Improved Approximation Algorithms for Bounded-Length Path Augmentation

Felix Hommelsheim[1]
University of Cologne, Germany

Abstract

The Forest Augmentation Problem (FAP) asks for a minimum set of additional edges (links) that makes a given forest 2-edge-connected while spanning all vertices. A key special case is the Path Augmentation Problem (PAP), where the input forest consists of vertex-disjoint paths. Grandoni, Jabal Ameli, and Traub [STOC'22] recently broke the long-standing 2-approximation barrier for FAP, achieving a 1.9973-approximation. A crucial component of this result was their 1.9913-approximation for PAP; the first better-than-2 approximation for PAP.

In this work, we present a 1.9412-approximation for bounded-length PAP, in which each path of the input forest has constant length $p_{\max}$. The running-time of our algorithm is $O(n^{p_{\max}})$. One of our key innovations is a $(\frac{11}{6} + \varepsilon)$-approximation preserving reduction to so-called structured instances, which simplifies the problem and enables our improved approximation. Additionally, we introduce a new relaxation inspired by 2-edge covers and analyze it via a corresponding packing problem, where the relationship between the two problems is similar to the relationship between 2-edge covers and 2-matchings. Using a factor-revealing LP, we bound the cost of our solution to the packing problem w.r.t. the relaxation and derive a strong initial solution. We then transform this solution into a feasible PAP solution, combining techniques from FAP and related connectivity augmentation problems, along with new insights.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Routing and network design problems

Keywords and phrases Approximation algorithms, path augmentation, connectivity augmentation, survivable network design

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.7

Category APPROX

Related Version *Full Version*: `https://arxiv.org/abs/2505.15324` [28]

Funding The work of the first author is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) by the grant HO 7562/2-1 (project ID: 573939419).

1 Introduction

Designing reliable networks that can withstand failures is a fundamental challenge in real-world applications. The field of *survivable network design* (SND) aims to construct cost-efficient networks that maintain connectivity even when some edges fail. Many natural SND problems are NP-hard (even APX-hard), justifying the study of approximation algorithms. A landmark result in this area is Jain's 2-approximation algorithm [31], which applies to a broad class of connectivity problems. However, breaking the 2-approximation barrier remains a major open question, even for many special cases of SND.

[1] A major part of this work was done while the author was affiliated with the University of Bremen, Germany.

In this paper, we focus on the Path Augmentation Problem (PAP), which is a special case of the Forest Augmentation Problem (FAP), a fundamental network augmentation problem. In the Forest Augmentation Problem we are given an undirected graph $G = (V, F \cup L)$, where F is a forest and $L \subseteq \binom{V}{2}$ is a set of additional edges (links) and the goal is to find a minimum-cardinality subset $S \subseteq L$ such that $(V, F \cup S)$ is 2-edge-connected[2]. The assumption that F is a forest is no restriction, as one can obtain an equivalent instance by contracting each 2-edge-connected component of F into a single vertex. The Path Augmentation Problem (PAP) is a special case of FAP where the given forest is composed of vertex-disjoint paths. We formally denote an instance of PAP by $G = (V, E(\mathcal{P}) \cup L)$, where $\mathcal{P}$ is the set of vertex-disjoint paths and $E(\mathcal{P})$ denotes its corresponding edge set. The goal is to find a minimum-cardinality subset $S \subseteq L$ such that $(V, E(\mathcal{P}) \cup S)$ is 2-edge-connected. Both problems are APX-hard [16, 18], ruling out the possibility of a Polynomial-Time Approximation Scheme (PTAS), unless P = NP. Until recently, no polynomial-time algorithm achieving a better-than-2 approximation was known for either problem.

Grandoni, Jabal Ameli, and Traub [25] were the first to break this barrier for PAP and FAP. In more detail, they obtained a 1.9913-approximation for PAP[3]. Combining this result with another result of [25] led to a 1.9973-approximation for FAP.

1.1 Our Contribution

In this paper, we consider *bounded-length PAP*, the special case of PAP in which each path in $\mathcal{P}$ contains at most $p_{\max}$ edges. This problem generalizes several well-established problems in connectivity augmentation, such as the unweighted 2-edge connected spanning subgraph problem (2ECSS), in which each path has length 0, or the Matching Augmentation Problem (MAP), in which each path has length 1; see Section 1.2 for more details. Our main result is a substantial improvement for bounded-length PAP.

▶ **Theorem 1.** *There is a* 1.9412*-approximation algorithm for PAP with running-time* $n^{O(p_{\max})}$.

To achieve this improvement, we introduce a new $(\frac{11}{6} + \varepsilon)$-approximation preserving reduction to structured instances, which is key to our approach. Additionally, we introduce a new relaxation of PAP, called 2-Edge Cover with Path Constraints (2ECPC), in which we have to compute a minimum-cardinality set of links $X \subseteq L$ such that $(V, E(\mathcal{P}) \cup X)$ is a 2-edge cover with the additional property that each path $P \in \mathcal{P}$ has two outgoing links. Instead of directly obtaining an approximate solution for 2ECPC, we consider a novel packing problem, called the Track Packing Problem (TPP). TPP is in some sense the corresponding packing problem to the covering problem 2ECPC, where the relationship between the two problems is inspired by the relationship between 2-edge covers and 2-matchings (precise definitions follow). We compute up to $|\mathcal{P}| + 2$ many solutions to TPP and analyze their cost using a factor-revealing linear program (LP). The resulting approximate solution serves as a suitable starting point for our framework. Later, this solution is turned into a feasible solution, highly relying on the fact that the input instance is structured. Our techniques also combine insights from [25] with methods used in other augmentation problems, together with new ideas tailored for structured instances of bounded-length PAP. We believe that

[2] A graph is k-edge-connected (kEC) if it remains connected after the removal of an arbitrary subset of $k - 1$ edges.

[3] In [25] a different result is shown with a bi-criteria approximation ratio. However, the result stated here for PAP is given implicitly in [25].

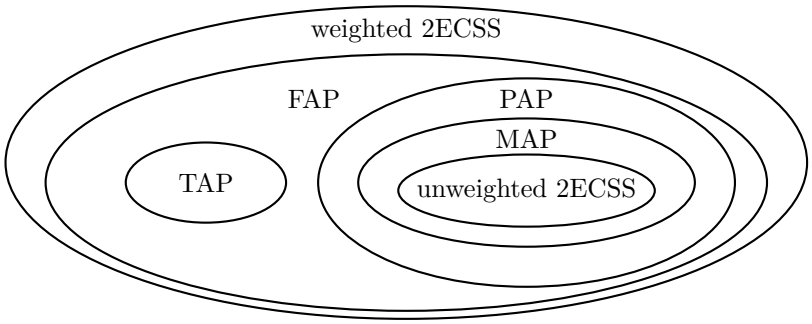

Figure 1 Hierarchical relation between weighted/unweighted 2ECSS, FAP, TAP, PAP, and MAP.

both the reduction to structured instances and the new relaxation 2ECPC as well as the corresponding packing problem TPP may be of independent interest for other connectivity augmentation problems.

1.2 Previous and Related Work

FAP generalizes several well-known network design problems, many of which have seen improvements beyond the 2-approximation barrier, see Figure 1. One important special case is the Tree Augmentation Problem (TAP), where F is a tree. Over the past decades, a sequence of results [1, 9, 10, 11, 14, 17, 19, 20, 26, 32, 36, 37, 38, 39] has led to a 1.393-approximation by Cecchetto, Traub, and Zenklusen [6]. The weighted version of TAP has also been extensively studied, culminating in a $1.5 + \varepsilon$ approximation by Traub and Zenklusen [42], which even holds for the more general connectivity augmentation problem. In the connectivity augmentation problem we are given a k-edge-connected graph G and additional links L and the task is to compute a set of links $X \subseteq L$ of minimum size such that $G \cup X$ is $(k+1)$-edge-connected. Before, the barrier of 2 for connectivity augmentation was breached by Byrka, Grandoni, and Jabal Ameli [5].

Another relevant special case of FAP and PAP is the unweighted 2-edge-connected spanning subgraph problem (unweighted 2ECSS), where $F = \emptyset$. Several works [12, 22, 30, 33, 35, 41] have led to better-than-2 approximations. Bosch-Calvo, Garg, Grandoni, Hommelsheim, Jabal Ameli, and Lindermayr [3] obtained a $\frac{5}{4}$-approximation, which was recently improved slightly by Hommelsheim, Lindermayr, and Liu [29].

The Matching Augmentation Problem (MAP) is the special case of PAP in which F is a matching, i.e., each path consists of a single edge. Cheriyan, Dippel, Grandoni, Khan, and Narayan introduced this problem and gave a $\frac{7}{4}$-approximation [7]. Cheriyan, Cummings, Dippel, and Zhu [8] improved this to $\frac{5}{3}$ and Garg, Hommelsheim, and Megow improved this further to $\frac{13}{8}$ [23]. An LP-based approximation algorithm for MAP with worse guarantee, yet simpler analysis has been developed by Bamas, Drygala, and Svensson [2].

Beyond edge connectivity, many of these problems have natural vertex connectivity counterparts, where the objective is to compute a 2-vertex connected spanning subgraph, ensuring connectivity despite a single vertex failure. For the vertex-connected version of unweighted 2ECSS, several algorithms obtain better-than-2 approximations [13, 24, 27, 33]. The current best-known result is a $\frac{4}{3}$-approximation by Bosch-Calvo, Grandoni, and Jabal Ameli [4].

