Approximating the Weight of Shallow Steiner Trees *

Guy Kortsarz † David Peleg ‡

December 16, 1998

Abstract

This paper deals with the problem of constructing Steiner trees of minimum weight with diameter bounded by d, spanning a given set of ν vertices in a graph. Exact solutions or logarithmic ratio approximation algorithms were known before for the cases of $d \leq 5$. Here we give a polynomial time approximation algorithm of ratio $O(\log \nu)$ for constant d, which is asymptotically optimal unless $P = NP$, and an algorithm of ratio $O(\nu^{\epsilon})$, for any fixed $0 < \epsilon < 1$, for general d.

Keywords: NP-hard problems, approximation algorithms, Steiner trees

1 Introduction

1.1 The problem

This paper considers the problem of finding low diameter Steiner trees of minimum weight. Given an n-vertex graph $G(V, E)$ and a subset $\mathcal{V} \subset V$ of ν vertices called *terminals*, we look for a tree spanning $\mathcal{V}$, with diameter bounded by d and minimum weight. We will be dealing with the case of unit length edges (i.e., where distance is measured by number of edges). Hereafter we refer to this problem as the Bounded Diameter Minimum Steiner Tree (*BDST*) problem. This problem arises in various contexts in communication network design. It has also been given some applications in the area of information retrieval in [BK90, BK91]. There, shallow trees are used to efficiently compress a collection

*An extended abstract of this work was presented at the Eighth ACM-SIAM Symp. on Discrete Algorithms, Jan. 1997, under the title "Approximating shallow-light trees".

†Department of Computer Science, The Open University of Israel, Ramat Aviv, Israel. E-mail: guyk@tavor.openu.ac.il.

‡Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: peleg@wisdom.weizmann.ac.il. Supported in part by a grant from the Israel Science Foundation and by a grant from the Israel Ministry of Science and Art.

of bits; the shorter the resulting tree, the faster the process of deciphering the message.

The *BDST* problem is clearly NP-hard (see problem ND4 in [GJ79]). In fact, it is shown in [BKP96] based on the results of [LY94, Fei96] that the problem has no better than $\ln n$ approximation ratio on an n-vertex graph. More precisely, even for the special case of a spanning tree ($\mathcal{V} = V$), for any $\epsilon > 0$, the *BDST* problem with $d = 4$ has no $\ln n - \epsilon$ approximation unless $NP \subseteq DTIME(n^{O(\log \log n)})$. This result easily extends to any fixed $d > 0$, yielding the following.

Proposition 1.1 **[BKP96]** *For any fixed $d > 0$ and any $\epsilon > 0$, the BDST problem has no $\ln n - \epsilon$ approximation unless $NP \subseteq DTIME(n^{\log \log n})$.* ∎

(This is a slight improvement over the hardness result of [MRS+98].)

Remark: It is also possible to prove a lower bound of $c \cdot \ln n$, for some $c < 1$, assuming $P \neq NP$. This follows from the hardness result in [RS97] for set cover. Therefore, one can get a lower bound with slightly smaller constant, under somewhat safer assumption.

A logarithmic-ratio approximation is given for this problem for $d = 4$ and $d = 5$ [BKP96]. Unfortunately, the approach of [BKP96] does not seem to extend to larger values of d.

The current paper introduces a different technique for handling the problem, and presents the first polynomial time approximation algorithm of ratio $O(\log \nu)$ for any *constant* d.

We also give the first algorithm for general d with ratio $O(\nu^\epsilon)$ for any fixed $0 < \epsilon < 1$. Our algorithms apply a careful combination of greedy selection and exhaustive search.

Another variant of the *BDST* problem, dealt with in earlier work [MRS+98], gives a "relaxed approximation" for the problem. An algorithm is said to have a (t, s) approximation ratio if it finds a tree whose weight is away from the optimum for trees of diameter d by a factor of at most t, but its diameter is bounded by $s \cdot d$, rather than d. Hence an approximation algorithm in the usual sense is a relaxed approximation algorithm with $s = 1$.

The algorithm presented in [MRS+98] gives a $((1+\epsilon)\log_2 \nu, 2\lceil \log_2 \nu \rceil)$ approximation for the problem. Specifically, given the bound d, a tree is produced, whose weight is only $(1+\epsilon)\log n$ away from the optimum for d, but whose diameter is bounded only by $2d \log n$, rather than by d. In fact, this result holds in a more general setting, in which an arbitrary nonnegative length $\ell(e)$ is associated with each edge e, and the diameter bound is specified in terms of these lengths. (In the unit length case, the $(1+\epsilon)$ term may be removed.) Our result can also be regarded as an improvement of [MRS+98] for fixed d, namely, we provide an $(O(\log \nu), 1)$ approximation.

We note that the constants in our approximation ratios are very large, namely, roughly $d \cdot 2^d$ in the case of fixed d, and $3^{1/\epsilon}$ in the case of general d

(and fixed ϵ). Hence for arbitrary d and ϵ, the resulting ratios are $O(d \cdot 2^d \cdot \log \nu)$ and $O(3^{1/\epsilon} \cdot \nu^{\epsilon})$. Determining the best possible constants is an intriguing open question. Following [KP97], better analysis for our algorithm for fixed d was provided in [CCC+98], establishing an improved ratio of $O(d \cdot \log \nu)$. Also, an improved $O(\nu^{\epsilon}/\epsilon^2)$ ratio algorithm was presented for general d.

1.2 Related work

Shallow-light trees were introduced and studied in [ABP90, ABP91, KRY95], as trees that simultaneously approximate well both a minimum-weight spanning tree and a shortest path tree, in the case where the weight and distance functions are identical. In these papers it is proven that there always exists a tree whose weight is away from that of the minimum spanning tree by only a constant factor, and whose diameter is away from that of the graph by only a constant factor. This can be considered as a relaxed $(O(1), O(1))$ approximation for the *BDST* problem in the special case where the bound d is $d = Diam(G)$, i.e., d is the diameter of the graph, and the weight and distance functions are identical. When the distance and weight functions are different, the existence of a tree whose weight is roughly that of the minimum spanning tree and whose diameter is roughly that of the graph is not guaranteed. In [KJ83], [KPP92] and [ZPD94] heuristics are proposed for the case of two different cost functions. These heuristics can be shown to produce bad (i.e., unbounded) ratios in some examples (cf. also [MRS+98]).

Hence the *BDST* problem can be thought of, in a sense, as a variant of the shallow-light tree problem where the weight and length functions are different. Nevertheless, it is clearly a more restrictive problem, in that the diameter of the sought tree is bounded by some specified value d (which may be much larger than the diameter) rather than by the diameter of the graph at hand.

A different problem which bears some resemblance to *BDST* is the k-MST problem, which requires finding a partial spanning tree of minimum weight among those spanning at least k nodes in the graph [RSM+94]. Yet another related tree problem is studied in [KR95].

Other kinds of greedy approximation algorithms for various NP-hard problems were extensively studied [Joh74, Lov75, Chv79, Dob82, Wol82, KP94]. The relaxed approach of multi-objective approximation was studied in [RMR+93], for the similarly structured problem of constructing a spanning tree of minimum weight among those whose *degree* is bounded by d.

Returning to the *BDST* problem, perhaps the simplest case of the *BDST* problem is where the lengths are uniform and $\nu = 2$. That is, we are given two vertices u and v, and we look for the lightest path from u to v, among the paths containing d or fewer edges. This problem is solvable in $O(E \cdot d)$ time using dynamic programming (cf. [War87, Has92, Phi93]).

However, consider the problem of $\nu = 2$ and arbitrary edge lengths. In this problem one looks for the lightest path from a vertex u to a vertex v, among

the paths of (weighted) length d or less. This seemingly simple problem is already (weakly) NP-hard. Nevertheless, this case enjoys a pseudo-polynomial algorithm, which is transformed by scaling techniques to a fully polynomial approximation scheme for the problem (cf. [War87, Has92, Phi93]).

