Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity

Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity

Dylan Hyatt-Denesik

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Hyatt-Denesik, D. V. P.

Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity by Dylan Hyatt-Denesik.
Eindhoven: Technische Universiteit Eindhoven, 2024. Proefschrift.

Cover design by Gino Weel, Grefo Press

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-6088-2

Keywords: Approximation Algorithms, Survivable Network Design, Connectivity Augmentation, Flexible Graph Connectivity

Printed by ADC Nederlands

Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr. S.K. Lenaerts, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 10 juli 2024 om 16:00 uur

door

Dylan Hyatt-Denesik

geboren te Canada

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	Prof. dr. E.R. van den Heuvel
Promotor:	Prof. dr. L. Sanita (Università Bocconi)
Copromotor:	Prof. dr. F.C.R. Spieksma
Promotiecommissieleden:	Prof. dr. M.T. de Berg
	Prof. dr. B. Speckmann
	Prof. dr. J. Byrka (Uniwersytet Wrocławski)
	Prof. dr. N. Olver (LSE)
	Prof. dr. V. Traub (Universität Bonn)
	Prof. dr. F. Grandoni (ISDIA)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Acknowledgments

The path to the completion of this dissertation has been a long and surprising one. I would not have been able to complete this journey without the help and support of many friends and family. I would like to take the time here to thank the many people who have been important on my path.

First, I would like to thank my supervisor, Laura Sanita. How could I have known, entering your office for the first time in Waterloo that it would culminate in completing my PhD all the way here in Eindhoven. I am endlessly grateful for the opportunities that working with you has given me. Your guidance, support, and encouragement throughout this journey have been invaluable, particularly during the height of the pandemic.

Furthermore, I would like to thank Frits. Whose encouragement and advice with navigating my PhD has proven invaluable time and again. Your patience in reading through this text and the valuable insight you've provided can hardly be overstated. In addition, I would like to thank you for leadership in the CO group as a whole, cultivating a warm and inviting atmosphere for everyone.

I would also like to extend my warm thanks to the members of my committee: Mark de Berg, Bettina Speckmann, Jarek Byrka, Neil Olver, Vera Traub, and Fabrizio Grandoni. Your feedback, suggestions, and constructive criticism have been vital in shaping this work. I am grateful for the time and effort you have invested in reviewing my thesis.

A special thanks goes to the CO group for creating such a welcoming and friendly environment. The fun lunches and stimulating conversations we shared over our meals, have been among the highlights of my time here. The most fond memories with the CO group would of course be during the Sport and Games days, where the small cost of sunburn and scrape knees paid for incredible memories.

I'd like to thank co-authors through the years, Haris, Waldo and Afrouz.

I've learned so much from working with all of you. Whether that be working through problems or writing things out there was always something I could learn from your experiences. In particular, I would like to acknowledge Afrouz – whose friendship and mentorship have meant a lot to me.

Next I would like to thank my fellow PhD students. The camaraderie formed with endless, often ridiculous, conversations have made the time spent worthwhile. The PhD weeks were an absolute highlight of every academic year for me, spending time hanging out in a stress free environment, collaborating on the given problems, and playing games into the night were all incredible memories.

To my non-academic friends I've made along the way, Luuk and Micky, you have been great friends over these last few years. The fact that we could meet almost every week and play games together has been an absolute joy. Sharing our hobbies of gaming and modelling and collecting miniatures to bring to the table each week has been an absolute joy.

Last, but certainly not least, I would like thank my closest friends and family. To my family, thank you all for believing in me, for listening to me talk about my work and troubles. A special thanks to my parents, for allowing Michelle and me to see you all during this time, despite living so far away.

To Josh and Kody, we haven't been able to see each other in person enough, but I've always been able to rely on either of you when I've felt down about my work or just wanted to talk. We all live far from each other, and the time zones make things difficult, but any time I'm able to spend with you both is time I treasure. I could not be happier that you two are able to be my seconds for my defense.

Finally, to my fiancee, Michelle, we began our relationship at the start of all this, and it's been an incredible five years. Thank you for being with me throughout all of this, even moving halfway across the world with me during the pandemic. I love you and I'm not sure I'd have been able to succeed at this endeavour without you. Thank you.

Dylan Hyatt-Denesik
Eindhoven, 2024

Abstract

Survivable network design deals with the design of a low-cost network that is resilient to the failure of some of its elements (like nodes or edges), and finds application in various settings such as, e.g., telecommunications and transportation. A fundamental problem in this area is *Connectivity Augmentation*, which asks to cheaply increase the connectivity of a given network from a value k to $k+1$. More formally, in Edge-(resp. Node-) Connectivity Augmentation, the input consists of a k-edge-connected (resp. k-node-connected) graph G and a set L of extra edges (called links), and the goal is to select the smallest subset $L' \subseteq L$ of links such that if we add L' to the graph G, then its connectivity increases by 1, i.e., it becomes $(k+1)$-edge-connected (resp. $(k+1)$-node-connected). Both Edge and Node-Connectivity Augmentation problems are NP-hard, and thus do not admit polynomial time algorithms, unless $P = NP$. The primary goals of this research are to devise efficient approximation algorithms for solving connectivity augmentation problems for various values of k, where an α-approximation algorithm is a polynomial time algorithm that provides solutions whose value is within a factor α away from the optimum. In this thesis, we consider both node-connectivity and edge-connectivity, though we give more emphasis to node-connectivity variant, as algorithmic results for this case are more scarce and often more difficult to achieve.