1.3 Notation

We assume that all instances of optimization problems studied in this paper admit a feasible solution, which can be tested in polynomial time. Throughout this paper, we consider an instance of PAP given by $G = (V, E(\mathcal{P}) \cup L)$, where $\mathcal{P}$ is a collection of disjoint paths, $E(\mathcal{P})$ denotes their edge set, and $L \subseteq \binom{V}{2}$ is a set of additional links. We define $E = E(\mathcal{P}) \cup L$ as the set of edges of G and set $p_{\max} := \max_{P \in \mathcal{P}} |E(P)|$ to be the maximum length of a path in $\mathcal{P}$. For $\mathcal{P}' \subseteq \mathcal{P}$ we write $V^{\text{end}}(\mathcal{P}')$ to denote the set of endpoints (endvertices) of all paths in $\mathcal{P}'$, i.e., vertices that have degree at most 1 w.r.t. $E(\mathcal{P}')$. We define $V^{\text{end}} := V^{\text{end}}(\mathcal{P})$. Every vertex of $V(G)$ that is not an endvertex has degree 2 w.r.t. $E(\mathcal{P})$; such vertices are called interior vertices (of some path $P \in \mathcal{P}$). We further define the following subsets of links: $L^{\text{end}} \subseteq L$ is the set of links uv such that $u, v \in V^{\text{end}}$. We use $L^{\text{end,in}} \subseteq L^{\text{end}}$ to denote the set of links of L^{end} whose endpoints belong to the same path, while $L^{\text{end,out}} = L^{\text{end}} \setminus L^{\text{end,in}}$ contains links of L^{end} that connect different paths. For a subgraph H of G we write $L(H)$ to denote the set of links contained in H. For $P \in \mathcal{P}$ and $u, v \in V(P)$, let P^{uv} be the path from u to v on P. We denote by $\text{OPT}(G)$ some optimal solution of G and by $\text{opt}(G)$ the cost of an optimal solution, i.e., the number of links in $\text{OPT}(G)$. If G is clear from context, we simply write OPT and opt, respectively.

Further, we mainly use standard graph notation. Given a set of vertices $U \subseteq V$, we let $G|U$ denote the multi-graph obtained from G by contracting U to a single vertex. For a subgraph H of G we simply write $G|H$ instead of $G|V(H)$. More generally, given a set $\mathcal{T} = \{T_1, \dots, T_k\}$ of vertex-disjoint sets $T_1, \dots, T_k \subseteq V(G)$, $G|\{T_1, \dots, T_k\}$ (or $G|\mathcal{T}$ in short) denotes the graph obtained by contracting each T_i into a single vertex. A set $X \subseteq V$ is a separator if $G \setminus X$ is disconnected. For $V' \subseteq V(G)$ we define $N_G(V')$ as the neighborhood of V' in G and $\delta(V')$ refers to the edge set with exactly one endpoint in V'. A bridge in G is an edge whose removal from G increases the number of connected components. A subgraph H is a 2-edge cover of G if $|\delta_H(v)| \geq 2$ for each $v \in V(G)$.

2 Overview of Our Approach

In this section, we provide an overview of our algorithmic approach to prove Theorem 1. At a high level, we follow the approach recently introduced for FAP and PAP [25], unweighted 2ECSS [3, 22, 29] and MAP [7, 8, 23]. Our approach has four main steps.

1. Reduction to Structured Instances. The first step is a reduction to structured instances. Specifically, we show that for any $\alpha \geq \frac{11}{6}$ and any small constant $\varepsilon > 0$, if an α-approximation exists for structured instances of bounded-length PAP, then an $(\alpha + \varepsilon)$-approximation exists for arbitrary bounded-length instances of PAP. These structured instances possess key properties that enable and simplify our subsequent algorithmic steps. This step has not been utilized in the previous work on FAP and PAP, except for a reduction to PAP without isolated vertices in [25]. We remark that this is the only place where we require the length of the paths in the instance of PAP to be bounded.

2. Computing a Low-Cost Starting Solution. The second step includes one of our key innovations. Here, we compute a *low-cost* starting solution $S \subseteq L$ such that the graph $H = (V, E(\mathcal{P}) \cup S)$ is not necessarily feasible but has important structural properties. In later steps, we will turn this starting solution into a feasible solution. To guide this process, we introduce a credit scheme that assigns to any spanning subgraph H' of G credits to vertices, edges, blocks, and components of H'. The total sum of credits of H' is denoted

by credits(H'), and we define cost(H') = $|H' \cap L|$ + credits(H'), i.e., the number of links in H' plus the total sum of credits assigned to H'. This scheme helps to maintain a cost measure that accounts not only for the number of selected links but also for structural penalties. Roughly speaking, the number of credits assigned to a solution reflects its distance to a feasible solution. To compute the starting solution, we introduce a novel relaxation of PAP, which we use as a lower bound on opt and which is related to 2-edge covers. Instead of computing a solution to the relaxation directly, we consider a closely related packing problem, where the relationship between the packing problem and the relaxation of PAP is similar to the relationship between 2-matchings and 2-edge covers[4]. Although it turns out that the packing problem and the relaxation are still NP-hard (see full version [28]), we can compute a solution S to the packing problem such that $H_0 = (V, E(\mathcal{P}) \cup S)$ satisfies cost(H_0) $\leq 1.9412 \cdot$ opt. In fact, we compute up to $|\mathcal{P}| + 2$ many solutions H and output the one minimizing cost(H). We analyze the cost of this solution using a factor-revealing LP. This (most likely infeasible) solution is used as a starting solution.

In the remainder, we step-by-step turn H_0 into solutions $H_1, H_2, \ldots$ that are "closer to being feasible", while always keeping the invariant that the new solution H_{i+1} satisfies cost(H_{i+1}) $\leq$ cost(H_i). Since the credits are non-negative and the starting solution satisfies cost(H_0) $\leq 1.9412 \cdot$ opt, eventually we will have a feasible solution using at most $1.9412 \cdot$ opt many links.

There are two reasons why a solution H is not yet feasible: 1) there is a bridge in H, and 2) there are several connected components in H. The next two steps deal with these reasons for infeasibility.

3. Bridge-Covering. The third step, called bridge-covering, ensures that the intermediate solution H' is bridgeless. Our goal is to add (and sometimes remove) links to (from) H such that the resulting graph H' is bridgeless, while keeping the invariant cost(H') $\leq$ cost(H). At this stage, we heavily exploit the fact that G is structured, which allows us to prove the existence of cheap augmenting link sets needed to decrease the number of bridges.

4. Gluing. In the fourth and final step, called gluing, we transform H into a feasible solution, i.e., a 2-edge-connected spanning subgraph. At this point, H consists of multiple 2-edge-connected components that must be merged into a single 2-edge-connected component. We achieve this by iteratively adding well-chosen cycles in the *component graph*, ensuring that each step maintains feasibility and preserves the approximation guarantee. The component graph $G^{\text{comp}}(H)$ arises from G by contracting each component of H to a single vertex. To prove the existence of such cycles we again heavily exploit that G is structured.

The remainder of this section provides a more detailed overview of these four steps, along with lemmas formalizing each transformation. The subsequent sections are dedicated to the individual steps, leading to the proof of Theorem 1, which follows from the four main lemmas established before.

3 Reduction to Structured Instances

Unlike previous algorithms for FAP and PAP [25], we introduce a reduction to structured instances, which plays a crucial role in our approach. Informally, a structured instance is one that avoids certain degenerate configurations that would complicate the design of good

[4] For a graph $G = (V, E)$ a 2-matching $M \subseteq E$ satisfies $\delta_M(v) \leq 2$ for all $v \in V$. A classical result states that one can compute a minimum 2-edge cover by computing a maximum 2-matching and greedily adding edges to each vertex $v \in V$ with $\delta_M(v) < 2$; see [40] for more details.

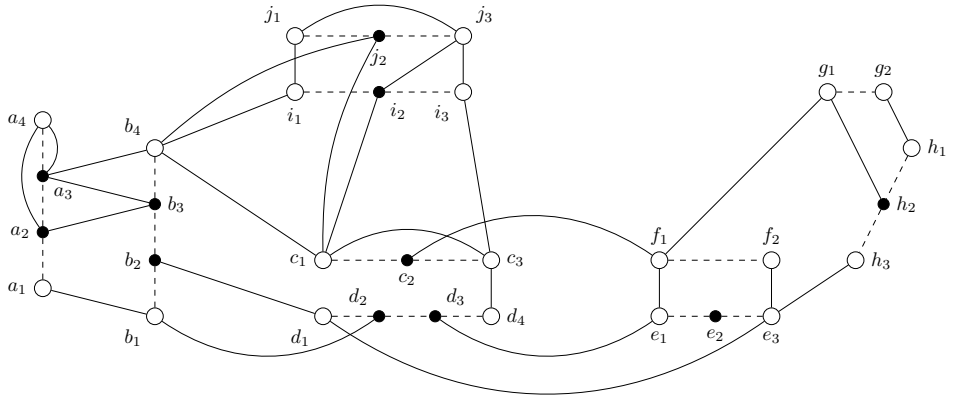

Figure 2 Non-structured PAP instance. Dashed edges are in $E(\mathcal{P})$, solid edges in L. Interior vertices of paths from $\mathcal{P}$ are filled, endvertices of paths from $\mathcal{P}$ are empty. The cycle $a_2a_3a_4a_2$ is a 1-contractible subgraph, the path $P = b_1b_2b_3b_4 \in \mathcal{P}$ is a path-separator, the path $c_1c_2c_3d_4d_3d_2d_1$ is a P^2-separator, the cycle $e_1e_2e_3f_2f_1e_1$ is a C^2-separator, and the path $j_1j_2j_3$ is a degenerate path.

approximation algorithms in our approach. We formally define structured instances and establish that solving bounded-length PAP on structured instances suffices to achieve a good approximation algorithm for general bounded-length PAP. We prove that for all $\alpha \geq \frac{11}{6}$ and any small constant $\varepsilon > 0$, if there exists a polynomial-time α-approximation for bounded-length PAP on structured instances, then there is a polynomial-time $(\alpha+\varepsilon)$-approximation for arbitrary instances of bounded-length PAP. This guarantees that improvements on structured instances directly translate into improvements for the general case.

To define structured instances, we introduce several graph properties, see Figure 2. First, structured instances do not contain contractible subgraphs. Roughly speaking, a subgraph is (strongly) α-contractible if at least a $\frac{1}{\alpha}$ fraction of its links must appear in any feasible solution, allowing us to safely contract it without a loss in the approximation guarantee. We ensure that such subgraphs can be identified and handled efficiently. Formally, we define α-contractible subgraphs as follows.

