) = \sum_{e \in P} r(e)$. The distance between u, v in H is $\mathrm{dist}_H(u,v) = L(P_{u,v})$ so that $P_{u,v}$ is the unique path between u, v in H. The diameter of H is $\mathrm{diam}(H) = \max_{u,v} \mathrm{dist}_H(u,v)$. Throughout, whenever we talk about the cost of a path or the cost of a tree we mean the cost under b-cost and whenever we say length or diameter we mean under r-cost. Assuming a spanning subtree $H(V,E')$ with cost at most B and diameter at most D exists, the shallow-light spanning tree problem is to find H. The more general shallow-light k-Steiner tree problem requires to select for an input k a tree spanning k terminal nodes that meets the diameter and cost bounds D and B, respectively. Even the shallow-light spanning tree ($k = n$) special case is NP-hard and also NP-hard to approximate within a factor better than $c \log n$ for some universal constant c [5]. Thus we focus on approximation algorithms. An (α, β) bi-criteria approximation algorithm for the shallow-light k-Steiner tree problem is an algorithm that delivers a tree H' with at least k terminals (vertices in T) whose diameter is at most $\alpha \cdot D$, and whose cost is at most β times the cost of a D-diameter minimum cost tree. The constraint that only $k < n$ nodes have to be picked makes this problem considerably harder than the usual shallow-light spanning tree problem, namely, the $k = n$ case.

The paper is organized as follows. In the rest of this section we discuss some different models of cost functions on the edges and the relations between them followed by a description of our techniques. Section 2 contains our main algorithm for the shallow-light k-Steiner tree as well as a reduction algorithm from buy-at-bulk k-Steiner tree.

1.1 Subadditive monotone functions

Our algorithm for the buy-at-bulk variant applies in a slightly more general setting of minimizing $\sum_e f_e$ with every f_e being a monotone subadditive function. This is related to economies of scale. Typically, a capacity (or bandwidth) on a link can be purchased in some discrete units $u_1 < u_2 < \ldots < u_r$ with costs $c_1 < c_2 < \ldots < c_r$ such that the cost per bandwidth decreases $c_1/u_1 > c_2/u_2 > \ldots > c_r/u_r$. The capacity units are sometimes referred to as cables or pipes. The cables induce a monotone concave (or more generally a sub-additive) function $f : \mathcal{R}^+ \rightarrow \mathcal{R}^+$ where $f(b)$ is the minimum cost of cables of total capacity at least b.

It is possible to deal with concave monotone functions as follows (see [2, 27]). The function f_e is approximated by a collection of simple piece-wise linear functions of the form $a + bx$. We replace an edge e with cost function f_e by a a collection of parallel edges, one for each of the simpler linear functions. Given a function $f : \mathcal{R}^+ \rightarrow \mathcal{R}^+$, and a fixed $\varepsilon \geq 0$, for integer $i \geq 0$ let $g_i : \mathcal{R}^+ \rightarrow \mathcal{R}^+$ be a linear function defined by $g_i(x) = f(a^i) + f(a^i)/a^i \cdot x$ where $a = (1+\varepsilon)$. It can be verified that if f is monotone and sub-additive then for all $x \geq 1$, $\frac{1}{2+\varepsilon} \min_i g_i(x) \leq f(x) \leq \min_i g_i(x)$.

1.2 Related work

In the buy-at-bulk multicommodity problem we are given p source-sink pairs, $\{s_i, t_i\}_{i=1}^p$. A subset E' of the edges is feasible if for every i, an s_i to t_i path exists in $G' = (V, E')$, namely, s_i, t_i belong to the same connected component in G'. The cost of E' is $\sum_{e \in E'} b(e) + \sum_i \operatorname{dist}_{G'}(s_i, t_i)$ where the distance is with respect to r, and the goal is to find a minimum cost feasible E'. If we are also given an integer $k \leq p$ and must find a solution that connects k (out of p) s_i, t_i pairs then we have the buy-at-bulk k-multicommodity problem. It is easy to see that the buy-at-bulk Steiner (but not k-Steiner) tree problem is a special case of the buy-at-bulk multicommodity problem in which all the sinks are at a single vertex (namely the root). The buy-at-bulk k-Steiner tree problem is a special case of the buy-at-bulk k-multicommodity problem. However, it is shown in [20] that if the buy-at-bulk k-multicommodity problem admits a polylogarithmic approximation ratio then so does the dense-k-subgraph problem (see [13]). For a long time now (11 years since the journal version and 14 years since the conference version [24]) the best known approximation algorithm for the dense k-subgraph problem is $O(n^{1/3-\varepsilon})$ for some positive $\varepsilon > 0$ [13], and it is widely believed that the dense k-subgraph problem admits no polylogarithmic approximation ratio. If indeed the dense k-subgraph problem admits no polylogarithmic approximation factor, then the result in our paper shows that the case of single source (but many sinks), namely buy-at-bulk k-Steiner tree, is provably easier to approximate than the general case of arbitrary source-sink pairs.

