Matching Augmentation via Simultaneous Contractions

Mohit Garg[1]
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Felix Hommelsheim
Faculty of Mathematics and Computer Science, Universität Bremen, Germany

Nicole Megow
Faculty of Mathematics and Computer Science, Universität Bremen, Germany

Abstract

We consider the matching augmentation problem (MAP), where a matching of a graph needs to be extended into a 2-edge-connected spanning subgraph by adding the minimum number of edges to it. We present a polynomial-time algorithm with an approximation ratio of $13/8 = 1.625$ improving upon an earlier 5/3-approximation. The improvement builds on a new α-approximation preserving reduction for any $\alpha \geq 3/2$ from arbitrary MAP instances to well-structured instances that do not contain certain forbidden structures like parallel edges, small separators, and contractible subgraphs. We further introduce, as key ingredients, the technique of repeated simultaneous contractions and provide improved lower bounds for instances that cannot be contracted.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases matching augmentation, approximation algorithms, 2-edge-connectivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.65

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2211.01912` [14]

Funding *Mohit Garg*: Supported by SERB Core Research Grant (CRG/2022/001176) on "Optimization under Intractability and Uncertainty".

1 Introduction

In the `Matching Augmentation Problem (MAP)`, we are given an undirected graph G, where each edge $e \in E(G)$ has a weight in $\{0, 1\}$, and all the zero-weight edges form a matching. The task is to compute a minimum weight 2-edge-connected spanning subgraph (2-ECSS) of G, which is a connected graph $(V(G), F)$ with $F \subseteq E(G)$ that remains connected on deleting an arbitrary edge.

`MAP` is a fundamental problem in the field of *survivable network design* and is known to be MAX-SNP-hard with several better-than-2 approximation algorithms [3, 5, 6]. Prior to this work, the best-known approximation ratio for `MAP` was $\frac{5}{3}$, achieved by Cheriyan et al. [6].

Both [5, 6] provide combinatorial algorithms for `MAP`, where the approximation ratios are achieved by comparing the outputs against the minimum-cardinality 2-edge-cover (D_2). A 2-*edge-cover* of an undirected graph is a spanning subgraph in which each node has a degree of at least 2, but it may not be connected. Thus, computing a D_2 is a relaxation

[1] A part of this work was done while the author was affiliated with the University of Bremen and the University of Hamburg.

of MAP. In contrast to solving MAP, a D_2 can be computed exactly in polynomial time by extending Edmonds' matching algorithm [10]. A weaker approximation result for MAP by Bamas et al. [3] follows a very different approach: the output is compared against another lower bound on an optimal MAP solution, obtained by solving a linear programming relaxation, the so-called Cut LP. The integrality gap of the Cut LP is at least $\frac{4}{3}$ [3].

Our result. We present a polynomial-time algorithm for MAP with an approximation ratio of $\frac{13}{8} = 1.625$, improving the previous best ratio of 5/3.

▶ **Theorem 1.** *There is a polynomial-time $\frac{13}{8}$-approximation algorithm for MAP.*

This improvement builds on a new α-approximation preserving reduction for any $\alpha \geq 3/2$ from arbitrary MAP instances to well-structured instances that do not contain certain forbidden structures like parallel edges, small separators, and contractible subgraphs. We further introduce, as key ingredients, the technique of repeated simultaneous contractions and provide improved lower bounds for instances that cannot be contracted.

Further related work. MAP sits between the minimum unweighted 2-ECSS and the minimum weighted 2-ECSS problems. For the minimum weighted 2-ECSS problem, improving known 2-approximations [2, 18, 19, 27] is a major open problem. Whereas for the unweighted case, in a recent breakthrough, Garg et al. [13] provided a 1.326-approximation, improving the earlier $\frac{4}{3}$-approximations [17, 24].

Research on the minimum weighted 2-ECSS problem assuming that the set of zero-weight edges in the input graph has a certain structure such as forest, spanning tree, matching, or disjoint paths has received a lot of attention recently. A general variant is the Forest Augmentation Problem (FAP), where the edges in the input graph have 0/1 edge weights. For FAP, only recently, Grandoni et al. [15] obtained a 1.9973-approximation, breaking the approximation barrier of 2. A famous special case of FAP is the unweighted Tree Augmentation Problem (TAP) where the zero-weight edges in the input graph form a single connected component. In a long line of research, several better-than-2 approximations have been achieved for TAP [1, 4, 7–9, 11, 12, 16, 20–23, 25, 26], culminating in a 1.393-approximation by Cecchetto et al. [4].

Notice that MAP is somewhat orthogonal to TAP in terms of connectivity as it has many small connected components as input instead of a single big one. Understanding both the extreme cases well, TAP and MAP, seems promising for making further progress for FAP.

Organization of the rest of the paper. Section 2 contains preliminaries and a high-level overview of our work along with some important definitions. Section 3 and Section 4 consist of the description of our reduction and algorithm, respectively, along with the corresponding theorem and lemma statements which we prove in the appendix of the full version of this paper [14]. Using these theorems and lemmas, in Section 5 we prove Theorem 1. In Section 6, we conclude with final remarks, pointing out the bottleneck for improving our algorithm.

In the full version [14] we have included detailed proofs of various lemmas in Appendices A–F, which makes our write-up lengthy. A lot of material, especially in Appendices A, C, and F are standard, but formally necessary; the new innovations are mainly contained in Appendices B, D, and E. While some proofs admit a case analysis, no single proof has too many cases. We have tried to keep the exposition reader-friendly and make the proofs easily verifiable at the expense of making the write-up a bit lengthy; a terser style might have saved some pages.

2 Technical overview

2.1 Preliminaries

We use standard notation for graphs. We consider weighted undirected graphs where each edge has a weight of either 0 or 1. A *MAP instance* consists of a graph G such that the zero-weight edges of G form a matching, and the task is to compute a minimum weight 2-edge-connected spanning subgraph (2-ECSS) of G, which is a connected subgraph $(V(G), F)$ which remains connected on deletion of an arbitrary edge. Without loss of generality, we may assume that the input graph G is 2-edge-connected; this can be checked in polynomial time by testing for each edge whether its deletion results in a disconnected graph.

Given a set of edges $F \subseteq E(G)$, $||F||$ denotes the weight of F, i.e., the number of unit edges in F. With slight abuse of notation, we denote the weight of a subgraph H of G by $||H||$. Thus, $||H|| = ||E(H)||$. Given a MAP instance G, let $\mathsf{OPT}(G)$ represent a 2-edge-connected spanning subgraph of G of minimum total weight $\mathsf{opt}(G) := ||\mathsf{OPT}(G)||$. When G is clear from the context we sometimes omit G.

Whenever we speak of components of a graph we refer to its *connected* components.

