Breaching the 2-Approximation Barrier for the Forest Augmentation Problem*

Fabrizio Grandoni
IDSIA, USI-SUPSI
Switzerland

Afrouz Jabal Ameli
IDSIA, USI-SUPSI
Switzerland

Vera Traub
Department of Mathematics
ETH Zurich, Switzerland

ABSTRACT

The basic goal of survivable network design is to build cheap networks that guarantee the connectivity of certain pairs of nodes despite the failure of a few edges or nodes. A celebrated result by Jain [Combinatorica'01] provides a 2-approximation for a wide class of these problems. However nothing better is known even for very basic special cases, raising the natural question whether any improved approximation factor is possible at all.

In this paper we address one of the most basic problems in this family for which 2 is still the best-known approximation factor, the Forest Augmentation Problem (FAP): given an undirected unweighted graph (that w.l.o.g. we can assume to be a forest) and a collection of extra edges (links), compute a minimum cardinality subset of links whose addition to the graph makes it 2-edge-connected. Several better-than-2 approximation algorithms are known for the special case where the input graph is a tree, a.k.a. the Tree Augmentation Problem (TAP), see e.g. [Grandoni, Kalaitzis, Zenklusen - STOC'18; Cecchetto, Traub, Zenklusen - STOC'21] and references therein. Recently this was achieved also for the weighted version of TAP [Traub, Zenklusen - FOCS'21], and for the k-connectivity generalization of TAP [Byrka, Grandoni, Jabal-Ameli - STOC'20; Cecchetto, Traub, Zenklusen - STOC'21]. These results heavily exploit the fact that the input graph is connected, a condition that does not hold in FAP.

In this paper we breach the 2-approximation barrier for FAP. Our result is based on two main ingredients. First, we describe a reduction to the Path Augmentation Problem (PAP), the special case of FAP where the input graph is a collection of disjoint paths. Our reduction is *not* approximation preserving, however it is sufficiently accurate to improve on a factor 2 approximation. Second, we present a better-than-2 approximation algorithm for PAP, an open problem on its own. Here we exploit a novel notion of *implicit credits* which might turn out to be helpful in future related work.

*The first author is partially supported by the SNF Excellence Grant 200020B_182865/1 and by the SNF Grant 200021_200731/1. The third author is supported by the Swiss National Science Foundation grant 200021_184622.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3520035

CCS CONCEPTS

• **Theory of computation → Routing and network design problems**; **Network optimization**; • **Mathematics of computing → Approximation algorithms**; **Paths and connectivity problems**; **Combinatorial optimization**.

KEYWORDS

approximation algorithms; connectivity augmentation; network design; combinatorial optimization

ACM Reference Format:
Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. 2022. Breaching the 2-Approximation Barrier for the Forest Augmentation Problem. In *Proceedings of the 54rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3520035

1 INTRODUCTION

Real networks are prone to failures. The basic goal of *survivable network design* is to build "cheap" networks that provide connectivity between given pairs of nodes despite the failure of a few edges or nodes (we will next focus on edge faults). Most natural survivable network design problems are NP-hard, therefore it makes sense to study them in terms of approximation algorithms. A celebrated result by Jain [18] provides a 2-approximation for a very wide family of such (edge-connectivity) problems. However, nothing better is known even for very basic special cases. Therefore a recent trend in the area is trying to breach the 2 approximation barrier for interesting special cases of survivable network design.

In this paper we focus on one of the most basic survivable network design problems for which the best-known approximation factor is still 2, namely the Forest Augmentation Problem (FAP): given an undirected unweighted graph (V, F) and a collection of extra edges $L \subseteq \binom{V}{2}$, called *links*, find a minimum cardinality subset of links $S \subseteq L$ such that $(V, F \cup S)$ is 2-edge-connected[1]. Observe that one obtains an equivalent problem by contracting each 2-edge-connected component of (V, F) into a single node, hence leading to a forest graph (motivating the name FAP). Therefore, we will assume in the following that (V, F) is a forest.

FAP generalized two well-studied problems for which better-than-2 approximation algorithms are known, namely the Tree Augmentation Problem (TAP) [1, 4, 9, 11, 12, 16, 21–24, 28, 29] and the 2-Edge-Connected Spanning Subgraph problem (2-ECSS) [7, 8, 13, 17, 19, 27]. Therefore a natural question is whether also FAP admits an approximation factor strictly below 2. Indeed, this is explicitly posed as an open problem, e.g., in [4–6]. This question

[1] We recall that a graph is k-edge connected if it remains connected even after removing an arbitrary subset of up to $k-1$ edges.

was open even in the special case of FAP where the input forest is a collection of disjoint paths, also known as the Path Augmentation Problem (PAP).

TAP is the special case of FAP where the forest consists of a single spanning tree. 2-ECSS is the problem of computing a 2-edge-connected subgraph of an input graph $G = (V, E)$ with the minimum number of edges. Hence 2-ECSS can be interpreted as the special case of FAP where all trees in the forest are singleton nodes (i.e. $F = \emptyset$). One can define a natural weighted generalization 2-WECSS of 2-ECSS with edge weights, where the goal is to compute a 2-edge-connected subgraph of minimum total weight. Then FAP is the special case of 2-WECSS with edge weights 0 (for $e \in F$) and 1 (for $e \in L$). Therefore an improved approximation for FAP is a first step in the direction of a similar result for 2-WECSS, a well-known open problem in the area.

In this paper we breach the 2-approximation barrier for FAP, namely we obtain the following result.

Theorem 1.1. *There is a polynomial-time deterministic* 1.9973*-approximation algorithm for FAP.*

In Section 3 we provide an overview of the main ideas behind the above result.

1.1 Previous and Related Work

TAP (hence FAP) is known to be APX-hard [21], thus in particular it does not admit a PTAS unless $P = NP$. Several better-than-2 approximation algorithms are known for this problem [11, 16, 22, 23], culminating with a 1.393 approximation by Cecchetto, Traub and Zenklusen [4].

Very recently the 2-approximation barrier was breached for the natural weighted version WTAP of TAP (with link weights) by Traub and Zenklusen [28], who presented a 1.694 approximation (which they improved to a $(1.5 + \varepsilon)$-approximation in [29]). This solved one of the main open (and simplest weighted) problems in the area (preliminary results in this directions, among others, appeared in [1, 9, 12, 16, 24]).

FAP and PAP belong to the family of *connectivity augmentation* problems, where the goal is to increase the connectivity of an existing network by adding links. One important problem in this area is the k-Connectivity Augmentation Problem (k-CAP): given a k-edge-connected graph $G = (V, E)$ and a collection of links $L \subseteq \binom{V}{2}$, find a minimum cardinality subset of links $S \subseteq L$ such that $G' = (V, E \cup S)$ is $(k + 1)$-edge-connected. Hence in particular TAP is the special case of k-CAP with $k = 1$. Known approximation-preserving reductions [10] show that k-CAP reduces to the case $k = 2$, a.k.a. the Cactus Augmentation Problem (CacAP). After some preliminary results on special cases [14], a better-than-2 approximation for CacAP was achieved recently by Byrka, Grandoni and Jabal-Ameli [2] via a (non-black-box) reduction to the Steiner Tree problem [3] (see also [25] for a black-box reduction to the same problem). This was improved to 1.393 by Cecchetto et al. [4] with a completely different approach, more similar in spirit to prior work on TAP.

Better-than-2 approximation algorithm are known for the Matching Augmentation Problem (MAP),i.e., the special case of FAP where the input forest is a matching [5, 6]. In more detail, Cheriyan, Dippel, Grandoni, Khan and Narayan [6] present a 7/4 approximation for MAP. This was later improved to 5/3 by Cheriyan, Cummings, Dippel and Zhu [5]. Finding a better-than-2 approximation for FAP is mentioned in [5, 6] as one of the main motivations to study MAP. We remark that this question was open even for the Path Augmentation Problem (PAP). The techniques in [5, 6] do not seem to extend even to paths of length 2.

Khuller and Vishkin [19] found the first better-than-2 (namely a 3/2) approximation for 2-ECSS. Cheriyan, Sebő and Szigeti [7] improved the approximation factor to 17/12. The current best known approximation factor for 2-ECSS is 4/3, and this can be achieved with two rather different approaches [17, 27]. Hunkenschröder, Vempala and Vetta [17] use a credit invariant (vaguely) similar to the one used in this paper. Sebő and Vygen [27] instead exploit an ear decomposition of the input graph with special properties. The natural generalization k-ECSS of 2-ECSS to k connectivity was studied, among others, by [8, 13].

2 PRELIMINARIES

In this section we first introduce some notation (Section 2.1) and then recap some algorithms from previous work that we will build on. Specifically, in Section 2.2 we explain the results on WTAP from [28] that we will use in our algorithm, while in Section 2.3 we describe a well-known 2-approximation algorithm for W2ECSS (and thus also for FAP). In what follows all algorithms will be deterministic and we will therefore not mention this explicitly anymore.

2.1 Notation

We use standard graph notation. In particular, given a graph $G = (V, E)$ and $\emptyset \neq R \subsetneq V$, by $\delta(R) \subseteq E$ we denote the edges with exactly one endpoint in R. If G is directed, then $\delta^-(R) \subseteq E$ are the edges (or arcs) entering R. For any $F \subseteq E$, we let $\delta_F(R) = \delta(R) \cap F$ and $\delta_F^-(R) = \delta^-(R) \cap F$. For an edge weight function w and $F \subseteq E$, $w(F) := \sum_{e \in F} w(e)$.

Let (V, F, L) be a considered instance of FAP. W.l.o.g. we assume that $(V, F \cup L)$ is 2-edge-connected (otherwise there is no feasible solution). By $n_{\text{comp}} := |V| - |F|$ we denote the number of connected components (or components for short) of the forest (in particular in a TAP instance $n_{\text{comp}} = 1$). By $\text{OPT} \subseteq L$ we denote an optimal solution to this instance and by $\text{opt} = |\text{OPT}|$ its size. Notice that $\text{opt} \geq n_{\text{comp}}$.

We will call the elements of L *links*. By *edges* we will mean both links and edges of the input forest.

2.2 Preliminaries on WTAP

Given a WTAP instance (V, F, L, w) and $\ell = \{a, b\} \in L$, we let P_ℓ be the (edge set of the) path in the tree (V, F) between endpoints a and b. We say that ℓ covers the edges of P_ℓ. Then a feasible solution to WTAP is a subset of links that covers all the edges. We say that $\ell' \in L$ is a shadow of $\ell \in L$ if $P_{\ell'} \subseteq P_\ell$. W.l.o.g. we can assume that all the possible shadows of a link ℓ are present in the input. Indeed, if this is not the case, we can add any missing shadow of ℓ with weight $w(\ell)$: any feasible solution for the new problem can be converted into a feasible solution for the original problem and vice versa. For similar reasons, if ℓ' is a shadow of ℓ, we can assume that $w(\ell') \leq w(\ell)$.