In the uniform version of the buy-at-bulk multicommodity problem all the buy values along edges are equal. The best approximation factor known for the uniform case is $O(\log n)$ due to the results of Awerbuch and Azar [3], Bartal [6] and Fakcharoenphol et al. [12]. Kumar et al. [25] and Gupta et al. [18] present a constant factor approximation algorithm for a the case the cost of buying each edge is equal to M times the cost of renting the edge (per unit length) for a fixed M. The single sink uniform version also admits constant-factor approximation algorithms [17, 19].

Meyerson et al. [27] study the buy-at-bulk Steiner tree or equivalently, the non-uniform single sink buy-at-bulk multicommodity problem for which they give a randomized $O(\log n)$-approximation algorithm that was derandomized by Chekuri, Khanna, and Naor [10] via an LP formulation. Note that none of these algorithms yield any polylogarithmic approximation ratio for the k-Steiner tree case. For the most general case, the best known ratio for the non-uniform buy-at-bulk multicommodity problem was $exp(O(\sqrt{\log n \log\log n}))$ by Charikar and Karagiozova [8]. Recently [9], we have improved this result to a polylogarithmic approximation factor using the results of this paper.

On the lower bound side, Andrews [1] showed that unless NP $\subseteq$ ZPTIME $(n^{\mathrm{polylog}\, n})$ the buy-at-bulk multicommodity problem has no $O(\log^{1/2-\varepsilon} n)$-approximation algorithm for any $\varepsilon > 0$. Under the same assumption, the uniform variant admits no $O(\log^{1/4-\varepsilon} n)$-approximation algorithm for any constant $\varepsilon > 0$. For the single sink case, Chuzhoy et al. [11] showed that the problem cannot be approximated better than $\Omega(\log\log n)$ unless NP $\subseteq$ DTIME($n^{\log\log\log n}$).

The buy-at-bulk k-Steiner tree problem generalizes the classic Steiner tree, k-MST, and more generally k-Steiner tree problems when the rent cost is zero. See for example [15]. As we mentioned above, the buy-at-bulk Steiner tree problem first was studied by Meyerson et al. [27], but we are not aware of any result on buy-at-bulk k-MST or buy-at-bulk k-Steiner tree.

The shallow-light $k-$Steiner tree problem generalizes the Shallow-light Steiner problem [26] which is the special case of $k = |T|$. It generalizes the k-MST problem [7, 4, 7, 14, 15] which is the case $D = \infty$ and also the bounded diameter spanning tree problem [22] which is the zero costs case. Even the $k = |T|$ special case is NP-hard and also NP-hard to approximate within a factor better than $c \log n$ for some universal constant c [5]. For $k = |T|$ an $(O(\log n), O(\log n))$-approximation algorithm is given in [26]. The constraint that only $k < n$ nodes have to be picked seems to make this problem harder to approximate than the usual shallow-light Steiner tree problem, namely, the $k = |T|$ case.

1.3 Our results

We give the first approximation algorithm for the buy-at-bulk k-Steiner tree problem:

Theorem 1.1 *Given an instance of the shallow-light k-Steiner tree problem with diameter bound D we can obtain a $\frac{k}{8}$-Steiner tree with diameter at most $O(\log n \cdot D)$ and cost at most $O(\log^3 n \cdot \mathrm{OPT})$, where* OPT *is the cost of an optimum shallow-light k-Steiner tree with diameter bound D.*

Theorem 1.2 *There is a polynomial time $O(\log^4 n)$-approximation algorithm for the buy-at-bulk k-Steiner tree problem*

Theorem 1.3 *The shallow-light k-Steiner subtree problem admits an $(O(\log^2 n), O(\log^4 n))$ bicriteria approximation algorithm that runs in polynomial time*

It is worth mentioning that Theorem 1.3 is one of the main tools we use to obtain the first polylogarithmic approximation algorithm for the non-uniform multicommodity buy-at-bulk problem in the subsequent work [9]. In the same place, we show that using rounding a solution to an LP formulation of the problem, we can improve the result of Theorem 1.2 to an $O(\log^3 n)$-approximation algorithm.

1.4 Equivalent models

There are three different models for the buy-at-bulk k-Steiner tree problem. This holds also for the other versions of the problem like the buy-at-bulk multicommodity problem. Below we describe these three models and show that in fact they are all equivalent.

Model A. The *unique cost* model: Every edge e of G is given with either a buy cost $b(e)$ or a rent cost $r(e)$. For buy edges we have to pay $b(e)$ to use them. For rent edges we have to pay $r(e) \cdot f(e)$ where $f(e)$ is the amount of demand routed over that edge, i.e., the number of terminals $t \in T - s$ for which e lies on the unique s,t-path in the Steiner tree solution.

Model B. The *rent or buy* model: Every edge e is given with both a buy cost $b(e)$ and a rent cost $r(e)$. For every edge in the solution, we have to decide whether to buy e and pay $b(e)$ or rent it and pay $r(e) \cdot f(e)$. However, it is not possible to both buy and rent the edge.

Model C. The *rent and buy* or *cost-distance* model: This is the model we defined earlier, i.e.

every edge is given with both a buy cost $b(e)$ and a rent cost $r(e)$. The cost we pay for every edge in the solution is $b(e) + r(e) \cdot f(e)$. This model first has been considered by Meyerson et al. [27].

The following theorem shows that all these three models are in fact equivalent. We will be working with Model C in the rest of the paper.