2 Preliminaries

2.1 Formal definitions

The *BDST* problem is formally defined as follows. Given a graph $G = (V, E)$ with a rational nonnegative weight function ω on the edges, a subset $\mathcal{V} \subseteq V$ and a parameter d, we look for the minimum weight tree spanning (i.e., containing) all the terminals $\mathcal{V}$, among the trees with diameter bounded by d. (The diameter is defined as the number of edges in the longest path in the tree, i.e., the edges are thought of as having unit length.)

For any edge subset E', denote the total weight of the edges in E' by $\omega(E') = \sum_{e \in E'} \omega(e)$. Given an instance of the *BDST* problem on G, $\mathcal{V}$ and ω, denote the optimum tree for this instance by $T^*(\mathcal{V}, G, \omega)$, and its weight by $W^*(\mathcal{V}, G, \omega) = \omega(T^*(\mathcal{V}, G, \omega))$. (Later on, we occasionally omit some or all of the parameters, when no confusion arises.)

We say that Algorithm A is an approximation algorithm for the problem with approximation ratio $\rho \geq 1$, if for every instance $(\mathcal{V}, G, \omega)$ of the problem it yields a spanning tree of diameter bounded by d and total weight bounded above by $\rho \cdot W^*(\mathcal{V}, G, \omega)$.

In what follows, we assume that the required bound d on the diameter is even, and denote $q = d/2$. Consequently, the algorithms we develop view the problem as looking for a tree of height q. Nevertheless, our results can all be modified to apply to the case of odd $d = 2q + 1$. This modification is described in Section 5.

For a tree T, let $V(T)$ and $E(T)$ denote the vertex and edge sets of the tree T, respectively. For a tree T rooted at a vertex z, and a child u of z, let T_u denote the subtree of T rooted at u, and let the *expanded subtree* of u, denoted $z \circ T_u$, be the tree T_u with the root z (and hence the edge (z, u)) added.

2.2 A recursive relation

The following three relations are used later on.

$$\ln(1 + x) \leq x \ , \quad \text{for all } x > -1 \tag{1}$$

$$\ln(1 - x) \geq \frac{x}{x - 1} \ , \quad \text{for all } 0 < x < 1 \tag{2}$$

For the third recursive relation, let k, r and s be positive rational numbers, with $k \geq 1$, and let c and W be non-negative rational numbers such that $W \geq c$.

Claim 2.1 *Let f be a rational nonnegative function obeying the following recursive relation:*

$$f(t) = \begin{cases} 0, & t \leq r \\ s \cdot c + f\left(\left(1 - \frac{c}{k \cdot W}\right) t\right), & t > r. \end{cases}$$

Then

$$f(t) \;\leq\; \max\left\{0 \;,\; sW\left(1 + \frac{1}{k-1} + k \cdot \ln(t/r)\right)\right\}.$$

Proof: By induction on t. The claim is clear for $t \leq r$, so it remains to consider $t > r$. Suppose that the claim holds for every $t' < t$. Let $\gamma = 1 - \frac{c}{k \cdot W}$. As $t \geq r$ and $W \geq c$, we have that

$$\begin{aligned} 1 + \frac{1}{k-1} + k \cdot \ln(t\gamma/r) &\geq 1 + \frac{1}{k-1} + k \cdot \ln \gamma \\ &\geq 1 + \frac{1}{k-1} + k \cdot \ln\left(1 - \frac{1}{k}\right) \\ &\geq 1 + \frac{1}{k-1} + \frac{1}{1/k - 1} \;=\; 0\,. \end{aligned}$$

(The third inequality follows from Inequality (2).) Combined with the inductive hypothesis for $\gamma t < t$, we have that

$$\begin{aligned} f(t) &= s \cdot c + f(\gamma t) \;\leq\; s \cdot c + s \cdot W\left(1 + \frac{1}{k-1} + k \cdot \ln(\gamma t/r)\right) \\ &= s \cdot c + s \cdot W\left(1 + \frac{1}{k-1} + k \ln(t/r)\right) + s \cdot W \cdot k \cdot \ln \gamma\,. \end{aligned} \tag{3}$$

Now, note that $\left(1 - \frac{c}{k \cdot W}\right)^{k \cdot W} \leq e^{-c}$ by Inequality (1), and therefore $s \cdot W \cdot k \cdot \ln \gamma \leq -s \cdot c$. Combined with Inequality (3), this gives the desired result. ∎

3 An approximation for constant even d

This section presents an $O(\log \nu)$-ratio approximation algorithm for the *BDST* problem, with time complexity $\tilde{O}(|V|\Delta^{q-1}\nu^{d-1})$, for an even integer $d = 2q$, where Δ is the maximum degree of a vertex in G. (The $\tilde{O}$ notation ignores polylogarithmic factors.) Hence for every fixed d, we obtain an $O(\log \nu)$-ratio approximation algorithm with polynomial time complexity.

3.1 Overview of the algorithm

Let us now describe in more detail the main ideas behind Procedure *ST*. For the purpose of approximation, the problem is treated as a cover problem. The

algorithm iteratively finds new trees. We maintain a set $\mathcal{U} \subseteq \mathcal{V}$ of yet uncovered terminals, i.e., terminals that did not appear in any of the previously constructed trees. The method employed is a combination of greedy selection and exhaustive search. This is done via a recursive procedure ST that attempts to find new subtrees of varying depths, with a small ratio of weight-to-gain, where "gain" is measured in terms of the number of new covered terminals. When all the terminals are "covered" i.e., $\mathcal{U}$ turns empty, all the trees are merged into a single tree using Procedure `Merge`.

Let us now describe in more detail the main ideas behind Procedure ST. Consider the optimum tree T^*. This tree is rooted at some node z, and has height q (or less) and weight W^*. We may separate T^* into its subtrees $T_1, T_2, \ldots, T_k$, rooted at the children of z. Clearly, all the trees T_i are edge- and vertex-disjoint. Note that each subtree T_i contributes weight $\omega(T_i)$ to $\omega(T^*)$, but also covers (i.e., contains) $|V(T_i) \cap \mathcal{V}|$ terminals.

Call the weight-to-gain ratio of the optimum tree T^* "the optimum ratio," and denote it by $r^* = \omega(T^*)/\nu$. A simple averaging argument reveals that for (at least) one of the subtrees T_i, the expanded subtree $z \circ T_i$ has a weight-to-gain ratio no greater than the optimum ratio r^*. This tree T_i is a good candidate to search for recursively. Thus our aim will be to find, in each iteration i, a good candidate subtree T_i, having height bounded by $q-1$, and finally to merge all the found trees T_i.

Note that the sub-task of finding the best T_i is of similar nature as the one we started with, hence we can only approximate the best T_i.

3.2 Fast merging of trees

Later on, we need a procedure that given J (overlapping) trees T_i, $1 \leq i \leq J$, all rooted at the same node z, and all with height at most ℓ, constructs a tree T of height at most ℓ with $V(T) = \bigcup_i V(T_i)$ and $E(T) \subseteq \bigcup_i E(T_i)$. Note that such a tree can be easily found by combining the edge sets of all the trees into a graph G, and constructing a BFS tree rooted at z for G (cf. also [RMR+93]). From a practical point of view, however, it is worth noting that this can be done in a much simpler and faster way, as follows.

For every vertex w and tree T_j containing w, define the *level* of w in T_j, denoted $\ell_j(w)$, as w's distance from the root z in T_j. Let $e_j(e)$ be the edge joining w to its parent in T_j. Let $e(w)$ be the edge corresponding to the minimum $\ell_j(w)$. It is easily seen that in order to merge the trees properly, one only needs to select the edge $e(w)$ for all w and add it into the merged tree. Call the procedure that merges the trees (i.e., that chooses the highest entering edge for any w) Procedure `Merge`.

3.3 The approximation algorithm

We now turn to describing our algorithm. The algorithm considers every possible root v, and attempts to construct the best possible tree rooted at v. For every v, the algorithm attempts to greedily cover "many" terminals of $\mathcal{U}$, the set of uncovered terminals. Recall that $q = d/2$ represents the desired depth of the sought tree.