Our algorithm for FAP will make use of recent insights on WTAP from [28]. Thus, we briefly summarize the statements from [28] which we will need. Consider a WTAP instance (V, F, L, w), that we interpret as rooted at some node r. We define an *up-link* as a link where one endpoint in an ancestor of the other endpoint in the tree (V, F). The approximation algorithm for WTAP from [28], a so-called relative greedy algorithm, first computes a 2-approximation $U \subseteq L$ that contains only up-links; it is well-known that this exists and is computable in polynomial time. Then the main insight from [28] is the following lemma that allows for improving this 2-approximate solution U to a $(1 + \ln(2) + \varepsilon)$-approximation APX (for any constant $\varepsilon > 0$).

Lemma 2.1 ([28]). [2] *Let (V, F, L, w) be a WTAP instance rooted at some node r with optimal solution* OPT*, and let $U \subseteq L$ be a feasible solution consisting only of up-links. For any $\varepsilon > 0$, one can in polynomial time construct a feasible WTAP solution* APX *with weight*

$$w(\text{APX}) \leq \left(1 + \ln\left(\frac{w(U)}{w(\text{OPT})}\right) + \varepsilon\right) \cdot w(\text{OPT}).$$

We will apply the above lemma to up-link solutions obtained in a different way. While in general there might not exist a better-than-2 approximation for WTAP that consists only of up-links, in some cases we can obtain such very good up-link solutions, as we will explain in Section 3. In particular, we highlight that we do not use the $(1 + \ln(2) + \varepsilon)$-approximation algorithm for WTAP as a black-box, but crucially use the properties of the relative greedy algorithm that allow for improving arbitrary up-link solutions, as stated in Lemma 2.1. For this reason, we do not use the recent improvements on [28] that lead to a $(1.5 + \varepsilon)$-approximation algorithm for WTAP [29].

2.3 A Simple 2-Approximation for 2-WECSS

We next sketch how to obtain a 2-approximation for a 2-WECSS instance (V, E, w) because we will need this algorithm from [20] later. Let us replace each undirected edge $e = \{a, b\}$ with two oppositely directed arcs (a, b) and (b, a) with weight $w(e)$. We call A this set of directed arcs and still use $w(\cdot)$ to denote their weight. Moreover, we fix an arbitrary vertex r as the root. If we can find an arc set $D \subseteq A$ such that every set $\emptyset \neq R \subseteq V \setminus \{r\}$ has two entering arcs, then clearly the set of undirected edges corresponding to D is a feasible solution to our 2-WECSS instance. We remark, that by Edmonds' disjoint branching theorem (see, e.g., Corollary 53.1c in [26]), such arc sets D are precisely those that contain two edge-disjoint spanning arborescences rooted at r.

Moreover, the cheapest such set D satisfies the following claim:

Lemma 2.2. *Given a 2-WECSS instance (V, E, w) with optimal weight* opt *and the corresponding directed arc set A, we have*

$$\min\left\{w(D) : D \subseteq A,\ |D \cap \delta^-(R)| \geq 2 \text{ for all } \emptyset \neq R \subseteq V \setminus \{r\}\right\} \leq 2 \cdot \text{opt}.$$

Proof. Indeed, by taking an optimal 2-WECSS solution and replacing every edge by the two corresponding directed arcs we obtain a feasible set $D \subseteq A$. □

[2] This lemma is not stated explicitly in this form in [28]. We provide a detailed explanation of how this statement follows from [28] in the full version [15] of this paper.

A 2-approximation for 2-WECSS (and hence for FAP) is implied from the above discussion and the following lemma.

Lemma 2.3. *Given a directed graph (V, A) with nonnegative arc-weights $w : A \to \mathbb{R}_{\geq 0}$ and a root vertex $r \in V$ we can find in polynomial time a minimum-weight set $D \subseteq A$ such that*

$$|D \cap \delta^-(R)| \geq 2 \quad \text{for all } \emptyset \neq R \subseteq V \setminus \{r\}.$$

Proof. By Edmonds' disjoint branching theorem (Corollary 53.1c in [26]), the desired sets D are precisely those sets that contain two disjoint spanning arborescences rooted at r. Hence, because the edge weights are non-negative, we can find a minimum-weight set $D \subseteq A$ as desired by computing a cheapest edge set that is the union of two disjoint arborescences. This can be done in polynomial time; see Theorem 53.10 in [26]. □

3 OVERVIEW OF OUR APPROACH

Our main result (Theorem 1.1) follows by combining two different algorithms: the first one (Section 4) provides a good approximation for the case that opt is much larger than n_{comp}, and the second one (Section 5) provides a good approximation for the complementary case. In particular, we prove the following.

Lemma 3.1. *Let $\varepsilon > 0$ be a constant. Given an instance (V, F, L) of FAP, we can compute in polynomial time a solution of size at most*

$$n_{\text{comp}} + \left(1 + \ln\left(2 - \frac{n_{\text{comp}}}{\text{opt}}\right) + \varepsilon\right) \cdot \text{opt}.$$

Lemma 3.2. *Given an instance (V, F, L) of FAP, we can compute in polynomial time a solution of size at most*

$$\tfrac{7}{4} \cdot \text{opt} + \tfrac{13}{4} \cdot (\text{opt} - n_{\text{comp}}).$$

Lemma 3.1 yields an approximation ratio ranging from 2 to $1 + \ln 2 + \varepsilon < 1.7$ as the ratio $\frac{n_{\text{comp}}}{\text{opt}}$ decreases from 1 to 0[3]. Hence, to improve on a 2 approximation, it is sufficient to have such an approximation for the case that $\frac{n_{\text{comp}}}{\text{opt}}$ is close to 1: this is achieved by the algorithm from Lemma 3.2. Theorem 1.1 follows easily.

Proof of Theorem 1.1. Consider the best solution among the ones returned by the algorithms from Lemmas 3.1 and 3.2. Let $\alpha \in [0, 1]$ be such that $n_{\text{comp}} = \alpha \cdot \text{opt}$. Then the approximation factor of this algorithm is at most $\min\{\alpha + 1 + \ln(2 - \alpha),\ \frac{7}{4} + \frac{13}{4}(1 - \alpha)\} + \varepsilon$. The worst-case ratio is obtained by choosing α so that the two terms in the minimum are equal, implying the claim. (This is the case for $\alpha \approx 0.923925$.) □

We next sketch the basic ideas behind the above two lemmas.

3.1 Overview of the Algorithm for a Small Number of Connected Components

First, we explain how we obtain Lemma 3.1, which yields a good approximation ratio if the number n_{comp} of connected components of the forest is much smaller than opt. In this case, it seems natural to consider the simple reduction to TAP: complete the forest (V, F) to a spanning tree by adding $n_{\text{comp}} - 1$ links and then apply an approximation algorithm for TAP to obtain a 2-edge-connected graph. Using a ρ-approximation algorithm for TAP this yields a

[3] For $n_{\text{comp}} = 1$, our algorithm is indeed identical to the one in [28].

FAP solution of size at most $n_{\text{comp}} - 1 + \rho \cdot \text{opt}$. At the moment, the best known approximation algorithm allows for choosing $\rho = 1.393$ [4]. While this yields a better-than-2 approximation if n_{comp} is sufficiently smaller than opt, i.e., if $n_{\text{comp}} \leq (2 - \rho - \delta)\text{opt}$ for some constant $\delta > 0$, this guarantee is not good enough for our purposes. In contrast, the statement of Lemma 3.1 is much stronger in the sense that it yields an improvement over the approximaton ratio of 2 as soon as $n_{\text{comp}} \leq (1 - \delta)\text{opt}$ for any arbitrary constant $\delta > 0$.

Thus, let us consider the following, slightly more involved reduction to TAP. Let $S \subseteq L$ be a 2-approximate solution for FAP. Then we can complete the forest (V, F) to a spanning tree (V, T) using only edges from S. This way, we obtain a tree (V, T) and a solution $S \setminus T$ for the TAP problem of augmenting (V, T) to a 2-edge-connected graph. We observe that $|T \cap S| = n_{\text{comp}} - 1$ and $|S \setminus T| = |S| - n_{\text{comp}} + 1 \leq 2 \cdot \text{opt} - n_{\text{comp}} + 1$. In particular $S \setminus T$ can only be an optimal solution to the TAP instance if $n_{\text{comp}} \approx \text{opt}$ and S is not already a better-than-2 approximation (in which case we are done). Because S is a 2-approximation for our FAP instance, we would immediately obtain a better-than-2 approximation if we could improve the TAP solution $S \setminus T$ by just a little bit, similar as it is done in Lemma 2.1. Unfortunately, we cannot apply Lemma 2.1 because $S \setminus T$ will in general consist not only of up-links.

In order to address this issue, we use the 2-approximation algorithm from Section 2.3 not as a black-box, but exploit that the computed solution S has stronger properties than just being a FAP solution that is not too expensive. More precisely, we show that from the directed arc set D computed in the algorithm from Section 2.3 we can obtain a spanning tree (V, T) with $F \subseteq T$ and a TAP solution consisting of at most $|S| - n_{\text{comp}} + 1$ many up-links. Then, applying Lemma 2.1, completes the proof of Lemma 3.1. In order to obtain the up-link solution from the directed arc set D, we use that directed links can in a certain sense be interpreted as up-links, an observation first made in [4]. For details on how we prove Lemma 3.1, we refer to Section 4.

3.2 Overview of the Algorithm for a Large Number of Connected Components

In view of Lemma 3.1, it makes sense to design approximation algorithms for the case that $\frac{n_{\text{comp}}}{\text{opt}}$ is close to 1. This motivates the following definition.

Definition 3.3. An algorithm for FAP is a (ρ, K)-approximation algorithm if it produces a solution of size at most

$$\rho \cdot \text{opt} + K \cdot (\text{opt} - n_{\text{comp}}).$$

If we can find a (ρ, K)-approximation algorithm for some $\rho < 2$ and any arbitrary constant $K \geq 0$, then together with Lemma 3.1, this implies a better-than-2 approximation for FAP.

We will show how to obtain a $(\frac{7}{4}, \frac{13}{4})$-approximation algorithm for FAP, which is precisely the statement of Lemma 3.2. To achieve this we exploit a relatively simple reduction to PAP. In order to simplify the notation, we can further impose that the PAP instance has no isolated nodes, i.e., all connected components of the forest (V, F) are paths with length at least 1.

Lemma 3.4. *Given a polynomial time (ρ, K)-approximation algorithm for PAP without isolated nodes for some constants $\rho \geq 1$ and $K > 0$, there is polynomial-time $(\rho, K + 2(\rho - 1))$-approximation algorithm for FAP.*

For a proof of Lemma 3.4 we refer to the full version of this paper [15]. We conclude that in order to prove Lemma 3.2, it suffices to show the following.

Lemma 3.5. *There is a $(\frac{7}{4}, \frac{7}{4})$-approximation algorithm for PAP without isolated nodes.*

In the rest of this section we sketch the proof of Lemma 3.5. The basic algorithm is analogous to known algorithms for MAP [5, 6] and 2-ECSS [17]. We start by building (in polynomial time) an infeasible partial solution S. Then we gradually modify S by adding (and sometimes removing) links until we obtain a feasible solution.