Theorem 1.4 *All the three models A, B, and C are equivalent.*

Proof: We show how each model can be formulated by the other models. Here, we write, e.g., $C \longrightarrow A$ short for "it is possible to model Model A via Model C".

1. $C \longrightarrow A$: We mimic model A by model C as follows. We will have the same set of vertices as in model A. An edge that does not have a buy cost (in A) is given 0 buy cost in C, and otherwise, it is given 0 rent cost. Note that by definition of Model C zero rent or buy costs do not affect the overall cost and so can be ignored. This is the same affect as in model A.
2. $B \longrightarrow C$: An edge $e = (v, u)$ in model C is replaced by two edges $e_1 = (v, w_e)$, $e_2 = (w_e, u)$ (in model B) for some new and specific for e vertex w_e. The buy cost of (v, w_e) is $b(e)$ and its rent cost is ∞. The rent cost of (w_e, u) is $r(e)$ and its buy cost is ∞. The new w_e nodes are not terminals. The edges e_1 cannot be rented and the edge e_2 cannot be bought. Observe that for the two serial edge e_1, e_2 to be used, a solution both has to pay $b(e)$ buy cost and $r(e)$ rent cost for using these edges. This mimics model C.
3. $A \longrightarrow B$: An edge $e = (u, v)$ in B is replaced by by two parallel edges e_1, e_2 between u, v (it can be replaced by paths of length 2 if parallel edges are needed to be avoided). The edge e_1 will have rent cost $r(e)$ and the edge e_2 will have buy cost $b(e)$. Clearly, if e_2 is bought then there is no need to rent e_1. Otherwise, e_1 needs to be rented, unless the (u, v) is not used. This is equivalent to choosing exactly either to rent or to buy e (but not both).

■

1.5 The technique used

Our technique is quite general and may find other applications in the future. Note that the number $t = |T|$ of terminals may be much larger than k. Fixing some optimum solution let a terminal be a *true terminal* if it belongs to the optimum solution.

We try to find a shallow-light $\frac{k}{8}$-Steiner tree that has cost at most $O(\log^3 n \cdot \mathrm{OPT})$ and diameter at most $O(D \log n)$. At any given time, our algorithm maintains a collection C of rooted trees. The goal is to increase the size of some of the trees in C so that one of them will eventually have at least $\frac{k}{8}$ terminals with cost and diameter bounded as above. The algorithm runs in rounds and in every round we may have several iterations (of some loop). At the beginning of every round, each terminal is a singleton tree. In every iteration of a round we perform a test. If the test result is successful, then we merge two trees of C into a larger connected component and go to the next iteration. If the test fails then we end this round, delete some of the terminals (temporarily), and start the next round of the algorithm with a new (smaller) set of terminals. As stated above, we initialize again each terminal to be a component of size 1, ignoring any mergers that were done in the last failed round.

The key property of the test is that the number of true terminals among the roots of the trees in C (which is exactly the number of true terminals deleted among all terminals deleted) in the case of a failure can be seen to be at most $c \cdot k/\log^2 n$ for some constant c. On the other hand, we make sure that the number of roots removed in case of a failure is $c' \cdot k/\log n$ for some constant c'. Thus, intuitively, in every failure we delete "many" terminals but only "few" true terminals.

It is shown that eventually there will be a round in which one of the trees will reach size $\frac{k}{8}$ or more. Moreover, the deleted true terminals prior to this round almost do not affect the cost of the output solution because a constant fraction of the true terminals always remains.

2 The algorithms

2.1 Reducing the buy-at-bulk k-Steiner tree problem to a shallow-light k-Steiner tree problem

In this section, we show how to prove Theorem 1.2 using Theorem 1.1. A bicriteria network design problem [26] (A, B, S) is defined by identifying two objective functions, A and B, and specifying a membership requirement in a class of subgraphs S. Typically, there is a budget constraint on the first objective and we seek to minimize the second objective function. This way, the (diameter, cost, k-Steiner tree) problem is naturally defined as follows: we are given an undirected graph $G(V, E)$ with terminal set T, an integer $k \leq |T|$, diameter bound D, and two cost functions $b : E \longrightarrow \mathbb{R}^+$ and $r : E \longrightarrow \mathbb{R}^+$ on the edges. Our goal is to find a minimum b-cost (i.e. minimizing the cost under the b function) Steiner tree with k terminals in G such that the diameter of the tree under the r-cost is at most D. We can assume that a particular terminal $s \in T$, called the root belongs to the solution (we can simply guess this node s). Therefore, we are solving the rooted shallow-light k-Steiner tree. We may relax the condition of requiring at least k terminals being in the solution to at least σk terminals be in the solution for some constant $\sigma \leq 1$. We call this variation the relaxed shallow-light k-Steiner tree.