The first problem considered is 1-Node-Connectivity Augmentation. We develop a 1.8596-approximation algorithm for this problem, which is currently the best known approximation factor. Our result exploits a connection to another famous connectivity problem, called the *Steiner Tree* problem, where the goal is to connect a subset of nodes of an edge-weighted network by selecting edges

with minimum total cost. In particular, our approximation bound is obtained by developing a different and simpler analysis of the iterative randomized rounding technique, introduced in the literature for the Steiner Tree problem.

As a byproduct of our analysis, we are also able to improve the best approximation bound known for a class of Steiner Tree instances, called *Steiner-claw free* instances. These are instances where the induced subgraph of Steiner nodes is of degree at most two, that is, it does not possess a claw subgraph. For such instances we provide a 1.354-approximation and show that our analysis is tight.

We then consider the problem of k-Node-Connectivity Augmentation for very high values of k. We first show that $(n-d)$-Node-Connectivity Augmentation is APX-hard for any $d \geq 4$, even when the edges have unit weights. Our results complement some previously known hardness results in the literature, and completely settles the complexity status of the problem. We subsequently provide a $\frac{4}{3}$-approximation for the case $d = 4$ and unit link weights, and a $\frac{3}{2}$-approximation for the case $d = 4$ and arbitrary link weights. The proof proceeds by considering the complement graph G' of our initial graph G. A subset of the edges of this complement graph corresponds to the links L in the problem description. The cuts in G that we are asked to augment correspond to a family F of subgraphs in G', and our goal is to select a subset of links such that every subgraph in F contains a selected link.

The last problem we consider is a recently introduced problem in the field of network design, called (p,q)-*Flexible Edge-(resp. Node-)Connectivity* problem. Here one is given a graph where the edges (resp. nodes) are partitioned into *safe* and *unsafe*, and we are asked to find a minimum size subgraph that is p edge-(resp. node-)connected, and the deletion of any set of q edges (resp. nodes) leaves the graph p edge-(resp. node-) connected. We provide the first better-than-2 approximation for $(1,1)$-Flexible Node-Connectivity, and also improve the currently best known approximation factors for $(1,1)$- and $(1,k)$-Flexible Edge-Connectivity.

Contents

List of Figures

Chapter 1

Introduction

1.1 Prologue

We begin this thesis with a toy example to ground the basic concepts that will be explored in this work. The first half of this chapter will focus on building up the concepts introduced in this toy example in an informal manner, to provide the reader with the intuition necessary to understand the second half of the chapter, where we state the main results of the thesis.

Suppose you are shown a board similar to Figure 1.1, with many pieces of red string, each strung between a pair of thumbtacks. There may be so many strings on this board that it looks more like a messy spider web of strings than anything coherent. Our problem is to pick the fewest possible number of strings, satisfying the following conditions: first, if one removes every string *not* selected, the strings that were picked are still connected to each other, and each thumbtack is connected to a picked string (a property we call *connectivity*). Second, if one also removes one of the selected strings, then there will be two thumbtacks that are not connected (a property we call *minimality*). While this problem may seem straightforward, a fundamental question arises: is a set of strings that is both connected and minimal, as well as the *fewest* strings necessary for this task? Note that the term *fewest* is synonymous with *minimum size*, distinct from the concept of *minimality*.

With some thought, it should be clear that the answer is yes. That is, if you have a minimal, connected set of strings, you actually have the *minimum* number of such strings. So we can nicely characterize a minimum size connected

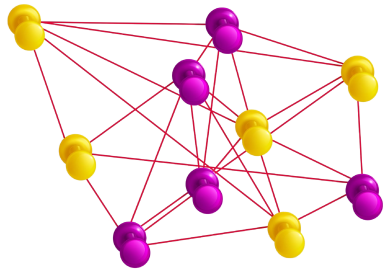


Figure 1.1: A set of thumbtacks and strings connecting the thumbtacks, as one might see on a cork board in a movie or television show.

set of strings: if you remove *any* string from the set, you disconnect the strings. Now consider a different, but related problem: can you describe a sequence of steps such that, given any board with strings and thumbtacks, if the steps are applied to the letter, the steps find a minimum size connected set of strings, if it exists?

We will return to this problem a few times in this chapter, as we provide notation that makes stating it easier, until we are ready to directly address it. The remainder of this chapter will be dedicated to expanding on these concepts so that the primary results presented in this thesis are sufficiently motivated. For complete formal definitions, see Chapter 2.

1.1.1 Graphs

The example of thumbtacks and strings on a board was not an arbitrary one. It was a thinly veiled metaphor for what is properly called a *graph*. What we called a "thumbtack", we call a *vertex* (plural: vertices), and what we called a "string" between two thumbtacks u and v, we call an *edge* e between vertices u and v. Note that an alternative name for vertices is *nodes*, which will at times be used interchangeably with vertices in this text. Observe that one can define any edge uniquely by the two vertices it connects, denoted as $e = uv$, calling u and v the *endpoints* of e. We say that the vertices u and v are *adjacent*, if there is an edge $e = uv$. Our convention, unless otherwise stated, is to denote the set of vertices by V, and the set of edges by E. Taken together, a set of vertices and edges form a graph G, typically defined as the pair $G = (V, E)$, or just G when the context is clear. See Figure 1.2 for an example of a graph.