In [5, 6, 17] the initial solution is a 2-edge-cover[4] with the minimum number of links. The number of links in such a 2-edge-cover provides a lower bound on opt. A cheapest 2-edge cover can be obtained by first computing a maximum cardinality matching (in L) on the leaves of the forest (V, F) and then adding an arbitrary incident link for every unmatched leaf. Instead of using a cheapest 2-edge-cover as a starting solution, we use such a maximum matching $M \subseteq L$ between the leaves of the input paths, but we exclude links that match the two endpoints of a single path in (V, F) (*bad links*). Intuitively, using such links is bad because they do not help to connect different connected components of the forest. For this reason, also the optimal solution can use these bad links only if $\text{opt} > n_{\text{comp}}$, which we exploit to show that if many leaves remain unmatched, opt must be much larger than n_{comp}. (See Lemma 5.3 for the precise statement.) Working with the matching M rather than a 2-edge-cover is useful because it allows us to exclude bad links (which we could not do in a 2-edge-cover), but also because it simplifies the later parts of the proof.[5]

In order to upper bound the size of the final solution we use a *credit assignment scheme* similarly to [5, 6, 17]. The basic idea is to assign (nonnegative fractional) credits to certain parts of the current graph $H = (V, F \cup S)$ (like links, nodes, components, 2-edge connected components etc.). Let $\text{credits}(H)$ be the total number of credits assigned to H. We show that the initial graph $H = (V, F \cup S)$ with $S = M$, and every intermediate graph $H = (V, F \cup S)$ satisfies the following invariant:

Invariant 1.

$$\text{credits}(H) + |S| \;\leq\; \tfrac{7}{4}\text{opt} + \tfrac{7}{4}\left(\text{opt} - n_{\text{comp}}\right).$$

At the end of our algorithm, S is a feasible solution and thus Lemma 3.5 follows immediately because $\text{credits}(H) \geq 0$.

Our credit assignment scheme however critically deviates from prior work in the following sense: typically a credit invariant is *explicit* meaning that, given H, one can compute in polynomial time the credits assigned to each part of H. We rather use an *implicit* credit invariant where part of the credits are assigned based on properties of the (unknown) optimal solution OPT. Our algorithm is able to work despite the lack of knowledge about the precise number of credits available. In prior related work the role of implicit

[4] We recall that a 2-edge cover of a graph is a subgraph where each node has degree at least 2.

[5] Alternatively, we could have worked with a weighted version of the 2-edge-cover where we make bad links more expensive, but working with the matching is overall simpler.

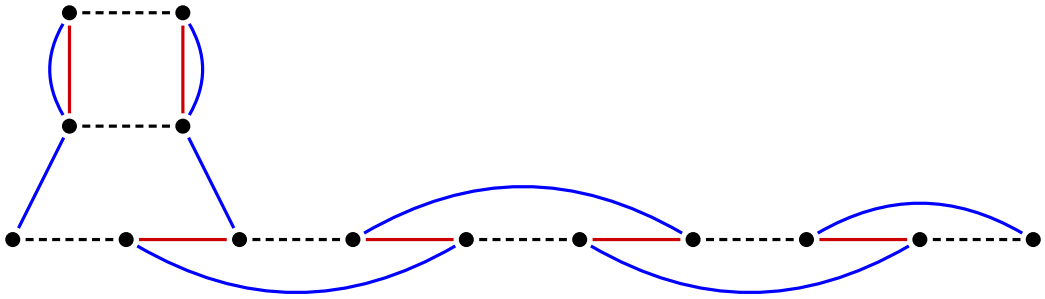


Figure 1: Example showing that even for $n_{\text{comp}} \approx \text{opt}$ we cannot ensure $|S| \leq \rho \cdot \text{opt}$ in the bridge covering step for any $\rho < 2$ if we never remove any links from S. Solid edges are links from L, dashed edges are edges of the forest F. The red links are a possible matching M chosen in the initialization step of our algorithm. The blue links are an optimal solution OPT. If the red matching M is chosen, we need to add all but two links from OPT to S in order to obtain a graph $(V, F \cup S)$ in which all connected components are 2-edge-connected. For a large enough number of vertices, i.e., making the path to the right long enough, this shows that we cannot obtain a better-than-2 approximation without removing some links in the bridge covering step.

credits is played by complicated preprocessing steps and more complex credit invariants and case analysis. We believe that implicit credits can be used to simplify and/or strengthen related results in the literature, and might be useful to address generalizations and variants of FAP in future work.

Similar to prior work on MAP [5, 6], our algorithm consists of two phases, called *bridge-covering* and *gluing*. At the end of the bridge-covering phase, every connected component of $H = (V, F \cup S)$ will be 2-edge-connected. In other words, H is bridgeless, i.e., none of its edges is a bridge[6]. During the gluing phase, we maintain the property that H is bridgeless and at the end of the gluing phase H will be 2-edge-connected. While this high-level structure is similar to prior work, we also introduce new concepts in this part, making the algorithm simpler and avoiding some issues that were in prior work addressed by rather complicated preprocessing steps.

For the bridge covering step we introduce the new concept of *alternating trails*, which we define in Section 5.4. Using this concept, our algorithm for bridge covering can be simply stated as repeatedly doing the following until H is bridgeless. Consider the graph resulting from H by contracting each 2-edge-connected component. This graph is a forest and its edges are precisely the bridges of H. Pick an arbitrary leaf of this forest, find an alternating trail that starts at this leaf and 'covers as many bridges as possible'. Then augment H along this trail. For a precise description of the algorithm and definitions of *alternating trails* and *augmentation* we refer to Section 5.4. Note that it is crucial that when augmenting along alternating trails we do not only add links to S in the bridge covering step, but also remove some links. Indeed, this is necessary to obtain a better-than-2 approximation even when $\frac{n_{\text{comp}}}{\text{opt}}$ is arbitrarily close to 1 as shown in Figure 1.

The final part of our algorithm is the gluing step in which we maintain a bridgeless graph H and transform it into a 2-edge-connected graph. Our algorithm will repeatedly find a cycle Q in the graph resulting from $(V, F \cup L)$ by contracting each 2-edge-connected component of H and add the links from Q to our current partial solution S (and thus to $H = (V, F \cup S)$). However, by adding only links and never removing any link in the gluing step it is impossible to obtain a better-than-2 approximation, even for $\text{opt} = n_{\text{comp}}$, as shown in Figure 2. Therefore, in some cases where we need this to achieve the desired approximation ratio, our algorithm will exploit what we call a *good cycle* (defined in Section 5.5) that allows us to add some links to S, but also to remove some links from S which become redundant.

Finally, we remark that we designed our approximation algorithm for PAP favouring simplicity over a (slightly) better approximation factor.

4 AN ALGORITHM FOR FORESTS WITH FEW CONNECTED COMPONENTS

In this section we prove Lemma 3.1, making use of Lemma 2.1. First of all, we consider the 2-WECSS instance $(V, F \cup L, w)$ corresponding to the input instance of FAP and we fix an arbitrary root $r \in V$. Moreover, as in Section 2.3, we again denote by A the arc set that, for every (undirected) edge in $F \cup L$, contains the two corresponding directed arcs. By Lemma 2.3, we can find a minimum-weight set $D \subseteq A$ of directed arcs that enters every set $\emptyset \neq R \subseteq V \setminus \{r\}$ at least twice. W.l.o.g. we can assume that D contains all the arcs corresponding to F (since they have weight zero).

Next, we construct a WTAP instance together with an up-link solution, to which we will later apply Lemma 2.1. Let S be the set of links corresponding to directed edges in D. Here, we define S to contain only a single copy of a link $\{u, v\}$ even if D contains both (u, v) and (v, u). The lemma below shows that $F \cup S$ contains a cheap spanning tree, which will be the input tree of the WTAP instance we construct.

Lemma 4.1. *The graph $(V, F \cup S)$ contains a spanning tree $(V, F \cup S_{tree})$ with $|S_{tree}| = n_{\text{comp}} - 1$.*

[6] A bridge is an edge whose removal would increase the number of connected components.

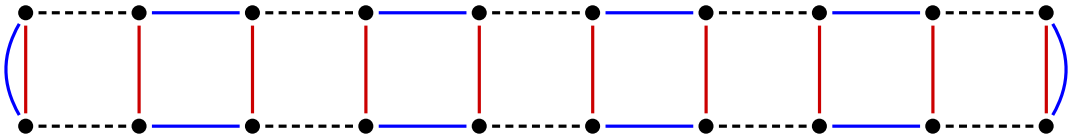


Figure 2: Example showing that we cannot obtain a better-than-2 approximation if we never remove any links in the gluing step. Solid edges are links from L, dashed edges are edges of the forest F. The red links are a possible matching M chosen in the initialization step of our algorithm. If this matching is chosen, all connected components of $(V, F \cup M)$ are 2-edge-connected and hence the bridge covering step does nothing. In order to obtain any feasible solution, we need to include all but two of the edges from the optimal solution, which is shown in blue. For a large enough number of vertices, this shows that we cannot obtain a better-than-2 approximation without removing some links in the gluing step and this holds even if $n_{\text{comp}} = \text{opt}$. .

Proof. First, we observe that (V, D) contains a spanning arborescence. This follows from the fact that every nonempty set of vertices that does not contain r has an entering arc in D and thus every vertex is reachable from r in (V, D). Thus, $(V, F \cup S)$ is connected. Because (V, F) is a forest, there exists a spanning tree containing all edges from F. This spanning tree has weight (i.e. number of links) $|V| - 1 - |F| = n_{\text{comp}} - 1$. □

Let $(V, F \cup S_{\text{tree}})$ be a spanning tree as in Lemma 4.1 and let $S_{\text{tap}} := S \setminus S_{\text{tree}}$. Moreover, let $D_{\text{tree}} \subseteq D$ and $D_{\text{tap}} \subseteq D$ be the arcs in D whose underlying undirected edges are contained in $F \cup S_{\text{tree}}$ and S_{tap}, respectively.

Because the choice of D implies that $F \cup S_{\text{tree}} \cup S_{\text{tap}}$ is 2-edge-connected, S_{tap} is a feasible solution for the WTAP instance with tree $(V, F \cup S_{\text{tree}})$. Moreover, we obtain the following even stronger property, the proof of which is inspired by an observation from [4].

Lemma 4.2. *Consider the WTAP instance with tree $G = (V, F \cup S_{\text{tree}})$ and link set consisting of all shadows of links in S_{tap}. Then one can find in polynomial time, a feasible solution for this instance that consists only of up-links and has weight (i.e., number of links) at most $|S_{\text{tap}}|$.*

Proof. We call a cut, a 1-cut of the spanning tree G, if it contains only a single edge of G. Recall that we can view WTAP as the problem of covering the 1-cuts of G by links. First, we show that for every 1-cut $\delta(R)$ with $\emptyset \neq R \subseteq V \setminus \{r\}$, there is a directed arc in D_{tap} that enters R. To prove this, we first observe that D contains at least two arcs entering R. One of these arcs might be an arc $(a, b) \in D_{\text{tree}}$ corresponding to the unique edge $\{a, b\} \in (F \cup S_{\text{tree}}) \cap \delta(R)$. However, because $\delta(R)$ is a 1-cut of the tree G, the other arc in D that enters R must be an element of $D_{\text{tap}} = D \setminus D_{\text{tree}}$.

To construct the desired up-link solution, we replace every directed link $(a, b) \in D_{\text{tap}}$ by the up-link $\{c, b\}$, where c is the lowest common ancestor of a and b in the tree G (with respect to the root r). See Figure 3. Note that $\{c, b\}$ is a shadow of the link $\{a, b\}$.