2.2 Algorithmic template and the previous $\frac{5}{3}$-approximation

The algorithm and analysis of Cheriyan et al. [6], for obtaining a $\frac{5}{3}$-approximate solution for MAP, exemplifies the general template for obtaining combinatorial algorithms for 2-edge-connected spanning subgraphs used in several works [5, 11, 17, 20]. We first explain this template, by giving an overview of the algorithm of Cheriyan et al. [6], and then in the next subsection highlight our approach where we alter this template to achieve our improvement.

The algorithm consists of two parts. The first part is a preprocessing step which constitutes a $\frac{5}{3}$-approximation preserving reduction from arbitrary MAP instances to well-structured instances that do not contain certain forbidden structures. This is a key element of their work, which helps them to improve upon an earlier $\frac{7}{4}$-approximation by getting rid of certain hard instances.

In the second part, they handle instances that do not contain any of the forbidden structures through a *discharging scheme*. Their algorithm starts by computing a minimum 2-edge-cover, D_2 (in polynomial time). Additionally, all the zero-edges are added to the D_2, so that the edges not in the D_2 are all unit-edges. Now, since a 2-ECSS is a 2-edge-cover, $||D_2||$ lower bounds opt, the weight of the minimum weight 2-ECSS. To output a $(1+c)$-approximate solution (for $c = \frac{2}{3}$), one has $(1+c)||D_2||$ *charge* to work with. This charge is used to buy the edges of the D_2 and a charge of c is distributed to each of the unit edges of the D_2. Now, they incrementally transform this D_2 into a 2-ECSS by adding edges to it. For each edge that is added, a charge of 1 is used up from the available charge (which is taken from nearby edges), i.e., their D_2 incrementally evolves into a 2-ECSS at the expense of *discharging*. This is an oversimplified view of their actual algorithm. In reality, sometimes they even delete edges in the process which results in gaining charge.

We briefly describe the two steps involved in transforming the D_2 into a 2-ECSS, namely *bridge covering* and *gluing*. Note that a D_2 can have several connected components. Some of these components can be 2-edge-connected, whereas some might have bridges (i.e., deleting those edges will result in increasing the number of components). The first step is to *cover* all the bridges one by one. Given a bridge, they add edges so that the bridge becomes part of a cycle; as a side effect, multiple components might merge into one. At the end of the bridge-covering step, their graph has only 2-edge-connected components. They ensure that

after using up the charge for buying the edges in the process, each component with at least 3 unit-edges has at least a charge of 2 leftover. Components with exactly 2 unit-edges (cycles of lengths 3 and 4) keep the initial charge of $2c = \frac{4}{3}$.

Next, in the gluing step, the components are merged into a single component using the leftover charge in the components resulting in a feasible solution. To see how this might be done, momentarily assume that all components have at least 3 unit-edges, i.e., having a leftover charge of at least 2. Here, one can simply contract each component into a single node, find a cycle in the contracted graph, and buy all the edges in that cycle. This will result in merging all the components corresponding to the nodes in the cycle into a single component. To be able to repeat such merges, we need to ensure that we have a leftover charge of at least 2 in this newly formed component. For a cycle of length $k \geq 2$, initially there was a charge of at least $2k$ in the corresponding components, and we need to buy exactly k edges. Thus, after the merge, the leftover charge in the new component is at least $2k - k = k \geq 2$, maintaining the charge invariant. Repeating this process eventually leads to a feasible solution. Unfortunately, this idea is not guaranteed to work if there are components with 2 unit-edges; a cycle with 2 nodes corresponding to such components will have a total charge of $2 \times \frac{4}{3} = \frac{8}{3}$, and after buying the 2 edges in the cycle, we will be left with a charge of only $\frac{2}{3}$. On repeating such merges, the graph will run out of charge before the gluing finishes. To handle such small components one needs to delete edges to gain charge.

2.3 Highlights of our approach and innovations for the $\frac{13}{8}$-approximation

Our approach follows the same broad framework explained above with some key innovations. Formal definitions will be given in later sections.

Preprocessing

For all $\alpha \geq \frac{3}{2}$, we provide an α-approximation preserving reduction from arbitrary MAP instances to *structured graphs*. Our list of forbidden structures subsumes the one by Cheriyan et al. [6] and consists of parallel edges, cut vertices, small separators, and *contractible subgraphs*.

Contractible subgraphs are a general form of contractible cycles as considered in [17]. A 2-edge-connected subgraph H of a graph G is contractible if each 2-ECSS(G) includes at least $\frac{1}{\alpha}||H||$ unit-edges from $G[V(H)]$. Since we are interested in only an α-approximate solution, we may contract $V(H)$ into a single node, solve the problem on the contracted graph, and add the edges of H to the solution without any loss in the approximation ratio. As an example, suppose a 6-cycle in G of weight 6 has 2 antipodal vertices that have degree 2 in G. Then, $\mathsf{OPT}(G)$ must include the 4 edges incident on these two vertices. As the cycle costs 6, and $\mathsf{OPT}(G)$ is guaranteed to pick at least weight 4 from the subgraph induced on the vertices of the cycle, for a $\frac{3}{2}$-approximation, it suffices to buy all edges of this cycle, contract it and solve the reduced problem. We can detect all contractible subgraphs with constant-size vertex sets in polynomial time and remove them during preprocessing. Interestingly, some intricate structures considered in [6] are simply contractible subgraphs.

We further exclude several small separators, which is crucial for our bridge covering and gluing steps as we have less charge at our disposal. Given a separator, we split the graph into two or three parts, recursively solve the problem on the smaller parts, and then combine the solutions arguing that the approximation ratio is preserved. If each of these parts has at least 5 vertices, this step is relatively straightforward. But for parts containing at most 4 vertices, the argument becomes significantly more challenging, in particular, since we are

aiming for a better guarantee than previous work. Handling the small separators forms a substantial part of our work consisting of several innovations. In particular, given a separator that splits the graph into two parts, the structure of the interaction of the separator with the parts is exploited in carefully constructing the subproblems. Here, we sometimes introduce *pseudo-edges*, representing possible connections via the other part, and suitably remove them during the combining step. Our reduction, which works for any $\alpha \geq \frac{3}{2}$, might be useful for future works. We only need $\alpha = \frac{13}{8}$ for our main result.

Bridge covering

Empowered by a stronger preprocessing, we can rule out more structures in the input graph, which enables us to obtain a bridgeless 2-edge-cover of G, even for an approximation ratio of $(1 + c)$, for $c = \frac{5}{8}$. In fact, our bridge-covering works even for $c = \frac{3}{5}$, so it might also be useful for future works. At the end of the bridge-covering step, we have the following charges left in the 2-edge-connected components: 2 in the *large* components (containing 4 or more unit-edges), $3c = \frac{15}{8} < 2$ in the *medium* components (containing 3 unit-edges) and $2c = \frac{5}{4} \ll 2$ in the *small* components (containing 2 unit-edges).

