

# Near-linear Time Algorithms for Approximate Minimum Degree Spanning Trees

Ran Duan, Haoqing He, and Tianyi Zhang

Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

duanran@mail.tsinghua.edu.cn

{hehq13,tianyi-z16}@mails.tsinghua.edu.cn

**Abstract.** Given a graph  $G = (V, E)$ , we wish to compute a spanning tree whose maximum vertex degree, i.e. tree degree, is as small as possible. Computing the exact optimal solution is known to be NP-hard, since it generalizes the Hamiltonian path problem. For the approximation version of this problem, a  $\tilde{O}(mn)$  time algorithm that computes a spanning tree of degree at most  $\Delta^* + 1$  is previously known [Fürer & Raghavachari 1994]; here  $\Delta^*$  denotes the minimum tree degree of all the spanning trees. In this paper we give the first near-linear time algorithm for this problem. Specifically speaking, we propose an  $\tilde{O}(\frac{1}{\epsilon^7}m)$  time algorithm that computes a spanning tree with tree degree  $(1 + \epsilon)\Delta^* + O(\frac{1}{\epsilon^2} \log n)$  for any constant  $\epsilon \in (0, \frac{1}{6})$ . Thus, when  $\Delta^* = \omega(\log n)$ , we can achieve approximate solutions with constant approximate ratio arbitrarily close to 1 in near-linear time.

**Keywords:** approximate algorithm · graph algorithm · minimum degree spanning tree

## 1 Introduction

Computing minimum degree spanning trees is a fundamental problem that has inspired a long time of research. Let  $G = (V, E)$  be an undirected graph, and we wish to compute a spanning tree of  $G$  whose tree degree, or maximum vertex degree in the tree, is the smallest. Clearly this problem is NP-hard as the Hamiltonian path problem can be reduced to it, and so we could only hope for a good approximation in polynomial time. The optimal approximation of this problem was achieved in [7] where the authors proposed a  $^1\tilde{O}(mn)$  time algorithm that computes a spanning tree of tree degree  $\leq \Delta^* + 1$ ; conventionally  $n = |V|, m = |E|$  and  $\Delta^*$  denotes the minimum tree degree of all the spanning trees. For convenience, the degree of a vertex usually means its tree degree in the current spanning tree.

### 1.1 Our results

The major result of this paper is a near-linear time algorithm for computing minimum degree spanning trees in undirected graphs. To the best of our knowledge, this is the first near-linear time algorithm for this problem. Formally we propose the following statement.

**Theorem 1.** *For any constant  $\epsilon \in (0, \frac{1}{6})$ , there is an algorithm that runs in  $O(\frac{1}{\epsilon^7}m \log^7 n)$  time which computes a spanning tree with tree degree at most  $(1 + \epsilon)\Delta^* + \frac{5}{16\epsilon^2} \log n$ .*

The core argument of Theorem 1 is that, starting from an arbitrary spanning tree, we repeatedly search for a sequence of distinct non-tree edges, named as *augmenting sequence* (The formal definition is given in Section 3.1), to modify the current spanning tree which immediately reduces the degree of some high-degree vertex. The idea of augmenting sequence is similar to [7], that is, given a fixed degree bound  $k$ , an augmenting sequence w.r.t. the current spanning tree and  $k$  is a sequence of vertex-disjoint non-tree edges  $(w_1, z_1), (w_2, z_2), \dots, (w_h, z_h)$  such that  $w_1, w_2, \dots, w_{h-1}$  have tree degree  $k - 1$  and  $w_h, z_1, z_2, \dots, z_h$  have tree degree  $< k - 1$ . Also there is an vertex  $w_0$  with tree degree  $\geq k$  on the tree path between  $w_1$  and  $z_1$ ,

<sup>1</sup>  $\tilde{O}(\cdot)$  hides poly-logarithmic factors.

and  $w_i$  for  $1 \leq i < h$  is on the tree path between  $w_{i+1}$  and  $z_{i+1}$  but not on the tree path between  $w_j$  and  $z_j$  for  $j > i + 1$ . Then we can add these edges  $(w_1, z_1) \cdots (w_h, z_h)$  to the spanning tree and delete the edges associated with  $w_0, \cdots, w_{h-1}$  on the cycles formed, so the total degree of vertices with degree  $\geq k$  will decrease by 1 but more degree- $(k-1)$  vertices may emerge.

In our process of searching, similar to the blocking flow approach [3] for max-flow, we first construct a layering of the graph by the shortest length of augmenting sequences, then each time find a shortest augmenting sequence in the layering and do such tree modification by this augmenting sequence, thus after near-linear time the shortest length of augmenting sequences would increase. We repeat this until the length of shortest augmenting sequence is longer than  $\frac{1}{\epsilon} \log n$ . When the shortest length of augmenting sequences exceeds  $\frac{1}{\epsilon} \log n$ , the number of layers also exceeds  $\frac{1}{\epsilon} \log n$ , so there are two adjacent layer whose ratio is at most  $1 + \epsilon$ , then if the number of augmenting sequences we found are not too large (not too many new degree- $(k-1)$  vertices emerge), we can argue a  $1 + O(\epsilon)$  approximation for the optimal solution  $\Delta^*$ . In the whole procedure of our algorithm, we can let  $k = (1 + O(\epsilon))\delta$  for the degree  $\delta$  of the current spanning tree, and make  $k$  increase by one after each iteration until in some iteration  $d_k$  is not significantly decreased. See Section 3.2.

## 1.2 Related work

There is a line of works that are concerned with low-degree trees in weighted undirected graphs. In this scenario, the target low-degree that we wish to compute is constrained by two parameters: an upper bound  $B$  on tree degree, an upper bound  $C$  on the total weight summed over all tree edges. The problem was originally formulated in [4]. Two subsequent papers [10,11] proposed polynomial time algorithms that compute a tree with cost  $\leq wC$  and degree  $\leq \frac{w}{w-1}bB + \log_b n$ ,  $\forall b, w > 1$ . This result was substantially improved by [2]; using certain augmenting path technique, their algorithm is capable of finding a tree with cost  $\leq C$  and degree  $B + O(\log n / \log \log n)$ . Results and techniques from [2] might sound similar to ours, but in undirected graphs we are actually faced with different technical difficulties. [2]'s result was improved by [8] where for all  $k$ , a spanning tree of degree  $\leq k + 2$  and of cost at most the cost of the optimum spanning tree of maximum degree at most  $k$  can be computed in polynomial time. The degree bound was later further improved from  $k + 2$  to the optimal  $k + 1$  in [15].

Another variant is minimum degree Steiner trees which is related to network broadcasting [13,14,5]. For undirected graphs, authors of [7] showed that the same approximation guarantee and running time can be achieved as with minimum degree spanning trees in undirected graphs, i.e., a solution of tree degree  $\Delta^* + 1$  and a running time of  $\tilde{O}(mn)$ . For the directed case, [5] showed that directed minimum degree Steiner tree problem cannot be approximated within  $(1 - \epsilon) \log |D|$ ,  $\forall \epsilon > 0$  unless  $\text{NP} \subseteq \text{DTIME}(n^{\log \log n})$ , where  $D$  is the set of terminals.

The minimum degree tree problem can also be formulated in directed graphs. This problem was first studied in [6] where the authors proposed a polynomial time algorithm that finds a directed spanning tree of degree at most  $O(\Delta^* \log n)$ . The approximation guarantee was improved to roughly  $O(\Delta^* + \log n)$  in [12,9] while the time complexity became  $n^{O(\log n)}$ . The problem becomes much easier when  $G$  is acyclic, as shown in [18], where a directed spanning tree of degree  $\leq \Delta^* + 1$  is computable in polynomial time. The approximation was greatly advanced to  $\Delta^* + 2$  in [1] by an LP-based polynomial time algorithm, and this problem has become more-or-less closed since then.