Now consider any 1-cut $\delta(R)$ with $R \subseteq V \setminus \{r\}$. We observe that if the directed link (a, b) enters R, then also the up-link $\{c, b\}$ covers this 1-cut, i.e., $\{c, a\} \in \delta(R)$. Indeed, because $\delta(R)$ is a 1-cut of G and R does not contain the root, the fact that $b \in R$ and $a \notin R$ implies that the lowest common ancestor c of a and b is not contained in R. □

We are now ready to prove Lemma 3.1.

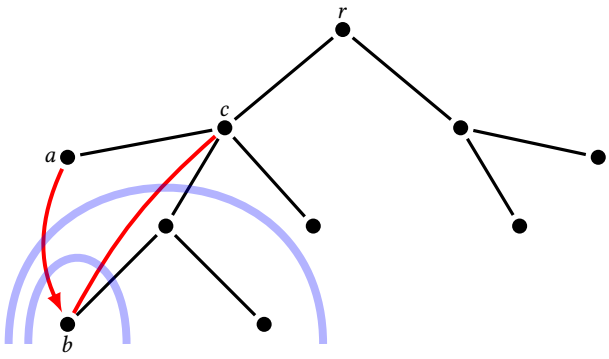


Figure 3: Illustration of the proof of Lemma 4.2. The spanning tree S_{tree} is shown in black. In red, there is a directed link (a, b) and the corresponding up-link $\{c, b\}$. The cuts $\delta(R)$ with $\emptyset \neq R \subseteq V \setminus \{r\}$ that contain precisely one edge of the tree S_{tree} and fulfill $(a, b) \in \delta^-(R)$, are shown in light blue. For each such cut, we have $\{b, c\} \in \delta(R)$.

Proof of Lemma 3.1. By Lemma 2.3, we can compute the sets D and S in polynomial time. Then by Lemma 4.1, we can compute a spanning tree $(V, F \cup S_{\text{tree}})$ in $(V, F \cup S)$ of weight $n_{\text{comp}} - 1$.

Clearly, the set $\text{OPT} \setminus S_{\text{tree}}$ is a feasible TAP solution to this instance with at most opt many links. In particular, the optimal solution value of the TAP instance $((V, F \cup S_{\text{tree}}), L \setminus S_{\text{tree}})$ is at most the optimal solution value opt for the original FAP instance. Moreover, by Lemma 4.2, we can construct an up-link solution for this instance with at most $|S_{\text{tap}}|$ many links. Using Lemma 2.2, we get

$$\begin{aligned} |S_{\text{tap}}| &= |S| - |S_{\text{tree}}| = |S| - n_{\text{comp}} + 1 \leq |D| - n_{\text{comp}} + 1 \\ &\leq 2 \cdot \text{opt} - n_{\text{comp}} + 1, \end{aligned}$$

and hence applying the algorithm from Lemma 2.1 yields a TAP solution with at most

$$\left(1 + \ln\left(2 - \frac{n_{\text{comp}} - 1}{\text{opt}}\right) + \varepsilon\right) \cdot \text{opt}$$

many links. The union of these links and the $n_{\text{comp}} - 1$ links in S_{tree} is the desired FAP solution. The claim follows by observing that replacing $n_{\text{comp}} - 1$ by n_{comp} only weakens the overall bound. □

5 ALGORITHM FOR FORESTS WITH MANY CONNECTED COMPONENTS

In this section we prove Lemma 3.2. From the discussion in Section 3, this reduces to proving Lemma 3.5, i.e., to providing a $(\frac{7}{4}, \frac{7}{4})$-approximation algorithm for PAP without isolated nodes. Therefore, in what follows we will assume that every connected component of (V, F) is a path of length at least 1.

5.1 Overview of Our Algorithm

Let us start with a brief recap of how our algorithm and its analysis work on a high level. In our algorithm we maintain a partial solution $S \subseteq L$, which is not necessarily a feasible solution. In the analysis, we maintain in addition a fractional set of (nonnegative) credits, which are distributed over the current graph $H = (V, F \cup S)$. Credits will be assigned to some of the vertices, edges, connected components, and 2-edge-connected components. Throughout the algorithm we maintain the invariant that the number $|S|$ of links in our partial solution plus the total number of credits credits(H) of H is at most $\frac{7}{4}\text{opt} + \frac{7}{4}(\text{opt} - n_{\text{comp}})$. At the end of the algorithm, S will be a feasible solution, i.e., $H = (V, F \cup S)$ will be 2-edge-connected. Then our invariant (and the fact that credits are nonnegative) implies $|S| \leq \frac{7}{4}\text{opt} + \frac{7}{4} \cdot (\text{opt} - n_{\text{comp}})$.

Our algorithm consists of three steps. First, we use a matching algorithm to compute an initial partial solution S. The goal of the second step, called bridge covering, is to ensure that every connected component of $H := (V, F \cup S)$ is 2-edge-connected. We describe this step in Section 5.4. Finally, in the third step, called gluing, we ensure that H becomes 2-edge-connected (see Section 5.5). Algorithm 1 describes our overall algorithm for PAP without isolated nodes.

Algorithm 1 Algorithm for PAP without isolated nodes

(1) **Initialization:**
Let $M \subseteq L$ be a maximum cardinality matching on the leaves of (V, F) that contains no bad link, i.e., no link with both endpoints in the same path in (V, F).
Initialize $S := M$.

(2) **Bridge covering:**
As long as $H = (V, F \cup S)$ has a connected component that is not 2-edge-connected, iterate the following:
- Let C be a connected component of H that is not 2-edge-connected and let x be a leaf of the tree T^C (arising from the contraction of each 2-edge-connected component of C).
- Let z be the vertex of the tree T^C that has maximum distance from x (in T^C) among all vertices that are reachable from x by an alternating trail (as defined in Section 5.4).
- Augment S along an alternating x-z trail (as defined in Section 5.4).

(3) **Gluing:**
As long as $H = (V, F \cup S)$ is not 2-edge-connected, iterate the following:
- If there is a good cycle Q, then glue H along Q (definitions in Section 5.5).
- Otherwise take an arbitrary cycle Q in the graph G_H (arising from the contraction of the 2-edge-connected components of H) and add the links of Q to S.

(4) Return S.

5.2 Credit Scheme and Invariants

In this section we show how we distribute credits and describe the invariants that we maintain during our algorithm. We emphasize that we will use credits only in the analysis of our algorithm. Therefore, the distribution of credits can depend on a fixed (but unknown) optimal solution OPT.

To define our credit scheme, we need the notion of *simple components*.

Definition 5.1 (simple components). A connected component of $H = (V, F \cup S)$ is called *simple* if it is a cycle that contains exactly 2 links.

Simple components will play a special role in the gluing step because they are the components that prevent us from only adding links (and never removing links) in the gluing step. We remark that in particular in the example in Figure 2 all connected components of $H = (V, F \cup M)$ are simple.

Let us now define the credit invariants. Consider any point of the algorithm, where we have a current partial solution $S \subseteq L$ and define the current subgraph to be $H := (V, F \cup S)$. Recall that by assumption, the graph H has no isolated vertices. We call a 2-edge-connected component of H *nontrivial* if it contains at least two vertices. The *2EC-blocks* of a connected component of H are its inclusionwise maximal 2-edge-connected subgraphs that are nontrivial.

Definition 5.2 (lonely vertex). If a vertex $v \in V$ does not belong to any nontrivial 2-edge-connected component of H, we call v a *lonely* vertex (of H).

For $H = (V, F \cup S)$, we assign credits according to the following rules, where we set $\varepsilon := \frac{1}{4}$.

(A) Vertices receive the following credits.
  (A1) Every leaf of H, i.e., every vertex v with $|\delta_{F \cup S}(v)| = 1$, receives 1 credit.
  (A2) A vertex v of H, receives $\frac{1}{2}(|\delta_{\text{OPT} \cup F}(v)| - 2)$ additional credits if it is a lonely vertex or it belongs to a simple component.
(B) Every bridge $\ell \in S$, i.e., every link $\ell \in S$ that is not part of a 2-edge-connected component of H, receives $1 - \varepsilon$ credits.
(C) Every connected component of H that contains at least one bridge receives 1 credit. Every connected component of H that is 2-edge-connected receives $2 - 2\varepsilon$ credit if it simple and 2 credits otherwise.
(D) Every 2EC-block of a connected component that contains bridges receives 1 credit.

We remark that the credits assigned according to (A1) and (A2) add up. We also observe that the credits assigned according to (A2) depend on the (unknown) optimal solution OPT. These *implicit credits* are needed in order to be able to compare to opt because using only a lower bound on opt that is based on the number of unmatched

Figure 4: Example showing that a lower bound based on the number of unmatched leaves in the initial matching M (which would be empty in this example) does not lead to a strong enough lower bound, which is why we introduce implicit credits. The dashed edges are edges of the forest F, while blue edges are links in OPT. Then, indeed, the number of unmatched leaves of the forest is 2 (and thus constant), but opt can be unbounded (by increasing the size of the above example in the obvious way).

leaves in M[7] is not sufficient to achieve any finite approximation guarantee, as one can see from the example in Figure 4.

We use credits(H) to denote the total number of credits of $H := (V, F \cup S)$ according to the above rules (A) – (D). To prove that our algorithm fulfills the desired approximation guarantee, we prove that we maintain Invariant 1, which in terms of ε can be restated as

$$\text{credits}(H) + |S| \;\leq\; (2-\varepsilon)\cdot \text{opt} + (1+3\varepsilon)\cdot\big(\text{opt} - n_{\text{comp}}\big)\,.$$

To this end, we will first show that Invariant 1 is fulfilled for the initial choice of $S = M$. Then we provide a procedure that maintains this invariant and at the end of which H is 2-edge-connected. In order to show that the invariant is maintained, we will show that the total credits$(H) + |S|$ never increases.

Besides Invariant 1, we will also maintain the following invariant.

Invariant 2. *Every lonely vertex of H has degree at most 2 in H.*

5.3 Invariants after the Initialization

Let $M \subseteq L$ be a maximum cardinality matching on the leaves of (V, F) that contains no link with both endpoints in the same path (we call such links *bad*). Our initial solution is $S = M$ and we let $H = (V, F \cup M))$. We next show that H fulfills Invariants 1 and 2. To this aim, we first relate the number of unmatched leaves of (V, F), which is the number of leaves of the initial graph $H = (V, F \cup M)$, to the size opt of an optimal solution.

Lemma 5.3. *The number $2n_{\text{comp}} - 2|M|$ of* unmatched *leaves, i.e., the number of leaves of (V, F) that do not have an incident edge in M, is at most $4\cdot\big(\text{opt} - n_{\text{comp}}\big)$.*

Proof. Let OPT $\subseteq L$ be an optimal solution and let $M_{\text{OPT}} \subseteq$ OPT be a maximal matching in OPT that contains no bad links. Then $|M| \geq |M_{\text{OPT}}|$ and thus it suffices to show that at most $4\cdot(\text{opt} - n_{\text{comp}})$ leaves of the forest (V, F) have no incident edge in the matching M_{OPT}.