Gluing

In the gluing step, we are able to merge all the medium components into large components even though medium components have strictly less than 2 charge. We are also able to handle some small components that have a particular configuration by deleting edges and gaining charge. Unfortunately, we were unable to handle all the small components as they have a minuscule charge. In the end, we are left with a *special* configuration that has only large (with charge ≥ 2) and small components (with charge $\frac{5}{4}$) which cannot be merged.

Two-edge-connecting special configurations

The small components of the special configuration originate in the initial D_2 and could not be merged. So we ask the following question. How *close* are these small components to OPT restricted to the vertices of the small components? Intuitively, if they are close, we should be able to do something algorithmically as we are roughly doing what OPT is doing on this part of the graph. Otherwise, if they are not close, we should be able to argue that OPT does much worse than what the D_2 does on this part of the graph, giving us an improved lower bound. Our main conceptual innovation is in articulating a notion of closeness and making this intuition work.

Method of simultaneous contractions

We now describe our measure of closeness. Let G be our input structured graph and let $H_1, \cdots, H_s$ be the small components of the special configuration obtained. We count the total number of unit-edges bought by OPT from the following subgraphs $G[V(H_1)], \cdots, G[V(H_s)]$. If this number is more than $\frac{8}{13}$ times the number of unit-edges in the small components of our special configuration, which is precisely $2s$, we say that the small components are close to OPT. Otherwise, they are not close.

Observe when the small components are indeed close, on average, each H_i is contractible, preserving an approximation ratio of $\frac{13}{8}$. Thus, algorithmically, we can simultaneously contract each $V(H_i)$ into a distinct single node, solve the problem on the reduced instance (which can be done recursively, as contracting vertices into nodes decreases the size of the graph), and add the edges of the small components to the solution, without incurring a loss in approximation.

When the small components are not close to OPT, i.e., the H_i's are not *simultaneously contractible*, we rely on the gluing step of Cheriyan et al. [6] using a charge of $\frac{4}{3}$ per small component instead of our original charge of $\frac{5}{4}$, increasing our cost. Our improvement, in this case, comes from improving the lower bound.

Note that it is not possible for us to check in polynomial time whether the small components are simultaneously contractible or not; so what should we do – contract, or use the gluing algorithm of Cheriyan et al. [6]? We do both and return the solution with a smaller weight and argue that in either scenario the algorithm performs well.

Improved lower bound

In the case when the small components are not simultaneously contractible, OPT picks at most $\frac{8}{13} \cdot 2s$ unit-edges from the $G[V(H_i)]$'s put together. Thus, at least $2s - \frac{16}{13}s = \frac{10}{13}s$ unit-edges are not picked from within the small components. We show that for each unit-edge not picked by OPT from this part, OPT buys on average at least $1 + \frac{1}{12}$ edges that go between different small components. To argue this, we crucially use the fact that the special configurations have a restricted structure, as certain merges are not possible in it. Thus, we show that in total the number of unit-edges used by OPT on the vertices of the small components is at least $\frac{8}{13} \cdot 2s + (1 + \frac{1}{12})\frac{10}{13}s = \frac{161}{78}s$, which is strictly more than $2s$, which is the number of unit-edges used by the D_2 on this part. Finally, through an elegant argument, we are able to use this improved lower bound on OPT restricted only to the vertices of the small components to show that it compensates for the increased cost incurred during gluing the small components.

2.4 Important definitions

We give some definitions that we need for the presentation of our algorithm.

▶ **Definition 2** ($f(\cdot)$). *Given a MAP instance G, let*

$$f(G) = \max\{\frac{13}{8} \cdot \mathsf{opt}(G) - 2, \mathsf{opt}(G)\}.$$

For a MAP instance G, we will compute a 2-ECSS of G with weight at most $f(G)$. Observe that the "-2" term gives us a slightly better bound than claimed, which we crucially exploit in our preprocessing.

▶ **Definition 3** (size of a graph $s(\cdot)$). *Given a graph G, its* **size** *is $s(G) = 10 \cdot |V(G)|^2 + |E(G)|$.*

We will show that the running time of our algorithm is upper bounded by a polynomial in the size of the input graph.

▶ **Definition 4** (notation graph contraction). *Given a graph G and a set of vertices $T \subseteq V(G)$, G/T denotes the graph obtained from G after contracting all the vertices in T into a single vertex. More generally, given disjoint vertex sets $T_1, \cdots T_k \subseteq V(G)$, $G/\{T_1, \cdots, T_k\}$ denotes the graph obtained from G after contracting vertices of each T_i into single vertices.*

Note that edges in G and $G/\{T_1, \cdots, T_k\}$ are in one-to-one correspondence. Given a subgraph H of the contracted graph, we use $\hat{H}$ to refer to the subgraph of G containing precisely those edges that correspond to the edges of H.

▶ **Definition 5** (contractible subgraphs). *Let $\alpha \geq 1$ and $t \geq 2$ be fixed constants. Given a 2-edge-connected graph G, a collection of vertex-disjoint 2-edge-connected subgraphs $H_1, H_2, ..., H_k$ of G is called* (α, t, k)-contractible *if $2 \leq |V(H_i)| \leq t$ for every $i \in [k]$ and every 2-ECSS of G contains at least $\frac{1}{\alpha}||\bigcup_{i\in[k]} E(H_i)||$ unit-edges from $\bigcup_{i\in[k]} E(G[V(H_i)])$.*

In our preprocessing, we will remove all $(\frac{13}{8}, 12, 1)$-contractible subgraphs, which we simply refer to as contractible subgraphs. Later, when considering a special configuration with n_s small components, we will work with a $(\frac{13}{8}, 4, n_s)$-contractible collection of small components; we will refer to the special configuration simply as $\frac{13}{8}$-simultaneously contractible.

2.5 Algorithm overview

Here, we give a brief overview of our main algorithm.

Step 1: Preprocessing: We apply our reduction to obtain a collection of subproblems of MAP on structured graphs (Section 3). We then assume that we are given some structured graph G.

Step 2: Bridge covering: We compute a D_2 in polynomial-time and apply bridge covering to obtain an *economical* bridgeless 2-edge-cover H – a bridgeless 2-edge-cover of low cost (Section 4.1).

Step 3: Special configuration: Given H, we compute a special configuration S of G (Section 4.2).

Step 4: Contract vs. glue: We compute two feasible solutions S_1 and S_2: S_2 is obtained by applying the algorithm of [6] to G and S (Section 4.3); S_1 is obtained by calling Step 1 for G_S, which arises from G by contracting each small component of S to a single vertex. Finally, we output $\arg\min\{||S_1||, ||S_2||\}$.

3 Preprocessing

We show that, for purposes of approximating MAP with any approximation ratio at least $\frac{3}{2}$, it suffices to consider MAP instances that do not contain certain forbidden configurations. These configurations are cut vertex, parallel edge, contractible subgraph, S_0, S_1, S_2, $S_{\{3,4\}}$, S_3, S_4, S_5, S_6, S'_3, S'_4, S'_5, and S'_6. The formal definitions of these structures are provided in Appendix B of the full version [14]. Each of these configurations is referred to as a **type** and is of constant size. A MAP instance with at least 20 vertices that does not contain any of these forbidden configurations is termed as **structured**.