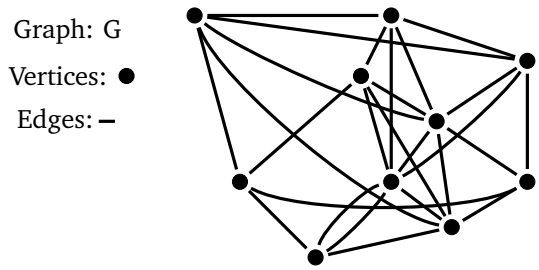


Figure 1.2: The example found in Figure 1.1 drawn as a graph $G = (V, E)$. The thumbtacks are replaced with the set of vertices, V, depicted by black dots, and the red strings are replaced with edges, E, depicted by black lines between the vertices.

Given graphs $G = (V, E)$ and $G' = (V', E')$, we call G' a *subgraph* of G when $V' \subseteq V$, $E' \subseteq E$, and we denote this by $G' \subseteq G$. A connected graph is called a *tree* if any subgraph of G found by removing a single edge from E is not connected. If a subgraph G' of G is a tree, then we can say that G' is a *subtree* of G. We say that a subgraph G' is a *spanning* subgraph of G when $V' = V$, and finally if G' is a tree, then we say it is a *spanning tree*.

Given this graph terminology, we can restate our problem: can you describe a sequence of steps such that, given any graph $G = (V, E)$, we can find a spanning subtree $T \subseteq G$, if such a T exists? This problem is known as the *Spanning Tree problem*. So we have terminology for one half of the problem statement. Namely, we have a way to describe the graph objects (graphs, trees, and subgraphs) we are interested in. In the next section, we will unpack this notion of "steps" in more detail.

1.1.2 Algorithms

In our statement of the Spanning Tree problem, we say our goal is to find a "sequence of steps" that we perform on a given graph. The technical term for these steps is, of course, an *algorithm*.

Informally, an algorithm is a well-defined series of computational steps that takes in a set of values as an *input* and returns some value, or set of values, as an *output*. Each *step* of the algorithm refers to a basic and "quick" operation

(quickness will be described in more detail later on). For our purposes, we can view a step as something as simple as reading or writing a value, or performing an arithmetic operation. With the terminology, the Spanning Tree problem becomes: given a graph $G = (V, E)$, design an algorithm that returns as output a spanning tree $T \subseteq G$.

As it turns out, we can design a very natural algorithm for this problem. Given an input graph $G = (V, E)$, begin with a subset $V' \subseteq V$ consisting of a single vertex $v \in V$ ($V' = \{v\}$), and an empty set of edges $F = \emptyset \subseteq E$. We will iteratively add vertices to V' and edges to F, until $V' = V$: find an edge $e \in E$ with one endpoint in V' and the other not in V', add e to F, and change V' to include both endpoints of e. When $V = V'$, we output subgraph $T = (V', F)$. Given the modest assumption that G is connected, it turns out that we can use the properties of minimality and connectivity to show that T is indeed a spanning tree. To see this, we observe two facts: first, in each iteration, when we add an edge to F, we require that it shares an endpoint with an edge already in F, thus we guarantee T is connected in each iteration. Second, it should be clear that in every iteration, T is minimally connected, hence T is a tree. Finally, since we stop when $V' = V$, T is a spanning tree of G. Summarizing, first, we have provided an algorithm that takes a graph as an input, and provides a subgraph as output. Second, we provided a sketch of an argument outlining the correctness of our algorithm. In fact, this algorithm is the well known Prim's Algorithm [77], which can be extended even when the edges have an associated cost, with a goal of finding a Spanning Tree of minimum cost, called the *Minimum* Spanning Tree problem. There are many well known algorithms for Minimum Spanning Tree problem [56, 61, 66, 76], including Prim's algorithm [77].

Throughout this thesis, we will focus on connectivity problems defined on graphs, and we will develop algorithms to solve these problems. When designing an algorithm, we will be principally concerned with two things:

1. Does the algorithm provably achieve its stated goals?

2. What is the runtime of the algorithm?

While the notion of an algorithm's correctness is not hard to get one's head around, the notion of an algorithm's runtime requires some unpacking.

There are a few ways to think about the runtime of an algorithm. Perhaps the most natural way is to simply have a clock start when the algorithm starts running, and stop when the algorithm terminates. This sort of real time measurement is important in many applications such as computer programs and

apps, where the quick response to the input of a user is critical to the user experience. But this kind of real time measurement is dependent on many factors that are not controlled by the algorithm itself, such as, e.g., hardware, programming language, specific implementation tricks etc.

Another way to describe the runtime of an algorithm is to consider how we define an algorithm. That is, an algorithm can be seen as a sequence of discrete, fast, computational steps, where the next step the algorithm takes depends on the input and what steps it has taken previously. Thus, we can describe the runtime of an algorithm in terms of the number of steps. The number of steps taken in an algorithm obviously depends on the input. For example, consider the Spanning Tree problem. If the input graph is already a tree, then all our algorithm needs to do is verify that fact. Compared to a graph that has many edges that can be removed, this obviously takes far fewer steps. So we must discriminate between how long an algorithm takes given the "best" case input, how long it takes given an arbitrary input, the so-called "average" case input, and how long it takes in the "worst" case input.

While there are many good reasons to focus on best or average runtimes, in this work, we are concerned with the worst case runtimes. To properly define runtimes in this sense, we need to define the *size* of the input. Formally, the size of an input is the length of a binary representation of the input, though we will typically describe the size of an input with respect to major facets of the input. For instance, in the Minimum Spanning Tree problem, given input $G = (V, E)$ and costs $c : E \to \mathbb{R}$, we will denote input size in terms of the number of vertices, $|V|$, the number of edges, $|E|$, and the values the cost function c takes.