We say an algorithm is an (α, β)-approximation algorithm for an (A, B, S)-bicriteria problem if in the solution produced the first objective (A) value is within factor at most α of the budget and the second objective (B) value is at most β times the minimum for any solution that is within the budget on A. Marathe et al. [26] gave an $(O(\log n), O(\log n))$-approximation algorithm for the (diameter, cost, Spanning tree) problem. In Theorem 1.1 we show how to obtain an $(O(\log n), O(\log^3 n))$-approximation algorithm for the relaxed shallow-light k-Steiner tree where the solution has at least $\frac{k}{8}$ terminals. More specifically, the algorithm takes another parameter M. If there is a shallow-light k-Steiner tree with diameter D and cost at most M, then the algorithm is guaranteed to returns a solution with diameter at most $O(\log n \cdot D)$ and cost at most $O(\log^3 n \cdot M)$; otherwise the algorithm may fail (not return any solution). So if $M \geq \text{OPT}$, where OPT is the cost of an optimum solution with diameter D, then the algorithm will return a solution of cost at most $O(\log^3 n \cdot M)$. We will start from a trivial upper bound for OPT and do a binary search to find a value of M for which the algorithm succeeds (in returning a solution of cost at most $O(\log^3 n \cdot M)$) while it fails for $M/2$. In this case, clearly $\text{OPT} \geq \frac{M}{2}$. Thus our solution of cost $O(\log^3 n \cdot M)$ and diameter at most $O(\log n \cdot D)$ is indeed an $(O(\log n), O(\log^3 n))$-approximation.

Similarly, the algorithm for buy-at-bulk k-Steiner tree takes a parameter M. Again, we do a binary search to find a value of M for which the algorithm succeeds (in returning a solution of cost at most $O(\log^4 n \cdot M)$) while it fails for $M/2$. We start with the trivial upper bound of $\sum_e c(e) + |T| \cdot r(e)$ for M. Throughtout the rest, we assume that $M \geq \text{OPT}$ is a given parameter.

Lemma 2.1 *If there is an (α, β)-approximation algorithm for the relaxed shallow-light rooted k-*

Steiner tree problem such that the solution has at least $\frac{k}{8}$ terminals, then we have an approximation algorithm for (rooted) buy-at-bulk k-Steiner tree problem that given a parameter $M \geq \mathrm{OPT}$ returns a solution of cost at most $O((\alpha+\beta)\log k \cdot M)$.

Proof: Consider the input graph $G(V,E)$ for the buy-at-bulk k-Steiner tree problem and assume that $M \geq \mathrm{OPT}$. We mark every vertex with r-distance larger than M to s as "to be ignored". Clearly these vertices cannot be part of any optimal solution. Then, while $k > 0$ we do the following steps:

1. Run the (α,β)-approximation algorithm **A** for the relaxed shallow-light $\frac{k}{2}$-Steiner tree with diameter (under r-cost) bounded by $D = \frac{4M}{k}$ and parameter M.
2. Mark all the terminals (except the root) of the solution of **A** as Steiner nodes.
3. Decrease k by the number of new terminals found in this stage.

Since the root belongs to all the (sub)trees found in each iteration of the while loop, at the end we will get a connected graph, call it H, which spans k terminals. We can easily get a k-Steiner tree of cost no more than the total cost of H by taking the union of shortest paths (with respect to rent) from each of the terminals in H to the root and deleting unused edges. So it is enough to upper bound the total cost of H and this is what we do next.

At some iteration let k' be the number of yet unspanned terminals. Consider an optimal solution H^* for buy-at-bulk k'-Steiner tree instance and iteratively delete every leaf of H^* with r-distance to s (the root) larger than $\frac{2M}{k'}$. We delete at most $\frac{k'}{2}$ terminals. Otherwise, more than $k'/2$ terminals have rent distance at least $2M/k' \geq 2\,\mathrm{OPT}/k'$ to the root s and this is a contradiction as the total cost is more than OPT. So we are left with a tree rooted at s containing at least $\frac{k'}{2}$ terminals. An (α,β)-approximation algorithm for the relaxed shallow-light $\frac{k'}{2}$-Steiner tree finds a tree containing s with at least $\frac{k'}{16}$ new terminals with r-distance to S is at most $\beta \cdot \frac{2M}{k'}$ and cost bounded by αM. Given the bound from the root s, this adds at most $k' \cdot \beta \cdot \frac{2M}{k'} = 2\beta M$ to the rent cost of the solution. The buy cost added is at most αM. So we have covered a constant fraction of the remaining terminals at cost at most αM and the diameter increase is at most $2\beta M$. By a standard set-cover like arguments (see [23]), after at most $O(\log k)$ iterations, we have a tree with k terminals whose total cost is at most $O((\alpha+\beta)\log k \cdot M)$. ■

Proof of Theorem 1.2:
By Theorem 1.1 there is an $(O(\log n), O(\log^3 n))$-approximation algorithm for the relaxed shallow-light rooted k-Steiner tree. Thus, using Lemma 2.1, with $\alpha = O(\log n)$ and $\beta = O(\log^3 n)$, and the arguments before Lemma 2.1 we have an $O(\log^3 n \log k)$-approximation algorithm and therefore an $O(\log^4 n)$-approximation algorithm for (rooted) buy-at-bulk k-Steiner tree. ■

Proof of Theorem 1.3: We use an algorithm similar to that of Lemma 2.1; iteratively apply the algorithm for Theorem 1.1. At each iteration, we obtain a Steiner tree rooted at r covering a constant fraction (namely at least $\frac{1}{8}$) of the required number of terminals, with cost at most $O(\log^3 n \cdot \mathrm{OPT})$ and diameter at most $O(\log n \cdot D)$. Thus after at most $O(\log k)$ iterations, we cover at least k terminals at cost at most $O(\log^4 n \cdot \mathrm{OPT})$ and diameter at most $O(\log^2 n \cdot D)$. ■

