Fast Distributed Approximation for TAP and 2-Edge-Connectivity[*][†]

Keren Censor-Hillel[1] and Michal Dory[2]

1 Technion, Department of Computer Science, Haifa, Israel
ckeren@cs.technion.ac.il

2 Technion, Department of Computer Science, Haifa, Israel
smichald@cs.technion.ac.il

Abstract

The *tree augmentation problem (TAP)* is a fundamental network design problem, in which the input is a graph G and a spanning tree T for it, and the goal is to augment T with a minimum set of edges Aug from G, such that $T \cup Aug$ is 2-edge-connected.

TAP has been widely studied in the sequential setting. The best known approximation ratio of 2 for the weighted case dates back to the work of Frederickson and JáJá, SICOMP 1981. Recently, a 3/2-approximation was given for the unweighted case by Kortsarz and Nutov, TALG 2016, and recent breakthroughs by Adjiashvili, SODA 2017, and by Fiorini et al., 2017, give approximations better than 2 for bounded weights.

In this paper, we provide the first fast *distributed* approximations for TAP. We present a distributed 2-approximation for weighted TAP which completes in $O(h)$ rounds, where h is the height of T. When h is large, we show a much faster 4-approximation algorithm for the unweighted case, completing in $O(D + \sqrt{n}\log^* n)$ rounds, where n is the number of vertices and D is the diameter of G.

Immediate consequences of our results are an $O(D)$-round 2-approximation algorithm for the minimum size 2-edge-connected spanning subgraph, which significantly improves upon the running time of previous approximation algorithms, and an $O(h_{MST} + \sqrt{n}\log^* n)$-round 3-approximation algorithm for the weighted case, where h_{MST} is the height of the MST of the graph. Additional applications are algorithms for verifying 2-edge-connectivity and for augmenting the connectivity of any connected spanning subgraph to 2.

Finally, we complement our study with proving lower bounds for distributed approximations of TAP.

1998 ACM Subject Classification G.2.2 Graph Theory: Graph algorithms

Keywords and phrases approximation algorithms, distributed network design, connectivity augmentation

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2017.21

[*] Supported in part by the Israel Science Foundation (grant 1696/14).
[†] The full version of the paper is available at https://arxiv.org/abs/1711.03359.

1 Introduction

The tree augmentation problem (TAP) is a central problem in network design. In TAP, the input is a 2-edge-connected[1] graph G and a spanning tree T of G, and the goal is to augment T to be 2-edge-connected by adding to it a minimum size (or a minimum weight) set of edges from G. Augmenting the connectivity of T makes it resistant to any single link failure, which is crucial for network reliability. TAP is extensively studied in the sequential setting, with several classical 2-approximation algorithms [9, 13, 15, 18], as well as recent advances with the aim of achieving better approximation factors [1, 5, 8, 21].

TAP is part of a wider family of *connectivity augmentation* problems. Finding a minimum spanning tree (MST) is another prime example for a problem in this family, but, although an MST is a low-cost backbone of the graph, it cannot survive even one link failure. Hence, in order to guarantee stronger reliability, it is vital to find subgraphs with higher connectivity. The motivation for considering TAP is for the case that adding any new edge to the backbone incurs a cost, and hence if we are already given a subgraph with some connectivity guarantee then we would naturally like to augment it with additional edges of minimum number or weight, rather than to compute a well-connected low-cost subgraph from scratch. Connectivity augmentation problems also serve as building blocks in other connectivity problems, such as computing the minimum k-edge-connected subgraph. A natural approach is to start with building a subgraph that satisfies some connectivity guarantee (e.g., a spanning tree), and then augment it to have stronger connectivity.

Since the main motivation for TAP is improving the reliability of distributed networks, it is vital to consider TAP also from the distributed perspective. In this paper, we initiate the study of distributed connectivity augmentation and present the first distributed approximation algorithms for TAP. We do so in the CONGEST model [29], in which vertices exchange messages of $O(\log n)$ bits in synchronous rounds, where we show fast algorithms for both the unweighted and weighted variants of the problem. In addition to fast approximations for TAP, our algorithms have the crucial implication of providing efficient algorithms for approximating the minimum 2-edge-connected spanning subgraph, as well as several related problems, such as verifying 2-edge-connectivity and augmenting the connectivity of any spanning connected subgraph to 2. Finally, we complement our study with proving lower bounds for distributed approximations of TAP.

1.1 Our Contributions

Distributed approximation algorithms for TAP

Our first main contribution is the first distributed approximation algorithm for TAP. In particular, our algorithm provides a 2-approximation for weighted TAP in the CONGEST model, summarized as follows.

▶ **Theorem 1.1.** *There is a distributed 2-approximation algorithm for weighted TAP in the CONGEST model that runs in $O(h)$ rounds, where h is the height of the tree T.*

The approximation ratio of our algorithm matches the best approximation ratio for weighted TAP in the sequential setting. Its round complexity of $O(h)$ is tight if $h = O(D)$, where D is the diameter of G. This happens, for example, when T is a BFS tree, and follows from a lower bound of $\Omega(D)$ rounds which we show in Section 6.

[1] A graph G is 2-edge-connected if it remains connected after the removal of any single edge.

However, the height h of the spanning tree T may be large, even if the diameter of G is small, which raises the question of whether the dependence on h is necessary. We address this question by providing an algorithm for *unweighted* TAP that has a round complexity of $O(D + \sqrt{n}\log^* n)$ rounds, which is significantly smaller for large values of h. This only comes at the price of a slight increase in the approximation ratio, from 2 to 4.

▶ **Theorem 1.2.** *There is a distributed 4-approximation algorithm for unweighted TAP in the CONGEST model that runs in $O(D + \sqrt{n}\log^* n)$ rounds.*

Applications

The key application of our TAP approximation algorithm is an $O(D)$-round 2-approximation algorithm for the minimum size 2-edge-connected spanning subgraph problem (2-ECSS), which is obtained by building a BFS tree and augmenting it to a 2-edge-connected subgraph using our algorithm.

▶ **Theorem 1.3.** *There is a distributed 2-approximation algorithm for unweighted 2-ECSS in the CONGEST model that completes in $O(D)$ rounds.*

The time complexity of our algorithm improves significantly upon the time complexity of previous approximation algorithms for 2-ECSS, which are $O(n)$ rounds for a $\frac{3}{2}$-approximation [22] and $O(D + \sqrt{n}\log^* n)$ rounds for a 2-approximation [33].

In addition, our weighted TAP algorithm implies a 3-approximation for *weighted* 2-ECSS. Other applications of our algorithms are an $O(D)$-round algorithm for verifying 2-edge-connectivity, and an algorithm for augmenting the connectivity of any connected spanning subgraph H of G from 1 to 2.

Lower bounds

We complement our algorithms by presenting lower bounds for TAP. We first show that approximating TAP is a global problem which requires $\Omega(D)$ rounds even in the LOCAL model [25], where the size of messages is not bounded.

▶ **Theorem 1.4.** *Any distributed α-approximation algorithm for weighted TAP takes $\Omega(D)$ rounds in the LOCAL model, where $\alpha \geq 1$ can be any polynomial function of n. This holds also for unweighted TAP, if $1 \leq \alpha < \frac{n-1}{2c}$ for a constant $c > 1$.*

Theorem 1.4 implies that if $h = O(D)$ then our TAP approximation algorithms have an optimal round complexity. We also consider the case of $h = \omega(D)$ and show a family of graphs, based on the construction in [32], for which $\Omega(h)$ rounds are needed in order to approximate weighted TAP, were $h = \Theta(\frac{\sqrt{n}}{\log n})$.