For a worst case instance of size n, we denote the number of steps this worst case takes by a function $f(n)$, and we are interested in the general behaviour of the function f. For example, suppose that, an algorithm takes $f(n) = n^2 - n + 1$ steps. Observe that $\frac{1}{2}n^2 \leq f(n) \leq n^2$ for all $n \geq 1$. So $f(n)$ can be understood as being in some sense "equivalent" in general to n^2. This is referred to as the *asymptotic behaviour* of the function. The asymptotic behaviour of functions will be more completely described in Chapter 2, we only define here the following:

Definition 1.1 (Asymptotic behaviour (Big-O notation))**.** *Given a function* $g : \mathbb{N} \to \mathbb{R}$*, we denote the class of functions* $\{f : \mathbb{N} \to \mathbb{R} \mid \exists n_0,\ c \in \mathbb{N},\ f(n) \leq cg(n),\ \forall n \geq n_0\}$ *by* $O(g(n))$.

With Big-O notation, we have a useful means to upper bound the runtime of an algorithm. With these tools, given a choice between two algorithms, we can determine which algorithm has a better runtime, at least in an asymptotic sense. For example, consider algorithms A_1 and A_2, and assume that in their

worst cases, Algorithm A_1 takes $O(2^n)$, and Algorithm A_2 takes $O(n^n)$ time. It is not hard to see that in this case A_1 is the superior algorithm, with a quicker worst case run time.

However, absent this sort of comparison, can we say that algorithm A_1 is itself an algorithm with a "good" runtime? By many conventions, one would say that A_1 does not have a good runtime, that is, it is not an *efficient* algorithm. An algorithm is called efficient when its worst case runtime, $f(n)$, is in $O(n^c)$ for some fixed $c \in \mathbb{R}_{\geq 0}$. That is, an efficient algorithm is an algorithm whose runtime can be bounded by a polynomial in its input size. Such an algorithm is called *polynomial time*.

Now, when we are meant to provide an efficient algorithm to solve a problem, the task seems clear: design an algorithm, check if it is polynomial time, and if not, then try another algorithm, until we find an efficient one. One would hope that with enough time and energy, we will converge on an efficient algorithm for any problem. Unfortunately this is unlikely to be the case, as we will see in the following section.

1.1.3 Complexity

In this section, we will continue the discussion about efficiency by introducing the concept of complexity classes of algorithms. Up to this point in the text, we have primarily dealt with *optimization* problems, which are characterized by the goal of minimizing or maximizing some well-defined objective function. However, to understand computational tractability, we will instead consider a category of problems that may, at first glance, appear more straightforward - *decision* problems. A decision problem takes the same input as an optimization problem, but instead of a potentially complex output function, the output is binary, often understood to be a yes or no. To begin our understanding of complexity, we informally define the following crucial classifications of decision problems.

Definition 1.2 (Classes P and NP)**.** *A decision problem is in the complexity class P (Polynomial time) if one can decide if an instance of the problem admits a yes answer in polynomial time.*

A decision problem is in the complexity class NP (Non-deterministic Polynomial time) if it satisfies the following condition: for any given candidate solution, x, of a given instance of the problem, the validity of x as a yes solution can be determined in polynomial time.

To give an example of a problem in NP, consider the decision version of the Minimum Spanning Tree problem: given graph G, is there a spanning tree with total cost at most a certain value Y? A candidate solution for this instance is obviously a subtree of the original graph, and we can readily determine if it is a yes instance simply by adding up the cost of the edges and comparing the sum to Y. Importantly, a problem in NP has no stated requirement that finding such a candidate solution is itself a problem in P. However, for the case of the Minimum Spanning Tree problem, we already know that we can compute the spanning tree of minimum cost using Prim's algorithm: apply Prim's algorithm to find tree T, sum up the cost of the edges of T, and compare the cost of this tree to the threshold value. If the cost of T is greater than the threshold, then no spanning tree will be less than the threshold. That is, the decision version of the Minimum Spanning Tree problem is in P. With this example, it is not hard to see that P is a subset of NP. A natural question one might ask is if P is equal to NP. We will return to this question shortly.

To further explore the relationship between complexity classes, we introduce the concept of *reductions*. Reductions allow us to formally express statements of the form "problem A is at least as hard as problem B".

Given two decision problems A and B, there is a polynomial time reduction from A to B if there is a polynomial time algorithm that takes an instance a of problem A and provides an instance b of problem B such that b is a yes instance of B if an only if a is a yes instance of A.

Observe that if we can verify a yes instance of B in polynomial time, then we can verify a yes instance of A by applying this reduction. Hence, using this reduction, if B can be solved in polynomial time, then A can be solved in polynomial time. Assume now that we know that the problem A is not solvable in polynomial time. By the definition of a reduction, if there was a polynomial time algorithm to solve B, then that would imply a polynomial time algorithm to solve A, which we know does not exist. Thus, using reductions, we have a technique to show that a problem is not polynomial time solvable, without needing to prove that fact directly (which can be quite complicated). For this reason, reductions are a cornerstone in complexity theory.

In this work, we often deal with problems that are in the class NP, and in particular, we are interested in the subset of problems in NP that are what we call NP-hard. We can informally understand a problem to be NP-hard if a polynomial time algorithm for it would consequently provide a polynomial time algorithm for all NP problems. Problems in NP that are NP-hard are called *NP-Complete*.