▶ **Definition 2** (α-contractible subgraph). *Let $\alpha \geq 1$ and $t \geq 1$ be fixed constants. Given a 2-edge-connected graph G, a collection $\mathcal{H}$ of vertex-disjoint 2-edge-connected subgraphs $H_1, H_2, \ldots, H_k$ of G, such that $G|\mathcal{H}$ is also an instance of* PAP*, is called* weakly (α, t, k)-contractible *if $1 \leq |L(H_i)| \leq t$ for every $i \in [k]$ and there exists some optimal solution that contains at least $\frac{1}{\alpha}|\bigcup_{i\in[k]} L(H_i)|$ links from $\bigcup_{i\in[k]} L(G[V(H_i)])$. Furthermore, $\mathcal{H}$ is* strongly (α, t, k)*-contractible if additionally every 2EC spanning subgraph of G contains at least $\frac{1}{\alpha}|\bigcup_{i\in[k]} L(H_i)|$ links from $\bigcup_{i\in[k]} L(G[V(H_i)])$.*

Every strongly (α, t, k)-contractible subgraph is also a weakly (α, t, k)-contractible subgraph. We say that $\mathcal{H}$ is an (α, t, k)-contractible subgraph if it is a weakly or strongly (α, t, k)-contractible subgraph. If α, t, k are clear from the context, we simply refer to α-contractible subgraphs or even contractible subgraphs. A typical example of a strongly contractible subgraph occurs if an endvertex v of some path $P \in \mathcal{P}$ is only adjacent to vertices on P: If u is the neighbor of v on P furthest away from v, then $(V(P^{uv}), P^{uv} \cup \{uv\})$ is strongly $(1, 1, 1)$-contractible, since $(V(P^{uv}), P^{uv} \cup \{uv\})$ is 2-edge-connected and every feasible solution must contain a link connecting v and $V(P^{uv})$. This is summarized in Property (P2) of Definition 4; see also Figure 2.

Second, structured instances avoid certain small separators. Specifically, we prevent the existence of path-separators, P^2-separators, and C^2-separators, which are roughly defined as follows (leaving out some technicalities, see full version [28]). A path-separator is a path $P \in \mathcal{P}$ that is a separator, i.e., $G \setminus V(P)$ is not connected. A P^2-separator is a separator that consists of two paths $P, P' \in \mathcal{P}$ such that there is a link in $L^{\text{end,out}}$ connecting one endvertex of P and one of P'. Note that $E(P) \cup E(P') \cup \{e\}$ is a simple path on $V(P) \cup V(P')$. Similarly, a C^2-separator is a separator that consists of two paths $P, P' \in \mathcal{P}$ such that there are two links in $e, e' \in L^{\text{end,out}}$ such that $E(P) \cup E(P') \cup \{e, e'\}$ is a cycle. Note that a C^2-separator is also a P^2-separator, but the additional edge required for C^2-separators allows us to obtain stronger results due to technical reasons omitted here. If an instance contains such a separator, we decompose it into smaller subinstances, solve each recursively, and merge their solutions while maintaining the desired approximation guarantee. Formal definitions are given in the full version [28]. We remark that we only require the length of the paths in the instance of PAP to be bounded for the reduction to avoid path-separators and P^2-separators.

A final obstacle to obtain structured instances is the presence of too many degenerate paths, which are paths $P \in \mathcal{P}$ with limited connectivity to the rest of the graph. When the fraction of degenerate paths exceeds the threshold $\varepsilon|\mathcal{P}|$, we identify certain well-connected subgraphs that we can contract and then solve the resulting subinstances recursively.

▶ **Definition 3** (Degenerate Path). *Let $P, P' \in \mathcal{P}$ be two distinct paths with endpoints u', v' and u, v, respectively, such that $u'u \in L$, $v'v \in L$ and $u'v' \in L$. We say that P' is a degenerate path if $N(u'), N(v') \subseteq V(P') \cup V(P)$.*

Using these definitions, we define structured instances.

▶ **Definition 4** ((α, ε)-structured Instance). *We call an instance $G = (V, \mathcal{P} \cup L)$ of bounded-length PAP (α, ε)-structured if the following properties are satisfied:*

(P0) $\text{opt}(G) > \frac{3}{\varepsilon}$.
(P1) *G does not contain strongly $(\alpha, t, 1)$-contractible subgraphs with $t \leq 10$ many links.*
(P2) *Every endpoint of a path $P \in \mathcal{P}$ has a link to some other path $P' \in \mathcal{P}$, $P' \neq P$.*
(P3) *G does not contain path-separators.*
(P4) *G does not contain P^2-separators.*
(P5) *G does not contain C^2-separators.*
(P6) *G contains at most $\varepsilon \cdot |\mathcal{P}|$ many degenerate paths.*
(P7) *Every vertex $v \in V$ is on some path $P \in \mathcal{P}$ that contains at least 2 vertices.*

If α and ε are clear from the context, we simply say G is structured. In the subsequent subsections we outline how these properties are useful to obtain good approximation algorithms. Our main lemma in this section is the following.

▶ **Lemma 5.** *For all $\alpha \geq \frac{11}{6}$ and constant $\varepsilon \in \left(0, \frac{1}{12}\right]$, if there exists a polynomial-time α-approximation algorithm for bounded-length* PAP *on (α, ε)-structured instances, then there exists a polynomial-time $(\alpha + \varepsilon)$-approximation algorithm for bounded-length* PAP *on arbitrary instances.*

We give a rough overview on how we obtain Lemma 5. Assume that G is not structured. In this overview, we focus only on three obstacles that prevent G from being structured: 1) G contains a contractible subgraph, 2) G contains a forbidden separator, or 3) G contains more than $\varepsilon|\mathcal{P}|$ many degenerate paths. If G contains one of these subgraphs, we can decompose G into at most 2 smaller subinstances, solve each recursively, and merge their solutions while maintaining the approximation guarantee. We outline our approach for each of these three cases.

Property (P1) is not met. We can safely contract each component of an α-contractible subgraph $\mathcal{H}$ into a single vertex and work with $G|\mathcal{H}$, as long as we only aim for an α-approximation. Indeed, any α-approximate solution for $G|\mathcal{H}$ can be turned into an α-approximate solution for G by adding the links of $\mathcal{H}$. In general, it is not straightforward to argue that a 2EC subgraph is weakly α-contractible. However, under certain conditions we can argue that this is indeed the case. We will show that we can find strongly $(\alpha, t, 1)$-contractible subgraphs in polynomial time if t is constant, i.e., if there is only one component and there are only a constant number of links in the contractible subgraph. Hence, after our preprocessing, we can assume that G does not contain any strongly $(\alpha, t, 1)$-contractible subgraphs for some constant t (here $t = 10$ suffices).

Property (P3), (P4), or (P5) is not met. Assume that G contains a forbidden separator X, resulting in $G \setminus X$ to decompose into two disjoint vertex sets V_1 and V_2 and there are no edges between V_1 and V_2. Here, we construct two sub-instances $G_1 = (G \setminus V_2)|X$ and $G_2 = (G \setminus V_1)|X$ arising from G by removing either V_1 or V_2 and contracting X into a single vertex. We recursively solve these sub-instances approximately and combine both solutions while preserving the desired approximation guarantee. By introducing a proper notion of *size* of an instance, one can ensure that the sum of the sizes of the two sub-instances is smaller than the size of the original instance and hence this step runs in polynomial time.

Property (P6) is not met. One of the main innovations in our reduction to structured graphs compared to previous work deals with degenerate paths. For degenerate paths, we can not do the same strategy as for separators and instead do the following: Let $\mathcal{P}' = \{P'_1, \dots, P'_\ell\}$ be a set of degenerate paths. We define two collections $\mathcal{K}$ and $\mathcal{C}$ of ℓ vertex-disjoint 2-edge-connected subgraphs, each subsuming $V(\mathcal{P}')$. We show that either $\mathcal{C}$ is weakly $(\alpha, 2, \ell)$-contractible or $\mathcal{K}$ is weakly $(\alpha, 1, \ell)$-contractible for any $\alpha \geq \frac{11}{6}$. Hence, either an α-approximate solution for $G|\mathcal{K}$ or an α-approximate solution for $G|\mathcal{C}$ can be turned into an α-approximate solution for G by adding the edges contained in $\mathcal{K}$ or $\mathcal{C}$, respectively.

Unfortunately, in general this recursive procedure leads to an exponential running time if $|\mathcal{P}'|$ is small: We need to recursively solve the two sub-instances $G|\mathcal{K}$ and $G|\mathcal{C}$ that both may have roughly the same size as the original instance. Since we might need to apply this reduction again in later steps in our reduction to structured graphs, this procedure can result in an exponential running time. However, we circumvent this by only applying the above procedure if some constant fraction ε of the paths in $\mathcal{P}$ is degenerate, and hence each sub-instance is significantly smaller than the original one, which overall results in a polynomial running time.

In light of Lemma 5, it remains to give a 1.9412-approximation for bounded-length PAP on $(\frac{11}{6}, \varepsilon)$-structured instances. Hence, for the remaining steps, we assume the instance $G = (V, E(\mathcal{P}) \cup L)$ is a $(\frac{11}{6}, \varepsilon)$-structured instance of bounded-length PAP. In fact, for the remainder of this paper we only require the instance to be a $(\frac{11}{6}, \varepsilon)$-structured instance of PAP; the requirement of bounded-length was only needed for the reduction to structured graphs.

4 Credit Scheme, Relaxation, and Starting Solution

Throughout our algorithm, we maintain a temporary solution $S \subseteq L$, which is not necessarily a feasible solution. To track progress and guide modifications, we assign certain credits to parts of the current graph $H = (V, E(\mathcal{P}) \cup S)$. These credits are used to ensure that

every transformation preserves an approximation bound while gradually converting H into a feasible solution. We first define key structural components and introduce a cost function that incorporates both the number of selected links and assigned credits.

We use the following notation of blocks from related works on unweighted 2ECSS [22, 3, 29] and MAP [7, 8, 23]. A block is a maximal 2-edge-connected subgraph containing at least 2 vertices. A block B is called *simple* if there is some $P \in \mathcal{P}$ such that $V(B) \subseteq V(P)$, it is called *small* if there are two distinct paths $P_1, P_2 \in \mathcal{P}$ such that $V(B) = V(P_1) \cup V(P_2)$, and *large* if there are three distinct paths $P_1, P_2, P_3 \in \mathcal{P}$ such that $V(P_1) \cup V(P_2) \cup V(P_3) \subseteq V(B)$. A connected component that is 2EC is called a 2EC *component*. We call 2EC components small if they contain exactly 2 links and large if they contain at least 3 links.