▶ **Theorem 1.5.** *For any polynomial function $\alpha(n)$, there is a $\Theta(n)-$vertex graph of diameter $\Theta(\log n)$ for which any (perhaps randomized) distributed $\alpha(n)$-approximation algorithm for weighted TAP with an instance tree $T \subseteq G$ of height $h = \Theta(\frac{\sqrt{n}}{\log n})$ requires $\Omega(h)$ rounds in the CONGEST model.*

Theorem 1.5 implies that our algorithm for weighted TAP is optimal on these graphs. In particular, there cannot be an algorithm with a complexity of $O(f(h))$ for a sublinear function f.

1.2 Technical overview of our algorithms

As an introduction, we start by showing an $O(h)$-round 2-approximation algorithm for *unweighted* TAP, which allows us to present some of the key ingredients in our algorithms. Later, we explain how we build on these ideas and extend them to give an algorithm for the weighted case, and a faster algorithm for unweighted TAP.

Unweighted TAP

A natural approach for constructing a distributed algorithm for unweighted TAP could be to try to simulate the sequential 2-approximation algorithm of Khuller and Thurimella [18]. In their algorithm, the input graph G is first converted into a modified graph G'. Then, the algorithm finds a directed MST in G', which induces a corresponding augmentation in G.

When considered in the distributed setting, this approach imposes two difficulties. The first is that we cannot simply modify the input graph, because it is the graph that represents the underlying distributed network, whose topology is given and not under our control. The second is in the directed MST procedure, as finding a directed MST efficiently in the distributed setting seems to be difficult. The currently best known time complexity of this problem is $O(n^2)$ for an asynchronous setting [14], which is trivial in the CONGEST model.

We overcome the above using two key ingredients. First, we bring into our construction the tool of computing lowest common ancestors (LCAs). We show that building G' and simulating a distributed computation over it can be done by an efficient computation of LCAs, and we achieve the latter by leveraging the *labeling scheme* for LCAs presented in [2].

Second, we drastically diverge from the Khuller-Thurimella framework by replacing the expensive directed MST construction by a completely different procedure. Roughly speaking, we show that the simple structure of G' allows us to find an optimal augmentation in G' efficiently by scanning the input tree T from the leaves to the root and performing the following procedure. Each vertex sends to its parent information about edges that may be useful for the augmentation since they *cover* many edges of the tree, and the vertices use the LCA labels in order to decide which edges to add to the augmentation.

While a direct implementation of this would result in much information that is sent through the tree, we show that at most two edges need to actually be sent by each vertex. Thus, applying the labeling scheme and scanning the tree T result in a time complexity of $O(h)$ rounds, where h is the height of T. Finally, we prove that an optimal augmentation in G' gives a 2-approximation augmentation for G, which gives a 2-approximation for unweighted TAP in $O(h)$ rounds.

Weighted TAP

Our algorithm for the unweighted case relies heavily on the fact that we can compare edges and decide which one is the best for the augmentation according to the number of edges they cover in the tree. However, once the edges have weights, it is not clear how to compare edges. This is because of the tension between light edges that cover only few edges and heavier edges that cover many edges. Therefore, Theorem 1.1, which applies for the weighted case, cannot be directly obtained according to the above description. Hence, sending two edges per vertex up in the tree, as we do for the unweighted case, is insufficient. That is, the number of edges that should be considered for the augmentation could be much larger and vertices cannot decide locally which edges are useful without feedback from their ancestors in the tree.

Nevertheless, we show how to overcome this by introducing a technique of having each vertex send to its parent edges with *altered weights*. The trick here is that we modify the weight that is sent for an edge in a way that captures the cost for covering each edge of the tree. This successfully addresses the competing needs of covering as many tree edges as possible, while using the lightest possible edges, and allows focusing on a smaller number of edges that may be useful for the augmentation. Finally, using standard pipelining, this gives a time complexity of $O(h)$ rounds for the weighted case as well.

Faster unweighted TAP

Both of our aforementioned algorithms rely on scanning the tree T, which results in a time complexity that is linear in the height h of the tree T. In order to avoid the dependence on h, one must be a able to add edges to the augmentation without scanning the whole tree.

However, if a vertex v does not get information about the edges added to the augmentation by the vertices in the whole subtree rooted at v, then it may add additional edges in order to cover tree edges that are already covered. But then we are no longer guaranteed to get an optimal augmentation in G', or even a good approximation for it.

Nevertheless, we are still able to show a faster algorithm for unweighted TAP, which completes in $O(D + \sqrt{n} \log^* n)$ rounds. The key ingredient in our algorithm is breaking the tree T into fragments and applying our 2-approximation for unweighted TAP algorithm on each fragment separately, as well as on the tree of fragments. Since our algorithm does not scan the whole tree, it may add different edges to cover the same tree edges, which makes the analysis much more involved. The approximation ratio analysis is based on dividing the edges to different types and bounding the number of edges of each type separately, using a subtle case-analysis. Although our algorithm does not find an optimal augmentation in G', it gives a 2-approximation for it, which results in a 4-approximation augmentation for the original graph G.

1.3 Related Work

Sequential algorithms for TAP

TAP is intensively studied in the sequential setting. Since TAP is NP-hard, approximation algorithms for it have been studied. The first 2-approximation algorithm for weighted TAP was given by Frederickson and JáJá [9], and was later simplified by Khuller and Thurimella [18]. Other 2-approximation algorithms for weighted TAP are the primal-dual algorithm of Goemans et al. [13], and the iterative rounding algorithm of Jain [15].

Recently, a new algorithm achieved a 1.5-approximation for unweighted TAP [21], and recent breakthroughs give approximations better than 2 for bounded weights [1,8]. Achieving approximation better than 2 for the general weighted case is a central open question. See [17,20] for surveys about approximation algorithms for connectivity problems. Also, the related work in [1] gives an overview of many recent sequential algorithms for TAP.

Related work in the distributed setting

While ours are the first distributed approximation algorithms for TAP itself, there are important related studies in the distributed setting.

MST. In the distributed setting, finding an MST, which is a minimum weight subgraph with connectivity 1, is a fundamental and well studied problem (see, e.g., [6,7,10,11,23,28]). The first distributed algorithm for this problem is the GHS algorithm that works in $O(n \log n)$ time [10]. Following algorithms improved the round complexity to $O(D + \sqrt{n} \log^* n)$ [11,23].

k-ECSS. For the minimum weight 2-edge-connected spanning subgraph (2-ECSS) problem, there is a distributed algorithm of Krumke et al. [22]. Their approach is finding a specific spanning tree and then augmenting it to a 2-edge-connected graph. In the unweighted case, they augment a DFS tree following the sequential algorithm of Khuller and Vishkin [19], which results in an $O(n)$-round $\frac{3}{2}$-approximation algorithm for 2-ECSS. In the weighted case they augment an MST and suggest a general $O(n \log n)$-round 2-approximation algorithm for weighted TAP, which gives an $O(n \log n)$-round 3-approximation algorithm for 2-ECSS. Our algorithms for TAP imply faster approximations for unweighted and weighted 2-ECSS.