Definition 1.3 (NP-hard)**.** *A decision problem X is classified as NP-*hard *if for every problem L in NP, there exists a polynomial time reduction from L to X. Formally, given any instance I_L of a problem L, there exists a polynomial time computable function f such that I_L is a 'yes' instance of L if and only if $f(I_L)$ is a 'yes' instance of X.*

*If X is also in NP, then X is said to be NP-*Complete*.*

As an important aside, it is not trivial to show that the class of NP-Complete problems is non-empty. It was first shown by Cook [30] that this class is in fact non-empty when he proved that the so-called *Boolean Satisfiability Problem* is NP-Complete. Subsequently, Karp [57] showed that another 21 problems are NP-Complete, using the method of reductions, and by now many problems have been shown to be NP-Complete.

With this in mind, it is crucial to note that there are no known polynomial time algorithms for any NP-Complete problem, and it is unknown if $P = NP$ or not. Indeed, the absence of known polynomial time algorithms for any NP-complete problem suggests the likelihood that $P \neq NP$. A natural question is if the set of NP-Complete problems consist of any interesting or meaningful problems. To answer this question, recall the Spanning Tree problem. In the Spanning Tree problem, we are given a graph and wish to find a minimum sized set of edges such that between any two vertices we can find a path in the edges we pick. We now consider a similar problem, where we require a stronger solution, where the goal is to find a minimum sized set of edges, such that between any two vertices, we can find *two* completely edge disjoint paths between these vertices. On the surface, this problem seems like it shouldn't be much harder.

Instead, it turns out this problem is the well-studied 2-*Edge Connected Spanning Subgraph* problem, which is known to NP-Complete. Even with such a small, natural increase in the scope of a problem, it is believed that there is no efficient algorithm for solving it. Thankfully, we may still be able to find solutions for certain NP-Complete problems that are "good enough" in some sense.

1.1.4 Approximation Algorithms

To this point, we established that for an NP-Complete problem, unless $P = NP$, we cannot find an optimal solution in polynomial time. It is then natural to wonder whether there is anything we can do. Fortunately, if we relax our requirement of finding an optimal solution, and we instead only require that the

solution *approximate* the optimal one, then we may be able to find a suitable algorithm. This approach gives rise to the concept of *Approximation Algorithms*, which are at the forefront of this thesis.

Definition 1.4 ([87]). *An α-approximation algorithm for an optimization problem is a polynomial time algorithm such that, for all instances of the problem, produces a solution whose cost is within a factor α of the cost of an optimal solution.*

A natural goal for designing an approximation algorithm is to find an α-approximation algorithm with α as close to 1 as possible. However, for some problems, unless $P = NP$, one cannot find a value for α less than some fixed value $\delta > 1$. Thus, in the study of approximation algorithms, there are two typical directions of research. The first direction is to find an α-approximation algorithm, where α is as small as possible (in the case of minimization problems). The second direction, is to determine a lower bound for δ (again, in the case of minimization problems). Taken together, the overall goal is to find δ and α that are as close to equal as possible. These two lines of research will be the main source of discussion in this work.

1.1.5 Network Design

The last topic we will cover in this prologue is to introduce the actual problems that will be examined in this thesis. We have seen a few examples of algorithmic questions related to graphs, but why are we interested in graphs in the first place? It should not be hard to see that graphs have an incredibly wide variety of applications, as they can be an abstract representation of transportation systems (vertices and edges representing locations and roads respectively), telecommunication networks (vertices and edges representing computers or phones and the connections between them) or social networks (vertices and edges representing individuals and their relationships).

The primary area of interest in this thesis is in connectivity problems on graphs. To motivate these sorts of problems, consider the following example that emphasizes the value of connectivity, particularly in sparse graphs. Suppose we have a set of locations relating to the oil industry; perhaps oil fields, refineries, depots, etc, represented with vertices, V. Pipelines pumping oil between these locations are represented by a set of edges, E. This creates a network for the production of oil. However, real-world issues, such as a pipeline running through an area prone to rock slides, can disrupt this flow. If the damaged

pipeline is the only connection between two locations, oil production could halt until repairs are made, leading to significant costs. To prevent this, an option is to construct additional pipelines connecting the locations, enhancing network connectivity. In terms of graph theory, given a graph $G = (V, E)$ representing the pipeline network, we aim to add the fewest additional edges, or "links," to ensure the network remains connected even if any single edge is removed. This ensures oil flow is maintained despite individual pipeline failures.

The above example motivates a well studied problem, the *Edge-Connectivity Augmentation Problem*. We say that a graph $G = (V, E)$ is *k-Edge-Connected* (resp. *k-Vertex-Connected)* if the deletion of any $k-1$ edges (resp. vertices) from G results in a connected subgraph of G. We can now formally define the Edge-Connectivity Augmentation Problem.

Definition 1.5 (k-Edge-CAP)**.** *Let $G = (V, E)$ be a k-edge-connected graph, and $L \subseteq \binom{V}{2}$. The goal is to compute a minimum cardinality set $L' \subseteq L$ such that $G' = (V, E \cup L')$ is a $(k+1)$-edge-connected graph.*

The k-Edge-CAP has a long history, which we will cover in more detail in the introduction of Chapter 3. For the moment it is interesting to note that it has long been known (see [25,31,59]) that there is an approximation-preserving reduction from the k-Edge-Connectivity Augmentation problem to the case where $k = 1$, known as *Tree Augmentation* problem (TAP), if k is odd, and to the case where $k = 2$, known as *Cactus Augmentation* problem (CacAP), if k is even. These reductions vastly restrict the space of problems one must consider to find a unifying approximation algorithm for the k-Edge-CAP.