We briefly describe some of the types that we forbid in a structured graph. Apart from cut vertex, parallel edge, and contractible subgraph, the other forbidden structures we consider can be broadly divided into two categories: (a) "Path-like"-separators and (b) "Component-like"-separators. Path-like separators are certain paths which when removed from the input graph disconnects it. Forbidding these structures in the structured graph is mostly used in the bridge-covering step. Roughly speaking, the absence of these structures helps us in finding sufficient credit while covering some path (consisting of bridges) between 2-edge connected blocks of the 2-edge-cover. Component-like structures, on the other hand, are certain 2-edge-connected subgraphs that when removed from the input graph disconnects it. Their absence from structured graphs is mainly exploited in the gluing step; it allows us to find certain cycles through some small components which help us gain credit that is needed for the gluing.

The reduction from MAP instances to structured graphs is given by the following algorithm where we assume ALG is an algorithm that works on structured graphs. Our reduction is essentially a divide-and-conquer algorithm. It searches for a forbidden configuration and if it detects one, it divides the problem into a few subproblems (at most 3) of smaller sizes, solves them recursively, and then combines the returned solutions into a solution for the original instance. In case there are no forbidden configurations in the input (the input is structured), it calls ALG to solve the problem.

Algorithm 1 Preprocessing.

function REDUCE(G)
  if G is simple and $|V(G)| \leq 20$ **then return** $\mathsf{opt}(G)$. ▷ by brute force

  Look for a forbidden configuration in G in the following type order:
  cut vertex, parallel edge, contractible subgraph, S_0, S_1, S_2, $S_{\{3,4\}}$, S_k, S'_k for $k \in \{3,4,5,6\}$.
  Stop immediately on detecting a forbidden configuration.

  if a forbidden configuration is detected **then**
    Call it F and let T be the type of F.
    $(H_1, H_2, H_3) = \mathsf{Divide}_T(G, F)$. ▷ H_2 and/or H_3 are always empty for certain types
    $H_i^* = \mathsf{Reduce}(H_i)$ for all $i \in \{1,2,3\}$.
    return $\mathsf{Combine}_T(G, H_1^*, H_2^*, H_3^*)$.
                                                ▷ G is now structured
  return $\mathsf{ALG}(G)$.

In the above algorithm, Divide_T and $\mathsf{Combine}_T$ are subroutines that are defined in Appendix B of the full version [14], which also contains proofs of the following lemmas.

▶ **Lemma 6.** *For all types T, Divide_T and $\mathsf{Combine}_T$ are polynomial time algorithms. Furthermore, given a MAP instance G and a type T, one can check in polynomial time whether G contains a forbidden configuration of type T.*

▶ **Lemma 7.** *Given a MAP instance G and a forbidden configuration F that appears in G of type T from the list L =(cut vertex, parallel edge, contractible subgraph, S_0, S_1, S_2, S_k, S'_k, $k \in \{3,4,5,6\}$) such that G does not contain any forbidden configuration of a type that precedes T in the list L and $\mathsf{Divide}_T(G, F) = (H_1, H_2, H_3)$, then the following statements hold:*

- **(i)** *for each $i \in \{1,2,3\}$ H_i is a MAP instance,*
- **(ii)** *$s(H_1) + s(H_2) + s(H_3) < s(G)$, and*
- **(iii)** *if for each $i \in \{1,2,3\}$, H_i^* is a 2-ECSS of H_i such that $||H_i^*|| \leq f(H_i)$, then $\mathsf{Combine}_T(G, H_1^*, H_2^*, H_3^*)$ is a 2-ECSS of G such that $||\mathsf{Combine}_T(G, H_1^*, H_2^*, H_3^*)|| \leq f(G)$.*

Using the above lemma, we can establish the following result: If for all structured graphs G, $\mathsf{ALG}(G)$ is a 2-ECSS of G such that $||\mathsf{ALG}(G)|| \leq f(G)$, then for all MAP instances G, $\mathsf{Reduce}(G)$ is a 2-ECSS of G such that $||\mathsf{Reduce}(G)|| \leq f(G)$.

We will produce an *admissible* ALG that calls Reduce on a smaller instance. Since Reduce and ALG call each other, we need a slightly stronger result, which is obtained using an induction argument. We first define an admissible algorithm.

▶ **Definition 8** (admissible)**.** *An algorithm* ALG *is* **admissible** *if the following holds. If for all MAP instances G with $s(G) \leq t$, $\mathsf{Reduce}(G)$ is a 2-ECSS of G such that $||\mathsf{Reduce}(G)|| \leq f(G)$, then for all structured graphs G such that $s(G) = t + 1$, $\mathsf{ALG}(G)$ is a 2-ECSS of G such that $||\mathsf{ALG}(G)|| \leq f(G)$. Furthermore, if $T(s)$ denotes the running time of* ALG *for structured graphs of size s, and $T'(s)$ denotes the running time of* Reduce *on MAP instances of size s', then $T(s) \leq T'(s-1) + poly(s)$.*

Our main results in this section are the following.

▶ **Theorem 9.** *If* ALG *is an admissible algorithm, then for all MAP instances* G, Reduce(G) *is a 2-ECSS of* G *such that* $||\text{Reduce}(G)|| \leq f(G)$.

▶ **Theorem 10.** *If* ALG *is an admissible algorithm, then* Reduce *runs in polynomial time.*

The proofs of the above two results follow from a straightforward application of Lemmas 6 and 7 and are included in Appendix A of the full version [14]. Now, if we can find an admissible ALG, Theorem 9 and Theorem 10 immediately imply Theorem 1. In the next subsections, we exhibit an admissible ALG.

4 Algorithm for structured graphs

We exhibit an admissible algorithm ALG that takes as input a structured graph G and outputs a 2-ECSS of G with weight at most $f(G)$. Our algorithm has three main steps. First, we compute an "economical" bridgeless 2-edge-cover of G. Then, with the aid of this 2-edge-cover, we compute a "special" configuration of G. We two-edge-connect the special configuration in two ways and return the solution with minimum weight. We define the relevant terms and explain these steps below.

Algorithm 2 Main algorithm for structured graphs.

function ALG(G) ▷ G is structured
  $H =$ economical bridgeless 2-edge-cover(G)
  $S =$ special configuration(G, H)
  $R =$ Contract-vs-Glue(G, S)
  return R

4.1 Computing an economical bridgeless 2-edge-cover

Given a structured graph G, we first compute an economical bridgeless 2-edge-cover of it. Before we define an economical bridgeless 2-edge-cover, we need to first define **small, medium,** and **large**, which is used to categorize a 2-edge-connected subgraph of G based on its weight.