Let H be a spanning subgraph of G, and let C be a connected component of H. A component is called *complex* if it contains bridges. For any such C, we define the graphs G_C and H_C as follows: G_C is the graph obtained from G by contracting every connected component of H except for C, as well as every block of C. H_C is the corresponding contraction of H. The structure of H_C consists of a tree T_C along with singleton vertices. A vertex in G_C or H_C that corresponds to a block in H is called a *block vertex*. If a block vertex corresponds to a simple block, it is called a *simple block vertex*. We identify the edges/links in G and the corresponding edges/links in G_C and H_C, resp.

A component C is called *trivial* if it consists of a single path $P \in \mathcal{P}$ (i.e., $V(C) = V(P)$). Otherwise, it is called *non-trivial*. A vertex $v \in V$ is called *lonely* if it does not belong to any 2-edge-connected component or block of H. For two paths $P, P' \in P$ with endpoints u, v and u', v', resp., we say that $uu' \in L$ is an *expensive link* if $N_G(v) \subseteq (V(P) \cup V(P')) \setminus \{v'\}$. If additionally $uu' \in L(H)$ is a bridge in H and v is a lonely leaf in H, then v is called an *expensive leaf vertex*.

For a given $H = (V, E(\mathcal{P}) \cup S)$, we assign credits as follows.

- **(A)** Every lonely vertex v that is a leaf of some complex component C receives 1 credit.
  - **(i)** It receives an additional $\frac{1}{4}$ credit if v is an expensive leaf vertex.
  - **(ii)** It receives an additional $\frac{1}{4}$ credit if there is a path in H_C from v to a simple block vertex of C without passing through a non-simple block vertex.
- **(B)** Every bridge $\ell \in S$ (i.e., a link that is not part of any 2-edge-connected component of H) receives $\frac{3}{4}$ credits.
- **(C)** Every complex component receives 1 credit.
- **(D)** Every non-simple 2-edge-connected block receives 1 credit.
- **(E)** Every 2-edge-connected component receives:
  - 2 credits if it is large or contains a degenerate path.
  - $\frac{3}{2}$ credits if it is small (i.e., has exactly 2 links) and does not contain a degenerate path.

For any spanning subgraph $H = (V, E(\mathcal{P}) \cup S)$, we define the total credit as credits(H), which is the sum of all assigned credits under these rules. We define cost$(H) :=$ credits$(H) + |S|$.

As an example, consider the right solution in Figure 3. There are three connected components: (1) The 2EC component containing v_1, which receives a credit of 2 (Rule (E)). (2) The complex component containing v_7. This component receives a credit of 1 (Rule (C)), the bridge u_7u_8 receives a credit of $\frac{3}{4}$ (Rule (B)), and the lonely leaf vertices u_7, u_8 receive a credit of 1 each (Rule (A)). (3) The complex component containing v_6. This component receives a credit of 1 (Rule (C)), the bridges $u_6w_3, w_3w_4, w_4u_5, v_5u_9, v_9w_{11}, w_{11}v_{10}$ receive a credit of $\frac{3}{4}$ (Rule (B)) each, and the lonely leaf vertices v_6, u_{10} receive a credit of 1 each (Rule (A)) (potentially plus $\frac{1}{4}$ if Rule (A) (i) applies).

To prove the approximation guarantee, we show that we maintain the following invariants.

▶ **Invariant 6.** $\text{cost}(H) \leq 1.9412 \cdot \text{opt}(G)$.

▶ **Invariant 7.** *Every lonely vertex of H has degree at most 2 in H. Every simple 2EC block B is part of some complex component C and has degree exactly 2 in H_C. Furthermore, for any complex component C, no two simple block vertices of C are adjacent to each other in H_C.*

▶ **Invariant 8.** *For every non-simple 2EC block B of H there are two distinct paths $P, P' \in \mathcal{P}$ such that $V(P) \cup V(P') \subseteq V(B)$.*

Our goal is to design a polynomial-time algorithm that computes a solution $S \subseteq L$ such that $H = (V, E(\mathcal{P}) \cup S)$ satisfies all invariants above. This is formalized in the following lemma.

▶ **Lemma 9.** *Given a structured graph, in polynomial time we can compute a set of links $S \subseteq L$ such that $H = (V, E(\mathcal{P}) \cup S)$ satisfies the Invariants 6-8.*

Next, we outline how to compute such a solution. The approaches in [3, 22, 7, 8, 23] for special cases of PAP, such as unweighted 2ECSS and MAP, use a relaxation of the problem, for which an optimum solution can be computed in polynomial time and a solution serves as a good starting solution. In particular, these prior works used a minimum 2-edge cover as a relaxation. For *structured instances* of these special cases of PAP, optimum 2-edge covers are usually not too far away from an optimum solution of the original problem (i.e., unweighted 2ECSS or MAP), and hence they serve as a good lower bound. However, for PAP, a 2-edge cover can be far away from an optimum solution. For example, if the 2-edge cover of an instance of PAP only contains edges of $E(\mathcal{P})$ and $L^{\text{end,in}}$, then this edge set only 2-edge-connects the paths individually, but it does not help to establish overall 2-edge-connectivity. This motivates us to define a more suitable relaxation for PAP, called 2-Edge Cover with Path Constraints (2ECPC).

4.1 A relaxation of PAP: The 2-Edge Cover with Path Constraints

The input consists of a structured instance of PAP where each path $P_i \in \mathcal{P}$ has exactly three vertices: two endpoints $u_i, v_i \in V^{\text{end}}(P_i)$ and an interior vertex w_i. The goal is to compute a minimum-cardinality link set $X \subseteq L$ such that $E(\mathcal{P}) \cup X$ is a 2-edge cover with the additional constraint that each path P_i is incident to at least two outgoing links in X. More formally, X has to satisfy the following:

- Every vertex $u \in V^{\text{end}}(P_i)$ is incident to at least one link in X, i.e., $|\delta_X(u)| \geq 1$.
- Each path $P_i \in P$ has at least two outgoing edges in X, i.e., $|\delta_X(P_i)| \geq 2$.

One can easily obtain an instance of 2ECPC from an instance of PAP by doing the following for each path: If the path contains at least 4 vertices, we contract all internal vertices to a single vertex, and if the path contains 2 vertices, we add a dummy vertex as an interior vertex. Recall that for structured instances, every vertex is part of some path $P \in \mathcal{P}$ containing at least 2 vertices (Property (P7)). There is a one-to-one correspondence between links in the graph of the instance of PAP and links in the graph of the instance of 2ECPC. One can verify that any feasible solution to the PAP instance is also feasible for its corresponding 2ECPC instance; hence 2ECPC is indeed a relaxation for PAP. Instead of computing an (approximate) solution to 2ECPC, we consider a packing problem, which we call the Track Packing Problem (TPP).

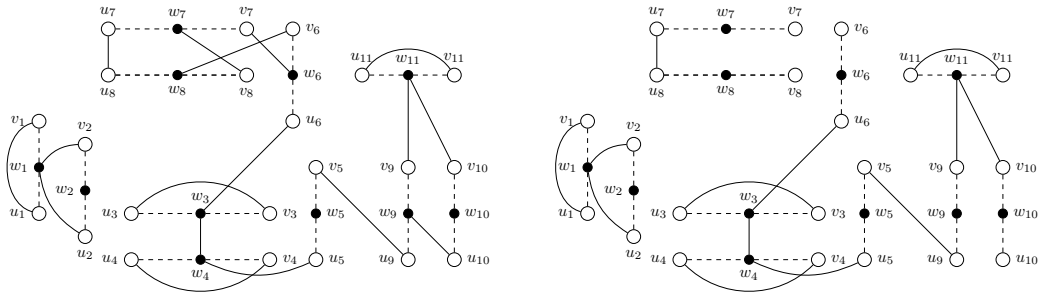

Figure 3 Dashed edges are in $E(\mathcal{P})$, solid edges are links of L. Filled vertices are interior vertices of paths from $\mathcal{P}$, empty vertices are endvertices of paths in $\mathcal{P}$. Left: Feasible solution Y for 2ECPC, since $E(\mathcal{P}) \cup Y$ is a 2-edge cover with $\delta_Y(P) \geq 2$ for each $P \in \mathcal{P}$. Right: Solution $X = \{T_1, T_2, T_3, T_4, T_5\}$ for the corresponding TPP instance with two tracks containing 1 link ($T_1 = \{u_7u_8\}$ and $T_2 = \{v_5u_9\}$), two tracks containing 3 links ($T_3 = \{u_2w_1, u_1v_1, w_1v_2\}$ and $T_4 = \{v_9w_{11}, u_{11}v_{11}, w_{11}v_{10}\}$) and one track containing 5 links ($T_5 = \{u_5w_4, u_4v_4, w_4w_3, u_3v_3, w_3u_6\}$). Note that $|Y| = 17 = 2|\mathcal{P}| - 5 = 2|\mathcal{P}| - |X|$.

We denote by OPT and OPT_C the optimum solutions for the PAP and the 2ECPC instance, respectively, and define opt and opt_C to be their objective values. It is easy to verify that any feasible solution to the PAP instance is also feasible for its corresponding 2ECPC instance, and hence 2ECPC is indeed a relaxation for PAP. We summarize this as follows.

▶ **Proposition 10.** *Let G be a structured instance of* PAP. *Then* $\text{opt} \geq \text{opt}_C$.

It is easy to see that $|\mathcal{P}| \leq \text{opt}_C \leq 2|\mathcal{P}|$: Observe that since the input graph G is structured, each endpoint u of some path $P \in \mathcal{P}$ is incident to a link $e \in \delta(P)$. Therefore, by picking such an edge for each endpoint u, there is a straightforward solution containing at most $2|\mathcal{P}|$ many links. Furthermore, note that any feasible solution must pick at least $|\mathcal{P}|$ many links. Therefore, we obtain the inequalities above.

Similar to other connectivity augmentation problems, we assume that the instance of 2ECPC is *shadow-complete*. In our setting this means the following: Let $P_i, P_j \in P$ such that $P_i \neq P_j$. If there is a link u_iu_j from an endpoint u_i of P_i to some other endpoint u_j of P_j, then we also add the *weaker* links u_iw_j, w_iu_j and w_iw_j, where w_i, w_j are the interior vertices of the paths P_i and P_j, respectively. Furthermore, if there is a link u_iw_j from an endpoint u_i of P_i to the interior vertex w_j of P_j, then we also add the *weaker* link w_iw_j, where w_i is the interior vertex of P_i. These weaker links are then also called the *shadows* of their original links. An instance that contains all such shadows is called shadow-complete.