There is a natural extension to the vertex-connected variants (Vertex-CAP).

Definition 1.6 (k-Vertex-CAP)**.** *Let $G = (V, E)$ be a k-vertex-connected graph, and $L \subseteq \binom{V}{2}$. The goal is to compute a minimum cardinality set $L' \subseteq L$ such that $G' = (V, E \cup L')$ is a $(k+1)$-vertex-connected graph.*

Approximation results on Vertex-Connectivity Augmentation are more scarce, even for the most basic generalization, known as the Vertex-Tree Augmentation problem (1-Vertex-CAP), which is the direct extension of TAP to the vertex-connected case. Chapters 3 and 5 are primarily concerned with finding approximations for this problem.

Another fundamental connectivity problem that central to this thesis is the *Steiner Tree problem*. In the Steiner Tree problem, we are given a graph and our goal is to find a subtree of the graph that contains a specific set of vertices, called terminals, of minimum cost.

Definition 1.7 (Steiner Tree Problem)**.** *Let $G = (V, E)$ by a connected graph, with edge costs $c : E \to \mathbb{R}_{\geq 0}$, and subset of vertices $R \subseteq V$ called* terminals*. The goal is to compute a minimum cost set of edges $F \subseteq E$ such that F induces a tree containing every terminal.*

The Steiner Tree problem is a very well studied problem with a long history, going all the way back to the list of NP-hard problems given by Gary and Johnson [43]. The problem has a very long history of approximation algorithms culminating in a $\ln(4) + \varepsilon$-approximation found by Byrka et al. [16]. A more detailed look at the history of this problem will be given in Chapter 4.

Much like the augmentation problems, there is also a variant of the Steiner Tree problem that revolves around the vertices.

Definition 1.8 (Vertex-Steiner Tree Problem)**.** *Let $G = (V, E)$ by a connected graph, with vertex costs $c : V \to \mathbb{R}_{\geq 0}$, and subset of vertices $R \subseteq V$ called* terminals*. The goal is to compute a minimum cost set of Steiner vertices $S \subseteq V \setminus R$ such that the subgraph induced by $S \cup R$ is connected.*

The remainder of this chapter will be dedicated to taking a brief overview of the results achieved.

1.2 Our results

The following section gives an outline of this thesis. Each subsection corresponds to a chapter of this work, which first introduces the relevant problem, and then highlights the primary results. Finally, the last subsection provides a short bibliography of the publications that led to the creation of this thesis.

1.2.1 1-Vertex-Connectivity Augmentation

In Chapter 3 we give a 1.8596-approximation algorithm for the NP-hard problem of augmenting a given 1-vertex-connected graph to be 2-vertex-connected. This improves upon the state-of-the-art approximation previously developed in the literature [71].

Theorem 1.1. *There is a* 1.8596*-approximation algorithm for* 1*-Vertex-CAP.*

The starting point of our work is a known reduction from our Connectivity Augmentation problem to some specific instances of the Vertex-Steiner Tree problem, and our result is obtained by developing a new and simple analysis

of the so-called iterative randomized rounding technique when applied to such Steiner Tree instances (this technique will be elaborated on in Chapter 3). Our results also imply a 1.8596-approximation for k-Edge-CAP. While this does not beat the best approximation factor known for this problem [19], a key point of our work is that the analysis of our approximation factor is less involved when compared to previous results in the literature. In addition, our work gives new insights on the iterative randomized rounding method, that might be of independent interest.

1.2.2 Steiner Claw Free instances

A Steiner-Tree instance is called a Steiner-Claw Free instance when every Steiner vertex is adjacent to at most 2 other Steiner vertices. In Chapter 4, we contribute with the following main result.

Theorem 1.2. *There is a* $(\frac{991}{732} + \varepsilon < 1.354)$*-approximation algorithm for Steiner Tree on Steiner-claw free instances.*

This approximation factor improves over the $\ln(4)+\varepsilon$-approximation for general Steiner Tree problem [16]. These instances were introduced by [35] in the context of studying a pair of relaxations for the Steiner Tree problem that have different algorithmic properties. In particular, combining our results with the ones of [35], we not only improve the approximation bound, but also get a much faster running time for our algorithm.

1.2.3 Augmentation of Highly Connected Graphs

In Chapter 5, we characterize completely the computational complexity status of the problem of k-Vertex-CAP, by showing hardness for all values of k which were not addressed previously in the literature. We here let $n := |V|$.

Theorem 1.3. *For any given* $d \geq 4$, $d \in O(n^c)$ *for any fixed constant* $c > 0$*, the* $(n-d)$*-Vertex-CAP problem is NP-hard.*

We then focus on k-Vertex-CAP for $k = n-4$, which corresponds to the highest value of k for which the problem is NP-hard. We improve over the previously best known approximation bounds for this problem [69], by developing a $\frac{4}{3}$-approximation algorithm for the problem, and a $\frac{3}{2}$-approximation algorithm for the weighted setting. The weighted setting is a generalization of k-Vertex-CAP where links have an associated weight and the goal is to minimize the total weight of the selected links.