First, we observe that every connected component of the forest (V, F) has at least two edges incident to it that have their other endpoint in a different connected component. Because bad links have both endpoints in the same connected component of the forest, this implies

$$\text{opt} \geq n_{\text{comp}} + \text{opt}_{\text{bad}}, \tag{1}$$

where opt_{bad} denotes the number of bad links in OPT. Let $V_{\text{leaf}} \subseteq V$ denote the set of leaves of the forest (V, F). Because every connected component of this forest is a path, we have

$$2\cdot n_{\text{comp}} = \big|V_{\text{leaf}}\big| \;=\; 2\cdot|M_{\text{OPT}}| + \big|\big\{v \in V_{\text{leaf}} : M_{\text{OPT}} \cap \delta(v) = \emptyset\big\}\big|.$$

Using the maximality of M_{OPT} and the fact that every leaf of (V, F) is the endpoint of a link in OPT, this implies

$$\begin{aligned}\text{opt} &\geq |M_{\text{OPT}}| + \big|\big\{v \in V_{\text{leaf}} : M_{\text{OPT}} \cap \delta(v) = \emptyset\big\}\big| - \text{opt}_{\text{bad}} \\ &= n_{\text{comp}} + \tfrac{1}{2}\cdot\big|\big\{v \in V_{\text{leaf}} : M_{\text{OPT}} \cap \delta(v) = \emptyset\big\}\big| - \text{opt}_{\text{bad}}.\end{aligned} \tag{2}$$

Adding up (1) and (2), we obtain

$$2\cdot(\text{opt} - n_{\text{comp}}) \geq \tfrac{1}{2}\cdot\big|\big\{v \in V_{\text{leaf}} : M_{\text{OPT}} \cap \delta(v) = \emptyset\big\}\big|. \qquad \square$$

We now show that $H = (V, F \cup M))$ fulfills Invariants 1 and 2.

Lemma 5.4. *$H = (V, F \cup M)$ satisfies Invariants 1 and 2.*

Proof. Invariant 2 follows from the fact that all vertices have degree at most 2 in F and this property is maintained by adding a matching on the leaves of (V, F).

Consider next Invariant 1. First, we observe that every connected component of H is either a path or a cycle. Moreover, because M contains no bad links, every cycle in H contains at least two links from M. Let us now give upper bounds on credits(H).

- The number of credits due to (A1) is the number of leaves of H, i.e., $2n_{\text{comp}} - 2|M|$. Therefore, we can upper bound this number of credits by
$$\begin{aligned}2n_{\text{comp}} - 2|M| &\overset{Lem.5.3}{\leq} \big(1-\tfrac{\varepsilon}{2}\big)\cdot\big(2\cdot n_{\text{comp}} - 2|M|\big) \\ &\quad + \tfrac{\varepsilon}{2}\cdot\big(4\cdot(\text{opt} - n_{\text{comp}})\big) \\ &= (2-\varepsilon)\cdot(n_{\text{comp}} - |M|) \\ &\quad + 2\varepsilon\cdot(\text{opt} - n_{\text{comp}}).\end{aligned}$$
- The number of credits according to (A2) is at most
$$\begin{aligned}\tfrac{1}{2}\sum_{v\in V}\big(|\delta_{\text{OPT}\cup F}(v)| - 2\big) &= \text{opt} + |F| - |V| \\ &= \text{opt} - n_{\text{comp}}.\end{aligned}$$
- The number of credits according to (B) is
$$(1-\varepsilon)\cdot|\{\ell \in M : \ell \text{ is a bridge of } H\}|.$$
- Let us now consider credit rule (C). The number of connected components of H that contain bridges is $\frac{1}{2}$ times the number of leaves of H, because each such connected component is a path. Thus, by Lemma 5.3, the connected components of H that contain bridges have at most
$$2\cdot(\text{opt} - n_{\text{comp}})$$
credits in total. Recall that every connected component that is 2-edge-connected is a cycle and contains at least two links from M. If it contains exactly two links from M it is simple. Thus, because $3\cdot(1-\varepsilon) \geq 2$, the number of credits of a connected component that is 2-edge-connected can be upper bounded by $(1-\varepsilon)$ times the number of links it contains (from M). Because these links are no bridges we can upper bound the total number of credits of connected components that are 2-edge-connected by
$$(1-\varepsilon)\cdot\big(|M| - |\{\ell \in M : \ell \text{ is a bridge of } H\}|\big).$$

[7] We will give such a bound in Lemma 5.3.

- Finally, because every connected component of H is either a cycle or a path, there are no 2EC-blocks in connected components containing bridges. Thus, there are no credits due to (D).

Summing up, we get

$$\begin{aligned}\text{credits}(H) &\leq (2-\varepsilon)\cdot(n_{\text{comp}}-|M|)\\ &\quad+(3+2\varepsilon)\cdot(\text{opt}-n_{\text{comp}})+(1-\varepsilon)\cdot|M|.\end{aligned}$$

Thus, $S = M$ implies

$$\begin{aligned}\text{credits}(H)+|S| &\leq (2-\varepsilon)\cdot(n_{\text{comp}}-|M|)\\ &\quad+(3+2\varepsilon)\cdot(\text{opt}-n_{\text{comp}})+(2-\varepsilon)\cdot|M|\\ &= (2-\varepsilon)\cdot\text{opt}+(1+3\varepsilon)\cdot(\text{opt}-n_{\text{comp}}).\end{aligned}$$

□

5.4 Bridge Covering using Alternating Trails

In this section we describe and analyze the bridge covering procedure. We use it to augment S in such a way that after the bridge covering step every connected component of $H = (V, F \cup S)$ is 2-edge-connected. In the following we assume that H has at least one bridge and describe an augmentation procedure that decreases the number of bridges of H by at least 1. We can then iteratively apply this procedure until every connected component of $H = (V, F \cup S)$ is 2-edge-connected.

Let C be a connected component of H that is not 2-edge-connected. We aim at "covering" at least one bridge of the component C. To this end we might use paths "going through" different connected components, but it is irrelevant for us what "happens inside these components". Thus, it is useful to consider the graph where these components are contracted. Moreover, we will not add or remove any links inside 2EC-blocks of C and thus we will also contract them.

Let G^C and H^C be the (multi-)graphs obtained from $(V, F \cup L)$ and H, respectively, by contracting every connected component except for C and contracting every 2EC-block of C. Then H^C is the union of a tree T^C with at least one edge (resulting from C by contracting its 2EC-blocks) and singletons (resulting from the contraction of connected components distinct from C). In the following we identify the edges/links in $(V, F \cup L)$ and the corresponding edges/links in G^C.

We will aim at "covering as many edges/bridges of the tree T^C as possible" by an *alternating trail* starting in x. See Figure 5 for an example.

We use the following definitions. A *trail* is a walk that uses every edge at most once. A *path* is a walk that visits every vertex at most once.

Definition 5.5. (alternating trail) Let z be a vertex of T^C that is distinct from x. We denote the x-z path in T^C by P^{xz} and the links in $S \cap P^{xz}$ by $\ell_1, \ldots, \ell_l$ in the order they appear on the path P^{xz} (starting from the link closest to x). We denote the endpoints of the link ℓ_i by u_i and v_i, where u_i is the endpoint closer to x in the path P^{xz}.

An *alternating x-z-trail* is a trail $P \subseteq L$ in G^C that has the following properties:

- P starts in x.

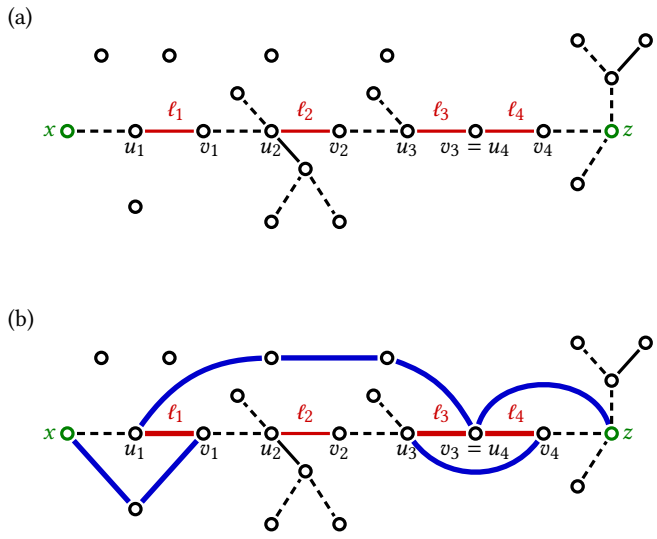


Figure 5: Picture (a) shows the graph G^C with a leaf x of T^C and a vertex z in T^C. Picture (b) shows an alternating x-z trail (bold edges). Solid edges are links from L, dashed edges are edges of the forest F. Links in $S \cap P^{xz}$ are red; edges in $P \setminus P^{xz}$ are shown in blue.

- P ends in the vertex $z \neq x$ in T^C.
- P consists alternatingly of
  - a path for which all interior vertices are not contained in the tree T^C, and
  - a link $\ell_i = \{u_i, v_i\}$ for some $i \in \{1, \ldots, l\}$, where P visits v_i before u_i.
- P visits every vertex outside of T^C at most once.
- The links in $P \cap P^x$ are visited in an order of increasing distance from x, i.e., for $i < j$ with $\ell_i, \ell_j \in P$, the link ℓ_i appears before ℓ_j on P.

Note that the last link of an alternating x-z trail P is always a link in $L \setminus P^{xz}$.

We will find an alternating trail P that start in x and ends in a vertex z of P^x, where z is as far away from x (in T^C) as possible. Then we will *augment* S along the alternating trail P. See Figure 6 for an example.

Definition 5.6 (augmenting along an alternating trail). If we *augment* the current partial solution $S \subseteq L$ of links by an alternating x-z trail P, this means that we replace S by $S \triangle P$. In other words, we remove the links in $P \cap S = P \cap P^{xz}$ from S and add the links in $P \setminus S = P \setminus P^{xz}$ to S.

Lemma 5.7. *Given a leaf x and an arbitrary vertex $z \neq x$ in the tree T^C, we can in polynomial time either find an alternating x-z trail or decide that no such trail exists.*

Proof. As in Definition 5.5, we denote the x-z path in T^C by P^{xz} and the links in $S \cap P^{xz}$ by $\ell_1, \ldots, \ell_l$ in the order they appear on the path P^{xz}. Also, we again denote the endpoints of the link ℓ_i by u_i and v_i, where u_i is the endpoint closer to x in the path P^{xz}. We

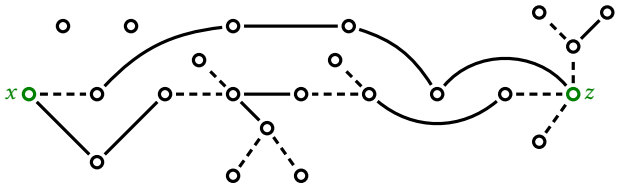


Figure 6: The result of the augmentation along the alternating trail from Figure 5 (in the graph G^C, i.e., before undoing contractions of 2EC-blocks and connected components of H).

construct a directed auxiliary graph with vertex set $\{x, \ell_1, \dots, \ell_l, z\}$ and arcs

- (ℓ_i, ℓ_j) if $i < j$ and there exists an u_i-v_j path in G^C for which all interior vertices are not contained in T^C;
- (x, ℓ_i) if there exists an x-v_i path in G^C for which all interior vertices are not contained in T^C;
- (ℓ_i, z) if there exists an u_i-z path in G^C for which all interior vertices are not contained in T^C.