The algorithm constructs the tree by invoking the procedure ST. This procedure has the following parameters. The set $\mathcal{U}$ contains the yet "uncovered" terminals. The parameter δ represents a "guess" concerning the number of terminals in the subtree. The parameter ℓ represents the current height-bound of the tree, and the vertex z is a desired root. In other words, the procedure assumes the existence of a tree rooted at z with height ℓ or less, containing at least δ terminals of $\mathcal{U}$.

An important point is that given this assumption, we examine the behavior of Procedure ST once it is invoked with this parameter set to δ/q rather than δ[1]. The reason we do not attempt to cover the entire set of δ terminals is the following. The task of finding the best sub-tree with δ terminals can not be solved exactly, but rather can be approximated. If we insist on covering δ new terminals in each invocation of ST, the following undesirable phenomenon might occur. In the case of $q = 2$ (or for that matter, for $q > 2$ when the recursion reaches height bound 2) we will find a tree whose weight is greater than the optimum by a $\log n$ factor (the situation in the case $q = 2$ resembles the situation in the weighted set-cover problems, cf. [Chv79]). When $q > 2$, the $\log n$ term will propagate upwards in the recursion tree, incurring a loss of roughly $\log^{q-1} n$-ratio in the approximation, that is, we get a polylogarithmic ratio. Hence we employ a relaxed approach, by which we seek to cover only δ/q terminals, and hence we gain a tree with much reduced weight. Since q is assumed to be constant, we do not lose too much by the fact that we have covered only δ/q terminals, and this choice is beneficial overall.

Algorithm 3.1 `Alg1`*(q)*

1. *For every vertex* $v \in V$ *do:*
   (a) $\mathcal{U} \leftarrow \mathcal{V}$; $T(v) \leftarrow \emptyset$.
   (b) **while** $\mathcal{U} \neq \emptyset$ **do**
      i. $T \leftarrow ST(\mathcal{U}, |\mathcal{U}|/q, q, v)$
      ii. $\mathcal{U} \leftarrow \mathcal{U} \setminus V(T)$
      iii. *Merge* T *into* $T(v)$ *using Procedure* `Merge`.
2. *Select the tree* $T(v)$ *with lowest weight* $\omega(T(v))$.

[1] Throughout, we assume that whenever a non-integral value is passed to an integer parameter of an invoked procedure, it is properly rounded.

Recall that given a child u of the root z and a tree T_u rooted at u, the expanded subtree $z \circ T_u$ is the tree obtained by adding the vertex z and the edge (z, u) to T_u.

Procedure $ST(\mathcal{U}, \delta, \ell, z)$

1. $T \leftarrow \{z\}; \quad s \leftarrow 0$
2. If $\ell = 1$ then do:
   (a) Pick the lightest δ edges connecting z to nodes in $\mathcal{U}$ (if any), and add them to T.
   (b) Return T
3. Else ($\ell > 1$) **repeat** while $s < \delta$
   (a) For every neighbor u of z, every $0 \leq h \leq \ell - 1$ and every $1 \leq m \leq \delta$ do:
   $\mathcal{U}' \leftarrow \mathcal{U} \setminus \{z\}$
   $T(u, m, h) \leftarrow ST(\mathcal{U}', m, h, u)$
   (b) Let
   $$r \leftarrow \min_{u,m,h} \left\{ \frac{\omega(T(u,m,h)) + \omega(u,z)}{|V(z \circ T(u,m,h)) \cap \mathcal{U}|} \right\}$$
   and let (w, m, h) be the triplet attaining r.
   (c) $s \leftarrow s + |V(z \circ T(w,m,h)) \cap \mathcal{U}|$
   $\mathcal{U} \leftarrow \mathcal{U} \setminus V(z \circ T(w,m,h))$.
   (d) Merge the tree $z \circ T(w,m,h)$ into T using Procedure `Merge`.
4. Return T

Finally let us make two important comments on the presentation of the algorithm. First, the algorithm has to ignore returned trees that do not contain "enough" (i.e., the specified number of) terminals. That is, whenever Procedure ST is invoked with $\mathcal{U}$, δ, ℓ and z, we make the implicit assumption that *there does exist* a tree rooted at z with height bounded by ℓ, containing at least δ terminals of $\mathcal{U}$. This assumption is indeed always true at the outer-level invocations of the procedure from Algorithm `Alg1`, but might be false in certain internal (recursive) invocations of the procedure. Whenever the assumption is false, the number of terminals in the tree returned by ST may be smaller than the desired δ. Any such "incorrect" tree must be ignored by the procedure when selecting the best tree to be merged, in Step (3b) in Procedure ST. In particular, this may result in the procedure having no candidate trees at all to return (indicating, of course, that its guess for δ was wrong), in which case the procedure will not return any tree.

Our second comment is that one must avoid reconsidering z as a possible root of a descendent tree, within an internal recursion of ST with z as the root.

These two issues are not explicitly accounted for in the algorithm as described, for simplicity of the presentation, but must be taken care of in an actual code. Similar comments apply for later algorithms.

3.4 Analysis

We next analyze the approximation ratio of the algorithm. We adopt the following notation. Throughout, we assume a tree T with total weight W, rooted at some vertex z whose height is bounded by ℓ. Denote the children of z by $u_1, \ldots, u_k$. For $1 \leq i \leq k$, let T_i denote the subtree of T rooted at u_i. We consider an uncovered set $\mathcal{U}$ of terminals. The tree T may contain only a subset of $\mathcal{U}$. (This assumption is needed for the situation in the internal levels of the recursion.) We denote by $\mathcal{U}_j$ the set of uncovered terminals left in $\mathcal{U}$ after j iterations of the loop in Line (3) of Procedure ST. (Note that each iteration is guaranteed to remove at least some terminals from $\mathcal{U}$. This holds true because whenever ST is called with "appropriate" parameters $(\mathcal{U}', m, h, u)$, for which a corresponding tree respecting them does exist, some terminals will necessarily be covered, and the exhaustive search ensures that some "appropriate" parameters will be inspected.) In particular, $\mathcal{U}_0 = \mathcal{U}$. Finally, we denote by t_j the number of terminals of $\mathcal{U}_j$ contained in the above tree T after j iterations, namely, $|V(T) \cap \mathcal{U}_j| = t_j$.

We further assume that the initial set of terminals of $\mathcal{U} = \mathcal{U}_0$ contained in T is at least δ, namely, $t_0 = |V(T) \cap \mathcal{U}| \geq \delta$.

Lemma 3.2 *For every subset of terminals $\mathcal{U} \subseteq V$ there exists some $1 \leq i \leq k$ such that*

$$\frac{\omega(z \circ T_i)}{|V(z \circ T_i) \cap \mathcal{U}|} \leq \frac{W}{|V(T) \cap \mathcal{U}|} .$$

Proof: Let us first suppose that $z \notin \mathcal{U}$. Then since $\sum_i \omega(z \circ T_i) = W$ and $\sum_i |V(z \circ T_i) \cap \mathcal{U}| = |V(T) \cap \mathcal{U}|$, we have

$$\frac{\sum_i \omega(z \circ T_i)}{\sum_i |V(z \circ T_i) \cap \mathcal{U}|} = \frac{W}{|V(T) \cap \mathcal{U}|} .$$

Consequently, there must be some i for which this ratio is no greater than the weighted average.