## 2 Preliminary

Let  $G = (V, E)$  be the graph we consider, and we assume  $G$  is a connected graph. Logarithms are taken at base 2. During the execution of our algorithm, a spanning tree  $T$  will be maintained. For every  $u \in V$ , let  $\deg(u)$  be the tree degree of  $u$  in  $T$ , and the degree of the spanning tree  $T$  is defined as  $\Delta = \max_{u \in V} \deg(u)$ . Our algorithm will repeatedly modify  $T$  to reduce its degree  $\Delta$ . Let  $\Delta^*$  denote the minimum tree degree of all the spanning trees. For each pair  $u, v \in V$ , let  $\rho_{u,v}$  be the unique tree path that connects  $u$  and  $v$

in  $\mathbb{T}$ . For each  $1 \leq k \leq n$ , define  $S_k = \{u \mid \deg(u) \geq k\}$  to be the set of vertices of degree at least  $k$ ,  $N_k = \{u \mid \deg(u) = k\}$  to be the set of vertices of degree exactly  $k$ , and  $d_k = \sum_{u \in S_k} \deg(u)$  to be the sum of degrees of vertices in  $S_k$ .

## 2.1 Boundary edge and boundary set

Boundary edge and boundary set are important concepts to get the lower bound of  $\Delta^*$ .

**Definition 1** For a graph  $G = (V, E)$  and a sequence of disjoint vertex subsets  $V_1, V_2, \dots, V_l \subseteq V$ , an edge  $(u, v) \in E$  is called a *boundary edge* if  $u \in V_i, v \in V_j$  for  $1 \leq i \neq j \leq l$ , or  $u \in V_i$  for some  $i$  but  $v \notin V_1 \cup \dots \cup V_l$ . A vertex set  $W$  is called a *boundary set* (with respect to  $V_1, V_2, \dots, V_l$ ) if, for any boundary edge  $(u, v)$ , at least one of  $u, v$  belongs to  $W$ .

**Lemma 1.** Let  $V_1, V_2, \dots, V_l \subseteq V$  be a sequence of disjoint vertex subsets,  $W$  be a boundary set and  $\Delta^*$  be the minimum tree degree of all the spanning tree in  $G$ . Then,  $\Delta^* \geq \frac{l-1}{|W|}$ .

*Proof.* By Definition 1, every set  $V_i$  can only be connected to other vertices by boundary edges, so for any spanning tree  $T$  of  $G$ , there are at least  $l - 1$  boundary edges connecting  $V_1, V_2, \dots, V_l$  in  $T$ . Then for any boundary edge  $(u, v)$ , at least one of  $u, v$  belongs to  $W$ . Thus by the pigeon-hole principle, there exists a  $u \in W$  whose tree degree is  $\geq \frac{l-1}{|W|}$ .

## 3 A $(1 + \epsilon)\Delta^* + O(\frac{1}{\epsilon^2} \log n)$ Approximation

Let  $\epsilon \in (0, \frac{1}{48})$  be a fixed parameter. This algorithm starts from an arbitrary spanning tree  $\mathbb{T}$  and keeps modifying  $\mathbb{T}$  to decrease its tree degree  $\Delta$ . It consists of two phases: the *large-step phase* and the *small-step phase*.

- In the large-step phase, as long as  $\Delta \geq \frac{10 \log^2 n}{\epsilon^3}$ , we repeatedly apply a near-linear time subroutine that, either  $\Delta$  is reduced to  $\leq (1 - \epsilon) \cdot \Delta$  or a spanning tree  $\mathbb{T}$  is returned with the guarantee that  $\Delta = (1 + O(\epsilon))\Delta^*$ .
- In the small-step phase, we need to deal with the situation where  $\frac{20 \log n}{\epsilon^2} \leq \Delta < \frac{10 \log^2 n}{\epsilon^3}$ . In this case we repeatedly run a weaker near-linear time subroutine that either  $\Delta$  is reduced by 1 or a spanning tree  $\mathbb{T}$  is returned with the guarantee that  $\Delta \leq (1 + O(\epsilon))\Delta^* + O(\frac{\log n}{\epsilon^2})$ .

Both the two phases rely on a degree reduction algorithm  $\text{AugSeqDegRed}(k)$ . The  $\text{AugSeqDegRed}(k)$  efficiently reduces the total degree of vertices with degree  $\geq k$  will decrease by 1 using an augmenting sequence technique.

For the rest of this section, we first propose and analyse the degree reduction algorithm  $\text{AugSeqDegRed}$  which underlies the core of our main algorithm. After that we specify how the large-step phase and the small-step phase work. Finally, we prove Theorem 1.

### 3.1 Degree reduction via augmenting sequences

For a fixed threshold  $k \leq \Delta$ , a simple idea is that we repeatedly look for non-tree edges that connect two vertices of tree degree  $\leq k - 2$  from different components of  $\mathbb{T} \setminus S_k$  and add these edges to  $\mathbb{T}$ , while at the same time we delete some edges incident on  $S_k$  to eliminate cycles, so the tree degrees of vertices become more balanced. In this algorithm, we continue to explore possibilities of improving the tree structure using the idea of augmenting sequence as in [7]. For a non-tree edge  $(u, v)$  that connects two different components of  $\mathbb{T} \setminus S_k$  where  $\deg(u) = k - 1$ , we try to add  $(u, v)$  to  $\mathbb{T}$  and delete some edge incident on  $S_k$  to eliminate cycles. In the same time, as  $\deg(u)$  increases to  $k$ , we keep looking for a sequence of distinct non-tree edges inside  $C_u$  to add to  $\mathbb{T}$  and delete a sequence of tree edges to eliminate cycles.

A difficulty is that when the degrees of some original  $(k - 1)$ -degree vertices decrease, it is hard to make the layering of the graph stable. Therefore, we define *marked vertices* instead of the concept of the vertices with degree  $\geq k - 1$ . Given a degree threshold  $k \leq \Delta$ , a vertex gets marked whenever its tree degree becomes  $k - 1$ , and it stays marked even if its tree degree becomes below  $k - 2$  afterwards. We only re-initialize the set of marked vertices when we change  $k$  in Section 3.2. Then we can define *augmenting sequence* formally.

**Definition 2 (augmenting sequence)** *An  $h$ -length augmenting sequence consists of a sequence of vertex-disjoint non-tree edges  $(w_1, z_1), (w_2, z_2), \dots, (w_h, z_h) \in E$  with the following properties.*

- (i)  $\exists w_0 \in \rho_{w_1, z_1} \cap S_k$ , and for all  $0 \leq i < h, w_i \in \rho_{w_{i+1}, z_{i+1}} \setminus (\bigcup_{j=i+2}^h \rho_{w_j, z_j})$ .
- (ii) All  $z_i$ 's are unmarked ( $\forall 1 \leq i \leq h$ );  $w_i$ 's are marked for all  $1 \leq i < h$  and  $w_h$  is unmarked.

Then the tree can be modified by the augmenting sequence  $(w_1, z_1), (w_2, z_2), \dots, (w_h, z_h)$  by:

**Lemma 2 (tree modification).** *Given an augmenting sequence  $(w_1, z_1), (w_2, z_2), \dots, (w_h, z_h) \in E$ , one can modify  $T$  such that  $d_k$  decreases and no vertices are added to  $S_k$ .*

*Proof.* We modify  $T$  in an inductive way. For  $i = h - 1, h - 2, \dots, 0$ , as  $w_i \in \rho_{w_{i+1}, z_{i+1}}$ , we can take an arbitrary tree edge  $(w_i, x) \in \rho_{w_{i+1}, z_{i+1}}$ , and then perform an update  $T \leftarrow T \cup \{(w_{i+1}, z_{i+1})\} \setminus \{(w_i, x)\}$  which guarantees that  $T$  is still a spanning tree. Because  $w_j \notin \rho_{w_{i+1}, z_{i+1}}$  for  $0 \leq j \leq i - 1$ , tree update  $T \leftarrow T \cup \{(w_{i+1}, z_{i+1})\} \setminus \{(w_i, x)\}$  does not change the connected components of  $T \setminus \{w_j\}$ , so the property that  $w_j \in \rho_{w_{j+1}, z_{j+1}} \setminus (\bigcup_{l=j+2}^h \rho_{w_l, z_l}), \forall 0 \leq j < i$  is preserved.