2.2 Algorithm for relaxed shallow-light k-Steiner tree

In this subsection we prove Theorem 1.1. Our algorithm is inspired by the algorithms of [26] (for shallow-light Steiner tree) and [4] for the (standard) k-MST problem. Recall that the input consists of a graph $G(V,E)$ with two edge costs b and r, D is a bound on the diameter under the r cost, $T \subseteq V$ is the set of terminals including the root s, k is the number of terminals we wish to cover, and ε is an error parameter. We also assume that OPT is the cost of an optimum solution and $M \geq \mathrm{OPT}$ is a given parameter.

First note that we can delete every vertex with r-distance to s larger than D (since they cannot be in the optimum solution). So all the vertices that remain are at distance at most D from s and so are at distance at most $2D$ from each other in G. Next we transform G into another graph, G^c, which we call the completion of G by doing the following. For every pair of vertices $u, v \in V$ we find a $(1+\varepsilon)$-approximate minimum cost u,v-path under b-cost with length (under r-cost) at most $2D$. Let $p^*(u,v)$ denote this cost. For this, we use the FPTAS algorithm of Hassin [21] which runs in time $O(|E|(\frac{n^2}{\varepsilon}\log\frac{n}{\varepsilon}))$. We add a new edge between u and v with b-cost equal to the cost of $p^*(u,v)$ and r-cost equal to the length of $p^*(u,v)$. Later on, in any solution of G^c that uses this new edge, we can replace it with path $p^*(u,v)$ in G at no extra cost and without increasing the length (diameter). Therefore:

Lemma 2.2 *If we have a bicriteria solution of cost X and diameter Y in G^c then we can find (in polynomial time) a solution of cost at most X and diameter at most Y in G.*

By this lemma, and since $G \subseteq G^c$, it is enough to work with graph G^c. By the argument before the lemma, we can assume G^c is a complete (multi)graph.

Before presenting the algorithm, we should note that the "rooted" and "un-rooted" versions of this problem are reducible to each other at the cost of a constant factor loss in the approximation ratio. Clearly, if we can solve the rooted version we can also solve the un-rooted version by simply trying all the terminals as the root and choose the smallest solution. On the other hand, if we have an algorithm for the un-rooted version we can do the following. Delete every node $v \in G^c$ for which the cost of edge sv is larger than $(1+\varepsilon)M$. Solve the un-rooted problem and if the solution does not contain the root v then add the root. This is done by arbitrarily adding an edge from v to some node in T. This will increase the cost by at most $(1+\varepsilon)M$ and the diameter by at most $2D$. Hence, it is enough to present an approximation algorithm for "un-rooted" shallow-light k-Steiner tree.

We focus on graph G^c and give an algorithm which finds a shallow-light $\frac{k}{8}$-Steiner tree in it that has cost at most $O(\log^3 n \cdot M)$ and diameter at most $O(D\log n)$. As said earlier, the algorithm runs in rounds and in every round we may have several iterations (of some loop). At every round we start with every terminal as a singleton connected component (tree). Initially, every terminal is the center of its own component (root of the tree). In every iteration of a round we perform a *test*. Each test has one of two outcomes: "success" or "failure". If the test is a success, we merge two connected components by connecting their centers (roots of the trees) using a path, making one of them the new center, and go to the next iteration. A single failure in an iteration inside a round causes the entire round to be a failure (so we end that round). After a failed round some of the centers (roots) of trees (which are terminals) are deleted, we exit the loop, and start the next round of algorithm with a new (smaller) set of terminals. As stated above we initialize again each terminal to be a component of size 1, ignoring any mergers that were done in the last failed round.

Our goal is to find a connected component (tree) containing at least $\frac{k}{8}$ terminals. We say that a round is *failure-free* if it has no failures at all. The number of connected components is reduced by 1 by every test that ends with success. Thus, a failure-free round will eventually end with a connected component with at least $\frac{k}{8}$ terminals. Clearly, either we fail at every round, in which case the set of terminals eventually turns empty, or we will eventually have a failure-free round. We later show that the first case above cannot happen.

Assume that in round i of the algorithm the number of terminals is t_i, where $t_1 = t = |T|$. At each iteration of the loop in each round i, we divide the connected components into $O(\log t_i)$ clusters, where cluster j (for $j \geq 3$) contains all the connected components for each of which the number of terminals is between $t_i/2^{j+1}$ and $t_i/2^j$.

Definition 2.1 *In every iteration of round i (for every $i \geq 1$), a cluster is called* light *if the total number of terminals in the union of the connected components in that cluster is at most $\frac{t_i}{2\log t_i}$. Otherwise, it is called* heavy.