The case where $z \in \mathcal{U}$ is handled almost identically. We still have $\sum_i \omega(z \circ T_i) = W$, and we use the fact that $\sum_i |V(T_i) \cap \mathcal{U}| = |V(T) \cap \mathcal{U}| - 1$. Therefore

$$\frac{\sum_i \omega(z \circ T_i)}{\sum_i |V(T_i) \cap \mathcal{U}|} = \frac{W}{|V(T) \cap \mathcal{U}| - 1} .$$

Consequently, by the same averaging argument there must be some i for which

$$\frac{\omega(z \circ T_i)}{|V(T_i) \cap \mathcal{U}|} \leq \frac{W}{|V(T) \cap \mathcal{U}| - 1} \ .$$

It follows that

$$\frac{\omega(z \circ T_i)}{|V(z \circ T_i) \cap \mathcal{U}|} \ = \ \frac{\omega(z \circ T_i)}{|V(T_i) \cap \mathcal{U}| + 1} \ \leq \ \frac{W}{|V(T) \cap \mathcal{U}|} \ .$$

(The last inequality follows since $\omega(z \circ T_i) \leq W$.) $\blacksquare$

It turns out that it is convenient to inspect the behavior of the procedure ST when run with the parameter δ/q rather than δ. Namely, we use the existence of T as above, i.e., a tree containing at least δ terminals of $\mathcal{U}$, in order to prove properties on the invocation of $ST(\mathcal{U}, \delta/q, \ell, z)$, in which ST is called to cover only δ/q new terminals, rather than δ.

In view of this, in what follows we consider each t in the range $t_0 - \delta/q \leq t \leq t_0$. Denote by $f(t)$ the cost incurred by the procedure in reducing the number of uncovered terminals from t to $t_0 - \delta/q$. Clearly, we are interested in $f(t_0)$, which is exactly the cost invested in covering δ/q terminals.

Lemma 3.3 *Each application of Procedure $ST(\mathcal{U}, \delta/q, \ell, z)$ returns a tree T_A such that*

$$|V(T_A) \cap \mathcal{U}| \ \geq \ \delta/q$$

and

$$\omega(T_A) \ \leq \ 2^{\ell-1} \cdot \left(1 + \frac{1}{q-1}\right)^{\ell-1} \cdot W \ .$$

Proof: The proof is by induction on ℓ. The claims are immediate for $\ell = 1$. Now suppose the claim holds up to $\ell - 1$ and consider ℓ. The algorithm clearly returns a tree T_A such that $|V(T_A) \cap \mathcal{U}| \geq \delta/q$, and we need to analyze the weight of T_A.

Note that initially no terminal is covered and procedure ST needs to cover δ/q terminals. Consider the beginning of the $(j+1)$st iteration of the loop in line (3) in Procedure ST. Recall that $\mathcal{U}_j$ is the current set of yet uncovered terminals after j iterations, and t_j is the number of terminals of $\mathcal{U}_j$ contained in T. Note that $t_j = t_0 - s_j$, where s_j is the number of terminals covered so far. (see line (3c) in Procedure ST). We still need to cover $\delta/q - s_j$ terminals.

By Lemma 3.2, there exists some child u_i of z such that

$$\frac{\omega(z \circ T_i)}{|V(z \circ T_i) \cap \mathcal{U}_j|} \ \leq \ \frac{W}{|V(T) \cap \mathcal{U}_j|} \ = \ \frac{W}{t_j} \ . \tag{4}$$

Let $m = |V(z \circ T_i) \cap \mathcal{U}_j|$, and let h_i be the height of T_i (recall that $h_i \leq \ell - 1$). One of the recursive applications of the procedure involves invoking

$ST(\mathcal{U}_j, m/q, h_i, u_i)$ and computing $T' = T'(u_i, m/q, h_i)$ for u_i, $m/q = |V(z \circ T_i) \cap \mathcal{U}_j|/q$ and h_i. By the inductive hypothesis of the lemma, the tree T' returned by this invocation satisfies $|V(T') \cap \mathcal{U}_j| \geq m/q$ and

$$\omega(T') \;\leq\; 2^{h_i - 1} \cdot \left(1 + \frac{1}{q-1}\right)^{h_i - 1} \cdot \omega(T_i) \tag{5}$$

By Inequality (4), the number of $\mathcal{U}_j$ nodes covered in the $(j+1)$st iteration is thus at least

$$m/q \;\geq\; \frac{\omega(z \circ T_i)}{qW} \cdot t_j \;. \tag{6}$$

In what follows, let e_i denote the edge connecting z to u_i. Recall that t_0 denotes the initial number of terminals of $\mathcal{U} = \mathcal{U}_0$ in T, and $t_0 \geq \delta$ by assumption. Thus the cost, $f(t_j)$ of reducing the number of uncovered terminals from t_j to $t_0 - \delta/q$ obeys the following recursive relation:

$$f(t_j) \;=\; 0 \qquad \text{for} \;\; t_j \leq t_0 - \delta/q \;.$$

In addition, Inequalities (5) and (6) guarantee that for $t_j > t_0 - \delta/q$,

$$\begin{aligned} f(t_j) \;&\leq\; 2^{\ell-2} \cdot \left(1 + \frac{1}{q-1}\right)^{\ell-2} \cdot \omega(T_i) + \omega(e_i) + f\left(\left(1 - \frac{\omega(z \circ T_i)}{qW}\right) \cdot t_j\right) \qquad (7)\\ &\leq\; 2^{\ell-2} \left(1 + \frac{1}{q-1}\right)^{\ell-2} \cdot \omega(z \circ T_i) + f\left(\left(1 - \frac{\omega(z \circ T_i)}{qW}\right) \cdot t_j\right). \end{aligned}$$

Denoting $c = \omega(z \circ T_i)$, $k = q$, $s = 2^{\ell-2}\left(1 + \frac{1}{q-1}\right)^{\ell-2}$ and $r = t_0 - \delta/q$, the above relation takes the form of Relation 2.1. Hence by Claim 2.1, this recurrence solves to give

$$f(t_j) \;\leq\; 2^{\ell-2} \cdot W \cdot \left(1 + \frac{1}{q-1}\right)^{\ell-2} \cdot \left(1 + \frac{1}{q-1} + q \ln\left(\frac{t_j}{t_0 - \delta/q}\right)\right).$$

Note that, since $t_0 \geq \delta$,

$$\frac{t_0}{t_0 - \delta/q} \;\leq\; \frac{1}{1 - 1/q} \;=\; 1 + \frac{1}{q-1} \;,$$

and therefore

$$q \cdot \ln\left(\frac{t_j}{t_0 - \delta/q}\right) \;\leq\; q \cdot \ln\left(1 + \frac{1}{q-1}\right) \;\leq\; q \cdot \frac{1}{q-1} \;=\; 1 + \frac{1}{q-1} \;.$$

(The first inequality follows from the fact that $t_j \leq t_0$, the second from Inequality (1).) Hence

$$f(t_0) \;\leq\; 2^{\ell-2} \cdot W \cdot \left(1 + \frac{1}{q-1}\right)^{\ell-2} \cdot 2 \cdot \left(1 + \frac{1}{q-1}\right)$$

completing the proof. ∎

Theorem 3.4 *Algorithm* `Alg1` *produces a spanning tree with height at most* q *and total weight bounded by* $eq \cdot 2^{q-1} \ln \nu \cdot W^*$.

Proof: Let v^* be the root of the optimal tree T^*. Note that Procedure ST is called to cover only $|\mathcal{U}|/q$ terminals, rather than $|\mathcal{U}|$. This agrees with the conditions of Lemma 3.3.

Hence by Lemma 3.3, when the algorithm tests for trees rooted at v^*, each invocation of Procedure ST (within Algorithm `Alg1`) when the set $\mathcal{U}$ of nodes remaining to be covered is of size m, will return a tree T such that $|V(T) \cap \mathcal{U}| \geq m/q$ and $\omega(T) \leq 2^{q-1} \cdot (1+1/(q-1))^{q-1} W^* \leq 2^{q-1} e W^*$. Thus $q \ln \nu$ consecutive applications of Procedure ST (within Algorithm `Alg1`) will reduce the number of remaining uncovered nodes from the initial $\mathcal{V}$ to $(1-1/q)^{q \ln \nu} \nu < 1$ at a total cost of at most $e \cdot 2^{q-1} q \ln \nu \cdot W^*$. ∎

3.5 The time complexity

Let Δ denote the largest degree in the graph. Let us denote by $T(\ell)$ the maximum possible time taken by Procedure ST when executed with ℓ as the height bound.