It is not hard to see that for any feasible instance of 2ECPC, a feasible solution to the shadow-complete instance can be turned into a feasible solution to the original instance of the same cost.

▶ **Lemma 11.** *For any feasible instance of* 2ECPC *arising from a structured graph, a feasible solution Y to the shadow-complete instance can be turned into a feasible solution Y' to the original instance such that $|Y'| \leq |Y|$.*

Hence, from now on we assume that our instance of 2ECPC is shadow-complete.

4.2 The Track Packing Problem

TPP is inspired by the relationship between 2-matchings and 2-edge covers, where the covering constraints are expressed as a packing problem. Formally, we construct an instance of TPP from a shadow-complete instance of 2ECPC (see full version [28] for formal definitions). For some link set $A \subseteq L$ let $V^{\text{end}}(A) \coloneqq V^{\text{end}}(\mathcal{P}) \cap V(A)$ be the set of vertices that are endpoints of some path and are incident to some link in A. A *track* $T \subseteq L$ with $T = Q(T) \cup I(T)$ is a subset of links such that the following conditions hold.

- $Q(T)$ is a simple path in G of length at least one, where the start vertex u and the endvertex v of $Q(T)$ are endpoints of some paths $P, P' \in \mathcal{P}$ (not necessarily $P \neq P'$) such that $u \neq v$,
- any interior vertex of $Q(T)$, i.e., a vertex from $V(Q(T)) \setminus \{u, v\}$, is the interior vertex of some path $P \in \mathcal{P}$,
- $I(T) \subseteq L$ contains exactly the following set of links: for each interior vertex $w_i \in V(Q(T)) \setminus \{u, v\}$, which is an interior vertex of some path $P_i \in \mathcal{P}$, there is a link $u_i v_i \in I(T)$, where $u_i, v_i \in V^{\text{end}}(P_i)$ are the endvertices of P_i, and
- if $|T| = 1$, i.e., if $|Q(T)| = 1$ and $|I(T)| = 0$, then the start vertex u of $Q(T)$ and the endvertex v of $Q(T)$ belong to distinct paths.

For any track T, we refer to $Q(T)$ and $I(T)$ as above. Observe that the number of links in a track is odd, since $|Q(T)| = |I(T)| + 1$. Note that a track of length 1 is simply a link in $L^{\text{end,out}}$, i.e., a link between two endvertices of two distinct paths.

The instance of TPP is now defined as follows. The input is a (shadow-complete) instance of 2ECPC of some structured graph G. Let $\mathcal{T}$ be the set of all tracks in G. The task is to select a maximum number of disjoint tracks from $\mathcal{T}$, where two tracks T, T' are disjoint if $V(T) \cap V(T') = \emptyset$.

In the following we establish a relationship between solutions of 2ECPC and its corresponding TPP instance. This connection is similar to the connection between 2-matchings and 2-edge covers. In particular, an optimum solution to one of the problems can be turned into an optimum solution to the other problem in polynomial time. We note that the additional condition required in Lemma 12 and Lemma 13 is always satisfied for structured graphs. The relationship between 2ECPC and TPP is highlighted in Figure 3.

▶ **Lemma 12.** *Given a solution X for a shadow-complete instance of* TPP*, in which each endvertex of a path is connected to a vertex of a different path, we can compute in polynomial time a solution Y to the corresponding* 2ECPC *instance of value $2|\mathcal{P}| - |X|$.*

▶ **Lemma 13.** *Consider a shadow-complete instance I of* 2ECPC*, in which each endvertex of a path is connected to a vertex of a different path. Given an inclusion-wise minimal solution Y to I such that $|Y| \leq 2|\mathcal{P}|$, we can compute in polynomial time a solution X to the corresponding* TPP *instance of value at least $2|\mathcal{P}| - |Y|$.*

We also note that TPP is NP-hard by a reduction from the NP-hard P_2-Packing problem [34]. In the P_2-Packing problem we are given a graph G and the task is to select a maximum number of vertex-disjoint P_2's, i.e., paths that contain exactly 2 edges.

▶ **Proposition 14.** *The Track Packing Problem is* NP*-hard, even on shadow-complete instances, in which each endvertex of a path is connected to a vertex of a different path.*

Note that due to the above relationship between 2ECPC and TPP established in Lemma 12 and Lemma 13, Proposition 14 also implies NP-hardness for 2ECPC.

▶ **Proposition 15.** *The 2-Edge Cover with Path Constraints is* NP*-hard.*

Despite this hardness, in the remaining part we show how to compute a solution to TPP which serves as a good starting solution for our algorithmic framework.

4.3 Computing a good starting solution

Our goal is to compute a solution to TPP. By its close relationship to 2ECPC, we can bound the cost of this solution in terms of opt_C, the optimum solution value for the instance of 2ECPC, and since 2ECPC is a relaxation of PAP, we can in turn bound its cost in terms of opt. We do not compute an approximate solution to TPP relative to its optimum solution value; instead we do the following: For a solution X to TPP we compute $H_X = (V, E(\mathcal{P}) \cup (X \cap L))$, and we want to find a solution X to TPP such that $\text{cost}(H_X) = |X \cap L| + \text{credits}(H_X) \leq 1.9412 \cdot \text{opt}$.

To find such a solution, we view TPP as a Set-Packing problem. Therein, we are given a set of elements U and a set of subsets $\mathcal{S} \subseteq 2^U$ of U and the task is to compute a set $X \subseteq \mathcal{S}$ of maximum cardinality such that for any two sets $S, S' \in X$ we have $S \cap S' = \emptyset$. Set-Packing is APX-hard and the state-of-the-art polynomial-time approximation algorithms for it achieve an approximation factor of $\frac{3}{1+k+\varepsilon}$ for any constant $\varepsilon > 0$, where $k \coloneqq \max_{S \in \mathcal{S}} |S|$ is the maximum size of a set in $\mathcal{S}$ [15, 21]. TPP can be reduced to Set-Packing by viewing each $v \in V^{\text{end}}(\mathcal{P})$ as an element and each track as a set, which exactly contains those vertices $v \in V^{\text{end}}(\mathcal{P})$ to which the track is incident to.

Using such set-packing algorithms, we define three different algorithms that compute sets of disjoint tracks, where the maximum size of a track we consider contains at most 4 vertices of $V^{\text{end}}(\mathcal{P})$. In total, we compute up to $|\mathcal{P}| + 2$ different solutions and output the solution X minimizing $\text{cost}(H_X)$. It can be easily shown that H_X satisfies Invariant 7 and Invariant 8. To show that H_X satisfies Invariant 6, that is, $\text{cost}(H_X) \leq 1.9412 \cdot \text{opt}$, we bound the cost of H_X in terms of variables that depend on the three algorithms and certain parameters of an optimum solution. Here, we heavily exploit the connection between optimum solutions of TPP and 2ECPC from above. We then derive a factor-revealing LP, which proves $\text{cost}(H_X) \leq \rho \cdot \text{opt}$ if the polyhedron is empty for a specific value of ρ. Indeed, for $\rho = 1.9412$ this is the case, implying Invariant 6. This then proves Lemma 9.

Throughout this section we fix an optimum solution OPT_P for the instance of TPP and assume that $\text{OPT}_P = \bigcup_{i=1}^{n} O_i$, where O_i is the set of tracks T in the optimum solution such that $|Q(T)| = i$. We define $\omega_i = |O_i|$. Hence, $|\text{OPT}_P| = \sum_{i=1}^{n} \omega_i$. We additionally let ω_0 be the number of vertices of $V^{\text{end}}(\mathcal{P})$ that are not incident to any track in OPT_P. Since each endvertex of $V^{\text{end}}(\mathcal{P})$ can be incident to at most one track and each track T with $|Q(T)| = i$ is incident to $2i$ distinct endvertices of $V^{\text{end}}(\mathcal{P})$, we have that $\omega_0 + \sum_{i=1}^{n} 2i \cdot \omega_i = 2|\mathcal{P}|$. Finally, let $\omega_1^e \leq \omega_1$ be the number of tracks T contained in OPT_P such that $|Q(T)| = 1$ and T corresponds to an expensive link.

Algorithm A

In Algorithm A we first only consider tracks of size 1, which are just links from $L^{\text{end,out}}$. In the first step we compute a maximum number of disjoint tracks of size 1, i.e., we compute a maximum matching $A_1 \subseteq L^{\text{end,out}}$, which can be done in polynomial time.

For the second step, let R_A be the set of endvertices of $V^{\text{end}}(\mathcal{P})$ that are not incident to some link of A_1. Let I_{R_A} be the resulting set-packing instance where we have one element for each vertex in R_A and consider only those tracks T that are only incident to vertices

from R_A and additionally satisfy $|Q(T)| = 2$ (i.e., $Q(T)$ contains exactly 2 links). We then compute a solution A_2 to I_{R_A} using the $\frac{3}{5+\varepsilon}$-approximation algorithm from [21] and output the solution $S_A = A_1 \cup A_2$.

Algorithm B

Algorithm B is similar to Algorithm A, but we compute a different matching in the first step. We aim at "guessing" the value ω_1^e, i.e., how many tracks of size 1 are contained in the optimum solution OPT_P that correspond to an expensive link. There are at most $|\mathcal{P}|$ many guesses, hence we can simply try all possible values.

Given a guess of ω_1^e, we compute a subset $B_1 \subseteq L^{\mathrm{end,out}}$ of maximum size, such that B_1 contains at most ω_1^e many expensive links. This can be done by guessing the size k of the maximum matching with at most ω_1^e many expensive links and giving each expensive link in $L^{\mathrm{end,out}}$ a cost of 1 and any other link of $L^{\mathrm{end,out}}$ a cost of 0 and computing a minimum-cost matching of size exactly k. The maximum k for which the cost of the solution is at most ω_1^e is then the desired solution. This gives us the edge set B_1, which corresponds to a maximum set of disjoint tracks of length 1 with at most ω_1^e many expensive links. Note that a minimum-cost matching of size exactly k in a weighted graph with n vertices (and non-negative weights) can be found by reducing it to a minimum-cost perfect matching problem by adding $n - 2k$ dummy vertices and connecting each dummy vertex to every original vertex with an edge of cost 0.