During the process, if for any  $\deg(z_i)$  ( $1 \leq i \leq h$ ) becomes  $k - 1$  during the process, mark  $z_i$ . By definition,  $d_k$  decreases as  $w_0$  loses a tree neighbour; plus, no vertices are newly added to  $S_k$  because all  $\deg(w_i), 1 \leq i < h$  are unchanged and  $\deg(w_h) \leq k - 2, \deg(z_i) \leq k - 2, \forall 1 \leq i \leq h - 2$ .

Now, back to the AugSeqDegRed algorithm. The core of this algorithm is that, if the currently shortest augmenting sequences have length  $h$  ( $h < 1 + \log_{1+\epsilon} n$ ), it searches for augmenting sequences of length  $h$  and applies Lemma 2 to decrease  $d_k$ . When there is no augmenting sequence of length  $h$ , it repeats this process for some larger  $h$ . Finally this algorithm terminates when  $h \geq 1 + \log_{1+\epsilon} n$  and we prove a lower bound on  $\Delta^*$  based on the structure of  $T$ .

First, we introduce the Layering algorithm which computes an auxiliary layering of the graph that will also help tree modification later. Initially set  $B_0 \leftarrow S_k$ . Inductively, suppose  $B_0, B_1, \dots, B_h, h \geq 0$  is already computed, then we compute the forest spanned by  $T \setminus (\bigcup_{i=0}^h B_i)$ ; for each  $u \in V \setminus (\bigcup_{i=0}^h B_i)$ , let  $C_u^h$  be the connected component of  $T \setminus (\bigcup_{i=0}^h B_i)$  that contains  $u$ . If there exists an edge  $(u, v) \in E$  such that both  $u, v$  are unmarked vertices, and that  $C_u^h \neq C_v^h$ , then the algorithm terminates and reports that the shortest length of augmenting sequences is equal to  $h + 1$ ; otherwise, we compute  $B_{h+1}$  to be the set of all marked vertices  $u \in V \setminus (\bigcup_{i=0}^h B_i)$  such that there exists an unmarked adjacent vertex  $v$  with  $C_u^h \neq C_v^h$ , and then continue until  $h > 1 + \log_{1+\epsilon} n$ . Note that whenever  $B_h = \emptyset, B_{h+1}, \dots, B_{\lceil 1 + \log_{1+\epsilon} n \rceil}$  are all empty. The pseudo code is shown in the Layering algorithm 1.

After we have invoked Layering and computed a sequence of vertex subsets  $B_0, B_1, \dots, B_h$  which naturally divides the graph into  $h + 2$  layers (including a layer of other vertices), every time we will find a length- $(h + 1)$  augmenting sequence  $(w_1, z_1), (w_2, z_2), \dots, (w_{h+1}, z_{h+1})$  such that  $w_i \in B_i$  for  $1 \leq i \leq h$ , then apply tree modifications of Lemma 2 by this augmenting sequence. Repeat this until there is no more length- $(h + 1)$  augmenting sequence any more. Then the difficulty in searching for shortest augmenting sequences is that, for a search that starts from a pair of adjacent and unmarked vertices  $u, v$  satisfying  $C_u^h \neq C_v^h$  and goes up the layers  $B_h, B_{h-1}, \dots, B_1, B_0$ , not every route can reach the top layer  $B_0$  because some previous  $(h + 1)$ -length augmenting sequences have already blocked the road. Therefore, a depth-first search needs to be performed. To save running time, some tricks are needed: if a certain vertex has been searched before by some previous  $(h + 1)$ -length augmenting sequences and has failed to lead a way upwards to  $B_0$ , then we *tag* this vertex so that future depth-first searches may avoid this tagged vertex; if a certain edge has been searched before, then we *tag* this edge whatsoever. The AugDFS algorithm may be a better illustration of this algorithm. The recursive algorithm AugDFS takes the layer number  $i$  and an edge  $(u, v)$  on layer  $i$  as input and keep

---

**Algorithm 1: Layering**

---

```
1  $B_0 \leftarrow S_k, h \leftarrow 0;$ 
2 while  $h < 1 + \log_{1+\epsilon} n$  do
3   compute the forest  $\{C_u^h\}$  spanned by  $T \setminus (\bigcup_{i=0}^h B_i)$ ;
4   if exists unmarked  $u, v$  such that  $(u, v) \in E, C_u^h \neq C_v^h$  then
5     break;
6   else
7     compute  $B_{h+1}$  to be the set of all marked vertices in  $u \in V \setminus (\bigcup_{i=0}^h B_i)$  such that there exists an
      unmarked adjacent vertex  $v$  with  $C_u^h \neq C_v^h$ ;
8      $h \leftarrow h + 1;$ 
9 return  $h$  and  $B_0, B_1, \dots, B_h;$ 
```

---

searching for edges between a vertex  $w \in (u, v) \cap B_{i-1}$  and an unmarked vertex  $z$  on layer  $(i-1)$ . If such an edge is found, invoking AugDFS with parameter  $(i-1, (w, z))$  and return the result plus  $(u, v)$ . The pseudo code is shown in the AugDFS algorithm 2. Later we will prove that AugDFS( $h+1, (u, v)$ ) always returns null or an augmenting sequence.

---

**Algorithm 2: AugDFS( $i, (u, v)$ )**

---

```
1 if  $i = 1$  then
2   return  $(u, v);$ 
3 for untagged  $w \in \rho_{u,v} \cap B_{i-1}$  do
4   for unmarked  $z$  such that  $(w, z)$  is untagged and  $C_z^{i-2} \neq C_w^{i-2}$  do
5      $p_{i-1} \leftarrow \text{AugDFS}(i-1, (w, z));$ 
6     tag  $(w, z);$ 
7     if  $p_{i-1} \neq \text{null}$  then
8       let  $p_i$  be  $p_{i-1}$  plus  $(u, v);$ 
9       return  $p_i;$ 
10  tag  $w;$ 
11 return null;
```

---

The upper-level AugSeqDegRed algorithm repeatedly applies Layering followed by several rounds of AugDFS. Each time AugDFS returns an augmenting sequence  $p$ , modify  $T$  by Lemma 1 via  $p$ . The repeat-loop ends when  $h \geq 1 + \log_{1+\epsilon} n$ . The pseudo code is shown in the AugSeqDegRed algorithm 3.

Before proving termination of AugSeqDegRed, we first need to argue some properties of Layering. The following lemma will serve as the basis for our future proof.

**Lemma 3 (the blocking property).** *Throughout each iteration of the repeat-loop in AugSeqDegRed, for any  $1 \leq i < h$  and any two adjacent vertices  $u, v \in V \setminus (\bigcup_{j=0}^i B_j)$  such that  $u$  is unmarked and  $C_u^i \neq C_v^i$ , then  $v \in B_{i+1}$ . (Recall that  $C_u^h$  is the connected component of  $T \setminus (\bigcup_{i=0}^h B_i)$  that contains  $u$ .)*

*Proof.* By rules of Layering, this blocking property holds right after Layering outputs them. This claim continuous to hold afterwards because tree modifications only merge components  $C_u^i$ 's and never splits any  $C_u^i$ 's.

Here is an important corollary of this Lemma 3.