Then, by the definition of alternating trails, every alternating x-z trail corresponds to a directed x-z path in this auxiliary graph. In particular, if an alternating x-z trail exist, z is reachable from x in the directed auxiliary graph. To find such an alternating trail, we compute a shortest x-z path $\overline{P}$ in the auxiliary graph (where shortest path means one with the minimum number of arcs).

We construct P from $\overline{P}$ by replacing every arc (ℓ_i, ℓ_j) of $\overline{P}$ by a u_i-v_j path whose internal vertices are not contained in T^C, and similarly replacing arcs (x, ℓ_i) and (ℓ_i, z) by x-v_i and u_i-z paths, respectiviely. We claim that P is an alternating x-z trail. To this end, we need to show that paths corresponding to different arcs of $\overline{P}$ are vertex disjoint, i.e., they have no common interior vertex. This follows from the fact that $\overline{P}$ is a shortest x-z path as one can see as follows.

Suppose there are arcs (ℓ_i, ℓ_j) and (ℓ_p, ℓ_q) in $\overline{P}$, appearing without loss of generality in this order on $\overline{P}$, where the u_i-v_j path and the u_p-v_q path corresponding to these arcs are not vertex disjoint. This would imply that v_q is reachable from u_i by a path whose internal vertices are not part of T^C. Hence, the auxiliary graph contains an arc (ℓ_i, ℓ_q). This is a contradiction to $\overline{P}$ being a shortest x-z path because we could use the arc (ℓ_i, ℓ_q) to shortcut it. A similar argument leads to a contradiction also if one (or both) of the arcs are of the form (x, ℓ_i) or (ℓ_i, z). □

The next lemma shows that when we augment S long an alternating x-z trail P, we merge all vertices and 2EC-blocks visited by P^{xz} into a single 2-edge-connected component, which also contains at least one vertex from each of the connected components of H that correspond to the vertices of P outside T^C.

Lemma 5.8. *$P \triangle P^{xz}$ is the edge set of a cycle in G^C containing all vertices visited by P or P^{xz}.*

Proof. First, we show that all vertices visited by P or P^{xz} have even degree in $P \triangle P^{xz}$. Indeed, because all such vertices have even degree in the disjoint union of the two x-z trails P and P^{xz}, they also have even degree in the symmetric difference $P \triangle P^{xz}$. (Here we use that $P \triangle P^{xz}$ arises from the disjoint union by removing pairs of parallel edges.) Moreover, no vertex has degree larger than 2 in $P \triangle P^{xz}$ because whenever an interior vertex of P^{xz} is visited by P, the trail P uses a link $\ell_i \in P \cap P^{xz}$ that is incident to this vertex. We conclude that all vertices visited by P or P^{xz} have degree 0 or 2 in $P \cap P^{xz}$. Hence, it remains to show that all these vertices are connected in $P \cap P^{xz}$.

To see this, we first observe that every vertex visited by P but not by P^{xz} is connected to some vertex visited by P^{xz} in $P \setminus P^{xz} \subseteq P \triangle P^{xz}$ (by the definition of an alternating trail). Hence it suffices to shows that every vertex v on P^{xz}, except for x, is connected to another vertex v' on P^{xz} that is closer to x (on P^{xz}). This is indeed the case because either $v = v_j$ for some $j \in \{1, \dots, l\}$ with $\ell_i \in P$, in which case a path in $P \setminus P^{xz}$ that precedes v in P connects v to a vertex $v' = u_i$ with $i < j$, or the edge $\{v', v\}$ preceding v on the path P^{xz} (viewed as a path starting at x) is contained in $P \triangle P^{xz}$. □

The next lemma will be crucial to prove that Invariant 1 is maintained when augmenting along P. Informally speaking, we use it to show that credits(H) decreases sufficiently (to pay for the increase of $|S|$) for one of the following reasons:

- our alternating trail reaches a leaf of T^C, in which case the required number of credits in H decreases due to either (A1) or (D);
- a vertex of degree at least 3 in T^C gets merged into another 2-edge-connected component; this vertex must correspond to a 2EC-block by Invariant 2; thus it had 1 credit (by (D)) before it was merged;
- we merge bridges in $D \cap P^{xz}$ or lonely vertices on P^{xz} into a nontrivial 2-edge-connected component, leading to a decrease of the credits in H required by (B) and (A2).

We remark that the statement of the below lemma is nontrivial only for $q \in \{0, 1\}$. Moreover, we highlight that the vertex set W of the path P^{xz} can contain both original vertices from V and vertices arising from the contraction of a 2EC-block of C.

Lemma 5.9. *Let z be a vertex that is furthest away from x in T^C among all vertices reachable from x by an alternating trail. Let P^{xz} be the x-z path in T^C and let W be the vertex set of P^{xz}. Then at least one of the following holds:*

- *z is a leaf of T^C;*
- *$W \setminus \{x\}$ contains a vertex with degree at least 3 in T^C;*
- *we have*

$$\sum_{w \in W \setminus \{x\}} \left(|\delta_{F \cup \mathrm{OPT}}(v)| - 2 \right) \;\geq\; 2 - q$$

where q is the number of links $\ell \in P^{xz} \cap S$.

Proof. Suppose that z is not a leaf of T^C and $W \setminus \{x\}$ contains only vertices of degree at most 2 in T^C. Then, because W is the vertex set of the x-z path in T^C, all vertices in $W \setminus \{x\}$ have degree exactly 2 in T^C. See Figure 7. Let $\overline{W}$ denote the set of vertices of T^C that are not contained in W, i.e., that are not visited by P^{xz}. We observe that all vertices in $\overline{W}$ have a larger distance from x in T^C than z. Here, we used that x is a leaf of T^C and all vertices on the x-z path P^{xz} in T^C, which has vertex set W, have degree 2 in T^C.

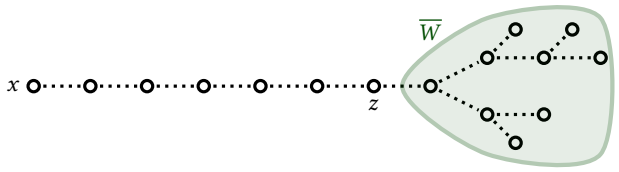


Figure 7: Illustration of T^C in the proof of Lemma 5.9.

Because $(V, \mathrm{OPT} \cup F)$ is 2-edge-connected, it contains two edge-disjoint paths P_1, P_2 from x to $\overline{W}$. Each of these paths visits at least one vertex in $W \setminus \{x\}$ due to the following. Let $i \in \{1, 2\}$ and let u be the first vertex on P_i that is distinct from x and contained in T^C; this vertex exists because the last vertex of P_i is an element of $\overline{W}$. Then the subpath of P_i from x to u is an alternating trail. By the choice of z, this implies $u \in W$.

For $q \geq 2$, the statement of the lemma is trivial and thus it suffices to distinguish the following two cases.

Case 1: $q = 0$

One of the edge-disjoint paths P_1, P_2, say P_1, does not contain the unique edge incident to x in P^{xz}. Let u_1 be the first vertex in W visited by P_1 and let v_1, v_2 be the last vertices in W visited by P_1, P_2, respectively. See Figure 8.

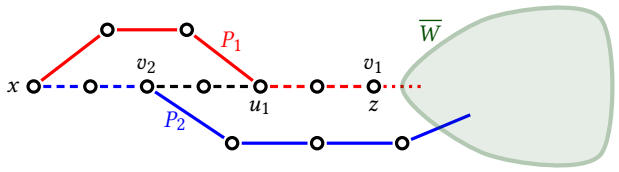


Figure 8: Example of Case 1. Dashed edges are part of the forest F, while solid edges are links, i.e., elements of L. Dotted edges can be any edges in $F \cup L$.

Then the edge e_0 by which P_1 enters u_1 is not contained in P^{xz}; here we used that P_1 does not contain the unique edge incident to x in P^{xz}. Moreover, because P_1 and P_2 end in $\overline{W}$, for each $i \in \{1, 2\}$ the vertex $v_i \in W$ is not the endpoint of P_i and thus there is an edge $e_i \in P_i$ by which P_i leaves v_i. By the definition of v_i, this edge is not contained in P^{xz}. Using that $q = 0$ implies $P^{xz} \subseteq F$, we conclude

$$\sum_{v \in W \setminus \{x\}} \left(|\delta_{F \cup \mathrm{OPT}}(v)| - 2\right) \geq 3 + \sum_{v \in W \setminus \{x\}} \left(|\delta_{P^{xz}}(v)| - 2\right) \geq 2 = 2 - q.$$

Case 2: $q = 1$

Let $\{u, v\}$ be the unique link in $P^{xz} \cap S$, where without loss of generality u is closer to x in P^{xz}. We assume $|\delta_{F \cup \mathrm{OPT}}(v')| = 2$ for all $v' \in W$ and derive a contradiction. Because $P^{xz} \setminus \{u, v\} \subseteq F$, this assumption implies that if a path P_i (with $i \in \{1, 2\}$) visits the vertex u, then the x-u subpath of P^{xz} is contained in P_i. Similarly, if a path P_i (with $i \in \{1, 2\}$) visits v, then the v-z subpath of P^{xz} is contained in P_i. Moreover, the assumption also implies that every vertex $w \in W \setminus \{x, z\}$ is visited by at most one of the paths P_1, P_2. Combining these observations with the fact that each of the paths P_1, P_2 visits at least one vertex in W, we obtain the following. One of the paths P_1, P_2, say P_1 starts with the x-u subpath of P^{xz} and then visits no further vertices in $W \setminus \{z\}$; the other path, P_2, contains the v-z subpath of P^{xz} and does not visit any vertex in $W \setminus \{x\}$ before v. See Figure 9.

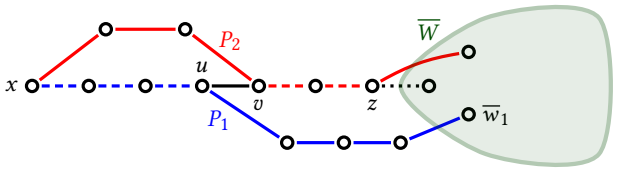


Figure 9: Example illustrating the proof of Case 2. Dashed edges are part of the forest F, while solid edges are links, i.e., elements of L. Dotted edges can be any edges in $F \cup L$.

If the x-v subpath of P_2 visits a vertex $\overline{w} \in \overline{W}$, then this contradicts the choice of z (because if we choose $\overline{w}$ to be the first vertex on P_2 that is contained in $\overline{W}$, the x-$\overline{w}$ subpath of P_2 would be an alternating trail).

Let $\overline{w}_1$ be the first vertex on P_1 that is contained in $\overline{W}$. (Such a vertex exists because P_1 ends in $\overline{W}$.) Next, we show that there exists an alternating x-$\overline{w}_1$ trail, leading to a contradiction. If the x-v subpath of P_2 and the u-$\overline{w}_1$ subpath of P_1 are not vertex disjoint, then there exists a x-$\overline{w}_1$ path that does not contain any vertex of T^C as an interior vertex; this path is an alternating x-$\overline{w}_1$ trail. Otherwise, we obtain an alternating x-$\overline{w}_1$ trail by concatenating the x-v subpath of P_2, the link $\{v, u\}$, and the u-$\overline{w}_1$ subpath of P_1. □

By Lemma 5.8, every vertex whose degree increases in the augmentation step becomes part of a 2EC-block and is not lonely. Thus, Invariant 2 is maintained. We now show that also Invariant 1 is maintained.