Another distributed algorithm for *unweighted* k-ECSS is an $O(k(D + \sqrt{n} \log^* n))$-round algorithm of Thurimella [33] that finds a sparse k-edge-connected subgraph. The general framework of the algorithm is to repeatedly find maximal spanning forests in the graph and remove their edges from the graph (this framework is also described in sequential algorithms [17, 26]). This gives a k-edge-connected spanning subgraph with at most $k(n-1)$ edges. Since any k-edge-connected subgraph has at least $\frac{kn}{2}$ edges, since the degree of each vertex is at least k, this approach guarantees a 2-approximation for unweighted k-ECSS.

Fault-tolerant tree structures. Another related problem is the construction of fault-tolerant tree structures. Distributed algorithms for constructing fault tolerant BFS and MST structures are given in [12], producing sparse subgraphs of the input graph G that contain a BFS (or an MST) of $G \setminus \{e\}$ for each edge e, for the purpose of maintaining the functionality of a BFS (or an MST) even when an edge fails. However, TAP is different from these problems in several aspects. First, we augment a specific spanning tree T rather then build the whole structure from scratch. In addition, since we need to preserve only connectivity when an edge fails and not the functionality of a BFS or an MST, optimal solutions for TAP may be much cheaper.

Additional related problems. Another connectivity augmentation problem studied in the distributed setting is the Steiner Forest problem [16, 24]. There are also distributed algorithms for finding the 2-edge-connected and 3-edge-connected components of a connected graph [30, 31], and distributed algorithms that decompose a graph with large connectivity into many disjoint trees, while almost preserving the total connectivity through the trees [4].

1.4 Preliminaries

For completeness, we first formally define the notion of edge connectivity.

▶ **Definition 1.6.** An undirected graph G is *k-edge-connected* if it remains connected after the removal of any $k-1$ edges.

The Tree Augmentation Problem (TAP). In TAP, the input is an undirected 2-edge-connected graph G with n vertices, and a spanning tree T of G. The goal is to add to T a minimum size (or a minimum weight) set of edges Aug from G, such that $T \cup Aug$ is 2-edge-connected. In the weighted version, each edge has a non-negative weight, and we assume that the weights of the edges can be represented in $O(\log n)$ bits.

▶ **Definition 1.7.** An edge e in a connected graph G is a *bridge* in G if $G \setminus \{e\}$ is disconnected.

▶ **Definition 1.8.** A non-tree edge $e = \{u, v\}$ *covers* the tree edge e' if e' is on the unique path in T between u and v, i.e., if e' is not a bridge in $T \cup \{e\}$.

A graph G is 2-edge-connected if and only if it does not contain bridges. Hence, augmenting the connectivity of T requires covering of all the tree edges.

Models of distributed computation. In the distributed CONGEST model [29], the network is modeled as an undirected connected graph $G = (V, E)$. Communication takes place in synchronous rounds. In each round, each vertex can send a message of $O(\log n)$ bits to each of its neighbors. The time complexity of an algorithm is measured by the number of rounds. Our algorithms work in the CONGEST model, where some of our lower bounds hold also in the stronger LOCAL model [25], where the size of messages is not bounded.

In the distributed setting, the input to TAP is a rooted spanning tree T of G with root r, whose height is denoted by h. The tree T is given to the vertices locally, that is, each vertex knows which of its adjacent edges is in T and which of those leads to its parent in T.[2] For each vertex $v \neq r$, we denote by $p(v)$ the parent of v in T. The output is a set of edges Aug, such that $T \cup Aug$ is 2-edge-connected. In the distributed setting it is enough that at the end of the algorithm each vertex knows which of the edges incident to it are added to Aug.

All the messages sent in our algorithms consist of a constant number of ids, labels and weights, hence the maximal message size is bounded by $O(\log n)$ bits, as required in the CONGEST model.

Roadmap. In Section 2, we describe our $O(h)$-round 2-approximation algorithm for unweighted TAP, and in Section 3 we extend it to the weighted case. In Section 4, we show applications of these algorithms, in particular for approximating 2-ECSS, and in Section 5 we present our faster algorithm for unweighted TAP. We present lower bounds for TAP in Section 6, and discuss questions for future research in Section 7. Full details of our algorithm for weighted TAP, as well as main proofs for it appear in Appendix A. Full details and proofs appear in the full version [3].

2 A 2-approximation for Unweighted TAP in $O(h)$ rounds

As an introduction, we start by describing an $O(h)$-round 2-approximation algorithm, A_{TAP}, for unweighted TAP. Here we give a high-level description of A_{TAP}. Full details and proofs appear in [3]. The general structure of A_{TAP} is as follows. It starts by building a related virtual graph G' as in [18]. Afterwards, we diverge completely from the approach of [18] since we cannot simulate it efficiently in the distributed setting, as explained in the introduction. Instead, A_{TAP} finds an optimal augmentation in G', and converts it to an augmentation Aug in G. We show that the size of Aug is at most twice the size of an optimal augmentation in G. All the communication in the algorithm is on the edges of the graph G, since G' is a virtual graph. In order to simulate the algorithm on G we use labels that represent the edges of G'.

Building G' from G. A_{TAP} starts by building a related *undirected* virtual graph G'. To simplify the presentation of the algorithm it is convenient to give an orientation to the edges of G'. However, we emphasize that G' is an undirected graph, that is, we do not address the notion of directed connectivity. The graph G' is defined as follows (as in [18]). The graph G' includes all the edges of T, and they are all oriented towards the root r of T. For every

[2] If a root and orientation are not given, we can find a root r and orient all the edges towards r in $O(h)$ rounds using standard techniques.

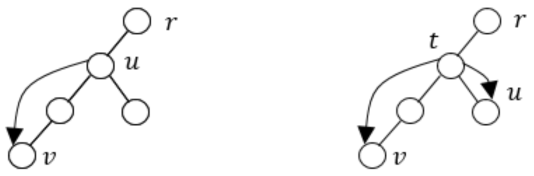

Figure 1 There are two cases for every non-tree edge in G. The left graph shows the first case, where the edge $\{u, v\}$ is between an ancestor and a descendant in T. The right graph shows the second case, where $t = LCA(u, v)$.

non-tree edge $e = \{u, v\}$ in G there are two cases (see Figure 1). If u is an ancestor of v in T, we add the edge $\{u, v\}$ to G', oriented from u to v. Otherwise, denote $t = LCA(u, v)$. In this case we add to G' the edges $\{t, u\}$ and $\{t, v\}$, oriented from t to u and to v, respectively.

In order to build the graph G' in the distributed setting, we use the *labeling scheme* for LCAs of Alstrup et al. [2]. This labeling scheme assigns labels of size $O(\log n)$ bits to the vertices of a rooted tree with n vertices, such that given the labels of u and v it is possible to infer the label of their LCA. The algorithm for computing the labels can be implemented in $O(h)$ rounds in the distributed setting.

Since G' is a virtual graph, the rest of the communication in the algorithm is only over the tree edges. In order to simulate the rest of the algorithm over G', it is enough that each vertex knows only the tree edges incident to it (which is its input), and the labels of the non-tree edges incoming to it in G'. Using the labeling scheme, each vertex learns this information by exchanging labels with all of its neighbors. For more details, see [3].

The Correspondence between G and G'. A non-tree edge $e = \{u, v\}$ in G covers all the edges in the unique path in T between u in v. To build G' from G, for each non-tree edge $e \in G$, we added one or two corresponding edges to G', which together cover exactly the same tree edges as e. This allows us to show that an optimal augmentation in G' gives a 2-approximation augmentation in G, when we replace each edge of the augmentation in G' by a corresponding edge in G. For full details and proofs see [3].