The second step in Algorithm B is identical to the second step of Algorithm A: Let R_B be the set of endvertices of $V^{\mathrm{end}}(\mathcal{P})$ that are not incident to some link of B_1. Let I_{R_B} be the resulting set-packing instance where we have one element for each vertex in R_B and consider only those tracks T that are only incident to vertices from R_B and additionally satisfy $|Q(T)| = 2$ (i.e., $Q(T)$ contains exactly 2 links). We then compute a solution B_2 to I_{R_B} using the $\frac{3}{5+\varepsilon}$-approximation algorithm from [21] and output the solution $S_B = B_1 \cup B_2$.

Algorithm C

In the third algorithm, we directly consider the corresponding set-packing problem in which we only consider tracks T with $|Q(T)| \leq 2$ (i.e., tracks T such that $|Q(T)| = 1$ or $|Q(T)| = 2$). We then compute a solution S_C to this set-packing instance using the $\frac{3}{5+\varepsilon}$-approximation algorithm from [21] and output S_C. We set $S_C = C_1 \cup C_2$, where C_i is the set of tracks T such that $|Q(T)| = i$.

Final Algorithm

The final solution is then computed as follows. For a solution $H = (V, E(\mathcal{P}) \cup S)$ we defined $\mathrm{cost}(H) \coloneqq \mathrm{credits}(H) + |S|$. For each guess $i \in \{0, 1, \ldots, |\mathcal{P}|\}$ of ω_1^e in Algorithm B we have a solution $H^i_{\mathrm{AlgB}} = (V, E(\mathcal{P}) \cup S^i_B)$ and for Algorithm A and C we have one solution $H_{\mathrm{AlgA}} = (V, E(\mathcal{P}) \cup S_A)$ and $H_{\mathrm{AlgC}} = (V, E(\mathcal{P}) \cup S_C)$, respectively. For each of the $|\mathcal{P}| + 3$ many solutions, we then compute their respective cost as defined above and output the solution with lowest cost. We let $H = (V, E(\mathcal{P}) \cup S)$ be this final solution.

We note that in fact one of the solutions computed in Algorithm B will be identical to the solution computed in Algorithm A. This is true if the guess of ω_1^e is 0 and hence in Algorithm B, we simply compute a maximum matching of $L^{\mathrm{end,out}}$. However, for notational purposes it will be more convenient to also consider Algorithm A with its solution S_A.

4.4 Bounding the cost of the solution

We now bound $\text{cost}(H)$. We bound the number of links in the optimum solution as follows.

▶ **Lemma 16.** *We have* $\text{opt}_C \geq 2|\mathcal{P}| - \sum_{i\geq 1} \omega_i$.

Proof. By Lemmas 12 and 13 we have that $\text{opt}_C = 2|\mathcal{P}| - \text{opt}_P$. By definition, we have that $\text{opt}_P = \sum_{i\geq 1} \omega_i$ and therefore obtain the result. ◀

Let S_B be the solution from Algorithm B which corresponds to the correct guess of ω_1^e. From now on, we only need this solution from Algorithm B and do not consider the solutions for the other guesses of ω_1^e. Let $S_A = A_1 \cup A_2$, where A_i is the set of tracks T from S_A such that $|Q(T)| = i$ and define $\alpha_i = |A_i|$ for $i = 1, 2$. Similarly, we define $S_B = B_1 \cup B_2$ and $S_C = C_1 \cup C_2$ and set $\beta_i = |B_i|$ and $\gamma_i = |C_i|$ for $i = 1, 2$. Furthermore, let $\alpha_0, \beta_0, \gamma_0$ be the number of vertices from $V^{\text{end}}(\mathcal{P})$ that are not incident to some track of S_A, S_B or S_C, respectively. Note that such vertices correspond to leaves in S_A and S_B, respectively. Additionally, let $\alpha_0^e \leq \alpha_0$, $\beta_0^e \leq \beta_0$, and $\gamma_0^e \leq \gamma_0$ be the number of such leaves as above that are expensive w.r.t. S_A, S_B and S_C, respectively. Finally, for $j = 0, 1, 2$ let α_1^j be the number of tracks $T = uv$ of size 1 in S_A such that exactly j vertices of $\{u, v\}$ intersect some track of size 1 in OPT_P. Similarly, for $j = 0, 1, 2$ we define β_1^j. Note that $\alpha_1 = \alpha_1^0 + \alpha_1^1 + \alpha_1^2$ and $\beta_1 = \beta_1^0 + \beta_1^1 + \beta_1^2$. First, we prove the following bound on the cost of a solution by giving it certain credits.

▶ **Lemma 17.** *Let S be a solution computed by Algorithm A, B, or C. If each track of length 1 receives a credit of $\frac{3}{4}$, each track of length 3 a credit of 2, each lonely leaf vertex a credit of $\frac{3}{2}$ if it is not expensive and $\frac{7}{4}$ if it is expensive, and each degenerate path an additional credit of $\frac{1}{2}$, then we can redistribute the credits such that the credit rules (A)-(E) are satisfied. Furthermore, Invariants 7 and 8 are satisfied.*

Next, we show that also Invariant 6 is satisfied. To do so, we establish the following lemmas.

▶ **Lemma 18.** *If we assign to H_{AlgA}, H_{AlgB} and H_{AlgC} credits according to Lemma 17, then we have* $\text{cost}(H_{\text{AlgA}}) \leq 3|\mathcal{P}| - \frac{5}{4}\alpha_1 - \alpha_2 + \frac{1}{4}\alpha_0^e + \frac{1}{2}\varepsilon'|\mathcal{P}|$, $\text{cost}(H_{\text{AlgB}}) \leq 3|\mathcal{P}| - \frac{5}{4}\beta_1 - \beta_2 + \frac{1}{4}\beta_0^e + \frac{1}{2}\varepsilon'|\mathcal{P}|$ *and* $\text{cost}(H_{\text{AlgC}}) \leq 3|\mathcal{P}| - \frac{5}{4}\gamma_1 - \gamma_2 + \frac{1}{4}\gamma_0^e + \frac{1}{2}\varepsilon'|\mathcal{P}|$, *where $\varepsilon'|\mathcal{P}|$ is the number of degenerate paths in G.*

Next, we obtain some further constraints on the variables we defined earlier.

▶ **Lemma 19.** *The following inequalities are valid:*

1. $\alpha_1 \geq \omega_1, \beta_1, \gamma_1$,
2. $\alpha_1 \geq \omega_1 + \alpha_1^0$,
3. $\alpha_1 \geq \omega_1 + \beta_1^0$,
4. $\beta_1 \geq \omega_1$,
5. $\alpha_1 \geq \alpha_0^e$,
6. $\beta_1 \geq \beta_0^e$,
7. $\gamma_1 \geq \gamma_0^e$,
8. $\omega_0 \geq \beta_0^e$,
9. $\alpha_1 = \alpha_1^0 + \alpha_1^1 + \alpha_1^2$,
10. $\beta_1 = \beta_1^0 + \beta_1^1 + \beta_1^2$,
11. $\alpha_2 \geq \frac{3}{5+\varepsilon} \cdot (\omega_2 - 2\alpha_1^0 - \alpha_1^1)$,
12. $\beta_2 \geq \frac{3}{5+\varepsilon} \cdot (\omega_2 - 2\beta_1^0 - \beta_1^1)$, *and*
13. $\gamma_1 + \gamma_2 \geq \frac{3}{5+\varepsilon} \cdot (\omega_1 + \omega_2)$.

Recall that $H = (V, E(\mathcal{P}) \cup S)$, where S is the best solution output by Algorithms A, B, and C w.r.t. their cost. Note that the following lemma proves that H satisfies Invariant 6.

▶ **Lemma 20.** *For $\varepsilon \leq 0.001$ we have* $\text{cost}(H) \leq 1.9412 \cdot \text{opt}(G)$.

Proof. Let $\varepsilon'|\mathcal{P}|$ be the number of degenerate paths. Note that $\text{opt}(G) \geq |\mathcal{P}|$, and therefore $\varepsilon'|\mathcal{P}| \leq \varepsilon' \text{opt}(G)$. We prove the statement by giving a factor-revealing LP, which proves $\text{cost}(H) \leq \rho \cdot \text{opt}(G)$ if the polyhedron is empty for a specific value of ρ. The parameter ρ will then give the desired approximation ratio. The factor-revealing LP is as follows. We use the same variables as before, but for simplicity we divide all constraints by $|\mathcal{P}|$. For ease of exposition, we do not replace each variable with a new variable, which is simply the old variable divided by $|\mathcal{P}|$. Instead, we simply use the same variable names.

We use the constraints we obtained by the preceding discussion and lemmas. That is, we use the constraints obtained from Lemma 16, Lemma 17, Lemma 18, Lemma 19, and constraints that we naturally obtained from the definitions of the variables. Additionally, we simplify the constraints on the optimum solution value as follows. From Lemma 16 we get $\text{opt}_C \geq 2|\mathcal{P}| - \sum_{i\geq 1} \omega_i$. Furthermore, by definition, we have $\omega_0 + \sum_{i=1}^{n} 2i \cdot \omega_i = 2$ (recall that we divide all numbers by $|\mathcal{P}|$). Since in all the constraints we used there is no other bound on γ_3, both inequalities can be simplified to $\text{opt}_C \geq 2 - \omega_1 - \omega_2 - \omega_3$ and $\omega_0 + 2\omega_1 + 4\omega_2 + 6\omega_3 \leq 2$. We then get the following polyhedron.