Lemma 5.10. *Let z be the vertex that is furthest away from x in T^C among all vertices reachable from x by an alternating trail and let P be an alternating x-z trail. When augmenting along P, the number of bridges in H reduces by at least 1. Moreover,* $\mathrm{credits}(H) + |S|$ *does not increase.*

Proof. By Lemma 5.8, when augmenting along P, we do not create any new bridges and all of the edges on the path P^{xz} become part of a 2-edge-connected component. This implies that the number of bridges of H decreases by at least 1.

Let k denote the number of links in $P \cap P^{xz}$. When augmenting along P, the cardinality of S increases by exactly $|P| - 2k$. Moreover, the number of connected components of H decreases by exactly $|P| - 1 - 2k$. We distinguish two cases.

Case 1: After the augmentation along P, the connected component containing x contains at least one bridge.

Each of the $|P| - 2k$ connected components that get merged when augmenting along P (including the component C) had at least

1 credit before the augmentation by (C). The connected component resulting from the augmentation has 1 credit and thus the number of credits due to (C) decreases by at least $|P| - 1 - 2k$. Therefore, it remains to show that the number of credits due to (A), (B), and (D) reduces by at least 1 when augmenting along P.

By Lemma 5.8, when augmenting along P, all vertices of the path P^{xz} become part of the same 2EC-block B and we do not create any new bridges in H. The block B has exactly 1 credit by (D). The credits of all vertices and blocks that are not contained in B remain unchanged. Moreover, the credits of links in $S \setminus P^{xz}$ do not change. Vertices and links in B have no credits anymore after the augmentation. Hence, it suffices to show that the total number of credits on vertices, bridges, and 2EC-blocks in P^{xz} was at least 2 before the augmentation.

Before the augmentation, x was a leaf of T^C and thus it was either a leaf of H or it resulted from the contraction of a 2EC-block. In both cases it had at least 1 credit by (A1) and (D).

Let W denote the vertex set of P^{xz}. If $W \setminus \{x\}$ contains a vertex that arose from the contraction of a 2EC-block, this block had 1 credit before the augmentation by (D). Otherwise, all vertices in $W \setminus \{x\}$ are lonely and hence they have degree at most 2 by Invariant 2. If z is a leaf of T^C it has 1 credit by (A1) before the augmentation. Otherwise, Lemma 5.9 implies that before the augmentation the total number of credits on bridges in P^{xz} (due to (B)) plus the total number of credits on $W \setminus \{x\}$ (due to (A2)) was at least 1. (Here, we used that all links in P^{xz} were bridges before the augmentation and that $1 - \varepsilon \geq \frac{1}{2}$.)

Case 2: After the augmentation along P, the connected component containing x is 2-edge-connected.

The connected component C had 1 credit before the augmentation. After the augmentation, the connected component containing x has at most 2 credits. The number of all other types of credits does not increase. Moreover, because the connected component containing x is 2-edge-connected after the augmentation, both x and z must be leaves of the tree T^C. Thus, x and z were either leaves of H or (contracted) 2EC-blocks of H. This implies that the total number of credits according to (A1) and (D) decreases by at least 2. □

5.5 Gluing

In this section we describe and analyze the gluing phase. For the purpose of gluing, we consider auxiliary graphs that are obtained from the graph $(V, F \cup L)$ by contracting some connected components of the current $H = (V, F \cup S)$, similarly to the bridge covering phase. We will merge several connected components of H by finding an affordable cycle in one of the auxiliary graphs. We begin by defining these auxiliary graphs, and our main tools, namely good cycles.

Recall that a graph G is called bridgeless if it does not contain any bridge (i.e. cut-edge). Let $F \cup S$ be a bridgeless 2-edge-cover of $(V, F \cup L)$ and $H = (V, F \cup S)$. Throughout the gluing phase $F \cup S$ will remain bridgeless and thus the connected components and the 2-edge-connected components of H coincide. Hence, we will sometimes refer to these as the components of H. We let G_H be the (multi-)graph obtained from $(V, F \cup L)$ after contracting each component of H into a single vertex. Notice that the edges of G_H correspond to a subset of the links of the original graph.

For a simple component C, we let $G_{H|C}$ be the (multi-)graph obtained from $(V, F \cup L)$ after contracting each connected components of H except for C into a single vertex. Then the edges of $G_{H|C}$ correspond to a subset of the links of the original graph plus the two paths $P_1 \subseteq F$ and $P_2 \subseteq F$ in the simple component C. Note that G_H and $G_{H|C}$ are not necessarily simple graphs and might have parallel edges. With a slight abuse of notation we identify the edges in G_H and $G_{H|C}$ with the edges in $(V, F \cup L)$.

Definition 5.11 (good cycle). A *good cycle of H in $(V, F \cup L)$ that affects a simple component C* is a cycle in $G_{H|C}$ that contains all the edges of the two paths P_1 and P_2 of C in F and has at least one more vertex.

See Figure 10. In the gluing phase, we will repeatedly merge components of H by *gluing* either along a cycle in G_H or along a good cycle in $G_{H|C}$ (for some simple component C). Next, we define these gluing operations.

Definition 5.12. Gluing H along a cycle Q of G_H is the process of adding the links of Q to H.

Gluing $H = (V, F \cup S)$ along a good cycle Q affecting a simple component C is the process of first removing the two links of C from S and then adding the links of Q to S.

Notice that it is possible that one link ℓ belonging to both C and Q is first removed and then added back. However this cannot happen for both links in the simple component C simultaneously. See Figure 10. We also remark that in order to obtain a better-than-2 approximation it is not sufficient to use only gluing operations that glue H along cycles in G_H as the example from Figure 2 shows.

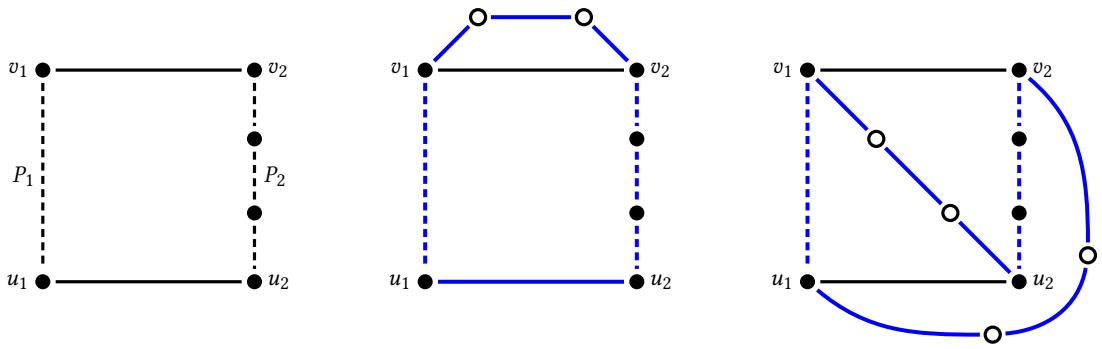
Lemma 5.13. *Given a bridgeless 2-edge-cover $F \cup S$ of $(V, F \cup L)$, there exists a polynomial time algorithm that computes a good cycle of $H = (V, F \cup S)$ in $(V, F \cup L)$ or verifies that no such good cycle exists.*

Proof. Fix a simple component C of H. We show that there exists a polynomial time algorithm that computes a good cycles of H in $(V, F \cup L)$ that affects C or verifies that no such good cycle exist. The claim follows by applying this algorithm to each simple component of H.

Let P_1 and P_2 be the paths of C that belong to F and let u_1 and v_1 be the endpoints of P_1 and u_2 and v_2 be the endpoints of P_2 such that $\{u_1, u_2\} \in S$ and $\{v_1, v_2\} \in S$ are the unique links of the simple component C. See Figure 10. We call a path in $G_{H|C}$ a *good path* if one of its endpoints is u_1 or v_1, the other endpoint is u_2 or v_2, and all its internal vertices do not belong to C, i.e., they are vertices that arose from the contraction of components of H. Every good cycle affecting C is the union of P_1, P_2 and two vertex disjoint good paths. More precisely, if there exists a good cycle affecting C then

(a) there exists a good v_1-v_2 path or a good u_1-u_2 path of length at least two, or
(b) there exist a good v_1-u_2 path and a good u_1-v_2 path such that at least one of these paths has length at least two.

We give a polynomial time algorithm that finds a good cycles whenever (a) of (b) holds. First we observe that for vertices $a \in \{u_1, v_1\}$



(a) a simple component (b) a good cycle (c) another good cycle

Figure 10: This figure shows a simple component C (a), and two different good cycles affecting C (shown in blue in (b) and (c)). Solid edges are links and dashed edges are edges of the forest (V, F). Filled vertices show elements of the original vertex set V, while other vertices resulted from the contraction of a component of H. Notice that when gluing along the good cycle depicted in (b), the link $\{u_1, u_2\}$ will remain in S. (Strictly speaking, according to Definition 5.12, we first remove it, but then add it again.) When gluing H along the good cycle depicted in (c), both links $\{v_1, v_2\}$ and $\{u_1, u_2\}$ will be removed from S.

and $b \in \{u_2, v_2\}$ we can in polynomial time find a good a-b path or decide that such a path does not exist (by searching for an a-b path in the graph arising from $G_{H|C}$ by removing all vertices of C (and their incident edges) except for the vertices a and b). We can also find a good a-b path of length at least two or decide that no such path exists by looking for a good a-b path in the graph where we remove the link $\{a, b\}$ (if it exists).

Our algorithm first checks if a good v_1-v_2 path P_v of length at least two exists. If this is the case, we obtain a good cycle by taking the union of P_1, P_v, P_2, and the link $\{u_1, u_2\}$. We proceed analogously if a good u_1-u_2 path of length at least two exists. Thus, we find a good cycle if (a) holds. Otherwise, we check if there exists a good v_1-u_2 path P_1' and a good v_2-u_1 path P_2'. If possible, we choose the good paths P_1' and P_2' such that they have length at least two. If (b) holds, both P_1' and P_2' exist and at least one of them has length at least two. The paths P_1' and P_2' must be vertex-disjoint as otherwise their union contains a good v_1-v_2 path of length at least two, in which case (a) holds. Then the union of P_1, P_1', P_2 and P_2' is a good cycle. □

Next we show that if we can glue H along a good cycle, our credit invariant (Invariant 1) is maintained. We also observe that the number of components of H decreases strictly.

Lemma 5.14. *Given a bridgeless 2-edge-cover $F \cup S$ of $(V, F \cup L)$ and a good cycle Q of $H = (V, F \cup S)$ in $(V, F \cup L)$, let $H' = (V, F \cup S')$ be the graph obtained by gluing H along Q. Then $F \cup S'$ is a bridgeless 2 edge-cover of $(V, F \cup L)$ such that* $\text{credits}(H) + |S| \geq \text{credits}(H') + |S'|$ *and H' has fewer connected components than H.*

Proof. Let C be the simple component of H that is affected by Q and let $C_1, ..., C_k$ be the other connected components of H such that their corresponding vertex in $G_{H|C}$ belongs to Q.