Finding an optimal augmentation in G'. In the graph G', all the edges that are not tree edges are between an ancestor and a descendant of it in T. This allows us to compare edges and define the notion of *maximal* edges. Intuitively, the notion of maximal edges would capture our goal that during the algorithm, when we cover a tree edge, we would like to cover it by an edge that reaches the highest ancestor possible, allowing us to cover many tree edges simultaneously. This motivates the following definition. Let v be a vertex in the tree, and let $e = \{u, w\}$ and $e' = \{u', w'\}$ be two edges between ancestors u, u' of v and descendants w, w' of v. We say that e is the *maximal* edge among e and e' if and only if u is an ancestor of u'. If $u = u'$ we can choose arbitrarily one of them to be the maximal edge.

In order to cover all tree edges of G', we assign each vertex $v \neq r$ in G' with the responsibility of covering the tree edge $\{v, p(v)\}$. In our algorithm A_{Aug} for finding an optimal augmentation in G', the tree T is scanned from the leaves to the root, and whenever a tree edge that is still not covered is reached, it is covered by the vertex responsible for it, using the maximal edge possible. For doing so, it is enough that each vertex v sends to its parent information about at most two edges. One of them is the maximal edge that was already added to the augmentation in order to cover the tree edge $\{v, p(v)\}$, and one is the maximal edge possible to cover the tree edge $\{v, p(v)\}$. We show that using the LCA labels of edges, vertices can compute easily if a tree edge is covered, and which edge is the maximal edge. The full description of the algorithm A_{Aug} is given in [3].

Correctness proof. Denote by A the solution obtained by A_{Aug}, and by A^* an optimal augmentation in G'. We show in [3] that A is an optimal augmentation in G'. The key ingredient is to show a one-to-one mapping from A to A^*. For each edge $e \in A$, we look at the tree edge $t(e) = \{v, p(v)\}$ where v is the vertex that decides to add e to the augmentation. We map e to an edge $e^* \in A^*$ that covers $t(e)$. Showing that this mapping is one-to-one is based on the fact that when we add a new edge to the augmentation, it is the maximal edge possible. In addition, we show that the time complexity of the algorithm is $O(h)$ rounds. Since an optimal augmentation in G' corresponds to a 2-approximation augmentation in G, this gives the following.

▶ **Theorem 2.1.** *There is a distributed 2-approximation algorithm for unweighted TAP in the CONGEST model that runs in $O(h)$ rounds, where h is the height of the tree T.*

3 A 2-approximation for Weighted TAP in $O(h)$ rounds

In this section, we give a high-level description of our algorithm for weighted TAP. Full details and main proofs appear in Appendix A. Our algorithm for weighted TAP, A_{wTAP}, has the same structure of A_{TAP}. It starts by building the same virtual graph G', and then it finds an optimal augmentation in G'. The only difference in building G' is that now each edge e is replaced by one or two edges in G' with the same weight that e has. The proof that an optimal augmentation in G' corresponds to an augmentation in G with at most twice the cost of an optimal augmentation in G is the same as in the unweighted case.

The difference is in finding an optimal augmentation in G'. In the unweighted case, for each vertex v, the only edge incoming to v in G' that was useful for the algorithm was the maximal edge. However, when edges have weights, potentially all the edges incoming to v may be useful for the algorithm, and we can no longer use the notion of *maximal* edges in order to compare edges. This is because of the tension between heavy edges that cover many edges of the tree, and light edges that cover less edges of the tree. To overcome this obstacle, we introduce a new technique of *altering* the weights of the edges we send in the algorithm. We next describe how our new approach allows us to find an optimal augmentation in G'.

Finding an Optimal Augmentation in G'. In the weighted case, it is not clear anymore how to compare edges. A vertex v may have many optional edges that cover $\{v, p(v)\}$, where one of them has the minimum weight, but other edges cover more tree edges. In order to cover the tree edge between v and its parent, it is best to add the edge with minimum weight. However, in order to cover additional tree edges, we may want to add one of the other edges. Since v alone does not have enough information to resolve this trade-off, one could have the vertices propagate all the edges up in the tree, but this would accumulate to too much information, which would render the algorithm very slow.

Instead, we pinpoint the exact source of the trade-off between length and weight of covering edges, as follows. Let min_v be the weight of the minimum weight edge that covers $\{v, p(v)\}$. The intuition behind our approach is that in order to cover the tree edge $\{v, p(v)\}$ we must pay at least min_v. Thus, min_v captures the cost of covering this tree edge. Therefore, before sending to its parent information about relevant edges, v alters their weights by reducing from them the weight min_v. In terms of $p(v)$, the reduced weights represent the extra cost incurred when choosing any of these edges in order to cover additional tree edges. Using the reduced weights in our algorithm instead of the original ones is crucial for selecting which edges to add to the augmentation, and allows to divide the weight of an edge in a way that captures the cost for covering each tree edge. In addition, we show that using this approach, sending information about at most h edges from each vertex to its parent suffices for selecting the best edges for the augmentation.

The Algorithm. Our algorithm consists of two traversals of the tree: from the leaves to the root and vice versa. As in A_{Aug}, each vertex v is responsible for covering the tree edge $\{v, p(v)\}$.

In the first traversal, each vertex v computes the weight min_v of the minimum weight edge that covers the tree edge $\{v, p(v)\}$ according to the weights of the edges it receives from its children, and the weights of the edges incoming to it. It also computes the weights of the minimum weight edges that cover the path from v to each of its ancestors u, according to the weights v receives in the algorithm. Then, v subtracts min_v from the weights of these edges, and sends them to its parent with the altered weights.

In the second traversal, we scan the tree from the root to the leaves. Each child v of r adds to the augmentation the edge having weight min_v. It informs the relevant child who sent it, if exists, and informs its other children it did not add their edges. Each internal vertex v receives from its parent a message that indicates whether one of the edges it sent was added to the augmentation by one of its ancestors or not. In the former case, v learns that this edge was added to the augmentation and forwards the message to the relevant child who sent it, if such exists. Otherwise, the tree edge $\{v, p(v)\}$ is still not covered, and v adds to the augmentation the edge having weight min_v. It informs the relevant child who sent it, if exists, and informs its other children that their edges were not added to the augmentation.

A full description of the algorithm is given in Appendix A.1.

Time analysis. In our algorithm, each vertex sends to its parent information about at most h edges. If each vertex waits to receive all the messages from its children, before sending messages to its parent, it would result in a time complexity of $O(h^2)$ rounds. However, using pipelining we get a time complexity of $O(h)$ rounds. The main intuition is that although each vertex v may receive h different messages from each of its children during the algorithm, in order for v to send to its parent $p(v)$ the message concerning an ancestor u, the vertex v only needs to receive one message from each of its children concerning the ancestor u. Hence, if all the vertices send the messages according to increasing order of heights of their ancestors, we can pipeline the messages and get a time complexity of $O(h)$ rounds. The full proof appears in [3].

Correctness proof. In Appendix A.3, we give a correctness proof for the algorithm. The challenge in establishing the correctness of our algorithm lies in the fact that the vertices use altered weights rather than the original ones. Nevertheless, we show that our intuition behind choosing these altered weights faithfully captures the essence of finding an augmentation in the weighted case. The key ingredient we use in our proof is giving a *cost* to each edge of T. For a tree edge $t = \{v, p(v)\}$, we assign the cost $c(t) = min_v$. We are then able to prove the following.

(I) The sum of the costs of the edges of T is equal to the cost of the augmentation obtained by the algorithm.