---

**Algorithm 3:** AugSeqDegRed( $k$ )

---

```
1 mark all degree  $k - 1$  vertices, unmark other vertices;
2 repeat
3   run Layering which computes  $h$  and  $B_0, B_1, \dots, B_h$ ;
4   untag all vertices and edges;
5   for  $(u, v) \in E$  such that  $u, v$  are unmarked and adjacent, and that  $C_u^h \neq C_v^h$  do
6      $p \leftarrow \text{AugDFS}(h+1, (u, v))$ ;
7     if  $p \neq \text{null}$  then
8       modify  $\mathbb{T}$  by augmenting sequence  $p$  via Lemma 2;
9 until  $h \geq 1 + \log_{1+\epsilon} n$ ;
10 return  $\mathbb{T}$ ;
```

---

**Corollary 1.** *Throughout each iteration of the repeat-loop, for any  $w \in B_i, 1 \leq i \leq h$ , suppose  $w$  is adjacent to an unmarked  $z$  such that  $C_w^{i-1} \neq C_z^{i-1}$ . Then  $\rho_{w,z}$  only contains vertices from  $V \setminus (\bigcup_{j=0}^{i-2} B_j)$ .*

*Proof.* Suppose otherwise, then there would be a vertex  $x \in \rho_{w,z} \cap B_j, j \leq i - 2$ , then in this case  $C_w^j \neq C_z^j$ , and thus by Lemma 3  $w \in B_{j+1}$  which is a contradiction as  $j + 1 < i$ .

Now we have the following lemmas:

**Lemma 4.** *If  $\text{AugDFS}(h+1, (u, v))$  returns a sequence of edges  $(w_1, z_1), \dots, (w_{h+1}, z_{h+1})$ , then  $w_i \in B_i$  for  $1 \leq i \leq h$ , and  $w_{h+1}, z_1, \dots, z_{h+1}$  are unmarked, also the edges are vertex-disjoint.*

*Proof.* The initial  $u, v$  are unmarked. From the algorithm, when calling  $\text{AugDFS}(i, (u, v))$ , we find a  $w \in \rho_{u,v} \cap B_i$  and  $z$  is unmarked, so the corresponding  $w_i \in B_i$  for  $1 \leq i \leq h$ , and  $w_{h+1}, z_1, \dots, z_{h+1}$  are unmarked, also the vertices  $\{w_i | 1 \leq i \leq h\}$  are distinct. To see that  $w_{h+1}, z_1, \dots, z_{h+1}$  are distinct, we argue that in one execution of  $\text{AugDFS}(i, (u, v))$ ,  $w$  and  $z$  have  $C_z^{i-2} \neq C_w^{i-2}$  but  $C_z^{i-3} = C_w^{i-3}$ , since if  $C_z^{i-3} \neq C_w^{i-3}$ ,  $w$  would be in  $B_{i-2}$  by the algorithm Layering. Thus  $w_{h+1}, z_1, \dots, z_{h+1}$  are in distinct components in  $\mathbb{T} \setminus (\bigcup_{i=0}^h B_i)$ .

**Lemma 5.** *In the AugSeqDegRed algorithm,  $\text{AugDFS}(h+1, (u, v))$  returns either null or an augmenting sequence.*

*Proof.* Assume a sequence of edges  $(w_1, z_1), \dots, (w_{h+1}, z_{h+1})$  is returned by  $\text{AugDFS}(h+1, (u, v))$ . Property (ii) in Definition 2 is proved by Lemma 4. Now let us focus on property (i). We can take an arbitrary  $w_0 \in \rho_{w_1, z_1} \cap B_0$  since  $C_{w_1}^0 \neq C_{z_1}^0$  by the algorithm. Also since  $w_i \in B_i, \forall 0 \leq i \leq h$ , by Corollary 1 we know  $\rho_{w_i, z_i}$  does not contain any  $w_j, 0 \leq j \leq i - 2$ , so property (i) holds.

The following statement concludes the AugSeqDegRed algorithm will terminate quickly.

**Lemma 6.** *In the AugSeqDegRed algorithm,  $h$  is increased by at least one during each repeat-loop, except the last one.*

*Proof.* By the rules of Layering, it is easy to see that at the beginning when Layering outputs  $B_0, B_1, \dots, B_h$ , the shortest length of augmenting sequence is equal to  $h + 1$ . So it suffices to prove that by the end of this iteration the shortest augmenting sequence has length  $> h + 1$ .

First we need to characterize all augmenting sequences using  $B_0, B_1, \dots, B_h$ . Let the sequence  $(w_1, z_1), (w_2, z_2), \dots, (w_l, z_l)$  be an arbitrary augmenting sequence and let  $w_0$  be the  $B_0$ -vertex on  $\rho_{w_1, z_1}$ . We argue  $l \geq h + 1$ , and more importantly, if  $l = h + 1$ , it must be  $w_i \in B_i, \forall 0 \leq i \leq h$ . We inductively prove that  $w_i \in \bigcup_{j=0}^i B_j$  for  $i = 0, 1, \dots, l - 1$ . The basis is obvious as is required by property (i) in Definition 2. Now assume  $w_i \in B_r$  for some  $r \leq i$ . Then, from algorithm Layering, it would not be hard to see  $w_{i+1} \in \bigcup_{j=0}^{r+1} B_j \subseteq \bigcup_{j=0}^{i+1} B_j$ . Now, since components  $\{C_u^r\}$  for  $r \leq h - 1$  are not connected by edges whose both

endpoints are unmarked by Lemma 3, so  $\rho_{w_l, z_l} \cap \bigcup_{j=0}^{h-1} B_j = \emptyset$ , and on the other hand  $w_{l-1} \in \rho_{w_l, z_l} \cap \bigcup_{j=0}^{l-1} B_j$ , so  $l \geq h+1$ . Plus, we can see from the induction that, when  $l = h+1$  it must be  $w_i \in B_i, \forall 0 \leq i \leq h$ .

For any unmarked and adjacent vertices  $u, v$  such that  $C_u^h \neq C_v^h$ , consider the instance of AugDFS with input  $(h+1, (u, v))$ . We make two claims.

- (1) If there is an  $(h+1)$ -length augmenting sequence ending with  $(u, v)$ , AugDFS would succeed in finding one.
- (2) If it has returned null, then there would be no  $(h+1)$ -augmenting sequence ending with  $(u, v)$  throughout the entire repeat-loop iteration.

If (1)(2) can be proved, then by the end of this repeat-loop iteration, there would be no  $(h+1)$ -length augmenting sequences because such augmenting sequence should end with a pair of adjacent unmarked vertices. Next we come to prove (1)(2).

- (1) The depth-first search of AugDFS exactly coincides with the conditions that  $w_i \in B_i$ , except that it skips all tagged vertices and edges. Now we prove that omitting tagged vertices and edges does not miss any  $(h+1)$ -length augmenting sequences. For an edge  $(w, z)$  to be tagged, either a further recursion AugDFS has succeeded or failed in finding an augmenting sequences; in the former case,  $C_w^{i-2}$  and  $C_z^{i-2}$  has been merged, and so the condition  $C_w^{i-2} \neq C_z^{i-2}$  would be violated afterwards; in the latter case, we would not need to recur on  $(w, z)$  since the components w.r.t.  $B_0, \dots, B_{i-2}$  also can only merge. For a vertex  $w$  to be tagged, we must have enumerated all of its untagged edges  $(w, z)$  but failed to find any augmenting sequences, and therefore any future depth-first searches on  $w$  would still end up in vain.
- (2) If AugDFS has once failed to find any augmenting sequences starting with  $(u, v)$ , then all vertices  $w \in \rho_{u,v} \cap B_h$  visited by this instance of AugDFS should be tagged and they would be omitted by all succeeding instances of AugDFS. Therefore  $\rho_{u,v} \cap B_h$  would stay unchanged since then. Hence, if we re-run AugDFS with  $h+1, (u, v)$ , it will return null without any recursion because all vertices in  $\rho_{u,v} \cap B_h$  are tagged.

Suppose AugSeqDegRed has terminated with  $B_0, B_1, \dots, B_{\lceil \log_{1+\epsilon} n + 1 \rceil}$ . We define *clean component*, a sequence of disjoint vertex subsets, and apply Lemma 1 to get the lower bound on  $\Delta^*$ .