Clearly H' is also a bridgeless 2-edge-cover. Furthermore, the vertices of C and the vertices of $C_1, \ldots, C_k$, form a single connected component C' in H'. Observe that we have removed 2 links and added $k + 2$ links, therefore $|S'| - |S| = k$. By the credit rule (C), we need 2 credits for the new component C' (which is not simple since it contains at least 3 links). We can compensate this with the credits of the components $C, C_1, \ldots, C_k$ of H, which are at least $(k + 1)(2 - 2\varepsilon) \geq k + 2$ (since $\varepsilon = \frac{1}{4}$ and $k \geq 1$). Thus, $\text{credits}(H) - \text{credits}(H') \geq k$ and therefore $\text{credits}(H) + |S| - \text{credits}(H') - |S'| \geq k - k = 0$.

Finally, we observe that the connected components of H not touched by Q are not modified, hence their credits do not change. □

It remains to consider the case where there is no good cycle. In order to prove the credit invariant, we first show that in this case every simple component contains a vertex with an implicit credit due to (A2). Such vertices will be called *rich*.

Definition 5.15 (rich vertex). We say that a vertex $v \in V$ is *rich* if $|\delta_{\text{OPT} \cup F}(v)| \geq 3$ and it belongs to a simple connected component of H.

Lemma 5.16. *Let C be a simple component of $H = (V, F \cup S)$ and suppose $C \neq H$. If there is no good cycle affecting C, then C contains a rich vertex.*

Proof. Let $P_1 \subseteq F$ and $P_2 \subseteq F$ be the paths of F in C, such that v_1, u_1 are the endpoints of P_1, v_2 and u_2 are the endpoints of P_2, and $\{v_1, v_2\}, \{u_1, u_2\} \in S$ are the two links contained in C.

Assume that the claim does not hold, i.e., C does not have any rich vertex. Then in $(V, F \cup \text{OPT})$ the degree of each vertex of C is exactly 2. Therefore, the internal vertices of P_1 and P_2 are not incident to any edge of OPT and each of their endpoints u_1, u_2, v_1, and v_2 is incident to exactly one edge of OPT.

Now consider the graph $G_{H|C}$. Since $(V, F \cup \text{OPT})$ is 2-edge-connected, there must exist two edge-disjoint paths P_1' and P_2' from the vertices of P_1 to the vertices of P_2 in $G_{H|C}$ that only contain links of OPT. Observe that, since by assumption C has no rich

vertex, the set of endpoints of P'_1 and P'_2 is exactly $\{u_1, u_2, v_1, v_2\}$ and none of the internal vertices of P'_1 and P'_2 belongs to C.

By a case analysis similar to the one in the proof of Lemma 5.13, the only possibility for $P'_1 \cup P'_2$ not to induce a good cycle affecting C is that $P'_1 \cup P'_2$ is a matching of size 2 between the endpoints of P_1 and the endpoints of P_2. However OPT must contain two links between the vertex set $V(C)$ of C and its complement $V \setminus V(C)$ (since $C \neq H$ and $(V, F \cup \text{OPT})$ is 2-edge-connected). Hence, in this case some endpoint of P_1 or of P_2 must be rich, a contradiction. □

Finally, we complete the analysis of the gluing phase by showing that the credit invariant (Invariant 1) is maintained and the runtime is polynomial.

Lemma 5.17. *Given a bridgeless 2-edge-cover H of $(V, F \cup L)$, the gluing phase of Algorithm 1 yields a 2-edge-connected spanning subgraph $H^* = (V, F \cup S^*)$ of $(V, F \cup L)$ such that $|S| + \text{credits}(H) \geq |S^*| + \text{credits}(H^*)$ in polynomial time.*

Proof. For this purpose, we show that if H is not already 2-edge-connected, then in every iteration of the gluing phase, we obtain in polynomial time a bridgeless subgraph $H' = (V, F \cup S')$ of $(V, F \cup L)$ from $H = (V, F \cup S)$ such that $|S| + \text{credits}(H) \geq |S'| + \text{credits}(H')$ and H' has fewer connected component than H. This implies that the number of iterations is linear in $|V|$.

First observe that if H has a good cycle, then using Lemmas 5.13 and 5.14 we are done. Therefore we can assume that no good cycle of H exists in $(V, F \cup L)$. Then, by Lemma 5.16 every simple component of H has at least one rich vertex, which has at least $\frac{1}{2}$ credits by (A2). Thus, every component of H is either simple with at least one rich vertex or is non-simple. In particular the credits of each simple component C plus the credits of its rich vertices add up to at least $(2 - 2\varepsilon) + \frac{1}{2} = 2$.

It remains to show that gluing $H = (V, F \cup S)$ along an arbitrary cycle $Q = (W, L')$ in G_H (such a cycle must exist because $(V, F \cup L)$ is 2-edge-connected) does not increase $\text{credits}(H) + |S|$. Let $H' = (V, F \cup S')$ be the resulting graph, where $S' = S \cup L'$ and C' is the new connected component formed during the gluing. Notice that C' is not simple since it contains strictly more than two links. Observe that $|S'| - |S| = |L'|$, and we need 2 credits for C' (due to the credit rule (C)). From the above discussion each of the $|L'|$ many connected component of H involved in Q can contribute with at least 2 credits. Thus,

$$|S| + \text{credits}(H) - |S'| - \text{credits}(H') \geq -|L'| + 2 \cdot |L'| - 2 \geq 0.$$

□

REFERENCES

[1] D. Adjiashvili. 2017. Beating approximation factor two for weighted tree augmentation with bounded costs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2384–2399.

[2] Jaroslaw Byrka, Fabrizio Grandoni, and Afrouz Jabal-Ameli. 2020. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to Steiner tree. In *ACM SIGACT Symposium on Theory of Computing (STOC)*. 815–825. https://doi.org/10.1145/3357713.3384301

[3] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2013. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM* 60, 1 (2013), 6:1–6:33.

[4] Federica Cecchetto, Vera Traub, and Rico Zenklusen. 2021. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *ACM SIGACT Symposium on Theory of Computing (STOC)*. 370–383. https://doi.org/10.1145/3406325.3451086

[5] Joseph Cheriyan, Robert Cummings, Jack Dippel, and J. Zhu. 2020. An Improved Approximation Algorithm for the Matching Augmentation Problem. *CoRR* abs/2007.11559 (2020). arXiv:2007.11559 https://arxiv.org/abs/2007.11559

[6] Joe Cheriyan, Jack Dippel, Fabrizio Grandoni, Arindam Khan, and Vishnu V. Narayan. 2020. The matching augmentation problem: a 7/4-approximation algorithm. *Math. Program.* 182, 1 (2020), 315–354. https://doi.org/10.1007/s10107-019-01394-z

[7] Joseph Cheriyan, András Sebö, and Zoltán Szigeti. 2001. Improving on the 1.5-Approximation of a Smallest 2-Edge Connected Spanning Subgraph. *SIAM J. Discret. Math.* 14, 2 (2001), 170–180. https://doi.org/10.1137/S0895480199362071

[8] Joseph Cheriyan and Ramakrishna Thurimella. 2000. Approximating Minimum-Size k-Connected Spanning Subgraphs via Matching. *SIAM J. Comput.* 30, 2 (2000), 528–560. https://doi.org/10.1137/S009753979833920X

[9] Nachshon Cohen and Zeev Nutov. 2013. A (1+ln 2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theor. Comput. Sci.* 489-490 (2013), 67–74. https://doi.org/10.1016/j.tcs.2013.04.004

[10] E. A. Dinits, A. V. Karzanov, and M. V. Lomonosov. 1976. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization* (1976), 290–306.

[11] G. Even, J. Feldman, G. Kortsarz, and Z. Nutov. 2009. A 1.8 Approximation Algorithm for Augmenting Edge-connectivity of a Graph from 1 to 2. *ACM Transactions on Algorithms* 5, 2 (2009), 21:1–21:17.

[12] S. Fiorini, M. Groß, J. Könemann, and L. Sanità. 2018. Approximating Weighted Tree Augmentation via Chvatal-Gomory Cuts. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 817–831.

[13] Harold N. Gabow and Suzanne Gallagher. 2012. Iterated Rounding Algorithms for the Smallest k-Edge Connected Spanning Subgraph. *SIAM J. Comput.* 41, 1 (2012), 61–103. https://doi.org/10.1137/080732572

[14] Waldo Galvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Krzysztof Sornat. 2019. On the Cycle Augmentation Problem: Hardness and Approximation Algorithms. In *Workshop on Approximation and Online Algorithms (WAOA)*.

[15] Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. 2021. Breaching the 2-Approximation Barrier for the Forest Augmentation Problem. *arXiv preprint arXiv:2112.11799* (2021).

[16] Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. 2018. Improved approximation for tree augmentation: saving by rewiring. In *ACM SIGACT Symposium on Theory of Computing (STOC)*. 632–645. https://doi.org/10.1145/3188745.3188898

[17] Christoph Hunkenschröder, Santosh S. Vempala, and Adrian Vetta. 2019. A 4/3-Approximation Algorithm for the Minimum 2-Edge Connected Subgraph Problem. *ACM Trans. Algorithms* 15, 4 (2019), 55:1–55:28. https://doi.org/10.1145/3341599

[18] Kamal Jain. 2001. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica* 21, 1 (2001), 39–60.

[19] Samir Khuller and Uzi Vishkin. 1994. Biconnectivity approximations and graph carvings. *J. ACM* 41, 2 (1994), 214–235.

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

[21] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. 2004. Hardness of Approximation for Vertex-Connectivity Network Design Problems. *SIAM J. Comput.* 33, 3 (2004), 704–720. https://doi.org/10.1137/S0097539702416736

[22] G. Kortsarz and Z. Nutov. 2016. A Simplified 1.5-Approximation Algorithm for Augmenting Edge-Connectivity of a Graph from 1 to 2. *ACM Transactions on Algorithms* 12, 2 (2016), 23:1–23:20.

[23] H. Nagamochi. 2003. An Approximation for Finding a Smallest 2-Edge-Connected Subgraph Containing a Specified Spanning Tree. *Discrete Applied Mathematics* 126, 1 (2003), 83–113.

[24] Zeev Nutov. 2017. On the Tree Augmentation Problem. In *European Symposium on Algorithms (ESA)*. 61:1–61:14. https://doi.org/10.4230/LIPIcs.ESA.2017.61

[25] Zeev Nutov. 2021. Approximation algorithms for connectivity augmentation problems. In *International Computer Science Symposium in Russia*. 321–338.

[26] Alexander Schrijver. 2003. *Combinatorial Optimization, Polyhedra and Efficiency*. Springer.

[27] András Sebö and Jens Vygen. 2014. Shorter tours by nicer ears: 7/5-Approximation for the graph TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica* 34, 5 (2014), 597–629. https://doi.org/10.1007/s00493-014-2960-3

[28] Vera Traub and Rico Zenklusen. 2022. A Better-Than-2 Approximation for Weighted Tree Augmentation. In *2021 IEEE Symposium on Foundations of Computer Science (FOCS)*. 1–12.

[29] Vera Traub and Rico Zenklusen. 2022. Local Search for Weighted Tree Augmentation and Steiner Tree. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 3253–3272.