▶ **Definition 11** (small, medium, large)**.** *For a weighted graph* G*, we call a 2-edge-connected subgraph* H *of* G **small** *if* $||H|| \leq 2$*,* **medium** *if* $||H|| = 3$*, and* **large** *if* $||H|| \geq 4$.

Note that for structured graphs, the only possible small components are cycles of length 3 or 4 with exactly 2 unit-edges, and medium components are cycles of length 3, 4, 5, or 6 with exactly 3 unit-edges.

▶ **Definition 12.** *A bridgeless 2-edge-cover* H *of a graph* G *is* **economical** *if all the zero-edges of* G *are in* H*,* $||H|| \leq \frac{13}{8} \cdot ||D_2(G)|| - 2n_\ell - \frac{15}{8}n_m - \frac{5}{4}n_s$*, where* n_ℓ, n_m*, and* n_s *are the number of large, medium, and small components of* H*, respectively. Furthermore, there exists a* D_2 *of* G *such that each small component of* H *is a small component of the* D_2.

Our main result for this subsection is as follows.

▶ **Theorem 13.** *Given a structured graph* G*, we can compute an economical bridgeless 2-edge-cover of* G *in polynomial time.*

To compute an economical bridgeless 2-edge-cover of G, we first find a D_2 of G and include all the zero-edges in it. Next, we transform this D_2 into a "canonical" D_2, which is defined in Appendix C of the full version [14]. Then, we cover the bridges of the canonical D_2 to get an "economical" bridgeless 2-edge-cover. All of this can be done in polynomial time. The details with the proof of Theorem 13 are in Appendix C of the full version [14].

4.2 Computing a special configuration

Next, given an economical bridgeless 2-edge cover H of a structured graph G, we compute a "special" configuration of G. A special configuration is a bridgeless 2-edge-cover that satisfies certain additional properties. In particular, it does not contain any medium components.

▶ **Definition 14** (Special configuration). *Given a structured graph G, we say H is a special configuration of G if*

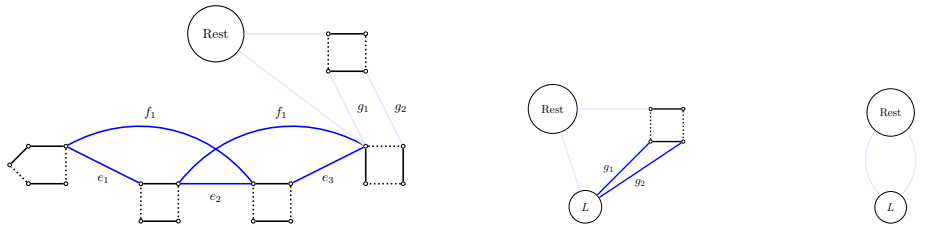
- **(i)** *H is an economical bridgeless 2-edge-cover of G,*
- **(ii)** *H does not contain medium-size components,*
- **(iii)** *G/H does not contain good cycles*
- **(iv)** *G/H does not contain open 3-augmenting paths, and*
- **(v)** *H does not contain a small to medium merge or small to large merge.*

The terms and notation used in conditions (iii)-(v) are formally defined in Appendix D of the full version [14]. Without going into details, we briefly describe the structures defined in (iii)-(v). A good cycle is a simple cycle C in G/H that contains either a) two large components, b) one large component and one small component containing a shortcut, or c) two small components each containing a shortcut. Here, we say a small component S is shortcut w.r.t. C if in $G[V(S)]$ there exists a Hamiltonian path from u to v of weight 1, where u and v are the vertices incident to C when S is expanded. Hence, there is a unit-edge in the small component S that is redundant for the 2-edge-connectivity of H after we add the edges of C to it. An open 3-augmenting path is a simple path P in G/H through 4 small components such that for each of the two interior small components there is a shortcut w.r.t. P. Finally, a small to medium merge (or small to large merge) is a set of 3 small components $S_1, S_2, S_3 \in H$ such that in $G' = G[V(S_1 \cup S_2 \cup S_3)]$ there exists a set of edges that form two medium components (or one large component) of weight precisely 6 spanning $V(G')$.

Essentially, these conditions restrict the structure of special configurations. For example, in the graph G/H (the graph obtained by contracting the various components of H into single nodes), there is no cycle that has 2 or more nodes corresponding to large components. In particular, a special configuration contains at least one small component or is already feasible. The restricted structure of special configurations will be crucially exploited while proving an improved lower bound on $\mathsf{OPT}(G)$.

From an economical bridgeless 2-edge-cover H, we obtain a special configuration by repeatedly searching for the four forbidden structures (properties (ii)-(v) above) and buying and selling certain edges such that we turn H into an economical bridgeless 2-edge cover H' with fewer components. One can show that searching for such structures can be done in polynomial time.

We briefly explain this process by an example: Figure 1a. The black edges correspond to the economical bridgeless 2-edge-cover H, where the dotted edges are of weight 0. The (bold and faint) blue edges are edges of G that are not in H. The 3 blue edges e_1, e_2, and e_3 form an open 3-augmenting path in G/H (as it can "shortcut" the two black unit edges adjacent to e_2). By the properties of structured graphs, one can show that the blue edges f_1 and f_2 must exist, and hence four components of H can be merged into one large component by



(a) open 3-augmenting path. **(b)** good cycle. **(c)** after merges.

Figure 1 An example of obtaining a special configuration.

buying all the 5 bold blue edges and selling the 2 black unit edges adjacent to e_2, which are "shortcut" by the open 3-augmenting path, to obtain Figure 1b; as initially the 4 components incident to the blue edges have a credit of $9 \times \frac{5}{8} \geq 5$, we buy 5 blue and sell 2 black edges to have a final credit of at least 2. One can show that this step is tight for our analysis with an approximation ratio of 13/8. Now, in Figure 1b the blue edges g_1 and g_2 form a good cycle in G/H (as it can be merged into a single large component). Initially, the credits in the large and small components that are part of this good cycle have a credit of $2 + 2 \times \frac{5}{8} \geq 3$. We buy the edges g_1 and g_2 and sell the unit edge adjacent to both g_1 and g_2 to form a single 2-edge-connected component having a credit of at least $3 - 2 + 1 = 2$, and thus the good cycle merges into a large component as shown in Figure 1c. In general, we show the following theorem, which is proved in Appendix D of the full version [14].

▶ **Theorem 15.** *Given a structured graph G and an economical bridgeless 2-edge-cover of it, we can compute a special configuration of G in polynomial time.*

4.3 Two-edge-connecting special configurations

Finally, we present the last part of our algorithm, which we call "Contract-vs-Glue", that converts a special configuration into a 2-edge-connected graph. Recall, a special configuration is an economical bridgeless 2-edge-cover of a structured graph that contains only small and large components and satisfies certain additional properties. Our algorithm computes two solutions and returns the one with a lower weight. The first solution is obtained by contracting the small components into single nodes and recursively computing the solution on the contracted graph (this is done by calling Reduce on the contracted graph) and then adding the edges in the small components to the solution after expanding it back. The second solution is obtained by following the "Gluing Algorithm" of Cheriyan et al. [6], which we call "Glue", and reproduce it below for completeness' sake.