We first analyze the time complexity of Procedure ST. As a first step, let us count the number of recursive calls executed each time the iteration (3a) in line (3) of Procedure ST is performed. A recursive call is performed in Line (3a) for every neighbor u of z and numbers m and h, for $1 \leq m \leq \nu$, $0 \leq h \leq q-1$. Thus the number of recursive calls is bounded by $\Delta \cdot \nu \cdot q$.

Next, let us observe that each iteration of line (3) covers at least one new terminal (and actually, it sometimes covers many more than one). It follows that the loop of Line (3) is iterated at most ν times. Therefore, the total number of recursive calls with $q-1$ performed by the algorithm is bounded by $\Delta \cdot \nu^2 q$.

Finally, note that for the merging procedure we only have to remember for every vertex its highest entering edge in any tree containing v to be merged. Thus the tree merging operations do not significantly affect the running time.

Overall, we have the following recursive relation for the time complexity of ST. First, $T(1) = O(\nu \log \nu)$, as the edges corresponding to the terminals of $\mathcal{V}$ need to be sorted. For $q > 1$,

$$T(q) = O(\Delta \cdot \nu^2 q) \cdot T(q-1),$$

which gives $T(q) = \tilde{O}((\Delta \cdot q)^{q-1} \nu^{d-1})$.

Now, let us analyze the time complexity of Algorithm `Alg1`. The algorithm invokes procedure ST for every vertex v as a candidate root. Each such invocation causes $O(q \ln \nu)$ additional recursive calls to ST. Thus in summary, we have $T(q) = \tilde{O}(q \cdot n \cdot (\Delta \cdot q)^{q-1} \nu^{d-1})$. This running time is polynomial for every fixed d. Note that the algorithm has better running time on graphs with low maximum degree.

As we shall see in Sect. 5, in case d is odd the time complexity is increased by an extra factor of $|E|$.

4 An approximation for the general case

In this section we give a polynomial-time $O(\mu_\epsilon)$-ratio approximation algorithm for the *BDST* problem, where $\mu_\epsilon = \nu^\epsilon$, for any fixed parameter $0 < \epsilon < 1$. Again, we assume d to be an even integer, $d = 2q$.

4.1 Overview of the algorithm

For the approximation algorithm, the problem is again treated as a cover problem. The algorithm considers every possible root u, and constructs a tree $T(u)$ rooted at it. It then picks the best of those trees as its output.

Let u be the "correct" root. The main idea in Procedure `Proc1` is the following. Given some fixed $\epsilon > 0$, the optimum tree T^* can be decomposed into roughly $\mu_\epsilon = \nu^\epsilon$ *edge disjoint* trees. By averaging one sees that at least one of these trees must have low weight. Therefore, the recursion is aimed at finding such a tree T_i, rooted at some vertex w. This tree would contain roughly $\nu^{1-\epsilon}$ terminals of $\mathcal{V}$ and weight roughly W^*/μ_ϵ. As will become clear, we are not able to find this exact tree T_i. Rather, we find a tree T' with at least $\epsilon \cdot \nu^{1-\epsilon}$ terminals, whose weight is $O(W^*)$, which implies an $O(\mu_\epsilon)$ to the weight of T_i. Repeated applications of this idea yield an $O(\mu_\epsilon \cdot \log \nu)$ approximation to the weight of T^*.

One critical issue that the procedure has to take care of is the height bound q on the resulting tree $T(u)$. We do not know a priori the "correct" height h' of the good tree T_i rooted at w. Hence, we search exhaustively for h' and w. Given a parameter k, Procedure `Proc1` tries recursively, for every $1 \leq h' \leq h$ and $w \neq u$, the possibility of covering roughly k/μ_ϵ terminals of $\mathcal{U}$ using a tree of height at most h' rooted at w. We then connect w to u by the lightest path among the paths of $h - h'$ or fewer edges from u to w (to get a combined tree of height h). Finally, the best tree among all pairs w and h' is returned by the procedure.

One problem with this approach is that we have no good bound on the weight of the path connecting u to the various chosen w. In particular, the paths found in the course of the procedure are *not* edge disjoint. The only recourse we have is to use the optimum weight W^* as a bound on the path weight.

Our main algorithm, denoted Algorithm `Alg2`, applies the above approach in iterations, invoking Procedure `Proc1` for $O(\mu_\epsilon \log \nu)$ times in order to cover all the ν terminals of $\mathcal{V}$. This algorithm, given in Subsection 4.3, yields an approximation ratio of roughly $\frac{3^{1/\epsilon}}{\epsilon^2}\nu^\epsilon$ for *BDST*. Since we deal with fixed ϵ, this is an $O(\nu^\epsilon)$ ratio approximation for the problem.

4.2 Spanning few leaves using low weight

The main tool used in the approximation is a procedure SP producing a relatively light tree containing a (presumably small) subset of the set $\mathcal{U}$ of unspanned terminals. Procedure SP greedily combines short paths with low weight leading to the terminals of $\mathcal{U}$. Let t denote the desired number of spanned terminals of $\mathcal{U}$ that should appear in the tree. Let u be a chosen root, and let h be a bound on the height. The procedure computes the light-short path leading to each terminal in $\mathcal{U}$ and combines the t lightest of these paths into a shallow, and relatively light tree. Procedure SP has a good performance ratio when t is small.

Algorithm 4.1 *SP(u,h,$\mathcal{U}$,t)*

1. *Compute for every terminal $w \in \mathcal{U}$, $w \neq u$, the weighted distance $dist_h(u,w)$ from u to w, namely, the weight of the lightest path from u to w, among the paths containing h or fewer edges.*
2. *Choose the t terminals in $\mathcal{U}$ with minimum $dist_h(u,w)$.*
3. *Merge these paths into a tree T using Procedure* `Merge` *and return T.*

Note that the problem that needs to be solved by the first step of the procedure is polynomial, as mentioned in Sect. 1.2. We discuss its time complexity later on.

4.3 The approximation algorithm for general d

We now describe procedure `Proc1`. This procedure gets as parameters a given root u, a subset $\mathcal{U}$ of the initial Steiner set $\mathcal{V}$ of unspanned terminals, a height bound h and a number t. The procedure is designed to construct a tree covering ϵt of the terminals of $\mathcal{U}$, for a fixed $0 < \epsilon < 1$. The procedure also uses the parameter $\mu_\epsilon = \nu^\epsilon$ (which is fixed throughout all invocations of the procedure, and depends only on the size of the *initial* set $\mathcal{V}$).

Algorithm 4.2 `Proc1`*(u, $\mathcal{U}$, h, t)*

1. *If $t \leq 3 \cdot \mu_\epsilon$, then call $T(u) \leftarrow SP(u,\mathcal{U},h,t)$ and return $T(u)$.*
2. */* $t > 3\mu_\epsilon$ */*
   $T(u) \leftarrow \emptyset$, $sp \leftarrow 0$, $t_0 \leftarrow t$.
3. **While** $sp < (t_0\epsilon)$ **do**
   (a) **For** *every vertex $w \neq u$ and every $1 \leq h' \leq h$, and for $w = u$ and $h' = h$,* **do:**
      i. $\mathcal{U}' \leftarrow \mathcal{U}$.

ii. *Call* $T(w,h') \leftarrow$ `Proc1`$(w, \mathcal{U}', h', t/(3\mu_\epsilon))$.

iii. *Compute the lightest path* $P_{h'}(u,w)$ *from* u *to* w *among the paths containing* $h - h'$ *or fewer edges.*

iv. *Let* $\mathcal{C}(w,h') = \omega(P_{h'}(u,w)) + \omega(T(w,h'))$ *be the "combined tree-path weight" for the pair* (w,h').

(b) *Let* $(z,\hat{h})$ *be the pair minimizing the "combined tree-path weight"* $\mathcal{C}(z,\hat{h})$.
Merge the edges of $T(z,\hat{h})$, $P_{\hat{h}}(u,z)$ *and* $T(u)$ *using procedure* `Merge`.