(II) The cost of any augmentation of G' is at least the sum of costs of the edges of T.

To show (1), for each edge e added to the augmentation, we assign a tree path P_e, as follows. For an edge $e = \{u, x\}$ added to the augmentation by a vertex v, where u is an ancestor of x in the tree, we assign the tree path P_e between x to $p(v)$. Then, we show that $w(e) = \sum_{t \in P_e} c(t)$. The proof is based on the fact that the cost of the tree edge $\{v, p(v)\}$

is min_v by definition, however, its cost is also $w(e) - \sum_{v' \in V'} min_{v'}$ where V' are all the vertices on the path between x and v, excluding v. This is since each vertex v' in this path reduces $min_{v'}$ from the weight of e before sending it to its parent, and since e is the edge v chooses to add to the augmentation. This gives $w(e) = min_v + \sum_{v' \in V'} min_{v'} = \sum_{t \in P_e} c(t)$. In addition, we show that the paths P_e are disjoint and their union is the entire tree T, which proves (1). The proof is based on the fact that the edges of the augmentation cover all tree edges, and that a vertex v adds an edge to the augmentation only if the tree edge $\{v, p(v)\}$ is not already covered by an edge added by one of its ancestors. We prove (2) in a similar way.

In conclusion, our algorithm gives an optimal augmentation in G', which gives a 2-approximation augmentation in G.

▶ **Theorem 1.1.** *There is a distributed 2-approximation algorithm for weighted TAP in the CONGEST model that runs in $O(h)$ rounds, where h is the height of the tree T.*

4 Applications

In this section, we discuss applications of our algorithms, and show they provide efficient algorithms for additional related problems.

Minimum Weight 2-Edge-Connected Spanning Subgraph. In the minimum weight 2-edge-connected spanning subgraph problem (2-ECSS), the input is a 2-edge-connected graph G, and the goal is to find the minimum weight 2-edge-connected spanning subgraph of G. Using A_{TAP} we have the following.

▶ **Theorem 1.3.** *There is a distributed 2-approximation algorithm for unweighted 2-ECSS in the CONGEST model that completes in $O(D)$ rounds.*

Proof. We apply A_{TAP} on G and a BFS tree T of G. Finding a BFS tree takes $O(D)$ rounds [29], and A_{TAP} takes $O(D)$ rounds since T is a BFS tree. The size of the augmentation Aug is at most $n-1$ because in the worst case we add a different edge in order to cover each tree edge. Hence, $T \cup Aug$ is a 2-edge-connected subgraph with at most $2(n-1)$ edges. Note that any 2-edge-connected graph has at least n edges, which implies a 2-approximation, as claimed. ◀

The above algorithm has a better time complexity compared to the algorithm of [22], which finds a $\frac{3}{2}$-approximation to 2-ECSS in $O(n)$ rounds. In the algorithm of [22], the augmented tree T is a DFS tree rather then a BFS tree. The same proof of [19, 22] gives that if we apply A_{TAP} on G and a DFS tree we also obtain a $\frac{3}{2}$-approximation to 2-ECSS in $O(n)$ rounds. For weighted 2-ECSS, using A_{wTAP} gives the following.

▶ **Theorem 4.1.** *There is a distributed 3-approximation algorithm for weighted 2-ECSS in the CONGEST model that completes in $O(h_{MST} + \sqrt{n} \log^* n)$ rounds, where h_{MST} is the height of the MST.*

Proof. We follow the same approach of [22]. We start by constructing an MST, which takes $O(D + \sqrt{n} \log^* n)$ rounds [23], and then we augment it using A_{wTAP} in $O(h_{MST})$ rounds.[3] Let $w(A)$ be the weight of an optimal solution A to weighted 2-ECSS. Since both the MST and an optimal augmentation have weights at most $w(A)$, and since our algorithm for

[3] We assume that the MST is unique. Otherwise, h_{MST} is the height of the MST we construct.

weighted TAP gives a 2-approximation, this approach gives a 3-approximation for weighted 2-ECSS. ◀

This algorithm has a better time complexity compared to the algorithm of [22], which takes $O(n \log n)$ rounds, with the same approximation ratio.

Increasing the Edge-Connectivity from 1 to 2. A_{wTAP} is a 2-approximation algorithm for TAP, but can also be used to increase the connectivity of any spanning subgraph H of G from 1 to 2. In order to do so, we start by finding a spanning tree T of H. Note that it is not enough to apply A_{TAP} on T and take the augmentation obtained, since edges from H can be added to the augmentation with no cost. Hence, we apply A_{wTAP} on G and T, where we set the weights of all the edges of H to be 0. The augmentation Aug we obtain is a set of edges such that $T \cup Aug$ is 2-edge-connected, which also implies that $H \cup Aug$ is 2-edge-connected. In addition, its cost is at most twice the cost of an optimal augmentation of H, because any augmentation of H corresponds to an augmentation of T with the same cost, and Aug is a 2-approximation to the optimal augmentation of T. The time complexity is $O(D_H)$ rounds where D_H is the diameter of H, since finding a spanning tree T of H takes $O(D_H)$ rounds and applying A_{wTAP} takes $O(D_H)$ rounds because it is the height of T.

Verifying 2-Edge-Connectivity. The algorithm A_{TAP} can be used in order to verify if a connected graph G is 2-edge-connected in $O(D)$ rounds, where at the end of the algorithm all the vertices know if G is 2-edge-connected.[4] We start by building a BFS tree T of G and then apply A_{TAP} to G and T. Note that when we find an optimal augmentation in G' by A_{Aug}, each vertex v is responsible to cover the tree edge $\{v, p(v)\}$. If the graph G is 2-edge-connected, all the edges can be covered. If the graph G is not 2-edge-connected, then there is a tree edge $\{v, p(v)\}$ that is a bridge in the graph, and hence cannot be covered by any edge in G. In such a case, v identifies that it cannot cover the edge and hence the graph is not 2-edge-connected. Therefore, after scanning the tree from the leaves to the root in A_{Aug}, we can distinguish between these two cases, which takes $O(D)$ rounds. The root r can distribute the information to all the vertices in $O(D)$ rounds as well.

5 A 4-approximation for Unweighted TAP in $\widetilde{O}(D + \sqrt{n})$ rounds

The time complexity of A_{TAP} and A_{wTAP} is linear in the height of T. When h is large, we suggest a much faster $O(D + \sqrt{n} \log^* n)$-round algorithm for unweighted TAP. Here we give a high-level description of the algorithm, full details and proofs appear in [3].

The structure of the algorithm is the same as the structure of A_{TAP}. It starts by building the same virtual graph G', and then it finds an augmentation in G'. However, since we want to reduce the time complexity, our algorithm cannot scan the whole tree anymore. Therefore, we can no longer use directly the LCA labeling scheme and the algorithm A_{Aug} for finding an optimal augmentation. To overcome this, we break the tree T into fragments, and we divide the algorithm into local parts, in which we communicate in each fragment separately, and to global parts, in which we coordinate between different fragments over a BFS tree. This approach is useful also in other distributed algorithms for global problems, such as finding an MST [23] or a minimum cut [27]. The challenge is showing that this approach guarantees a

[4] A verification algorithm with the same complexity can also be deduced from the edge-biconnectivity algorithm of Pritchard [30].

good approximation. Since our algorithm does not scan the whole tree T it may add different edges in order to cover the same tree edges, which makes the analysis much more involved.