Theorem 1.4. *There is a $\frac{4}{3}$-approximation algorithm for $(n-4)$-Vertex-CAP, and a $\frac{3}{2}$-approximation algorithm for weighed $(n-4)$-Vertex-CAP.*

1.2.4 Flexible Graph Connectivity

In Chapter 6, we deal with an interesting generalization of the 2-Edge Connected Spanning Subgraph (2ECSS) that has been introduced by Adjiashvili et al. [2], called *flexible* graph connectivity. Specifically, Adjiashvili et al. [2] considered a scenario in which not all edges are subject to potential failures. The set of edges is partitioned into *safe* and *unsafe*, and the goal is to construct a network resilient to the failure of unsafe edges. A formal definition is given below.

Definition 1.9 (Flexible Graph Connectivity Problem (FGC))**.** *Given a graph $G = (V, E)$ and a partition of E into safe edges E_S and unsafe edges E_U, the goal is to find the smallest subset E' of E such that (1) $H = (V, E')$ is 1-edge-connected, and (2) for every unsafe edge $e \in E_U \cap E'$, the graph $(V, E' \setminus \{e\})$ is 1-edge-connected.*

We first improve the previously best known approximation of this problem [14].

Theorem 1.5. *There is a $\frac{10}{7}$-approximation algorithm for FGC.*

The theorem is proved by giving a refined analysis of the algorithm developed in [14]. Their algorithm relies on an approximation algorithm for 2ECSS as a subroutine, whose analysis is used mainly when the size of the optimal solution is large enough compared to the number of vertices n. Our improvement stems from realizing that 2ECSS can be approximated better than the current best known factor whenever the optimal solution is large compared to n.

In addition to FGC, we define the vertex-connectivity version of the problem, investigated first in [8]. To state it, we use the notion of a *cut-vertex* of a graph. A cut-vertex of a graph is a vertex u such that if we remove u and all its incident edges the number of connected components of the graph increases.

Definition 1.10 (Flexible Vertex-Connectivity Problem (FVC))**.** *Given a graph $G = (V, E)$ and a partition of V into safe vertices V_S and unsafe vertices V_U, the goal is to find the smallest subset E' of E such that (1) $H = (V, E')$ is connected, and (2) for every vertex $u \in V_U$, u is not a cut-vertex of H.*

Our last result of Chapter 6 yields the first approximation algorithm for FVC with an approximation factor strictly better than 2. Its proof constitutes the

most technical part of this chapter. It combines two different main algorithms, which rely on non-trivial combinatorial ingredients, like ear-decompositions and matroid intersection, which will be described in more detail in Chapter 6.

Theorem 1.6. *There is a $\frac{11}{7}$-approximation algorithm for FVC.*

1.2.5 Publications and Joint Works

Part of the results of Chapter 3 are a joint work with H. Angelidakis and L. Sanità, and were published in the journal Mathematical Programming

- Haris Angelidakis, Dylan Hyatt-Denesik, and Laura Sanità. Node connectivity augmentation via iterative randomized rounding. *Mathematical Programming*, pages 1–37, 2022.

The remainder of Chapter 3 and Chapter 4 are a joint work with A. Jabal Ameli and L. Sanità, and appeared in the proceedings of ICALP 2023 (the journal version is in preparation).

- Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanità. Finding Almost Tight Witness Trees. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik

The results in Chapter 5 are a joint work with W. Gálvez, A. Jabal Ameli, and L. Sanità and are currently under submission. An early version can be found on arXiv.

- Waldo Gálvez, Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Node connectivity augmentation of highly connected graphs. *arXiv preprint arXiv:2311.17010*, 2023

Finally, the results in Chapter 6 are a joint work with A. Jabal Ameli and L. Sanità and are currently under submission. An early version can be found on arXiv.

- Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Improved approximations for flexible network design. *arXiv preprint arXiv:2404.08972*, 2024

Chapter 2

Preliminaries

To aid in understanding the main chapters of this thesis, we introduce in this section some essential concepts. Many concepts presented here were informally introduced in Chapter 1, such as graph theoretical notions, complexity classes, and approximation algorithms. These concepts will be more formally defined here.

In Section 2.1, we define graphs and graph problems (these definitions and more can be found in [86]). In Section 2.2 we formally define notions of algorithms, complexity, algorithm runtimes, and asymptotic notation. In Section 2.3 we formally define approximation algorithms, which are of core interest to this thesis. The definitions in Sections 2.2 and 2.3 can be found in [6, 80, 87].

2.1 Graphs and Graph Problems

A graph G is defined as an ordered pair $G = (V, E)$, where V is a non-empty set, whose elements are called *vertices* (or sometimes *nodes*), and E is a set of unordered pairs of distinct vertices, called *edges*. An edge $e \in E$ connecting vertices $u, v \in V$ is denoted by $e = uv$. If $e = uu$, then we call e a *loop*. We call the pair $G = (V, A)$ a *directed graph* if V is a set of vertices, and A is a set of *arcs* (or directed edges), where each arc $a \in A$ is an ordered pair $(u, v) \in V \times V$. A graph $G = (V, E)$ is called a *multigraph* when it may contain two edges between the same pair of vertices, called *parallel* edges. Similarly, a graph is called *simple* if it contains no parallel edges, and no loops.

Given a graph G, when it is not clear from context, we may denote the vertices or edges of a graph $G = (V, E)$ by $V(G)$ or $E(G)$ respectively.