(c) $sp \leftarrow sp + |(\mathcal{U} \cap V(T(z,\hat{h})))|$
$\mathcal{U} \leftarrow \mathcal{U} \setminus V(T(z,\hat{h}))$
$t \leftarrow t - |(\mathcal{U} \cap V(T(z,\hat{h})))|$

4. *Return the resulting tree* $T(u)$.

Finally we describe the main algorithm, `Alg2`, that iteratively applies Procedure `Proc1` to cover all the terminals of $\mathcal{V}$.

Algorithm 4.3 `Alg2`$(\mathcal{V},q)$

1. **For** *every* $u \in V$ **do:**

(a) $T(u) \leftarrow \emptyset$, $\mathcal{U} \leftarrow \mathcal{V}$.

(b) **While** $\mathcal{U} \neq \emptyset$ **do***:*

i. *Call* `Proc1`$(u, \mathcal{U}, q, |\mathcal{U}|)$ *and let* $T'(u)$ *denote the tree returned by the call*

ii. $\mathcal{U} \leftarrow \mathcal{U} \setminus V(T'(u))$.

iii. *Merge* $T'(u)$ *into* $T(u)$ *using Procedure* `Merge`.

2. *Choose the best tree* $T(u)$

4.4 Analysis

We next analyze the approximation ratio of the algorithm. First, consider Procedure `Proc1`. Throughout, we apply the following conventions. Let $T^*(u,\mathcal{U},t,h)$ denote the minimum weight tree among the trees of height h or less rooted at u and spanning at least t terminals of $\mathcal{U}$, and let $W^*(u,\mathcal{U},t,h)$ denote its weight.

Recall that $\mu_\epsilon = \nu^\epsilon$. For every $t > 0$ define

$$\alpha(t,\epsilon) \;=\; \max\left\{\frac{\log t}{\log \mu_\epsilon}, 1\right\}.$$

(We denote this quantity by α, whenever no confusion may arise.) Note that $\alpha \geq 1$ by definition, and that for $t = \nu$ (which is the largest value of t used)

we have $\alpha = 1/\epsilon$. Hence $1 \leq \alpha(t,\epsilon) \leq 1/\epsilon$. The quantity $\alpha(t,\epsilon)$ represents (roughly) the height of the recursion tree in Procedure `Proc1` when called with t. For example, consider the case of $t = \nu$. The procedure starts with ν uncovered terminals. In the first recursive call Procedure `Proc1` is required to cover roughly $\nu^{1-\epsilon}$ terminals of $\mathcal{V}$. Then in the second internal level of the recursion the procedure has to cover roughly $\nu^{1-2\epsilon}$ terminals, and so on.

Let us define

$$f(\epsilon,t,u,\mathcal{U},h) \;=\; \mu_\epsilon \cdot \alpha(t,\epsilon) \cdot \left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)+1} \cdot W^*(u,\mathcal{U},t,h).$$

In what follows we prove that Procedure `Proc1`, when invoked with the parameters $(u,\mathcal{U},h,t)$, yields a tree of height bounded by h, containing at least $\epsilon \cdot t$ terminals of $\mathcal{U}$, and whose weight is no more than $f(\epsilon,t,u,\mathcal{U},h)$. As said earlier, the main term in this resulting approximation ratio is the term $\mu_\epsilon = \nu^\epsilon$. The other terms are fixed for fixed ϵ.

Before we prove the above claim, we need to analyze the approximation ratio of Procedure SP.

Claim 4.4 *Invoked with parameters u, $\mathcal{U}$, t and h, Procedure SP finds a tree T_{SP} containing at least t terminals of $\mathcal{U}$, with weight $\omega(T_{SP}) \leq t \cdot W^*(u,\mathcal{U},t,h)$.*

Proof: We may assume that all the leaves of $T^*(u,\mathcal{U},t,h)$ belong to $\mathcal{U}$ (for otherwise we may discard the ones that are not in $\mathcal{U}$) and that the number of leaves in the tree is bounded by t. Let S^* denote the sum of the edge-weights of all root-to-leaf paths in T^*. Since every edge is counted no more than t times in this sum, we have $S^* \leq t \cdot W^*(u,\mathcal{U},t,h)$. Now consider the respective sum, S_{SP} for T_{SP}. By the greedy rule used to choose this tree in Procedure SP, it is easily seen that $S_{SP} \leq S^*$. Since the weight of T_{SP} is no greater than S_{SP}, the claim follows. ∎

Next, we rely on the following tree decomposition lemma.

Lemma 4.5 *Consider a set $\mathcal{U} \subseteq \mathcal{V}$ and a spanning tree T of height h rooted at some vertex u, containing all the terminals $\mathcal{U}$. Then for every $j < |\mathcal{U}|$, the tree T contains a subtree T_j with at least $j/3$ and no more than j terminals of $\mathcal{U}$, and weight bounded by $j \cdot \omega(T)/|\mathcal{U}|$.*

Proof: Let us start by proving that T contains at least $|\mathcal{U}|/j$ *edge-disjoint* trees T_i, each T_i containing $\Theta(j)$ terminals of $\mathcal{U}$. To prove that result, we first claim that there exists a vertex $w \in T$ with the following properties:

1. The subtree $T(w)$ of T rooted at w contains more than $j/3$ terminals of $\mathcal{U}$, and
2. each subtree of $T(w)$ rooted at a child of w contains fewer than $j/3$ terminals of $\mathcal{U}$.

Such a vertex w can be found by starting at u and walking down the tree. Specifically, if u has a child z with more than $j/3$ terminals of $\mathcal{U}$ in its subtree $T(z)$, we walk to this child. This process is repeated until the desired vertex w is found.

We now find a tree T_1 with $j/3 \leq |V(T_1) \cap \mathcal{U}| \leq 2j/3$ terminals of $\mathcal{U}$ (rooted at w) as follows. We start with $T_1 = \emptyset$. Iteratively, add to T_1 the subtrees rooted at the children of w one by one, in an arbitrary order, until the first time that the total number of terminals from $\mathcal{U}$ in T_1 is at least $j/3$. By the above Property (2), the last subtree of w added has fewer than $j/3$ terminals of $\mathcal{U}$, which implies that the resulting tree T_1 satisfies $j/3 \leq |V(T_1) \cap \mathcal{U}| \leq 2j/3$.

To get the trees $T_2, T_3, \ldots$, and so on, we just extract all the edges and vertices of $T_1 \setminus \{w\}$ from T (leaving w itself in T), and repeat the entire process. Consider the first time the number of terminals in the remaining tree drops below j. By the above considerations it follows that the last tree contains no more than j but *more than* $j/3$ terminals, as claimed. As all the trees but the last one contain between $j/3$ and $2j/3$ terminals, it follows that the number of trees produced is greater than $|\mathcal{U}|/j$.

The lemma now follows by an averaging argument, based on the edge-disjointness of the trees T_i. ∎

We are now ready to prove the main lemma.

Lemma 4.6 *Procedure* `Proc1` *produces a tree containing at least* $t \cdot \epsilon$ *terminals of* $\mathcal{U}$*, whose weight is no greater than* $f(\epsilon, t, u, \mathcal{U}, h) = \mu_\epsilon \cdot \alpha(t, \epsilon) \cdot \left(\frac{3}{1-\epsilon}\right)^{\alpha+1} \cdot W^*(u, \mathcal{U}, t, h)$.

Proof: The lemma is proved by induction on t. For $t \leq 3 \cdot \mu_\epsilon$, the algorithm runs procedure SP. The desired ratio follows from Claim 4.4 since SP has an approximation ratio of t.