Building G' from G. In order to build G' from G we use the labeling scheme for LCAs that we used in A_{TAP}. However, applying this scheme directly takes $O(h)$ rounds. We show how to compute all the relevant LCAs more efficiently in $O(D+\sqrt{n})$ rounds. The idea is to apply the labeling scheme on each fragment separately to obtain *local labels*, and to apply the labeling scheme on the tree of fragments to obtain *global labels*. We show that using the local and global labels, and additional information on the structure of the tree of fragments, each vertex can compute all the edges incoming to it in G'. For full details see [3].

Finding an optimal augmentation in G'. In order to find an augmentation in G', we need to cover tree edges between fragments (*global edges*) and tree edges in the same fragment (*local edges*). For doing so, we start by computing all the maximal edges that cover the global edges. To cover all the global edges, one approach could be to add all these maximal edges to the augmentation. However, this cannot guarantee a good approximation. Instead, we apply A_{Aug} on the tree of fragments in order to cover all the global edges. Then, we apply it on each fragment separately in order to cover the local edges in the fragment that are still not covered. This algorithm requires coordination between different fragments, since each vertex v needs to learn if the tree edge $\{v, p(v)\}$ is already covered after the first part of the algorithm. In addition, although the second part is applied on each fragment separately, a vertex v may need to add an edge incoming to another fragment to cover the tree edge $\{v, p(v)\}$. For achieving an efficient time complexity, we show how to use only $O(\sqrt{n})$ different messages for the whole coordination of the algorithm. For full details see [3].

Approximation Ratio. We show that this approach gives a 2-approximation to the optimal augmentation in G'. Denote by A the solution obtained by the algorithm and by A^* an optimal augmentation in G'. In the correctness proof of A_{TAP} we show a one-to-one mapping from A to A^*, but this mapping is no longer one to one here. However, if we could show that each edge in A^* is mapped to by at most two edges from A, we can obtain a 2-approximation. Unfortunately, this does not hold either.

Our approach is to divide the edges in A to two types A_1, A_2 as follows. We map each edge $e \in A$ to a corresponding path P_e in T. If P_e contains an internal vertex with more than one child in the tree we say that $e \in A_1$, otherwise $e \in A_2$. Then, we show that $|A_1| \leq 2|A^*|$, and $|A_2| \leq 2|A^*|$. The main idea is that the number of edges in A_1 is related to the degrees of internal vertices in T, which affects the number of leaves in the tree. We use this in order to show that $|A_1| \leq 2\ell$ where ℓ is the number of leaves in T. Note that ℓ is a lower bound on the size of any augmentation in G', since we need to add to the augmentation a different edge in order to cover each one of the leaves. This gives $|A_1| \leq 2|A^*|$.

In order to show that $|A_2| \leq 2|A^*|$, we use the fact that the edges of A_2 correspond to tree paths with a simple structure. This allows us to show a mapping between A_2 to A^* in which each edge in A^* is mapped to by at most two edges from A_2, giving $|A_2| \leq 2|A^*|$.

In conclusion, $|A| = |A_1 \cup A_2| \leq 4|A^*|$. A more delicate analysis extending these ideas gives $|A| \leq 2|A^*|$. This gives a 2-approximation to the optimal augmentation of G', which results in a 4-approximation to the optimal augmentation in G. A full proof of the approximation ratio is given in [3], and gives the following.

▶ **Theorem 1.2.** *There is a distributed 4-approximation algorithm for unweighted TAP in the CONGEST model that runs in $O(D + \sqrt{n}\log^* n)$ rounds.*

6 Lower Bounds

An $\Omega(D)$ Lower Bound for TAP in the LOCAL model. We show that TAP is a global problem, which admits a lower bound of $\Omega(D)$ rounds, even in the LOCAL model where the size of messages is unbounded.

▶ **Theorem 1.4.** *Any distributed α-approximation algorithm for weighted TAP takes $\Omega(D)$ rounds in the LOCAL model, where $\alpha \geq 1$ can be any polynomial function of n. This holds also for unweighted TAP, if $1 \leq \alpha < \frac{n-1}{2c}$ for a constant $c > 1$.*

The proof is based on considering two similar graphs G_1, G_2, defined as follows. The graph G_1 consists of a path T of $n = 2k+1$ vertices $\{v_0, v_1, ..., v_{2k}\}$, and the edges $\{v_{2i}, v_{2(i+1)}\}$ for $0 \leq i < k$, where $G_2 = G_1 \cup \{v_0, v_{2k}\}$. In G_2, adding the edge $\{v_0, v_{2k}\}$ already covers all the edges of the tree T, where in G_1 we must add many edges to the augmentation. However, a vertex in the middle of the path T cannot distinguish between the graphs G_1, G_2 in less than $\Omega(D)$ rounds. Based on these ideas, we show that approximating TAP takes $\Omega(D)$ rounds. The full proof appears in [3].

A Lower Bound for weighted TAP in the CONGEST model. By Theorem 1.4, when $h = O(D)$ our algorithms A_{TAP}, A_{wTAP} are optimal up to a constant factor. But what about the case of $h = \omega(D)$ for the CONGEST model? In [3], we show a family of graphs where $h = \omega(D)$, in which $\Omega(h)$ rounds are needed in order to approximate weighted TAP, where $h = O(\sqrt{n})$. The lower bound is proven using a reduction from the 2-party set-disjointness problem, in which there are two players, Alice and Bob, that have to decide whether their input strings are disjoint. Our construction is based on a construction presented in [6, 32]. In order to use this construction for showing lower bounds for TAP, we add to it additional parallel edges[5] and give weights to the edges in such a way that all the edges of the input tree T can be covered by parallel edges of weight 0, except for k edges, $\{e_i\}_{i=1}^{k}$. The edge e_i may be covered either by a corresponding parallel edge e_i^A, or by a distant edge e_i^B that closes a cycle that contains e_i (see Figure 2 for an illustration). However, the weights of the edges e_i^A and e_i^B depend on the i'th bit in the input strings of Alice and Bob, such that there is a light edge that covers e_i if and only if this bit equals 0 at least in one of the input strings. It follows that all the k edges can be covered by light edges if and only if the input strings of Alice and Bob are disjoint.

Using a known lower bound on the communication complexity of set-disjointness, we prove Theorem 1.5. Full details and proofs appear in [3].

▶ **Theorem 1.5.** *For any polynomial function $\alpha(n)$, there is a $\Theta(n)-$vertex graph of diameter $\Theta(\log n)$ for which any (perhaps randomized) distributed $\alpha(n)$-approximation algorithm for weighted TAP with an instance tree $T \subseteq G$ of height $h = \Theta(\frac{\sqrt{n}}{\log n})$ requires $\Omega(h)$ rounds in the CONGEST model.*

7 Discussion

In this paper, we present the first distributed approximation algorithms for TAP. Many intriguing problems remain open. First, can we get efficient distributed algorithms for TAP with an approximation ratio better than 2? In the sequential setting, achieving an approximation better than 2 for weighted TAP is a central open question. However, there

[5] We also show a construction with no parallel edges.