Lemma 2.3 *In every iteration of round i (for every $i \geq 1$) there are at least $\frac{t_i}{2}$ terminals in heavy clusters.*

Proof: There are at most $\log t_i$ light clusters as there are at most $\log t_i$ clusters in total, and therefore they have a total of at most $\frac{t_i}{2}$ terminals. The rest of the terminals must belong to heavy clusters. ∎

In any round i and any iteration of this round, we compute the light and heavy clusters. Assuming that there are at least $\frac{k}{2}$ terminals remaining in G^c, we show (in the Main Lemma) that there is a heavy cluster with at least two connected components. Then we pick such a heavy cluster arbitrarily, say cluster C_j. Assume that all the components of C_j have between p and $2p$ terminals where $p = t_i/2^{j+1}$. For every two components in C_j we consider the edge connecting their centers (recall that since we are in G^c this edge may be obtained from the approximate minimum cost path with length at most $2D$ between those vertices in G). Two connected components c_a and c_b in C_j are called *reachable* if the cost of the edge connecting their centers is at most $16\log^2 t \cdot pM/k$. If two centers are reachable, by charging each terminal in the two component by $8\log^2 t \cdot M/k$, we can pay for the cost of this merger. In that case, at any given time, the total cost of a component is at most the total amount we have charged to the terminals in that component, which we show later is at most $O(\log^3 n \cdot M/k)$. This can be used to obtain an upper bound on the cost of each connected component. We test to see if there is a pair of reachable connected components in C_j. If there is such a pair of components, then we merge the components by adding the edge between their centers and then charge every terminal in the two components by $8\log^2 t \cdot M/k$. Since there are at least $2p$ vertices in c_a and c_b combined, the total charge is enough to pay for the cost of connecting the two components. We make one of the centers of c_a or c_b (arbitrarily) to be the new center of the new (merged) component and proceed to the next iteration of this round.

Otherwise, if our test fails because there are no two reachable centers in C_j (i.e. the cost of every edge between the centers of components in C_j is larger than $16\log^2 t \cdot M \cdot p/k$) then we delete all the centers (roots) of the connected components of C_j (which are all terminals). Assuming that C_j has x_j clusters, we set $t_{i+1} = t_i - x_j$, and then exit the loop and start round $i+1$. Below is the formal description of the algorithm.

1. Set the counter i (for round) to 1 and let $t_1 = t = |T|$.
2. Every terminal is a connected component by itself and is the center of that component.
3. Repeat until there is a connected component with $\frac{k}{8}$ terminals:
   (a) Compute light and heavy clusters.
   (b) Throw away (ignore) every heavy cluster which has only one connected component and pick an arbitrary heavy cluster, say C_j, which has at least two components.

(c) If there are two components c_a and c_b in C_j such that the cost of the edge connecting their centers is at most $16 \log^2 t \cdot pM/k$ then we do the following merger:
/* The test succeeded */
   i. Merge the components by adding that edge.
   ii. Charge every node in the two components by $8 \log^2 t \cdot M/k$.
   iii. Make one of the two centers the center of the new (merged) component and goto step (a).

(d) Otherwise, /* The test failed */
   i. Delete all the centers of components of C_j and reset the charges of all nodes to 0.
   ii. Set $t_{i+1} = t_i - x_j$ where x_j is the number of components of C_j.
   iii. Set $i = i + 1$, exit this loop and goto Step 2.

Below we show that every component participates in at most $O(\log n)$ merger operations and each time the diameter increases by at most $2D$. This will be enough to show that the diameter will always be in $O(D \log n)$. Also, as said earlier, we show that the cost of each component with p terminals is at most $O(\log^3 n \cdot pM/k)$. Finally, we show that the algorithm succeeds in that there is a failure-free round before all the terminals are deleted in Step 3(d).

Lemma 2.4 *In any round $i \geq 1$, every component participates in at most $O(\log n)$ merger operations.*

Proof: Each time a component participates in a merger the number of terminals of the components it belongs to is multiplied by at least $\frac{3}{2}$. This follows as the size of the large component is at most $2p$ for some integer p and of the smaller one at least p. Therefore there are at most $O(\log n)$ (or more precisely $O(\log k)$) iterations involving that component. ∎

Lemma 2.5 *In any round $i \geq 1$ of algorithm, for every component c_a that may be obtained from σ merge operations, the length (under r-cost) between the center of c_a and any other node in c_a is at most $2\sigma D$.*

Proof: The proof is by induction on σ and noting the fact that whenever we merge two components the length of the edge we add (between the centers) is at most $2D$. ∎

Corollary 2.6 *In any round $i \geq 1$, every component has diameter at most $O(D \log n)$, always.*

Proof: Follows from Lemmas 2.4 and 2.5. ∎

Lemma 2.7 *In any round $i \geq 1$, every terminal is charged at most $O(\log n)$ times and the total charge of every terminal is $O(\log^3 n \cdot M/k)$.*

Proof: Recall that every time a terminals is charged, the number of terminals in its new (merged) cluster grows by at least a 3/2 factor. Thus, each terminal participates in at most $O(\log n)$ mergers before we find a component with $\frac{k}{8}$ terminals or before the round fails (after which the charges are all reset to zero). Furthermore, each time a terminal is charged $8 \log^2 t \cdot M/k$. So the total charge of every terminal at any given time is $O(\log^3 n \cdot M/k)$ ∎