**Definition 3** After an instance of AugSeqDegRed has been executed, for an arbitrary component  $C_u^h, 0 \leq h \leq \lceil \log_{1+\epsilon} n + 1 \rceil$ ,  $u \in V \setminus (\bigcup_{i=0}^h B_i)$ , it is called *clean* if all vertices in  $C_u^h$  are unmarked.

**Lemma 7.** For any  $0 \leq h < \lceil \log_{1+\epsilon} n + 1 \rceil$ , suppose  $T \setminus (\bigcup_{i=0}^h B_i)$  has  $l$  clean components, then a lower bound holds that  $\Delta^* \geq \frac{l-1}{\sum_{i=0}^{h+1} |B_i|}$ .

*Proof.* Since  $h < \lceil \log_{1+\epsilon} n + 1 \rceil$ ,  $B_h$  is not the last one, so there is no edge connecting two unmarked vertices in different components of  $T \setminus (\bigcup_{i=0}^h B_i)$ . By Lemma 3, any edge that connects a clean components of  $T \setminus (\bigcup_{i=0}^h B_i)$  outwards must be incident on a vertex in  $\bigcup_{i=0}^{h+1} B_i$ , so  $\bigcup_{i=0}^{h+1} B_i$  is a boundary set w.r.t. clean components. Therefore by Lemma 1 we have  $\Delta^* \geq \frac{l-1}{|\bigcup_{i=0}^{h+1} B_i|} = \frac{l-1}{\sum_{i=0}^{h+1} |B_i|}$

It is not hard to see that one execution of AugSeqDegRed takes near-linear time as summarized by the following lemma, whose proof will be given in the appendix.

**Lemma 8.** There is an implementation of AugSeqDegRed that runs in  $O(\frac{1}{\epsilon^2} m \log^2 n)$  time.

### 3.2 Large-step phase and small-step phase

The large-step phase and small-step phase are described in the ImprovedMDST algorithm 4. In the large-step phase, we deal with the case  $\Delta \geq \frac{10 \log^2 n}{\epsilon^3}$ . It works by invoking AugSeqDegRed with an incremental parameters  $k$  from  $(1-2\epsilon)\Delta + 1$  if  $d_{k-1} \leq 2d_k$ . Within each iteration, if AugSeqDegRed fails to reduce  $d_k$  by a factor of  $(1 - \frac{\epsilon^2}{2 \log n})$ , then the algorithm reports a lower bound on  $\Delta^*$  and returns  $T$  immediately.

Otherwise, increase  $k$  by 1 and continue until  $d_k$  becomes 0. Since  $d_{k+1} \leq d_k$ ,  $d_k$  will become 0 in at most  $O(\log^2 n/\epsilon^2)$  iterations. Once  $d_k = 0$ , set  $\Delta = k$  and repeat the while-loop.

In the small-step phase, we only deal with  $\frac{20 \log n}{\epsilon^2} \leq \Delta < \frac{10 \log^2 n}{\epsilon^3}$ . Set  $c = 6(2 + \log_{1+\epsilon} n)$  and define a potential:

$$\phi(\mathbb{T}) = \sum_{i=\Delta+1-\log n}^{\Delta} c^i \cdot |N_i|$$

The small-step phase works by repeatedly selecting a degree  $k$  that maximizes  $c^k \cdot |N_k|$  and then run `AugSeqDegRed(k)` until  $\Delta$  decreases. Similar with the large-step phase, if `AugSeqDegRed(k)` fails to reduce  $d_k$  significantly, then the algorithm reports a lower bound on  $\Delta^*$  and returns  $\mathbb{T}$  immediately. Clearly  $k$  must be larger than  $\Delta - \log n > \frac{19 \log n}{\epsilon^2}$ .

---

#### Algorithm 4: ImprovedMDST

---

```

1 Let  $\mathbb{T}$  be a spanning tree of  $G$ ;
  /* Large-step phase */
2 while  $\Delta \geq \frac{10 \log^2 n}{\epsilon^3}$  do
3    $k = (1 - 2\epsilon)\Delta + 1$ ;
4   while  $d_k > 0$  do
5     if  $d_{k-1} \leq 2d_k$  then
6        $d \leftarrow d_k$ ;
7       run AugSeqDegRed( $k$ );
8       if  $d_k > (1 - \frac{\epsilon^2}{2 \log n}) \cdot d$  then
9         return  $\mathbb{T}$ ;
10     $k = k + 1$ ;
11    $\Delta = k$ ;
  /* Small-step phase */
12 while  $\Delta \geq \frac{20 \log n}{\epsilon^2}$  do
13   while  $\Delta$  has not changed do
14     pick a  $k \in \arg \max_{i \in [\Delta+1-\log n, \Delta]} \{c^i \cdot |N_i|\}$ ;
15      $d \leftarrow d_k$ ;
16     run AugSeqDegRed( $k$ );
17     if  $d_k > (1 - \frac{\epsilon^2}{2 \log n}) \cdot d$  then
18       return  $\mathbb{T}$ ;
19 return  $\mathbb{T}$ ;

```

---

#### Running time

In the large-step phase, since every iteration  $d_k$  is shrink by a factor of  $\leq (1 - \frac{\epsilon^2}{2 \log n})$ , so  $d_k$  will become zero in  $O(\log^2 n/\epsilon^2)$  iterations. We have:

**Lemma 9.** *The running time of the large-step phase is bounded by  $O(\frac{1}{\epsilon^5} m \log^5 n)$ .*

*Proof.* From the previous subsection we already know `AugSeqDegRed` runs in  $O(\frac{1}{\epsilon^2} m \log^2 n)$  time, so here we only need to upper bound the total number of times `AugSeqDegRed` gets invoked before  $\Delta < \frac{10 \log^2 n}{\epsilon^3}$  or a spanning tree  $\mathbb{T}$  is returned within a while-loop. Next we only focus on the previous cases because it takes



a longer running time. In this case, at the end of each iteration,  $d_k \leq (1 - \frac{\epsilon^2}{2 \log n}) \cdot d$ . The inside while-loop would break when  $k \geq (1 - 2\epsilon)\Delta + \frac{2 \log^2 n}{\epsilon^2}$  because by the time

$$d_k \leq \left(1 - \frac{\epsilon^2}{2 \log n}\right)^{\frac{2 \log^2 n}{\epsilon^2}} \cdot d_{(1-2\epsilon)\Delta} \leq \frac{d_{(1-2\epsilon)\Delta}}{n} < 1$$

As  $(1 - 2\epsilon)\Delta + \frac{2 \log^2 n}{\epsilon^2} \leq (1 - \epsilon)\Delta$  when  $\Delta \geq \frac{10 \log^2 n}{\epsilon^3}$ , which means  $\Delta$  has been reduced by a factor of at most  $1 - \epsilon$  in the end of each while-loop and there are at most  $O(\frac{1}{\epsilon} \log n)$  while-loops within the large-step phase. In summary, the total running time of the large-step phase is  $O\left(\frac{\log^2 n}{\epsilon^2} m \times \frac{\log^2 n}{\epsilon^2} \times \frac{\log n}{\epsilon}\right) = O\left(m \cdot \frac{\log^5 n}{\epsilon^5}\right)$

In the small-step phase, since every iteration  $\phi(\mathbb{T})$  is shrink by a factor of  $\leq 1 - \frac{\epsilon^2}{5 \log^2 n}$ , after  $O(\frac{\log^3 n}{\epsilon^2})$  rounds,  $\phi(\mathbb{T})$  will be smaller than  $c^\Delta$ .

**Lemma 10.** *The running time of the small-step phase is bounded by  $O(\frac{1}{\epsilon^7} m \log^7 n)$ .*

*Proof.* We already know `AugSeqDegRed` runs in  $O(\frac{1}{\epsilon^2} m \log^2 n)$  time. Now we study how many rounds of `AugSeqDegRed` could be invoked before  $\Delta$  changes or this algorithm returns  $\mathbb{T}$  within a while-loop. We only focus on the previous cases because it takes a longer running time. For one execution of `AugSeqDegRed`, let  $N'_k, d'_k, \mathbb{T}'$  ( $k \in [\Delta + 1 - \log n, \Delta]$ ) be snapshots of  $N_k, d_k, \mathbb{T}$  right before we execute `AugSeqDegRed`, and here we consider the case when  $\Delta$  is not changed and  $d_k \leq (1 - \frac{\epsilon^2}{2 \log n}) \cdot d'_k$ .