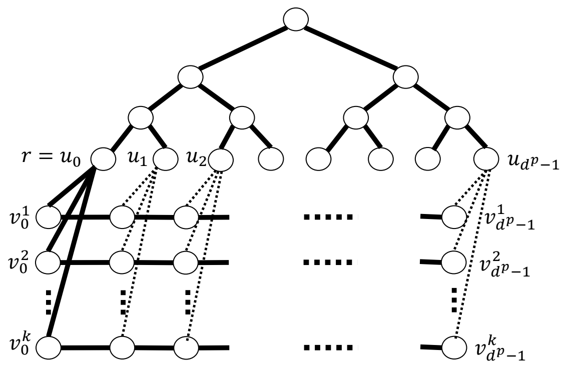

Figure 2 The structure of the graph G. The edges of the input tree T are marked with solid lines, other edges are marked with dashed lines. The edge $e_i = \{r, v_0^i\}$ can be covered either by a parallel edge e_i^A, or by the corresponding edge $e_i^B = \{u_{d^p-1}, v_{d^p-1}^i\}$ (additional options are expensive).

are several recent algorithms achieving better approximations for unweighted TAP [5, 21] or for weighted TAP with bounded weights [1, 8].

Second, can we show algorithms for weighted TAP and weighted 2-ECSS that are more efficient? For *unweighted* TAP, we showed an algorithm with a time complexity of $O(D + \sqrt{n}\log^* n)$ rounds. In addition, for *unweighted* k-ECSS, the $O(k(D + \sqrt{n}\log^* n))$-round algorithm of Thurimella [33] gives a 2-approximation. However, currently there are no algorithms for the *weighted* problems with a similar time complexity.

There are many additional connectivity augmentation problems, such as increasing the edge connectivity from k to $k + 1$ or to some function $f(k)$, as well as augmentation for increasing the vertex connectivity. Such problems have been widely studied in the sequential setting, and a natural question is to design distributed algorithms for them.

Finally, it is interesting to study TAP and additional connectivity problems also in other distributed models, such as the dynamic model where edges or vertices may be added or removed from the network during the algorithm. An interesting question is how to maintain highly-connected backbones when the network can change dynamically.

References

1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2384–2399. SIAM, 2017.

2 Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory of Computing Systems*, 37(3):441–456, 2004.

3 Keren Censor-Hillel and Michal Dory. Fast distributed approximation for TAP and 2-edge-connectivity. *arXiv:1711.03359*, 2017. URL: `https://arxiv.org/abs/1711.03359`.

4 Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing (PODC)*, pages 156–165. ACM, 2014.

5 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part II. *Algorithmica*, pages 1–44, 2015.

6 Michael Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing*, 36(2):433–456, 2006.

7 Michael Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 157–163. ACM, 2017. doi:10.1145/3087801.3087823.

8 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. A 3/2-approximation algorithm for tree augmentation via chvátal-gomory cuts. *CoRR*, abs/1702.05567, 2017. arXiv:1702.05567.

9 Greg N Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

10 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

11 Juan A Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

12 Mohsen Ghaffari and Merav Parter. Near-optimal distributed algorithms for fault-tolerant tree structures. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 387–396. ACM, 2016.

13 Michel X Goemans, Andrew V Goldberg, Serge A Plotkin, David B Shmoys, Eva Tardos, and David P Williamson. Improved approximation algorithms for network design problems. In *SODA*, volume 94, pages 223–232, 1994.

14 P Humblet. A distributed algorithm for minimum weight directed spanning trees. *IEEE Transactions on Communications*, 31(6):756–762, 1983.

15 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

16 Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012.

17 Samir Khuller. Approximation algorithms for finding highly connected subgraphs. In *Approximation algorithms for NP-hard problems*, pages 236–265. PWS Publishing Co., 1996.

18 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

19 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM (JACM)*, 41(2):214–235, 1994.

20 Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

21 Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Transactions on Algorithms (TALG)*, 12(2):23, 2016.

22 Sven O Krumke, Peter Merz, Tim Nonner, and Katharina Rupp. Distributed approximation algorithms for finding 2-edge-connected subgraphs. In *International Conference On Principles Of Distributed Systems (OPODIS)*, pages 159–173. Springer, 2007.

23 Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing (PODC)*, pages 238–251. ACM, 1995.

24 Christoph Lenzen and Boaz Patt-Shamir. Improved distributed steiner forest construction. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing (PODC)*, pages 262–271. ACM, 2014.

25 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

26 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1):583–596, 1992.

27 Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *International Symposium on Distributed Computing*, pages 439–453. Springer, 2014.

28 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time-and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 743–756. ACM, 2017.

29 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

30 David Pritchard. Robust network computation. Master's thesis, MIT, 2005.

31 David Pritchard and Ramakrishna Thurimella. Fast computation of small cuts via cycle space sampling. *ACM Transactions on Algorithms (TALG)*, 7(4):46, 2011.

32 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

33 Ramakrishna Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing (PODC)*, pages 28–37. ACM, 1995.

A A 2-approximation for weighted TAP in $O(h)$ rounds: full details

In this section, we give full details and main proofs for the algorithm A_{wTAP}. We show the following.

▶ **Theorem 1.1.** *There is a distributed 2-approximation algorithm for weighted TAP in the CONGEST model that runs in $O(h)$ rounds, where h is the height of the tree T.*

As explained in Section 3, the algorithm A_{wTAP} has the same structure of A_{TAP}, but it differs from it in the algorithm for finding an optimal augmentation in G'. In Section A.1, we describe our algorithm for finding an optimal augmentation in G'. In Section A.2, we analyze the time complexity of the algorithm, and in Section A.3 we prove its correctness.

A.1 The algorithm

A description of the algorithm is given in Algorithm 1. For simplicity of presentation, we start by describing an algorithm which takes $O(h^2)$ rounds. Later, in Section A.2, we explain how using pipelining we improve the time complexity to $O(h)$ rounds.

Technical Details:

We assume in the algorithm that each vertex knows all the ids of its ancestors in T. We justify it in the next claim. Note that when we construct G', if $\{u, v\}$ is an edge between an ancestor u and its descendant v in T, v learns the label of u according to the LCA labeling scheme and not the id of u. However, once v learns about the ids and labels of all its ancestors, it knows the id of u as well, and can use it in the algorithm.

▶ **Claim 1.1.** *All the vertices can learn the ids and labels of all their ancestors in $O(h)$ rounds.*

Algorithm 1 Finding an Optimal Augmentation in G'.

```
The code is for every vertex $v \neq r$

1: Initialization:
2: $e_{v,u} \leftarrow$ the minimum weight edge incoming to $v$ that covers the path between $v$ and its
   ancestor $u$ or $\perp$ if there is no such edge.
3: $w_v(u) \leftarrow w(e_{v,u})$ for each ancestor $u$ of $v$ such that $e_{v,u} \neq \perp$, and $w_v(u) \leftarrow \infty$ otherwise.
4: $A_v \leftarrow$ the union of $v$ and its children in $T$.
5: $Aug_v \leftarrow \emptyset$

6: First Traversal:
7: if $v$ is a leaf then
8:     for each ancestor $u$ of $v$: $sender_v(u) \leftarrow v$
9: else
10:    wait for receiving $w_{v'}(u)$ for all ancestors $u$ of $v$, from each child $v'$ of $v$
11:    for each ancestor $u$ of $v$: $w_v(u) \leftarrow min_{v' \in A_v} w_{v'}(u)$, $sender_v(u) \leftarrow argmin_{v' \in A_v} w_{v'}(u)$
12: end if
13: $min_v \leftarrow w_v(p(v))$
14: for each ancestor $u$ of $v$: $w_v(u) \leftarrow w_v(u) - min_v$
15: for each ancestor $u \neq p(v)$ of $v$ send $(u, w_v(u))$ to $p(v)$

16: Second Traversal:
17: $u \leftarrow p(v)$
18: if $v$ is not a child of $r$ then
19:    wait for a message $m$ from $p(v)$
20:    if $m \neq \perp$ then $u \leftarrow m$
21:    end if
22: end if
23: $s \leftarrow sender_v(u)$
24: if $s = v$ then
25:    $Aug_v \leftarrow Aug_v \cup \{e_{v,u}\}$
26: else
27:    send $u$ to $s$
28: end if
29: for each child $v' \neq s$ of $v$ send $\perp$ to $v'$
```