By this lemma, if the algorithm terminates with a $\frac{k}{8}$-Steiner tree then the cost of the tree is at most $O(\log^3 n \cdot M)$. Also, by Corollary 2.6 the diameter is at most $O(D \log n)$. Thus we only need to argue that the algorithm does find a $\frac{k}{8}$-Steiner tree and for that we need to show that the algorithm terminates before the number of terminals goes down below $\frac{k}{8}$. Since at every failed round the number of terminals is reduced, after at most t rounds the number of terminals becomes zero unless the algorithm terminates earlier with a feasible solution. Hence, if we show that the the set of terminals can never be smaller than $\frac{k}{2}$ then it means that the algorithm terminates before we have fewer than $\frac{k}{2}$ terminals. We also need to prove that we can perform step 3(b) of algorithm (i.e. find a heavy cluster with at least two connected components). These are proved in our main lemma, below. For that, we use the following pairing lemma:

Lemma 2.8 [26] *Let T be an arbitrary tree and let $v_1, v_2, \ldots, v_{2q}$ be an even number of vertices in T. There exists a pairing of the v_i (into q pairs) so that the unique paths joining the respective pairs are edge-disjoint.*

In the following lemma, we claim some properties on terminals not previously discarded by some failed round. We fix some optimal tree OPT and use that tree for proving these claims. We use OPT to refer to both the optimal solution and its cost. As some of the terminals in OPT may have been deleted by the failed rounds, the original OPT as defined *over* G does not exist any longer (the removal of deleted terminals may have destroyed that tree). Nevertheless, we can still use this original OPT to prove properties on G^c.

Lemma 2.9 (Main Lemma) *Let k_i be the number of terminals of* OPT *that are in G^c at the beginning of any round $i \geq 1$. If $k_i \geq \frac{k}{2}$ then:*

1. *There is at least one heavy cluster with at least two connected components (so we can perform step 3(b) of the algorithm).*
2. *$k_{i+1} \geq \frac{k}{2}$.*

Proof:

1) By Lemma 2.3 there are at least $\frac{t_i}{2} \geq \frac{k_i}{2} \geq \frac{k}{4}$ terminals in heavy clusters. Throw away every cluster with only one connected component. These components have a total of at most $\frac{k}{8} + \frac{k}{16} + \ldots < \frac{k}{4}$ terminals. Therefore, there is at least one heavy cluster with at least two components.

2) Note that by definition of k_i, $k_1 = k$ and $k_i \leq t_i$. Suppose at some iteration of round i and for some heavy cluster C_j chosen by the algorithm, no pair of centers are reachable to each other; so we have to delete all the centers of C_j from G^c. Assume that all the components of C_j have size between p_i and $2p_i$.

Proposition 2.10 *The number of centers of components of C_j that belong to* OPT *(namely, the number of true terminals among centers) is at most $k/(8p_i \log^2 t)$.*

Proof: Otherwise, using the pairing lemma (Lemma 2.8), we can pair those centers in OPT such that the paths connecting the pairs in OPT are all edge-disjoint. By averaging, there is at least one path with cost at most $16p_i \log^2 t \cdot \mathrm{OPT}/k \leq 16p_i \log^2 t \cdot M/k$ contradicting our assumption (because if there was such a path we would have merged the two components). ■

Therefore, by Proposition 2.10, the number of terminals of OPT in G^c goes down by a factor of at most $1 - 1/(8p_i \log^2 t)$. On the other hand, since C_j is a heavy cluster and we have at most $2p_i$ nodes in every component of C_j, there are at least $t_i/(2\log t_i)/(2p_i) = t_i/(4p_i \log t_i)$ components in C_j. This is also a lower bound on the number of centers (terminals) that are deleted in round i. Therefore, the number of terminals in G^c goes down by a factor of at least $1 - 1/(4p_i \log t_i)$. Hence:

$$k_i \left(1 - \frac{1}{8p_i \log^2 t}\right) \leq k_{i+1} \leq t_{i+1} \leq t_i \left(1 - \frac{1}{4p_i \log t_i}\right) \leq t_i \left(1 - \frac{1}{4p_i \log t}\right)$$

We now use the following two inequalities:

$$\text{If } x \leq 1/2, \text{ then } 1 - x \geq e^{-2x} \tag{2}$$

and

$$1 - x \leq e^{-x} \tag{3}$$

Using Inequality (2) and since $1/(8p_i \log^2 t) < 1/2$, it follows that $1-1/(8p_i \log^2 t) \geq e^{-1/(4p_i \log^2 t)}$. On the other hand from Inequality (3): $(1 - \frac{1}{4p_i \log t}) \leq e^{-1/(4p_i \log t)}$. Thus

$$\begin{aligned} k \cdot exp\left(-\sum_{\ell=1}^{i} \frac{1}{4p_\ell \log^2 t}\right) &\leq k \prod d_{\ell=1}^{i} \left(1 - \frac{1}{8p_\ell \log^2 t}\right) \leq k_{i+1} \leq t_{i+1} \\ &\leq t \prod_{\ell=1}^{i} \left(1 - \frac{1}{4p_\ell \log t}\right) \leq t \cdot exp\left(-\sum_{\ell=1}^{i} \frac{1}{4p_\ell \log t}\right). \end{aligned}$$

Note that both sequences t_i and k_i are decreasing but at different rates and t_i is lower bounded by k_i. Also, $\sum_{\ell=1}^{i} \frac{1}{4p_\ell} \leq \log t \cdot \ln t$, because for this value $t_{i+1} \leq t \cdot e^{-\ln t} = 1$. Plugging this upper bound on $\sum_{\ell=1}^{i} \frac{1}{4p_\ell}$ in the k_{i+1} lower bound we get that $k_{i+1} \geq k \ln t / \log t \geq k/2$. Therefore, k_j is always at least $k/2$. ■

Acknowledgments: The first author would like to thank Kamal Jain and Kunal Talwar for some initial discussions on the buy-at-bulk k-Steiner tree problem.