Algorithm 3 Contract-vs-Glue.

function CONTRACT-VS-GLUE(G, S) ▷ G is structured, S is a special configuration of G
  if S is a 2-ECSS of G **then return** S
  Let $H_1, \cdots, H_k$ be the small components of S. ▷ now S must have small components
  $G_1^* = \mathsf{Reduce}(G/\{H_1, \cdots, H_k\})$
  $S_1 = (V(G), E(\hat{G}_1^*) \cup \bigcup_{i \in [k]} E(H_i))$
  $S_2 = \mathsf{Glue}(G, S)$
  return $\arg\min\{||S_1||, ||S_2||\}$

We will be using the following lemmas to prove our main result.

▶ **Lemma 16.** *Let G be a structured graph, S be a special configuration of G with small components $H_1, \cdots, H_k$, and let $G_1^* = \mathsf{Reduce}(G/\{H_1, \cdots, H_k\})$. If G_1^* is a 2-edge-connected spanning subgraph of $G/\{H_1, \cdots, H_k\}$, then $(V(G), E(\hat{G}_1^*) \cup \bigcup_{i\in[k]} E(H_i))$ is a 2-edge-connected spanning subgraph of G. Furthermore, if $H_1, \cdots, H_k$ is $(\frac{13}{8}, 4, k)$-contractible in G and $||G_1^*|| \leq f(G/\{H_1, \cdots, H_k\})$, then $||(V(G), E(\hat{G}_1^*) \cup \bigcup_{i\in[k]} E(H_i))|| \leq f(G)$.*

The first statement in the above lemma is straightforward to see. The second part is obtained by specializing Lemma 33 of Appendix B of the full version [14] to our parameters.

The following lemma is proved implicitly in [6]. For completeness, we give a full proof in Appendix F of the full version [14].

▶ **Lemma 17.** *Let G be a structured graph and S be a special configuration of G with n_ℓ large and n_s small components. Then, $\mathsf{Glue}(G, S)$ is a 2-edge-connected spanning subgraph of G with $||\mathsf{Glue}(G, S)|| \leq ||S|| + 2n_\ell + \frac{4}{3}n_s - 2$.*

Also, we prove the following lower bound result. Informally, if a 2-ECSS includes t fewer edges from within the small components of a special configuration, it must include $t(1 + \frac{1}{12})$ edges going between different small components. The proof is given in Appendix E of the full version [14]..

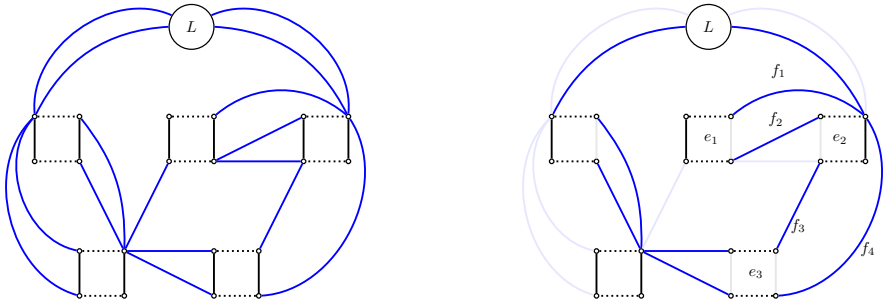
▶ **Lemma 18.** *Let G be a structured graph and S be a special configuration of G with k small components: $H_1, \cdots, H_k$. Let R be any 2-edge-connected spanning subgraph of G such that $2k - \sum_{i\in[k]} ||R[V(H_i)]|| = t$, then $\sum_{i<j\leq k} e_R(V(H_i), V(H_j)) \geq (1 + \frac{1}{12})t$, where $e_R(A, B)$ represents the number of unit-edges going between vertex sets A and B in R.*

Here, we give an intuition on how to prove Lemma 18. Fix a structured graph G together with a special configuration S and a 2-edge-connected spanning subgraph R as specified in Lemma 18. In order to simplify things, here we assume that each small component H_i of S is a cycle of length 4 such that G does not contain any of the diagonals of H_i. Furthermore, let $H_1, H_2, \ldots, H_\ell$ be the set of small components in S.

An edge between two small components is called **crossing**, whereas an edge inside a small component is called **inside**. Informally speaking, Lemma 18 states that, on average, for each inside edge $e \in E(S)$ of some small component of S that is not present in R, $E(R)$ has to contain at least $1 + \frac{1}{12}$ crossing edges. First, one can show that the vertices incident to an inside edge e that is not present in $E(R)$ cannot be adjacent to vertices of a large component of S, as otherwise this implies that S contains a good cycle, contradicting that S is special. Hence, each vertex incident to an inside edge e that is not present in $E(R)$ must be incident to at least one crossing edge in R.

In order to show that R contains sufficiently many crossing edges, we define an assignment ξ that distributes for each inside or crossing edge of R a total charge of one to the small components $H_1, \ldots, H_\ell$. The sum over all charges of edges incident to some component H_i then defines the *load* of the component H_i. Note that, by construction, the total load over all small components is equal to the number of inside and crossing edges in R.

Each inside edge contributes one to the charge of its component, while each crossing edge distributes a charge of one to the components incident to it: if only one of the two unit edges of $E(S)$ adjacent to a crossing edge is shortcut (absent) in R, then the component with the shortcut edge receives a charge of one from that crossing edge. Otherwise, both incident components receive a charge of $\frac{1}{2}$. Consider for example Figure 2b, where the bold edges represent R. By the above assignment, the components incident to f_3 receive a charge of $\frac{1}{2}$ each from f_3, while only the component containing e_3 receives a charge of 1 from f_4. The total load of the components (from left to right, top to bottom) then is 3, $\frac{5}{2}$, 2, 2, and $\frac{7}{2}$, respectively.



(a) Special Configuration. **(b)** Optimal Solution.

Figure 2 Example: special configuration, optimal solution, and lower bound.

From our assignment, one can easily argue that the load of each small component is ≥ 2. Furthermore, if there are no two shortcut edges that are adjacent to a crossing edge, then it clearly follows that Lemma 18 holds. In fact, in this case, we could replace the $1 + \frac{1}{12}$ by 2 in the lemma – a much stronger result.