We now assume the claim holds for every $t' < t$, and prove it for t. First, recall that every recursive call to `Proc1` (see line (3(a)ii) in `Proc1`) involves spanning only $t' = t/3\mu_\epsilon$ terminals. Also note that $t \leq |\mathcal{U}|$. Using Lemma 4.5 with $j = 3t'$, we get that there exists a subtree $T(w, h')$ rooted at some vertex w and containing at least t' terminals of $\mathcal{U}$, with height h' and weight bounded by

$$W^*(w, \mathcal{U}, h', t') \;\leq\; W^*(u, \mathcal{U}, h, t)/\mu_\epsilon. \tag{8}$$

In line (3a) of Procedure `Proc1`, all the possible choices for the height bound h' and roots w are checked, including, in particular, the "correct" pair h', w described in Equation (8).

Consider the combined tree-path weight $\mathcal{C}(w, h')$ produced for this pair h', w in Step (3(a)iv) of Procedure `Proc1`. This weight is composed of the weights of the returned tree $T(w, h')$ and the path $P_{h'}(u, w)$ connecting u and w. We bound these weights separately.

First, in what follows we prove that the weight of $T(w,h')$ is bounded by

$$\begin{aligned}\omega(T(w,h')) &\leq \alpha(t,\epsilon)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)}\cdot W^*(u,\mathcal{U},t,h) - W^*(u,\mathcal{U},t,h)\\ &= \frac{(1-\epsilon)f(\epsilon,t,u,\mathcal{U},h)}{3\mu_\epsilon} - W^*(u,\mathcal{U},t,h)\ . \qquad (9)\end{aligned}$$

We divide the proof into two cases. We start with the main case in which $t' > \mu_\epsilon$, and therefore by definition $\alpha(t',\epsilon) > 1$. This implies, by definition again, that $\alpha(t',\epsilon) < \alpha(t,\epsilon) - 1$.

Now, the induction hypothesis implies that the weight of the produced tree is bounded by

$$f(\epsilon,t',w,\mathcal{U},h') \leq \mu_\epsilon\cdot\alpha(t',\epsilon)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t',\epsilon)+1}\cdot W^*(w,\mathcal{U},t',h')\ .$$

Since $\alpha(t',\epsilon) < \alpha(t,\epsilon) - 1$,

$$\begin{aligned}f(\epsilon,t',w,\mathcal{U},h') &\leq \mu_\epsilon\cdot(\alpha(t,\epsilon)-1)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)}\cdot W^*(w,\mathcal{U},t',h')\\ &\leq \frac{1-\epsilon}{3}\left(\mu_\epsilon\cdot(\alpha(t,\epsilon)-1)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)+1}\right)\cdot\frac{W^*(u,\mathcal{U},h,t)}{\mu_\epsilon}\ .\end{aligned}$$

The last inequality follows from Eq. (8). Hence

$$\begin{aligned}f(\epsilon,t',w,\mathcal{U},h') &\leq \frac{1-\epsilon}{3}\left((\alpha(t,\epsilon)-1)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)+1}\right)\cdot W^*(u,\mathcal{U},h,t)\\ &\leq \frac{1-\epsilon}{3}\alpha(t,\epsilon)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)+1}\cdot W^*(u,\mathcal{U},h,t) - \left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)}\cdot W^*(u,\mathcal{U},h,t).\end{aligned}$$

Now, recalling that $\alpha(t,\epsilon) > 1$ we have:

$$f(\epsilon,t',w,\mathcal{U},h') \leq \frac{(1-\epsilon)f(\epsilon,t,u,\mathcal{U},h)}{3\mu_\epsilon} - W^*(u,\mathcal{U},h,t)\ ,$$

as required in Inequality (9).

Finally, we inspect the simpler case, in which $t' \leq \mu_\epsilon$. In this case, by Claim 4.4 the invocation with t' returns a tree $T(w,h')$ of weight no larger than $t'\cdot W^*(w,\mathcal{U},h',t')$. By Inequality (8) we get that this weight is no larger than $W^*(u,\mathcal{U},h,t)$. Now, we get again that

$$W^*(u,\mathcal{U},t,h) \leq \alpha(t,\epsilon)\cdot\left(\frac{3}{1-\epsilon}\right)^{\alpha(t,\epsilon)}\cdot W^*(u,\mathcal{U},t,h) - W^*(u,\mathcal{U},t,h)\ ,$$

as required by Inequality (9). This last inequality again follows since $\alpha(t,\epsilon) > 1$.

In both cases, we get that the bound in Inequality (9) holds for the weight $T(w,h')$. Now we add the weight of $P_{h'}(u,w)$ to this bound. Clearly the weight of $P_{h'}(u,w)$ is no more than W^*, and thus the total combined weight is bounded by $(1-\epsilon)f(\epsilon,t,u,\mathcal{U},h)/3\mu_\epsilon$. Note that in line (3b), Procedure `Proc1` picks *the best* pair. Thus, its corresponding combined path-tree weight is no greater than the above bound.

In summary, each iteration of Line 3 of Procedure `Proc1` covers at least $t\epsilon/(3\cdot\mu_\epsilon)$ terminals of $\mathcal{U}$ using weight bounded by $(1-\epsilon)f(\epsilon,t,u,\mathcal{U},h)/3\mu_\epsilon$.

In order to bound the total weight invested by procedure `Proc1` throughout the entire collection of iterations of Line 3 of `Proc1`, we need to know how many iterations of Line 3 have occurred. Recall that *sp* represents the number of terminals covered by `Proc1` so far (see Line 3c in `Proc1`). The number of covered terminals, *sp*, must reach $\epsilon\cdot t_0$. We now analyze the number of required iterations.

To bound the number of iterations, we write the following recursive relation for the number R_i of yet uncovered terminals left after i iterations.

$$R_0 \;=\; t_0, \qquad \text{and} \qquad R_i \;\leq\; t_0\left(1-\frac{\epsilon}{3\mu_\epsilon}\right)^i \quad \text{for} \quad i>0. \tag{10}$$

The inequality for $i > 0$ follows from the fact that each recursive call to `Proc1` is with $t' = t/(3\mu_\epsilon)$, and each such call covers $\epsilon t'$ terminals. We look for the smallest i such that

$$R_i \;\leq\; t_0(1-\epsilon)\;, \tag{11}$$

indicating that at least $\epsilon\cdot t_0$ terminals were covered.

It turns out that fixing $i = 3\mu_\epsilon/(1-\epsilon)$ satisfies the desired Inequality (11). To see this, we note that by Inequality (10), it suffices to prove that for this choice of i,

$$\left(1-\frac{\epsilon}{3\mu_\epsilon}\right)^{3\mu_\epsilon/(1-\epsilon)} \;\leq\; 1-\epsilon\;.$$

Taking logs from both sides, it remains to prove that for any $k > \epsilon$,

$$\frac{k}{1-\epsilon}\cdot\ln\left(1-\frac{\epsilon}{k}\right) \;\leq\; \ln(1-\epsilon)\,.$$

Using Inequality (1) on the left-hand side, we only need to show that

$$\frac{-\epsilon}{1-\epsilon} \;\leq\; \ln(1-\epsilon)\;,$$

or

$$\frac{\epsilon}{1-\epsilon} \;\geq\; \ln\left(\frac{1}{1-\epsilon}\right) \;=\; \ln\left(1+\frac{\epsilon}{1-\epsilon}\right),$$

which follows using Inequality (1) once again.

The total number of iterations is therefore bounded by $3\mu_\epsilon/(1-\epsilon)$. Combined with the fact that in each iteration of Line 3 the weight added is at most $(1-\epsilon)f(\epsilon,t,u,\mathcal{U},h)/3\mu_\epsilon$, the lemma follows. ∎

The following theorem is proved similarly to Theorem 3.4.

Theorem 4.7 *For every fixed ϵ, Algorithm* `Alg2` *returns a proper tree with weight bounded by $O(\nu^\epsilon \log\nu \cdot W^*)$.* ∎

4.5 The time complexity

Let $|V| = n$, $|E| = m$. As a first step, we find for every pair of vertices s and t, and every height bound $h \leq q$, the lightest path from s to t among the paths containing h or fewer edges. This takes $O(n^2 \cdot m \cdot q)$ time, since we have n^2 pairs of vertices, and for each pair it takes $m \cdot q$ time units to find the appropriate collection of "short paths". This procedure takes care of the information needed in line (3(a)iii) in `Proc1`.