$$
\begin{aligned}
\text{opt}_C &\geq 2 - \omega_1 - \omega_2 - \omega_3 \\
\text{cost}(H) &\geq \rho \cdot \text{opt}_C \\
\text{cost}(H) &\leq 3 - \frac{5}{4}\alpha_1 - \alpha_2 + \frac{1}{4}\alpha_0^e + \frac{1}{2}\varepsilon' \\
\text{cost}(H) &\leq 3 - \frac{5}{4}\beta_1 - \beta_2 + \frac{1}{4}\beta_0^e + \frac{1}{2}\varepsilon' \\
\text{cost}(H) &\leq 3 - \frac{5}{4}\gamma_1 - \gamma_2 + \frac{1}{4}\gamma_0^e + \frac{1}{2}\varepsilon' \\
2 &\geq \omega_0 + 2\omega_1 + 4\omega_2 + 6\omega_3 \\
\alpha_1 &\geq \omega_1, \beta_1, \gamma_1 \\
\alpha_1 &\geq \omega_1 + \alpha_1^0 \\
\alpha_1 &\geq \omega_1 + \beta_1^0 \\
\beta_1 &\geq \omega_1 \\
\alpha_1 &\geq \alpha_0^e \\
\beta_1 &\geq \beta_0^e \\
\gamma_1 &\geq \gamma_0^e \\
\omega_0 &\geq \beta_0^e \\
\alpha_1 &= \alpha_1^0 + \alpha_1^1 + \alpha_1^2 \\
\beta_1 &= \beta_1^0 + \beta_1^1 + \beta_1^2 \\
\alpha_2 &\geq \frac{3}{5+\varepsilon} \cdot (\omega_2 - 2\alpha_1^0 - \alpha_1^1) \\
\beta_2 &\geq \frac{3}{5+\varepsilon} \cdot (\omega_2 - 2\beta_1^0 - \beta_1^1) \\
\gamma_1 + \gamma_2 &\geq \frac{3}{5+\varepsilon} \cdot (\omega_1 + \omega_2) \\
\omega_0, \omega_1, \omega_2, \omega_3 &\geq 0 \\
\alpha_0^e, \alpha_1, \alpha_1^0, \alpha_1^1, \alpha_1^2, \alpha_2 &\geq 0 \\
\beta_0^e, \beta_1, \beta_1^0, \beta_1^1, \beta_1^2, \beta_2 &\geq 0 \\
\gamma_0^e, \gamma_1, \gamma_1^0, \gamma_1^1, \gamma_1^2, \gamma_2 &\geq 0 \ .
\end{aligned}
$$

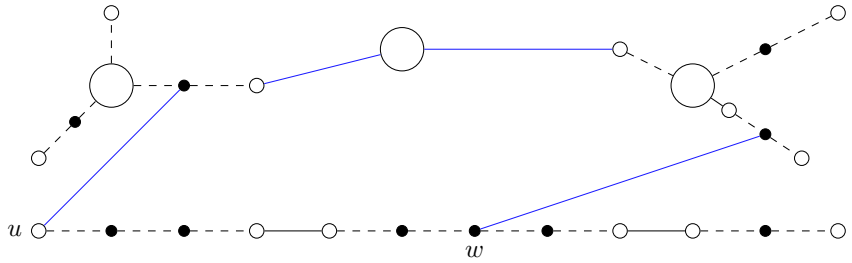

Figure 4 Example for the bridge-covering process: a pseudo-ear covering the witness path from u to w. H consists of the dashed edges (in $E(\mathcal{P})$) and solid black edges (links of H). Filled vertices are interior vertices of paths from $\mathcal{P}$, empty vertices are endvertices of paths in $\mathcal{P}$. The big circles represent 2EC blocks or components. The blue edges form a pseudo-ear Q^{uw} from u to w. All bridges, vertices and blocks on the path from u to w in H will be in a 2EC block in $H \cup Q^{uw}$.

By plugging this into a standard solver and setting $\rho \geq 1.9412$, $\varepsilon \leq 0.001$ and $\varepsilon' \leq 0.0001$, we obtain that the polyhedron is empty and hence we obtain the result. ◀

5 Bridge-Covering

After computing a good starting solution, our next goal is to ensure that the current partial solution $H = (V, E(\mathcal{P}) \cup S)$ is bridgeless. This process, called *bridge-covering*, eliminates all bridges in H while preserving Invariants 6-8. In particular, we show the following lemma.

▶ **Lemma 21.** *Let G be some structured instance of* PAP *and $H = (V, E(\mathcal{P}) \cup S)$ be a solution satisfying the Invariants 6-8. In polynomial time we can compute a solution $H' = (V, E(\mathcal{P}) \cup S')$ that satisfies Invariants 6-8 and contains no bridges.*

Throughout this section we assume that H contains some complex component C (a component that contains bridges), as otherwise the lemma holds trivially. We prove Lemma 21 by showing that we can iteratively turn H into a solution H' such that H' also satisfies the Invariants 6-8 and additionally H' contains fewer bridges than H (except for one case that we need for technical reasons). Applying this procedure exhaustively proves Lemma 21.

Intuitively, as long as our current solution H contains complex components, i.e., it contains bridges, our goal is to add certain links to H in order to "cover" the bridges and, at the same time, maintain the Invariants 6-8. To do so, it will be convenient to add so-called pseudo-ears to H, defined as follows. Similar definitions have been used in [8] and [23].

▶ **Definition 22** (Pseudo-ear and Witness Path)**.** *Let C be a complex component of H and $u, v \in V(C)$ be distinct vertices in G_C. A* pseudo-ear *Q^{uv} from u to v is a simple path in G_C that starts in u and ends in v and does not visit any vertex of C except u and v. The unique path W^{uv} from u to v in H_C is called the* witness path *of Q^{uv}. Note that the edges of W^{uv} are bridges of C.*

An example for a pseudo-ear is given in Figure 4. We first show that a pseudo-ear Q^{uv} can be found in polynomial time, if one exists. Furthermore, pseudo-ears are useful in the following sense. If, for some pseudo-ear Q^{uv}, its witness path W^{uv} contains "enough" credit, e.g., if W^{uv} contains 2 block-nodes, 2 lonely vertices, or a lonely vertex or block node and 2 bridges from $H \cap L$, then we can add Q^{uv} to H to obtain the graph H'. Afterward, H' is a graph that contains fewer bridges than H, the number of connected components in H' is at most the number of connected components in H, and H' satisfies all invariants.

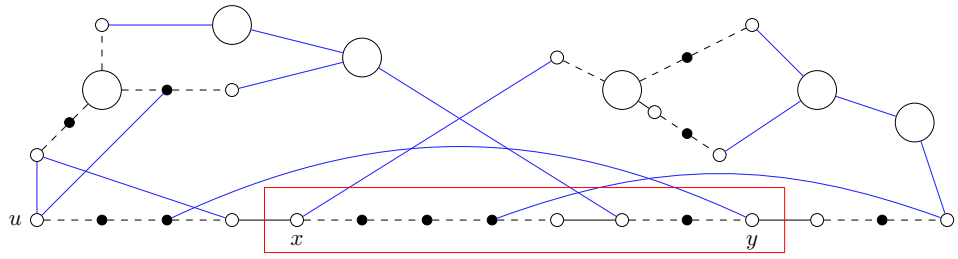

Figure 5 No witness path of a pseudo-ear starting in u contains 2 bridges of $H \cap L$. Note that the two paths P_1 and P_2 containing x and y (vertices in red box), respectively, form a separator in G (in fact, a P^2-separator).

Using pseudo-ears, we then show that in polynomial time we can find link sets $R \subseteq L \setminus S$ and $F \subseteq S$, such that $(H \cup R) \setminus F$ has fewer bridges than H and also satisfies all invariants. This then proves Lemma 21. It is crucial for this step that G is structured: We usually argue that G must have a desired pseudo-ear, as otherwise the absence of it implies that G is not structured, a contradiction.

We give an intuition for how we use structured graphs to prove Lemma 21 (see also Figure 5): If from a lonely leaf vertex u (or a leaf block node) there exists a pseudo-ear such that its witness path contains at least one other leaf vertex, one block node, or 2 bridges in L, then we can add this pseudo-ear to H, and we reduced the number of bridges while satisfying all invariants, as desired. Otherwise, if there exists no such pseudo-ear, then this complex component C must look locally like a path. Furthermore, the absence of this pseudo-ear implies that in C there must be two paths $P_1, P_2 \in \mathcal{P}$ such that $V(P_1) \cup V(P_2)$ is a separator in G. However, since C locally looks like a path, there is a link e in C connecting one endvertex of P_1 with one endvertex of P_2 in C. But then P_1 and P_2 together with e form a P^2-separator, which is a contradiction to G being structured (Property (P4)). Thus, there must be a desired pseudo-ear, which we can find in polynomial time, and hence we make progress. This is the whole intuition for proving Lemma 21. However, there are many corner cases that we have to deal with, which makes the proof a bit lengthy.

6 Gluing Algorithm

After the bridge-covering step, the graph $H = (V, P \cup S)$ consists of multiple 2-edge-connected components that remain disconnected from each other. The final step of our algorithm, called *gluing*, ensures that these components are merged into a single 2-edge-connected graph while maintaining the approximation guarantee. We show the following.

▶ **Lemma 23.** *Let G be some structured instance of* PAP *and $H = (V, E(\mathcal{P}) \cup S)$ be a solution satisfying the Invariants 6-8 that contains no bridges. In polynomial time we can compute a solution $H' = (V, E(\mathcal{P}) \cup S')$ satisfying the Invariants 6-8 such that H' is a 2-edge-connected spanning subgraph.*

Throughout this section we are given a solution S such that $H = (V, E(\mathcal{P}) \cup S)$ consists of several 2-edge-connected components, and each component contains at least the vertex set of two distinct paths $P, P' \in \mathcal{P}$. According to the credit scheme (Part (E)), each component has a credit of 2 if it is large, or it is small and contains a degenerate path. Otherwise, if it is small and not degenerate, it has a credit of $\frac{3}{2}$. Recall that a 2EC component is small if it contains exactly 2 links.

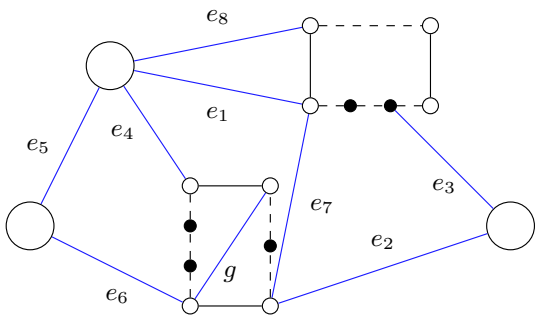

Figure 6 Example for the gluing algorithm and a good cycle. The bridgeless 2-edge-cover consists of large components (big circles) and small components (short cycles). H consists of the dashed edges (in $E(\mathcal{P})$) and solid black edges (links of H). Blue edges are edges in $E(G) \setminus E(H)$. The cycle $e_4e_5e_6$ is a good cycle in the component graph as it contains 2 large components. The cycle $e_4e_1e_7$ is also a good cycle as it shortcuts a small component using the Hamiltonian path through g.