References

[1] M. Andrews, *Hardness of buy-at-bulk network design.*, in Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS '04), 2004, pp. 115–124.

[2] M. Andrews and L. Zhang, *Approximation algorithms for access network design*, Algorithmica, 34 (2002), pp. 197–215.

[3] B. Awerbuch and Y. Azar, *Buy-at-bulk network design*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), IEEE Computer Society, 1997, pp. 542–547.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, SIAM J. Comput., 28 (1999), pp. 254–262.

[5] J. Bar-Ilan, G. Kortsarz, and D. Peleg, *Generalized submodular cover problems and applications*, Theoretical Computer Science, 250 (2001), pp. 179–200.

[6] Y. Bartal, *On approximating arbitrary metrices by tree metrics*, in Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC '98), New York, NY, USA, 1998, ACM Press, pp. 161–168.

[7] A. Blum, R. Ravi, and S. Vempala, *A constant-factor approximation algorithm for the k mst problem (extended abstract)*, in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC '96), New York, NY, USA, 1996, ACM Press, pp. 442–448.

[8] M. Charikar and A. Karagiozova, *On non-uniform multicommodity buy-at-bulk network design*, in STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, New York, NY, USA, 2005, ACM Press, pp. 176–182.

[9] C. Chekuri, M. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour, *Polylogarithmic approximation algorithm for non-uniform multicommodity buy-at-bulk*, in Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 677–686.

[10] C. Chekuri, S. Khanna, and J. Naor, *A deterministic algorithm for the cost-distance problem*, in Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (SODA '01), Philadelphia, PA, USA, 2001, Society for Industrial and Applied Mathematics, pp. 232–233.

[11] J. Chuzhoy, A. Gupta, J. S. Naor, and A. Sinha, *On the approximability of some network design problems*, in Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '05), Philadelphia, PA, USA, 2005, Society for Industrial and Applied Mathematics, pp. 943–951.

[12] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. System Sci., 69 (2004), pp. 485–497.

[13] U. Feige, G. Kortsarz, and D. Peleg, *The dense k-subgraph problem*, Algorithmica, 29 (2001), pp. 410–421.

[14] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 302–309.

[15] N. Garg, *Saving an epsilon: a 2-approximation for the k-mst problem in graphs*, in Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC '05), New York, NY, USA, 2005, ACM Press, pp. 396–402.

[16] S. Guha, A. Meyerson, and K. Munagala, *Hierarchical placement and network design problems*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS '00), Washington, DC, USA, 2000, IEEE Computer Society, pp. 603–612.

[17] S. Guha, A. Meyerson, and K. Munagala, *A constant factor approximation for the single sink edge installation problems*, in Proceedings of the thirty-third annual ACM symposium on Theory of computing (STOC '01), New York, NY, USA, 2001, ACM Press, pp. 383–388.

[18] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden, *Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem*, in Proceedings of the 44rd Symposium on Foundations of Computer Science (FOCS '03), IEEE Computer Society, 2003, pp. 606–615.

[19] A. Gupta, A. Kumar, and T. Roughgarden, *Simpler and better approximation algorithms for network design*, in Proceedings of the thirty-fifth ACM symposium on Theory of computing (STOC '03), ACM Press, 2003, pp. 365–372.

[20] M. Hajiaghayi and K. Jain, *The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema*, in Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithms (SODA '06), Philadelphia, PA, USA, 2006, Society for Industrial and Applied Mathematics, pp. 631–640.

[21] R. Hassin, *Approximation schemes for the restricted shortest path problem*, Math. Oper. Res., 17 (1992), pp. 36–42.

[22] R. Hassin and A. Levin, *Minimum restricted diameter spanning trees*, in Approx 2002, 2002, pp. 175–184.

[23] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278. Fifth Annual ACM Symposium on the Theory of Computing (Austin, Tex., 1973).

[24] G. Kortsarz and D. Peleg, *On choosing a dense subgraph*, in Proceedings of 34th Annual Symposium on Foundations of Computer Science (FOCS'93), IEEE Computer Society, 1993, pp. 692–701.

[25] A. Kumar, A. Gupta, and T. Roughgarden, *A constant-factor approximation algorithm for the multicommodity rent-or-buy problem*, in Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS '02), Washington, DC, USA, 2002, IEEE Computer Society, pp. 333–.

[26] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III, *Bicriteria network design problems*, J. Algorithms, 28 (1998), pp. 142–171.

[27] A. Meyerson, K. Munagala, and S. Plotkin, *Cost-distance: two metric network design*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS '00), IEEE Computer Society, 2000, p. 624.

[28] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, *Approximating the single-sink link-installation problem in network design*, SIAM J. on Optimization, 11 (2000), pp. 595–610.