Next we analyse how  $\phi(\mathbb{T})$  has decreased. The potential before the change is  $\phi(\mathbb{T}') = \sum_{i=\Delta+1-\log n}^{\Delta} c^i \cdot |N'_k|$ . Every time a tree modification to  $\mathbb{T}$  was made on `AugSeqDegRed`, at least one vertex in  $S_k$  would lose a tree edge and at most  $2 + \log_{1+\epsilon} n$  vertices with degree  $< k - 1$  would gain a tree edge, and then the total loss of  $\phi(\mathbb{T})$  would be at least

$$\begin{aligned} & (c^k - c^{k-1}) - (2 + \log_{1+\epsilon} n) \cdot (c^{k-1} - c^{k-2}) \geq (c^{k-1} - c^{k-2})(c - 2 - \log_{1+\epsilon} n) \\ & = c^k \cdot \left(1 - \frac{1}{c}\right) \cdot \left(1 - \frac{2 + \log_{1+\epsilon} n}{c}\right) \geq c^k \cdot \left(1 - \frac{1}{c}\right) \cdot \frac{5}{6} > 0.8 \cdot c^k \end{aligned}$$

After executing `AugSeqDegRed`,  $d_k$  has decreased by  $d'_k - d_k \geq \frac{\epsilon^2}{2 \log n} d'_k \geq \frac{\epsilon^2}{2 \log n} \cdot k |N'_k|$ , and so there are at least  $(d'_k - d_k)/(2k) \geq \frac{\epsilon^2}{4 \log n} \cdot |N'_k|$  modifications via Lemma 2 to  $\mathbb{T}$ . Therefore,

$$\phi(\mathbb{T}) \leq \phi(\mathbb{T}') - (0.8 \cdot c^k) \cdot \left(\frac{\epsilon^2}{4 \log n} \cdot |N'_k|\right) \leq \left(1 - \frac{0.2\epsilon^2}{\log^2 n}\right) \phi(\mathbb{T}')$$

The second inequality holds by maximality of  $c^k \cdot |N'_k|$  which implies  $c^k \cdot |N'_k| \geq \frac{1}{\log n} \cdot \phi(\mathbb{T}')$ .

In a nutshell,  $\phi(\mathbb{T})$  has decreases by a factor of at most  $1 - \frac{0.2\epsilon^2}{\log^2 n}$ . As long as  $\Delta$  has not changed,  $\phi(\mathbb{T})$  belongs to the interval  $(c^\Delta, n \cdot c^\Delta)$ , and consequently,  $\phi(\mathbb{T})$  could suffer at most  $-\log_{1 - \frac{0.2\epsilon^2}{\log^2 n}} n \cdot c = O(\frac{\log^3 n}{\epsilon^2})$  rounds of `AugSeqDegRed` before  $\Delta$  decreases. There are at most  $O(\frac{\log^2 n}{\epsilon^3})$  while-loops in the small-step phase because each while-loop reduces  $\Delta$  by at least 1.

In summary, the total running time of the small-step phase is  $O\left(\frac{\log^2 n}{\epsilon^2} m \times \frac{\log^3 n}{\epsilon^2} \times \frac{\log^2 n}{\epsilon^3}\right) = O\left(m \cdot \frac{\log^7 n}{\epsilon^7}\right)$

## Approximation guarantee

When a spanning tree  $\mathbb{T}$  is returned within the large-step phase or the small-step phase, the vertex subsets  $B_0, B_1, \dots, B_{\lceil 1 + \log_{1+\epsilon} n \rceil}$  created by `AugSeqDegRed` satisfies the blocking property. By Lemma 7, there is a lower bound on  $\Delta^*$  for each vertex set  $B_h, 0 \leq h \leq \lceil 1 + \log_{1+\epsilon} n \rceil$  as long as we get the lower bound on the number of clean components in  $\mathbb{T} \setminus (\bigcup_{i=0}^h B_i)$ . The following two statements show the lower bound on  $\Delta^*$ .

**Lemma 11.** For any vertex subset  $B$  and any spanning tree  $T$ , the number of connected components in  $T \setminus B$  is at least  $\sum_{u \in B} \deg(u) - 2|B| + 2$ .

*Proof.* Note that there are at least  $\sum_{u \in B} \deg(u) - |B| + 1$  tree edges incident on  $B$ , and so removing all of these edges would break  $T$  into  $\geq \sum_{u \in B} \deg(u) - |B| + 2$  components. Therefore, excluding singleton components formed by  $B$ , there are  $\geq \sum_{u \in B} \deg(u) - 2|B| + 2$  components are from  $T \setminus B$ .

**Lemma 12.** If a spanning tree  $T$  is returned within the large-step phase or the small-step phase and  $k$  is the parameter of the last invoked `AugSeqDegRed`, for any  $0 \leq h \leq \lceil 1 + \log_{1+\epsilon} n \rceil$ , the number of clean components in  $T \setminus (\bigcup_{i=0}^h B_i)$  is more than  $k \cdot (1 - 4\epsilon) \sum_{i=0}^h |B_i| + 1$  for  $\epsilon \in (0, \frac{1}{48})$ . Furthermore,

$$\Delta^* \geq k(1 - 4\epsilon) \cdot \frac{\sum_{i=0}^h |B_i|}{\sum_{i=0}^{h+1} |B_i|}$$

*Proof.* By Lemma 11, the number of tree components in  $T \setminus (\bigcup_{i=0}^h B_i)$  is at least

$$\sum_{u \in \bigcup_{i=0}^h B_i} \deg(u) - 2 \left| \bigcup_{i=0}^h B_i \right| + 2$$

Let  $d'_k, d'_{k-1}, S'_{k-1}$  and  $S'_k$  be snapshots of  $d_k, d_{k-1}, S_{k-1}$  and  $S_k$  right before the last instance of `AugSeqDegRed` started and let  $M$  be the set of all marked vertices  $\notin S'_{k-1}$  (i.e., vertices that are initially unmarked) by the end of `AugSeqDegRed`. Then, the number of clean components in  $T \setminus (\bigcup_{i=0}^h B_i)$  is at least

$$\sum_{u \in \bigcup_{i=0}^h B_i} \deg(u) - 2 \sum_{i=0}^h |B_i| + 2 - |M \cup S'_{k-1}|$$

The argument consists of a lower bound on  $\sum_{u \in \bigcup_{i=0}^h B_i} \deg(u)$  and an upper bound on  $|M \cup S'_{k-1}|$ .

(1) Lower bound on  $\sum_{u \in \bigcup_{i=0}^h B_i} \deg(u)$ .

By the Layering algorithm  $B_0 = S_k$ , then we have  $\sum_{u \in B_0} \deg(u) = d_k$ .