Our approach here is to find *good* cycles in the component graph $G^{\text{comp}}(H)$, that is, the graph arising from G by contracting each 2-edge-connected component of H to a single vertex. Adding cycles from the component graph to H reduces the number of components and does not introduce any bridges. By iteratively adding such cycles, we eventually end up with a single 2-edge-connected component, and we are done.

Good cycles are cycles that either contain 2 components with credit 2 each, or a small component that can be *shortcut*. A small component C can be shortcut w.r.t. a cycle K of $G^{\text{comp}}(H)$ if we can replace $E(C)$ by a Hamiltonian path $Q \subseteq G[V(C)]$ such that $|Q \cap L| < |E(C) \cap L|$ and the vertices of C that are incident to K are the endvertices of Q. If a good cycle contains a shortcut, we then update H to $H' = (H \setminus E(C)) \cup K \cup Q$ and refer to this operation as *augmenting* H with K. Two examples of good cycles can be found in Figure 6. We then show that 1) augmenting H with a good cycle results in a graph with fewer components that is bridgeless and satisfies all invariants, 2) we can find good cycles in polynomial time if one exists, and 3) the component graph of a structured graph always admits a good cycle. This then proves Lemma 23. To prove 3), we use that G is structured: We show that every small component that does not contain a degenerate path is incident to 2 edges in G that can be used to construct a shortcut. Furthermore, since G is structured and contains no C^2-separator, these 2 edges are contained in some cycle in the component graph $G^{\text{comp}}(H)$, which we can find in polynomial time.

7 Proof of Theorem 1

Theorem 1 now easily follows from Lemmas 5-23.

Proof of Theorem 1. For structured graphs G, we first use Lemma 9 to compute a solution S such that $H = (V, E(\mathcal{P}) \cup S)$ satisfies Invariants 6-8. Lemma 21 now guarantees that we can transform H into a solution H' that is bridgeless and satisfies Invariants 6-8. Finally, Lemma 23 ensures that we can obtain in polynomial time a 2-edge-connected spanning subgraph $H'' = (V, E(\mathcal{P}) \cup S'')$ satisfying Invariants 6-8. In particular, H'' is a feasible solution and satisfies $\text{cost}(H'') = |S''| + \text{credits}(H'') \leq 1.9412 \cdot \text{opt}$. Since $\text{credits}(H'') \geq 0$, we have $|S''| \leq 1.9412 \cdot \text{opt}$, implying that our algorithm is a polynomial-time 1.9412-approximation for structured graphs. Finally, Lemma 5 then guarantees that this leads to a $(1.9412 + \varepsilon)$-approximation for arbitrary instances of bounded-length PAP. However,

we mention here that the guarantee of 1.9412 from Lemma 9 has some slack, and hence we can choose ε small enough (still constant) so that we have a 1.9412-approximation for bounded-length PAP. ◀

8 Conclusion and Future Work

In this paper we designed a 1.9412-approximation for the Path Augmentation problem, where the length of each path is bounded. We established several key ingredients: (1) a $(\frac{11}{6}+\varepsilon)$-approximation preserving reduction to structured instances of bounded-length PAP; (2) a new relaxation of the problem together with a related packing problem, which we used to obtain a good starting solution using a factor-revealing LP; (3) leveraging concepts of other connectivity augmentation problems to turn the starting solution into a feasible solution.

There might be room for improvement, in particular in bounding the cost of the starting solution. Obtaining a better bound on the cost of the starting solution, while still maintaining the credit invariants, directly gives an improved approximation ratio for bounded-length PAP. However, to improve the approximation ratio below $\frac{11}{6}$ requires new insights.

We emphasize that the requirement of bounded-length instances of PAP is only needed for the reduction to structured graphs; the remaining parts of the algorithm also work for unbounded-length PAP as long as the input is structured. Hence, it would be nice to see if this assumption can be dropped without any loss in the approximation guarantee.

References

1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–26, 2018. doi:10.1145/3182395.

2 Étienne Bamas, Marina Drygala, and Ola Svensson. A simple lp-based approximation algorithm for the matching augmentation problem. In Karen I. Aardal and Laura Sanità, editors, *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2022. doi:10.1007/978-3-031-06901-7_5.

3 Miguel Bosch-Calvo, Mohit Garg, Fabrizio Grandoni, Felix Hommelsheim, Afrouz Jabal Ameli, and Alexander Lindermayr. A 5/4-approximation for two-edge connectivity. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 653–664, 2025. doi:10.1145/3717823.3718275.

4 Miguel Bosch-Calvo, Fabrizio Grandoni, and Afrouz Jabal Ameli. A 4/3 approximation for 2-vertex-connectivity. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 29:1–29:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.29.

5 Jaroslaw Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: A reduction to steiner tree. *SIAM J. Comput.*, 52(3):718–739, 2023. doi:10.1137/21M1421143.

6 Federica Cecchetto, Vera Traub, and Rico Zenklusen. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 370–383, 2021. doi:10.1145/3406325.3451086.

7 Joe Cheriyan, Jack Dippel, Fabrizio Grandoni, Arindam Khan, and Vishnu V Narayan. The matching augmentation problem: a 7 4-approximation algorithm. *Mathematical Programming*, 182(1-2):315–354, 2020. doi:10.1007/s10107-019-01394-z.

8 Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu. An improved approximation algorithm for the matching augmentation problem. *SIAM Journal on Discrete Mathematics*, 37(1):163–190, 2023. doi:10.1137/21M1453505.

9 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part i: stemless tap. *Algorithmica*, 80:530–559, 2018. doi:10.1007/s00453-016-0270-4.

10 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part ii. *Algorithmica*, 80:608–651, 2018. doi:10.1007/s00453-017-0275-7.

11 Joseph Cheriyan, Howard Karloff, Rohit Khandekar, and Jochen Könemann. On the integrality ratio for tree augmentation. *Operations Research Letters*, 36(4):399–401, 2008. doi:10.1016/j.orl.2008.01.009.

12 Joseph Cheriyan, András Sebo, and Zoltán Szigeti. Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph. *SIAM Journal on Discrete Mathematics*, 14(2):170–180, 2001. doi:10.1137/S0895480199362071.

13 Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. Comput.*, 30(2):528–560, 2000. doi:10.1137/S009753979833920X.

14 Nachshon Cohen and Zeev Nutov. A (1+ ln2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theoretical Computer Science*, 489:67–74, 2013. doi:10.1016/j.tcs.2013.04.004.

15 Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 509–518. IEEE, 2013. doi:10.1109/FOCS.2013.61.

16 Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 281–290. ACM/SIAM, 1999. doi:10.5555/314500.314573.

17 Guy Even, Jon Feldman, Guy Kortsarz, and Zeev Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Transactions on Algorithms (TALG)*, 5(2):1–17, 2009. doi:10.1145/1497290.1497297.

18 Cristina G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *J. Algorithms*, 28(1):105–124, 1998. doi:10.1006/jagm.1998.0931.

19 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 817–831. SIAM, 2018. doi:10.1137/1.9781611975031.53.

20 Greg N Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981. doi:10.1137/0210019.

21 Martin Fürer and Huiwen Yu. Approximating the-set packing problem by local improvements. In *International Symposium on Combinatorial Optimization*, pages 408–420. Springer, 2014. doi:10.1007/978-3-319-09174-7_35.

22 Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. Improved approximation for two-edge-connectivity. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2368–2410. SIAM, 2023. doi:10.1137/1.9781611977554.ch92.

23 Mohit Garg, Felix Hommelsheim, and Nicole Megow. Matching augmentation via simultaneous contractions. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, pages 65:1–65:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.65.

24 Naveen Garg, Santosh S. Vempala, and Aman Singla. Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 103–111. ACM/SIAM, 1993. doi:10.5555/313559.313618.

25 Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. Breaching the 2-approximation barrier for the forest augmentation problem. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1598–1611, 2022. doi:10.1145/3519935.3520035.

26 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 632–645, 2018. doi:10.1145/3188745.3188898.

27 Klaus Heeger and Jens Vygen. Two-connected spanning subgraphs with at most $\frac{10}{7}OPT$ edges. *SIAM J. Discret. Math.*, 31(3):1820–1835, 2017. doi:10.1137/16M1091587.

28 Felix Hommelsheim. Improved approximation algorithms for path and forest augmentation via a novel relaxation. *CoRR*, abs/2505.15324, 2025. doi:10.48550/arXiv.2505.15324.

29 Felix Hommelsheim, Alexander Lindermayr, and Zhenwei Liu. A better-than-5/4-approximation for two-edge connectivity. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1468–1520. SIAM, 2026. doi:10.1137/1.9781611978971.54.

30 Christoph Hunkenschröder, Santosh Vempala, and Adrian Vetta. A 4/3-approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Transactions on Algorithms (TALG)*, 15(4):1–28, 2019. doi:10.1145/3341599.

31 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21:39–60, 2001. doi:10.1007/s004930170004.

32 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of algorithms*, 14(2):214–225, 1993. doi:10.1006/jagm.1993.1010.

33 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM (JACM)*, 41(2):214–235, 1994. doi:10.1145/174652.174654.

34 David G Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245, 1978. doi:10.1145/800133.804353.

35 Yusuke Kobayashi and Takashi Noguchi. An approximation algorithm for two-edge-connected subgraph problem via triangle-free two-edge-cover. *arXiv preprint arXiv:2304.13228*, 2023. doi:10.4230/LIPIcs.ISAAC.2023.49.

36 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):1–20, 2015. doi:10.1145/2786981.

37 Guy Kortsarz and Zeev Nutov. Lp-relaxations for tree augmentation. *Discrete Applied Mathematics*, 239:94–105, 2018. doi:10.1016/j.dam.2017.12.033.

38 Hiroshi Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126(1):83–113, 2003. doi:10.1016/S0166-218X(02)00218-4.

39 Zeev Nutov. On the tree augmentation problem. *Algorithmica*, 83:553–575, 2021. doi:10.1007/s00453-020-00765-9.

40 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, Berlin, Heidelberg, 2003. URL: https://link.springer.com/book/9783540443896.

41 András Sebő and Jens Vygen. 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*, pages 1–34, 2014. doi:10.1007/s00493-014-2960-3.

42 Vera Traub and Rico Zenklusen. Better-than-2 approximations for weighted tree augmentation and applications to steiner tree. *J. ACM*, 72(2):16:1–16:40, 2025. doi:10.1145/3722101.