Next, for every vertex u and height bound h, we sort the paths leading from u to the terminals of $\mathcal{V}$ by increasing weights, and store them in a one dimensional array corresponding to u and h. This takes care of the information needed in Procedure SP, and requires $\tilde{O}(n^2 \cdot q)$ time.

Again, the leading factor in the time complexity comes from the recursion. Let $T(\mathcal{U})$ denote the time complexity of `Proc1` when run with $\mathcal{U}$ (putting aside for the moment the $\tilde{O}(n^2 \cdot q)$ term). The running time of Procedure `Proc1` is given by the following. For $|\mathcal{U}| \leq \mu_\epsilon$, $T(\mathcal{U}) = O(n)$, as we simply pick the first t lightest paths in the (sorted) array corresponding to u and h, leading to unspanned terminals. Now, $T(\mathcal{U}) = O(n \cdot q \cdot \mu_\epsilon)T(\mathcal{U}^{1-\epsilon})$. This follows since the number of iterations of line (3) in Procedure `Proc1` is bounded by $O(\mu_\epsilon)$ (see the discussion in the previous section). Now, in each such iteration, there is a recursive call for every $0 \leq h' \leq q$ and w.

Therefore, the running time is bounded by

$$T(\mathcal{V}) \;\leq\; \tilde{O}(n^{1/\epsilon-1} \cdot q^{1/\epsilon-1} \cdot \mu_\epsilon^{1/\epsilon-1}) \cdot T(\mu_\epsilon) \;=\; \tilde{O}(n^{1/\epsilon} \cdot q^{1/\epsilon-1} \cdot \nu^{1-\epsilon}) \;.$$

In summary, the running time of the main algorithm, `Alg2`, is $\tilde{O}(n^{1/\epsilon+1} \cdot q^{1/\epsilon-1} \cdot \nu + n^2 \cdot m \cdot q)$. This ensures polynomial running time for every *fixed* ϵ.

Again, as we shall immediately see, in case d is odd the time complexity is increased by an extra factor of $|E|$.

5 Handling the odd-diameter case

Let us now show how our solutions can be modified to handle an *odd* diameter bound $d = 2q + 1$. In a tree of diameter $2q + 1$, there are two adjacent centers

(in the graph-theoretic sense). These are the two end-vertices v and w of the *middle edge* $e = (v, w)$ in any path of length $2q + 1$ in the tree. Assuming we know this edge, it is possible to transform the problem of finding the lightest tree of diameter $2q + 1$ in the graph G into an equivalent problem of finding the lightest tree of diameter $2q$ in a modified graph G', constructed as follows. First, contract the two vertices v and w into a single combined vertex u of degree $deg(v) + deg(w) - 2$. Then, assign each edge (z, u) the minimum of the two weights $\omega(z, v)$ and $\omega(z, w)$. It is clear that the two problems are equivalent, and the solution T for G can be obtained from the solution T' of G' by de-contracting the node u appropriately (linking a neighbor z of u to v if $\omega(z, v) \leq \omega(z, w)$ and to w otherwise). Note that we add $\omega(e) = \omega(v, w)$ to the weight of the tree.

Moreover, one can verify that approximability of the *BDST* problem with $d = 2q$ on G' directly translates into approximability of the *BDST* problem on G with $d = 2q + 1$ with the same ratio (or better).

Thus, the following simple procedure translates an approximation algorithm for the even case into one for the odd case with the same ratio (or better). Go over the edges one by one. For each edge $e_i = (v_i, w_i)$ contract the edge e_i and get a combined vertex u_i. Approximate the *BDST* problem with $d = 2q$ and u_i as the chosen root, using the algorithm for the even case. Let T_i be the resulting tree in the approximation. Compute the sum $S_i = \omega(T_i) + \omega(e_i)$. Let j be the index achieving the minimum for this sum. De-contract e_j appropriately, and return the resulting tree. This increases the running time by a factor of $|E|$.

6 Discussion

The approximation algorithm for fixed d is optimal up to constants, due to the logarithmic lower bound on the approximation ratio. However, it is not known whether it is possible to get rid of the dependency on d in the approximation ratio of the algorithm (which was recently improved to $O(d \log \nu)$ [CCC+98]). Furthermore, it may certainly be possible to improve the running time.

For general d, there is still a large gap between our lower and upper bounds. It is interesting to note that *BDST* admits an $O(\log^2 \nu)$ ratio algorithm that runs in slightly super-polynomial time, namely in time $n^{O(\log n)}$. Getting a logarithmic (or at least polylogarithmic) approximation algorithm for *BDST* is still an open problem. Given the existence of a slightly super-polynomial time, polylogarithmic approximation ratio algorithm, it is our feeling that it is likely that there exists a *polynomial* time, polylogarithmic-ratio approximation algorithm for the *BDST* problem.

Acknowledgment

Thanks are due to two anonymous referees, whose remarks have helped to significantly improve the presentation of this paper.

References

[ABP90] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 177–187, 1990.

[ABP91] B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light-weight spanners. Unpublished Manuscript, 1991.

[BK90] A. Bookstein and S.T. Klein. Construction of optimal graphs for bit-vector compression. In *Proc. 13th ACM SIGIR Conference*, pages 327–342, 1990.

[BK91] A. Bookstein and S.T. Klein. Compression of correlated bit-vectors. *Information Systems*, 16:387–400, 1991.

[BKP96] J. Bar-Ilan, G. Kortsarz, and D. Peleg. Generalized submodular cover problems and applications. In *Proc. 4th Israel Symp. on Theory of Computing and Systems*, pages 110–118, 1996.

[CCC+98] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms*, pages 192–200, 1998.

[Chv79] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4:233–235, 1979.

[Dob82] G. Dobson. Worst case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, 7:515–531, 1982.

[Fei96] U. Feige. A threshold of $\ln n$ for approximating set cover. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 314–318, 1996.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[Has92] R. Hassin. Approximation schemes for restricted shortest path problems. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. of computer and system sciences*, 9:256–278, 1974.

[KJ83] B. Kadaba and J. Jaffe. Routing to multiple destinations in computer networks. In *IEEE Trans. on Communication*, pages 343–351, 1983.

[KP94] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *J. Algorithms*, pages 222–236, 1994.

[KP97] G. Kortsarz and D. Peleg. Approximating shallow-light trees. In *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, pages 173–182, 1997.

[KPP92] V.P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicasting for multimedia applications. In *Proc. IEEE INFOCOM*, 1992.

[KR95] P. N. Klein and R. Ravi. A nearly best possible approximation for node-weighted Steiner trees. *Journal of Algorithms*, 19:104–115, 1995.

[KRY95] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning and shortest paths trees. *Algorithmica*, 14:305–321, 1995.

[Lov75] L. Lovász. On the ratio of integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. of the ACM*, 41:960–981, 1994.

[MRS+98] Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Bicriteria network design problems. *J. of Algorithms*, 28:142–171, 1998.

[Phi93] C. Philips. The network inhibition problem. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 776–785, 1993.

[RMR+93] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt. Many birds with one stone: Multi-objective approximation algorithms. In *Proc. 25th ACM Symp. on The Theory of Computing*, pages 438–447, 1993.

[RS97] R. Raz and S. Safra. A sub constant error probability low degree test, and a sub constant error probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 475–484, 1997.

[RSM+94] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrantz, and S.S. Ravi. Spanning trees short or small. In *The 5th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 546–555, January 1994.

[War87] A. Warburton. Approximating of pareto optima in multy-objective, shortest path problems. *Operations Research*, 35:70–79, 1987.

[Wol82] L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:385–393, 1982.

[ZPD94] Q. Zhu, M. Parsa, and W.W.M. Dai. An iterative approach for delay-bounded minimum Steiner tree construction. Technical Report UCSC-CRL-94-39, UC Santa Cruz, 1994.