Proof. In order to do this, at the first round each vertex sends to its children its id and label. In the next round, each vertex sends to its children the id and label it received in the previous round, and we continue in the same way until each vertex learns about all its ancestors. Clearly, after h rounds each vertex learns all the ids and labels of all its ancestors. ◀

▶ **Claim 1.2.** *If a vertex v adds $e_{v,u}$ to Aug_v in line 25 of its algorithm, then $e_{v,u} \neq \perp$.*

Proof. Since G' is 2-edge-connected, we can cover all tree edges by edges from G'. Hence, the minimum weight of an edge that covers some tree edge is never infinite. It follows that if a vertex v adds $e_{v,u}$ to Aug_v, then $e_{v,u} \neq \perp$. ◀

A.2 Time analysis

We next analyze the time complexity of the algorithm. In the second traversal of the tree, each parent sends to each of its children one message, which takes $O(h)$ rounds. In the first traversal of the tree, each vertex sends to its parent at most h edges. If each vertex waited until it got all the messages from its children before sending messages to its parent, it would result in a time complexity of $O(h^2)$ rounds. We show how to get a time complexity of $O(h)$ rounds for this part using pipelining. To show this, we carefully design each vertex v to send the messages $(u, w_v(u))$ in increasing order of heights of its ancestors.

The main intuition is that although each vertex v may receive h different messages from each of its children during the algorithm, in order for v to send to its parent $p(v)$ the message concerning an ancestor u, the vertex v only needs to receive one message from each of its children concerning the ancestor u. Hence, if all the vertices send the messages according to increasing order of heights of their ancestors, we can pipeline the messages and get a time complexity of $O(h)$ rounds. We formalize this intuition in the next lemma, which we prove in the full version.

▶ **Lemma A.3.** *A vertex v at height i sends to its parent until round $i + j$ the message $(u, w_v(u))$ such that u is an ancestor of v at height j.*

From the lemma we get that by round $2h$ all the children of r learn about the minimum weight edge that covers the tree edge between them and r, so the first traversal is completed after $O(h)$ rounds. It follows that the overall time complexity of the algorithm is $O(h)$ rounds as needed, giving the following.

▶ **Lemma A.4.** *Algorithm 1 completes in $O(h)$ rounds.*

A.3 Correctness Proof

▶ **Lemma A.5.** *Algorithm 1 finds an optimal augmentation in G'.*

Proof. Note that the solution obtained by the algorithm is an augmentation of G' because each vertex v adds an edge in order to cover the tree edge $\{v, p(v)\}$ if it is not already covered by an edge which one of its ancestors decides to add to the augmentation. In order to show that the augmentation is optimal we give costs to the edges of T such that the sum of the costs is equal to both the cost of the solution obtained by the algorithm and the cost of an optimal augmentation of G'. Hence, we conclude that the cost of the solution obtained by the algorithm is optimal.

Giving costs to the edges of T:

Fix a vertex $v \neq r$ and let $t = \{v, p(v)\}$. We define $c(t) = min_v$ (the value of $w_v(p(v))$ in line 13 of the algorithm).

For an edge $e = \{u, x\}$ that covers t, such that u is an ancestor of x in T, let P be the path of tree edges between x and $p(v)$ in T. For a vertex v' such that $\{v', p(v')\} \in P$, let $P_{v'}$ be the path of tree edges between x and v'. Note that min_v is the weight of the minimum weight edge covering the tree edge $t = \{v, p(v)\}$ according to the weights v receives in the algorithm. Denote this edge by e_v.

▶ **Claim 1.6.** $w(e_v) = \sum_{t' \in P} c(t')$.

Proof. Let $e_v = \{u, x\}$, where u is an ancestor of x in T. For each vertex v' on the path between x and v, e_v is the minimum weight edge covering the path between v' and its ancestor $p(v)$, according to the weights v' receives in the algorithm, as otherwise we get a contradiction to the definition of e_v. Each vertex on this path reduces $min_{v'}$ from the weight of e_v it receives before sending it to its parent. Denote by V' all the vertices on the path between x and v, excluding v. It follows that

$$c(t) = min_v = w(e_v) - \sum_{v' \in V'} min_{v'} = w(e_v) - \sum_{t' \in P_v} c(t'),$$

which gives $w(e_v) = \sum_{t' \in P} c(t')$. ◀

▶ **Claim 1.7.** *The sum of the costs of the edges of T is equal to the cost of the solution obtained by the algorithm.*

Proof. We map each edge e added to the augmentation to a path P_e of tree edges, such that:

(I) The paths that correspond to different augmentation edges are disjoint, and their union is the entire tree T. That is, $P_e \cap P_{e'} = \emptyset$ for $e \neq e'$, and $\cup P_e = T$.

(II) The weight of e is equal to the sum of costs of tree edges in the corresponding path, i.e., $w(e) = \sum_{t' \in P_e} c(t')$.

Let $e = \{u, x\}$ be an edge added to the augmentation, such that u is an ancestor of x in T. Let v be the vertex that decides to add e to the augmentation. Note that v decides to add e to the augmentation because it covers the tree edge $\{v, p(v)\}$, which is not covered yet by an edge that one of v's ancestors decides to add to the augmentation. We map e to the tree path P_e that consists of all the tree edges on the path between x and $p(v)$. Note that e covers all the edges on this path (and it may also cover other tree edges, on the path between $p(v)$ and u in T). This divides the tree edges to disjoint paths because the vertices on the path between x and $p(v)$ do not decide to add other edges to the augmentation, since all the relevant tree edges are already covered by e. In addition, these paths include all tree edges because the edges added to the augmentation cover all tree edges. This proves (I).

Note that v adds e to the augmentation because the tree edge $\{v, p(v)\}$ is not covered yet. So v chooses e because it is the minimum weight edge e_v that covers $\{v, p(v)\}$. By Claim 1.6, it holds that $w(e_v) = \sum_{t' \in P} c(t')$ where $P = P_e$ is the path of tree edges between x and $p(v)$. This proves (II). (I) and (II) complete the proof that the cost of all the edges added to the augmentation is equal to the sum of costs of the edges in T. ◀

A similar proof shows that the cost of any augmentation of G' is at least the sum of costs of the edges of T. The idea is to map a subset of edges in the augmentation to disjoint paths in T, such that the union of the paths is the entire tree. We show that the weight of each edge is at least the sum of costs of the tree edges on the corresponding path, which proves the claim. The proof is in the full version. From the above and from Claim 1.7 we have that the cost of the augmentation obtained by the algorithm is smaller or equal to the cost of any augmentation of G', hence the solution obtained by the algorithm is optimal. This completes the proof of Lemma A.5. ◀

By Lemmas A.5 and A.4, Algorithm 1 finds an optimal augmentation A' in G' in $O(h)$ rounds. In the full version, we show that A' corresponds to a 2-approximation augmentation in G, and that building G' takes $O(h)$ rounds. This gives the following.

▶ **Theorem 1.1.** *There is a distributed 2-approximation algorithm for weighted TAP in the CONGEST model that runs in $O(h)$ rounds, where h is the height of the tree T.*