For any vertex  $u \in \bigcup_{i=1}^h B_i$ ,  $\deg(u) = k - 1$  by the time  $u$  was first added to some  $B_i$ . After that,  $\deg(u)$  could only decrease when we modify  $T$  by an augmenting sequence  $(w_1, z_1), \dots, (w_t, z_t)$  where  $u = w_j$  for some  $1 \leq j \leq t$ . Since  $t \leq \lceil 1 + \log_{1+\epsilon} n \rceil$ , during a tree modification, at least one vertex in  $S_k$  loses one degree and at most  $\lceil 1 + \log_{1+\epsilon} n \rceil$  vertices in  $\bigcup_{i=1}^h B_i$  lose one degree separately. As the total number of the degree loss in  $S_k$  is  $(d'_k - d_k)$ , we have

$$\sum_{u \in \bigcup_{i=1}^h B_i} \deg(u) \geq (k - 1) \sum_{i=1}^h |B_i| - (d'_k - d_k) \lceil 1 + \log_{1+\epsilon} n \rceil$$

Since  $d_k > (1 - \frac{\epsilon^2}{2 \log n}) \cdot d'_k$ , we get a lower bound on  $\sum_{u \in \bigcup_{i=0}^h B_i} \deg(u)$ ,

$$\begin{aligned} \sum_{u \in \bigcup_{i=0}^h B_i} \deg(u) &\geq d_k + (k - 1) \sum_{i=1}^h |B_i| - (d'_k - d_k) \lceil 1 + \log_{1+\epsilon} n \rceil \\ &\geq (k - 1) \sum_{i=1}^h |B_i| + \left(1 - \frac{\epsilon^2}{2 \log n}\right) d'_k - \frac{\epsilon^2}{2 \log n} (2 + \log_{1+\epsilon} n) d'_k \\ &\geq (k - 1) \sum_{i=1}^h |B_i| + \left(1 - \frac{3\epsilon^2}{2 \log n} - \frac{\epsilon}{2}\right) d'_k \end{aligned}$$

(2) Upper bound on  $|M|$ .

The argument is similar to (1). An unmarked vertex  $u$  is marked only when we modify  $T$  by an augmenting sequence  $(w_1, z_1), \dots, (w_t, z_t)$  where  $u = z_j$  for some  $1 \leq j \leq t$  or  $u = w_t$ . Since  $t \leq 1 + \log_{1+\epsilon} n$ , during a tree modification, at least one vertex in  $S_k$  loses one degree and at most  $2 + \log_{1+\epsilon} n$  unmarked vertices are marked. Then we get an upper bound on  $|M|$ .

$$|M| \leq (d'_k - d_k)(2 + \log_{1+\epsilon} n) \leq \epsilon \cdot d'_k$$

(3) Upper bound on  $|S'_{k-1}|$ .

First we claim  $\frac{d'_k}{d'_{k-1}} \geq \frac{1}{\epsilon(k-1)}$  when  $k \geq \frac{19 \log n}{\epsilon^2} \geq \frac{19}{\epsilon}$  and  $n > 2^\epsilon$ . In the large-step phase, the inequality holds since  $d'_k \geq \frac{1}{2}d'_{k-1}$ . In the small-step phase, by maximality of  $c^k \cdot |N_k|$ , we have  $|N_k| \geq \frac{1}{c} \cdot |N_{k-1}|$ . Then,

$$\frac{d'_k}{d'_{k-1}} = \frac{\sum_{i=k}^{\Delta} i |N_i|}{\sum_{i=k-1}^{\Delta} i |N_i|} > \frac{k |N_k|}{k |N_k| + (k-1) |N_{k-1}|} \geq \frac{1}{1 + \frac{c(k-1)}{k}} > \frac{1}{c+1} \geq \frac{1}{\epsilon(k-1)}$$

The last inequality holds by  $c = 6 \log_{1+\epsilon} n + 12 \leq \frac{12 \ln 2}{\epsilon} \log n + 12$ . Then we have the upper bound on  $|S'_{k-1}|$ :

$$|S'_{k-1}| \leq \frac{d'_{k-1}}{k-1} \leq \epsilon \cdot d'_k$$

Summing up (1)(2)(3), for  $n > 2$  and  $\epsilon \in (0, \frac{1}{48})$ ,  $k \geq \frac{19 \log n}{\epsilon^2} \geq \frac{19}{\epsilon}$ , we have

$$\begin{aligned} & \sum_{u \in \bigcup_{i=0}^h B_i} \deg(u) - 2 \sum_{i=0}^h |B_i| + 2 - |M \cup S'_{k-1}| \\ & \geq \left(1 - \frac{3\epsilon^2}{2 \log n} - 2.5\epsilon\right) \cdot d'_k + (k-1) \sum_{i=1}^h |B_i| - 2 \sum_{i=0}^h |B_i| + 2 \\ & \geq (1 - 2.6\epsilon) \cdot k |B_0| - 2 |B_0| + (k-3) \sum_{i=1}^h |B_i| + 2 \\ & > k(1 - 2.6\epsilon) \cdot |B_0| - 2 |B_0| + (k-3) \sum_{i=1}^h |B_i| + 2 \geq k(1 - 4\epsilon) \cdot \sum_{i=0}^h |B_i| + 2 \end{aligned}$$

Apply Lemma 7, we conclude the proof

$$\Delta^* \geq \frac{k(1 - 4\epsilon) \cdot \sum_{i=0}^h |B_i| + 1}{\sum_{i=0}^{h+1} |B_i|} \geq k(1 - 4\epsilon) \cdot \frac{\sum_{i=0}^h |B_i|}{\sum_{i=0}^{h+1} |B_i|}$$

In the following two statements, we combine all the inequalities for each  $B_h$ ,  $0 \leq h \leq \log_{1+\epsilon} n$  and get the upper bound on  $\Delta$  with  $\Delta^*$  in both the large-step phase and the small-step phase.

**Lemma 13.** *When a spanning tree  $T$  is returned within the large-step phase, it must be  $\Delta \leq (1 + 8\epsilon) \cdot \Delta^*$  for  $\epsilon \in (0, \frac{1}{48})$ .*

*Proof.* Consider the most recent execution of AugSeqDegRed before returning. By the previous subsection, this instance of AugSeqDegRed has created a sequence of disjoint vertex subsets  $B_0, B_1, \dots, B_{1+\log_{1+\epsilon} n}$  that satisfies the blocking property. By the pigeon-hole principle, there exists an  $h$  such that  $\frac{\sum_{i=0}^h |B_i|}{\sum_{i=0}^{h+1} |B_i|} \geq \frac{1}{1+\epsilon}$ . Then by Lemma 12,

$$\Delta^* \geq k(1 - 4\epsilon) \cdot \frac{1}{1 + \epsilon} > \frac{1 - 6\epsilon + 8\epsilon^2}{1 + \epsilon} \Delta$$

or equivalently,  $\Delta \leq \frac{1+\epsilon}{1-6\epsilon+8\epsilon^2} \Delta^* < (1 + 8\epsilon) \Delta^*$  when  $\epsilon \in (0, \frac{1}{48})$ .

**Lemma 14.** *When a spanning tree  $T$  is returned within the small-step phase, it must be  $\Delta \leq (1 + 6\epsilon)\Delta^* + \log n$  for  $\epsilon \in (0, \frac{1}{48})$ .*

*Proof.* Consider the most recent execution of `AugSeqDegRed` before returning. By the previous subsection, this instance of `AugSeqDegRed` has created a sequence of disjoint vertex subsets  $B_0, B_1, \dots, B_{1+\log_{1+\epsilon} n}$  that satisfies the blocking property. By the pigeon-hole principle, there exists an  $h$  such that  $\frac{\sum_{i=0}^h |B_i|}{\sum_{i=0}^{h+1} |B_i|} \geq \frac{1}{1+\epsilon}$ . Then by Lemma 12,

$$\Delta^* \geq k(1 - 4\epsilon) \cdot \frac{1}{1 + \epsilon} > \frac{1 - 4\epsilon}{1 + \epsilon} (\Delta - \log n)$$

or equivalently,  $\Delta \leq \frac{1+\epsilon}{1-4\epsilon} \Delta^* + \log n < (1 + 6\epsilon)\Delta^* + \log n$  for  $\epsilon \in (0, \frac{1}{48})$ .

Now we can finish the proof of Lemma 1

*Proof (Proof of Lemma 1).* We claim that, for any constant  $\epsilon \in (0, \frac{1}{6})$ , the `ImprovedMDST` algorithm computes a spanning tree with tree degree  $(1 + \epsilon)\Delta^* + \frac{5}{16\epsilon^2} \log n$  in  $O(\frac{1}{\epsilon^7} m \log^7 n)$  time (by resetting  $\epsilon \rightarrow 8\epsilon'$  where  $\epsilon'$  is the  $\epsilon$  in previous analysis).