Hence, we may assume that there are some shortcuts that share crossing edges, e.g. edges e_1, e_2, and e_3 in Figure 2b. However, in this case, the edges f_2 and f_3 (which form an *open 2-augmenting path*) cannot be extended to an open 3-augmenting path (since S is special); the edges f_1 and f_4 have to go back to the component containing e_2. One can show that in this case (since there are also no good cycles or local merges), the average load of the components containing e_1, e_2, and e_3 is at least $\frac{5}{2}$. In the remaining case when there are no open 2-augmenting paths in R, using a similar argument we can also show that the average load of a component is at least $2 + \frac{1}{6}$. This load assignment then implies the statement of Lemma 18.

5 ALG is admissible

As noted earlier, from Theorems 9 and 10, it follows that if we can show ALG is admissible, Theorem 1 follows. Thus, we will now focus on proving that ALG is admissible.

▶ **Lemma 19.** ALG *is admissible.*

Before we proceed with the proof, we develop some key definitions and propositions that will be used in the proof. Throughout this subsection, G is a structured graph and S is a special configuration of G with small components $H_1, \cdots, H_{n_s}$.

▶ **Definition 20** (simultaneously-contractible)**.** *We say S is $\frac{13}{8}$-simultaneously contractible if the small components of S are $(\frac{13}{8}, 4, n_s)$-contractible in G.*

▶ **Definition 21** ($\mathsf{OPT}^L, \mathsf{OPT}^R, D_2^L, D_2^R$)**.** *We partition the vertex set of G in two sets: $V(G) = L \cup R$, where L consists of the vertices in the large components of the special configuration S and R is the set of remaining vertices, i.e., the set of vertices in the small components of S. Let OPT^L be the edges of $\mathsf{OPT}(G)$ that have at least one endpoint incident on a vertex in L, and OPT^R be the remaining edges of $\mathsf{OPT}(G)$, i.e., the edges whose both endpoints are in R. D_2^L and D_2^R are defined analogously: D_2^L is the set of edges of $D_2(G)$ that are incident on at least one vertex of L and $D_2^R = E(D_2(G)) \setminus D_2^L$. $\mathsf{opt}^L, \mathsf{opt}^R, d_2^L$, and d_2^R are defined to be $||\mathsf{OPT}^L||, ||\mathsf{OPT}^R||, ||D_2^L||$, and $||D_2^R||$, respectively.*

The following relationships are immediate.

▶ **Proposition 22.**

$||\mathsf{OPT}(G)|| := \mathsf{opt} = \mathsf{opt}^L + \mathsf{opt}^R$.

$||D_2(G)|| := d_2 = d_2^L + d_2^R$.

The following proposition is key to proving our bound.

▶ **Proposition 23.**

$\mathsf{opt}^L \geq d_2^L$

Proof. Assume for contradiction $\mathsf{opt}^L < d_2^L$. Observe $\mathsf{OPT}^L \cup D_2^R$ forms a 2-edge-cover of G, since each vertex of L has at least 2 edges incident on it from OPT^L (as OPT is a feasible 2-ECSS of G) and each vertex of D_2^R has 2 edges incident on it from D_2^R (as D_2^R are the edges of D_2 restricted to the small components of S, which were originally small in D_2). But

$$||\mathsf{OPT}^L \cup D_2^R|| = \mathsf{opt}^L + d_2^R < d_2^L + d_2^R = d_2,$$

which contradicts the fact that D_2 is a minimum 2-edge-cover of G. ◀

Now, we are ready to prove that ALG is admissible.

Proof of Lemma 19. Fix a structured graph G. To show ALG is admissible, we need to show two properties: (i) $\mathsf{ALG}(G)$ is a 2-edge-connected spanning subgraph of G with $||\mathsf{ALG}(G)|| \leq f(G)$ under the assumption that $\mathsf{Reduce}(G')$ is a 2-edge-connected spanning subgraph of G' with $||\mathsf{Reduce}(G')|| \leq f(G')$ for all MAP instances G' of size strictly smaller than the size of G, and (ii) $T(s(G)) \leq T'(s(G)-1) + poly(s)$, where T is the running time of ALG and T' is the running time of Reduce. Note that (ii) follows from the fact that each of the three steps in ALG takes polynomial time and in the final step, namely Contract-vs-Glue, ALG calls the subroutine Reduce only once on a smaller graph. Thus, we will focus on proving (i) below.

Note that ALG on input G first computes a special configuration S and then applies the algorithm Contract-vs-Glue on (G, S). If S is 2-ECSS, Contract-vs-Glue returns S, and $||S|| \leq \frac{13}{8}d_2 - 2n_\ell - \frac{5}{4}n_s$, where $n_\ell = 1$ is the number of large components and $n_s = 0$ is the number of small components in S (since S is an economical bridgeless 2-edge-cover of G). Thus, $||\mathsf{ALG}|| \leq f(G)$. Otherwise, S must contain at least one small component as observed in Section 4.2.

Let $H_1, \ldots, H_{n_s}$ be the small components of S. Contract-vs-Glue on (G, S) computes two solutions S_1 and S_2 and returns the one with lower weight. Recall S_1 is obtained by contracting the small components of S, calling Reduce on it, and then expanding the contracted nodes and adding back the edges of the small components. S_2 is computed by calling the $\mathsf{Glue}(G, S)$ subroutine. In either case, the output is guaranteed to be a 2-edge-connected spanning subgraph of G from Lemmas 16 and 17.

Now, to show $||\mathsf{ALG}(G)|| \leq f(G)$, we have two cases based on whether the special configuration S is $\frac{13}{8}$-simultaneously contractible in G. If S is a $\frac{13}{8}$-simultaneously contractible in G, then by invoking Lemma 16 (whose precondition holds since the contracted graph has size strictly smaller than G and then we have the guarantee that $||\mathsf{Reduce}(G')|| \leq f(G')$ for all MAP instances G'), we have $||\mathsf{ALG}(G)|| \leq ||S_1|| \leq f(G)$ and we are done.

In the case S is not $\frac{13}{8}$-simultaneously contractible in G, we will first lower bound opt and then upper bound $||S_2||$ to show $||\mathsf{ALG}(G)|| \leq f(G)$.

Lower bound on opt

From Propositions 22 and 23 we have

$$\mathsf{opt} = \mathsf{opt}^L + \mathsf{opt}^R \geq d_2^L + \mathsf{opt}^R.$$

We now focus on lower bounding opt^R. Recall OPT^R consists of edges whose both endpoints are contained in $\bigcup_{i\in[n_s]} V(H_i)$. We categorize the edges of OPT^R into two types.

- An edge of OPT^R is **inside** if both its endpoints belong to the same $V(H_i)$ for some i.
- An edge of OPT^R is **crossing** if its endpoints lie in distinct $V(H_i)$ and $V(H_j)$ for some $i \neq j$.

Since S is not $\frac{13}{8}$-simultaneously contractible in G, the number of unit-edges that are inside is at most $\frac{8}{13} \cdot 2n_s$. Let us say the number of inside edges is exactly t less than the number of unit-edges in the small components of S, i.e., $2n_s - t$, and this number is at most $\frac{8}{13} \cdot 2n_s$.

Now, to lower bound the number of unit-edges that are crossing, we invoke Lemma 18, which states that the number of unit-edges going between $V(H_i)$ and $V(H_j)$ for all $i \neq j$ is at least $(1 + \frac{1}{12})$t.

Thus, we have the following lower bound for opt.

$$\begin{aligned}\mathsf{opt} = \mathsf{opt}^L + \mathsf{opt}^R &\geq d_2^L + \mathsf{opt}^R = d_2^L + ||\text{inside}|| + ||\text{crossing}|| \\ &\geq d_2^L + (2n_s - t) + \left(1 + \frac{1}{12}\right) t,\end{aligned}$$

where $2n_s - t \leq \frac{8}{13} \cdot 2n_s$. The lower bound is minimized when t is kept as small as possible, i.e., when $2n_s - t = \frac{8}{13} \cdot 2n_s$, i.e., for $t = \frac{5}{13} \cdot 2n_s$. Thus,

$$\mathsf{opt} \geq d_2^L + \frac{8}{13} \cdot 2n_s + \left(1 + \frac{1}{12}\right) \frac{5}{13} \cdot 2n_s = d_2^L + \left(\frac{16}{13} + \frac{10}{12}\right) n_s = d_2^L + \frac{161}{78} \cdot n_s.$$

Upper bound on $||\mathsf{ALG}(G)||$

Since S is an economical bridgeless 2-edge-cover of G, we have

$$||S|| \leq \frac{13}{8} \cdot d_2 - 2n_\ell - \frac{5}{4} \cdot n_s,$$

where n_ℓ and n_s denote the number of large and small components of S, respectively. Also, from Lemma 17, we have

$$||S_2|| = ||\mathsf{Glue}(G, S)|| \leq ||S|| + 2n_\ell + \frac{4}{3} \cdot n_s - 2.$$

Combing the two bounds we obtain

$$||S_2|| \leq \frac{13}{8} \cdot d_2 + \frac{1}{12} \cdot n_s - 2.$$

Now we can split d_2 as $d_2^L + d_2^R$, and use the fact that $d_2^R = 2n_s$ to obtain our bound.

$$\begin{aligned}||\mathsf{ALG}(G)|| \leq ||S_2|| &\leq \frac{13}{8} \cdot d_2 + \frac{1}{12} \cdot n_s - 2 = \left(\frac{13}{8} \cdot d_2^L + \frac{13}{8} \cdot 2n_s\right) + \frac{1}{12} \cdot n_s - 2 \\ &= \frac{13}{8} \cdot d_2^L + \frac{10}{3} \cdot n_s - 2 \leq \frac{13}{8}\left(d_2^L + \frac{160}{78} \cdot n_s\right) - 2 \leq \frac{13}{8} \cdot \mathsf{opt} - 2 \leq f(G),\end{aligned}$$

where the second last inequality follows from the lower bound on opt obtained above. ◀

6 Conclusion

In this work, we presented a $\frac{13}{8}$-approximation for MAP, which is a fundamental problem in network design. While several of our steps also work for smaller approximation ratios, two of our steps are tight for $\frac{13}{8}$: First, constructing a special configuration is tight for $\frac{13}{8}$. In particular, the merge involving 3-augmenting paths, in the worst case, uses all the available credits. On the other hand, such a merge could not be avoided, as their absence from special configurations helps us improve the lower bound later. Furthermore, the lower bound is tight. Hence, simply obtaining a better construction of a special configuration is not enough as one has to improve upon the lower bound as well. Finally, even if one can resolve these two issues, our approximation ratio would still be tight for 1.6 at two places: First, constructing a bridgeless 2-edge-cover is tight for 1.6, even though we believe that this result can be strengthened. Second, in the construction of special configuration, handling the medium components is also tight for precisely 1.6. Hence, also here new ideas are needed in order to obtain an approximation ratio below 1.6.

Our result builds on a new $\frac{3}{2}$-approximation preserving reduction to instances not containing certain structures including small separators and contractible subgraphs. Furthermore, we introduced the method of simultaneous contractions and improved lower bounds to achieve our main result. These techniques seem general and applicable to other problems in network design.

References

1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Trans. Algorithms*, 15(2):19:1–19:26, 2019.

2 Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

3 Étienne Bamas, Marina Drygala, and Ola Svensson. A simple lp-based approximation algorithm for the matching augmentation problem. In *IPCO*, volume 13265 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2022.

4 Federica Cecchetto, Vera Traub, and Rico Zenklusen. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *STOC*, pages 370–383. ACM, 2021.

5 Joe Cheriyan, Jack Dippel, Fabrizio Grandoni, Arindam Khan, and Vishnu V. Narayan. The matching augmentation problem: a $\frac{7}{4}$-approximation algorithm. *Math. Program.*, 182(1):315–354, 2020.

6 Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu. An improved approximation algorithm for the matching augmentation problem. In *ISAAC*, volume 212 of *LIPIcs*, pages 38:1–38:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

7 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part I: stemless TAP. *Algorithmica*, 80(2):530–559, 2018.

8 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part II. *Algorithmica*, 80(2):608–651, 2018.

9 Nachshon Cohen and Zeev Nutov. A (1+ln2)(1+ln2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theor. Comput. Sci.*, 489-490:67–74, 2013.

10 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

11 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 Trans. Algorithms*, 5(2):21:1–21:17, 2009.

12 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In *SODA*, pages 817–831. SIAM, 2018.

13 Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. Improved approximation for two-edge-connectivity. In *SODA (to appear)*, 2023. `arXiv:2209.10265`.

14 Mohit Garg, Felix Hommelsheim, and Nicole Megow. Matching augmentation via simultaneous contractions. *arXiv preprint*, 2022. `arXiv:2211.01912`.

15 Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. Breaching the 2-approximation barrier for the forest augmentation problem. In *STOC*, pages 1598–1611. ACM, 2022.

16 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *STOC*, pages 632–645. ACM, 2018.

17 Christoph Hunkenschröder, Santosh S. Vempala, and Adrian Vetta. A 4/3-approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Trans. Algorithms*, 15(4):55:1–55:28, 2019.

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

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

20 Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms*, 12(2):23:1–23:20, 2016.

21 Guy Kortsarz and Zeev Nutov. Lp-relaxations for tree augmentation. *Discret. Appl. Math.*, 239:94–105, 2018.

22 Hiroshi Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discret. Appl. Math.*, 126(1):83–113, 2003.

23 Zeev Nutov. On the tree augmentation problem. *Algorithmica*, 83(2):553–575, 2021.

24 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. *Comb.*, 34(5):597–629, 2014.

25 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *FOCS*, pages 1–12. IEEE, 2021.

26 Vera Traub and Rico Zenklusen. Local search for weighted tree augmentation and steiner tree. In *SODA*, pages 3253–3272. SIAM, 2022.

27 David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Comb.*, 15(3):435–454, 1995.