By Lemma 9 and Lemma 10, we know the total running time of the `ImprovedMDST` algorithm is bounded by  $O(\frac{1}{\epsilon^7} m \log^7 n)$ . The degree analysis is divided into three cases:

- If  $T$  is returned after the small-step phase, then  $\Delta < \frac{20 \log n}{64\epsilon^2} = \frac{5}{16\epsilon^2} \log n$  for  $\epsilon \in (0, \frac{1}{6})$ .
- If  $T$  is returned during the large-step phase, by Lemma 13,  $\Delta \leq (1 + \epsilon) \cdot \Delta^*$  for  $\epsilon \in (0, \frac{1}{6})$ .
- If  $T$  is returned during the small-step phase, by Lemma 14,  $\Delta \leq (1 + \frac{3\epsilon}{4})\Delta^* + \log n$  for  $\epsilon \in (0, \frac{1}{6})$ .

In summary, the `ImprovedMDST` algorithm computes a spanning tree with tree degree  $(1+\epsilon)\Delta^* + \frac{5}{16\epsilon^2} \log n$  in  $O(\frac{1}{\epsilon^7} m \log^7 n)$  time.

## References

1. Bansal, N., Khandekar, R., Nagarajan, V.: Additive guarantees for degree-bounded directed network design. *SIAM Journal on Computing* **39**(4), 1413–1431 (2009)
2. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: What would Edmonds do? augmenting paths and witnesses for degree-bounded MSTs. *Lecture notes in computer science* **3624**, 26 (2005)
3. Dinitz, Y.: Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.* **11**, 1277–1280 (01 1970)
4. Fischer, T.: Optimizing the degree of minimum weight spanning trees. Tech. rep., Cornell University (1993)
5. Fraigniaud, P.: Approximation algorithms for minimum-time broadcast under the vertex-disjoint paths mode. *AlgorithmsESA 2001* pp. 440–451 (2001)
6. Fürer, M., Raghavachari, B.: An NC approximation algorithm for the minimum degree spanning tree problem. In: *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing.* pp. 274–281 (1990)
7. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms* **17**(3), 409–423 (1994)
8. Goemans, M.X.: Minimum bounded degree spanning trees. In: *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on.* pp. 273–282. IEEE (2006)
9. Klein, P.N., Krishnan, R., Raghavachari, B., Ravi, R.: Approximation algorithms for finding low-degree subgraphs. *Networks* **44**(3), 203–215 (2004)
10. Könemann, J., Ravi, R.: A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing.* pp. 537–546. ACM (2000)
11. Könemann, J., Ravi, R.: Primal-dual meets local search: approximating mst's with nonuniform degree bounds. In: *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing.* pp. 389–395. ACM (2003)
12. Krishnan, R., Raghavachari, B.: The directed minimum-degree spanning tree problem. In: *FSTTCS. vol. 2245,* pp. 232–243. Springer (2001)

13. Ravi, R.: Rapid rumor ramification: Approximating the minimum broadcast time. In: Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on. pp. 202–213. IEEE (1994)
14. Ravi, R., Marathe, M.V., Ravi, S., Rosenkrantz, D.J., Hunt III, H.B.: Many birds with one stone: Multi-objective approximation algorithms. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing. pp. 438–447. ACM (1993)
15. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 661–670. ACM (2007)
16. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM (JACM) **32**(3), 652–686 (1985)
17. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (Apr 1975). <https://doi.org/10.1145/321879.321884>, <http://doi.acm.org/10.1145/321879.321884>
18. Yao, G., Zhu, D., Li, H., Ma, S.: A polynomial algorithm to compute the minimum degree spanning trees of directed acyclic graphs with applications to the broadcast problem. Discrete Mathematics **308**(17), 3951–3959 (2008)

## A Implementation and running time of AugSeqDegRed

We present an implementation of AugSeqDegRed that runs in  $O(\frac{1}{\epsilon^2}m \log^2 n)$  time. We discuss some implementation details of Layering, AugDFS and AugSeqDegRed, and analyse their contributions to the total running time in a single run of AugSeqDegRed.

### (1) Layering.

For every instance of Layering, computing the forest  $\{C_u^h\}_{u \in V \setminus (\cup_{i=0}^h B_i)}$  can be done in a single pass of breath-first search which takes  $O(m)$  time. Computing  $B_{h+1}$ , if necessary, is easily done by scanning the edge set  $E$  which also takes  $O(m)$  time. As the while-loop iterates for at most  $1 + \log_{1+\epsilon} n$  times, and due to Lemma 6 Layering is invoked for at most  $1 + \log_{1+\epsilon} n$  times, the overall contribution of Layering is  $O(\frac{1}{\epsilon^2}m \log^2 n)$ .

### (2) AugSeqDegRed.

Excluding the contributions of AugDFS and Layering, all AugSeqDegRed does is simply un-tagging all vertices and edges, scanning the edge set  $(u, v) \in E$  and deciding if  $C_u^h \neq C_v^h$  as well as modifying tree  $T$ . As tree components only get merged and never split, we can use the union-find data structure [17] to support querying whether  $C_u^h \neq C_v^h$  in  $O(\alpha(n))$  time. Every tree modification involves insertions and deletions of  $O(\frac{1}{\epsilon} \log n)$  edges, as well as merging  $O(\frac{1}{\epsilon} \log n)$  pairs of some tree components  $C_u^i$ . Using the link-cut tree, every edge insertion and deletion takes update time  $O(\log n)$ , and every component-merging takes time  $O(\alpha(n))$ . Since every tree modification merges two components in  $T \setminus S_k$ , there can be at most  $O(n)$  tree modifications throughout AugSeqDegRed. Therefore, the overall contribution of tree modifications is  $O(\frac{1}{\epsilon}n \log^2 n)$ . Hence, AugSeqDegRed’s exclusive contributions to the total running time would be  $O(\frac{1}{\epsilon}m\alpha(n) \log n + \frac{1}{\epsilon}n \log^2 n)$ .

### (3) AugDFS.

Now we analyse the overall time complexity induced by AugDFS invoked on line-5 of AugSeqDegRed. There are two technical issues to be resolved.

#### (a) How to enumerate untagged vertices $\in \rho_{u,v} \cap B_{i-1}$ ?

For each  $u_i \in B_i, \forall 0 \leq i \leq h$ , assign  $u_i$  a weight of  $i$ ; vertices that do not belong to any  $B_i$  have weight  $h + 1$ . By Corollary 1, to enumerate vertices  $\in \rho_{u,v} \cap B_{i-1}$ , it suffices to enumerate the lightest vertices on  $\rho_{u,v}$ , which can be done using a link-cut tree data structure [16] built on  $T$ , each enumeration taking  $O(\log n)$  amortized time. When a vertex gets tagged, we change its weight to  $h + 1$ , and so future enumerations on  $\rho_{u,v} \cap B_{i-1}$  may skip this tagged vertex.

#### (b) How to enumerate unmarked $z$ connected by an untagged edge $(w, z)$ such that $C_z^{i-2} \neq C_w^{i-2}$ ?

Each  $w$  decrementally maintains a list of all its neighbours. While we scan the list, if the next edge  $(w, z)$  satisfies both conditions that  $C_z^{i-2} \neq C_w^{i-2}$  and  $z$  is unmarked, then the algorithm starts a new iteration and recur; either way we cross the edge  $(w, z)$  off the list. In this way, every edge appears for at most once. Thus the total time of this part is  $O(m\alpha(n))$ ; the additional  $\alpha(n)$  factor comes from the union-find data structure that helps deciding if  $C_z^{i-2} \neq C_w^{i-2}$ .

Note that (a)'s running time is always dominated by (b)'s, then the overall complexity of AugDFS is  $O(m\alpha(n) + \frac{1}{\epsilon}n \log^2 n)$ .

Summing up (1)(2)(3), the total running time of is dominated by time complexity of Layering which is  $O(\frac{1}{\epsilon^2}m \log^2 n)$ .