Given a graph $G = (V, E)$ and a subset $V' \subseteq V$, the *subgraph* H of G by V' is the subgraph $H = (V', E')$, where $E' \subseteq E$ is a subset of the edges in E that have both endpoints in V'. If H is a subgraph of G, we say that G is a *supergraph* of H. One way to construct subgraphs (and supergraphs) is through arithmetic-like operations of vertex and edge removal and addition. Given a graph $G = (V, E)$, a subset of vertices $V' \subseteq V$, and a subset of edges $E' \subseteq E$, we denote by $G - V' - E' \coloneqq (V \setminus V', E \setminus E')$. That is, the subgraph of G obtained by deleting the vertices in V' (and their incident edges), and deleting the edges E'. Given a graph $G = (V, E)$, a set of additional vertices V' not in V, and a set of additional edges E' not in E, we can denote by $G + V' + E' \coloneqq (V \cup V', E \cup E')$, the supergraph obtained by adding the vertices in V' to G, and then adding the edges E', where each edge in E' must have its endpoints in $V \cup V'$.

Given a graph $G = (V, E)$, and a subset of vertices $U \subseteq V$, we define $G[U] = (U, \{u_1 u_2 \in E | u_1, u_2 \in U\})$, the *induced subgraph* of U. Similarly, we can define an induced subgraph on a subset of edges $F \subseteq E$, and by abuse of notation this is denoted by $G[F] = (\{v \in V | \exists u \in V, uv \in F\}, F)$. We also say a graph $H = (V', E')$ is a spanning subgraph of G if H is a subgraph of G and $V' = V$.

Given a graph $G = (V, E)$ and a subset of vertices $U \subseteq V$, we denote by G/U the (multi-)graph obtained by first replacing the vertices of U by a single vertex $\hat{U}$, and then for every edge $uv \in E$ such that $u \in U$ and $v \in V \setminus U$ we add an edge $\hat{U}v$ to E (note that there can be multiple copies of the same edge $\hat{U}v$ in G/U, and there will be no loops on $\hat{U}$). We sometimes refer to this as *contracting* G by the vertices U. We can define a similar operation on a subset of edges $F \subseteq E$, by taking the vertex sets of connected components of (V, F), and contracting G by these sets one by one. We denote this operation by G/F.

In an undirected graph $G = (V, E)$, the degree of a vertex v, denoted $\deg(v)$, is the number of edges incident to v. For an edge $e = uv$, both vertices u and v are said to be incident to e, and e contributes to the degrees of both u and v. If the graph allows loops, each loop at v contributes 2 to the degree of v. A graph is called a *matching* if each vertex has degree at most 1, and a *perfect matching* if each vertex has degree exactly 1.

A graph $P = (V, E)$ is a *path* if the vertices may be enumerated as $V = \{v_0, v_1, \ldots, v_k\}$, and the edges can be enumerated as $E = \{v_0 v_1, v_1 v_2, \ldots, v_{k-1} v_k\}$. A graph $C = (V, E)$ is a *cycle* if $V = \{v_0, v_1, \ldots, v_k\}$, $v_k v_0 \in E$, and $C - \{v_k v_0\}$ is a path. A graph is called *acyclic* if it does not contain a cycle as a subgraph.

A well known result in graph theory is the *Handshaking Lemma*, which relates the number of edges in a graph to the sum of vertex degrees. In particular,

it implies that the sum of vertex degrees must be even.

Lemma 2.1 (Handshaking Lemma)**.** *In any undirected graph $G = (V, E)$, we have $\sum_{v \in V} \deg(v) = 2|E|$.*

Proof. Consider an undirected graph $G = (V, E)$. The sum of the degrees of all vertices is obviously $\sum_{v \in V} \deg(v)$. By definition, each edge in the graph has two endpoints. Therefore, each edge contributes exactly two to this sum (one for each of its endpoints), and there are $|E|$ edges. Therefore, we have $\sum_{v \in V} \deg(v) = 2|E|$. □

A graph $G = (V, E)$ is said to be *connected* if for every pair of vertices $u, v \in V$, there exists a path in G between u and v, and is called disconnected otherwise. Given a connected graph $G = (V, E)$, a vertex $v \in V$ is called a *cut-vertex* if $G - \{v\}$ is disconnected, and an edge $e \in E$ is called a *bridge* if $G - \{e\}$ is disconnected.

A graph $T = (V, E)$ is a *tree* if it is connected and has $|V| - 1$ edges. Equivalently, a graph $T = (V, E)$ is a tree if it is connected and acyclic. Again, equivalently, a graph $T = (V, E)$ is a tree if every edge $e \in E$ is a bridge. The vertices of a tree that have degree one are called the leaves of the tree.

We define $\lambda(u, v)$ to be the maximum number of pairwise edge-disjoint paths from u to v in G. A graph $G = (V, E)$ is said to be *k-edge-connected* if for any two vertices $u, v \in V$, $\lambda(u, v) \geq k$. Similarly, we define $\kappa(u, v)$ to be the maximum number of internally vertex disjoint paths from u to v in G. A graph $G = (V, E)$ is said to be *k-vertex-connected* if for any two vertices $u, v \in V$, $\kappa(u, v) \geq k$.

Theorem 2.1 ([86])**.** *Given a graph $G = (V, E)$, the following are equivalent*

1. *For all $u, v \in V$, $\kappa(u, v) \geq k$*
2. *For any $k - 1$ vertices $v_1, \ldots, v_{k-1} \in V$, the subgraph $G - \{v_1, \ldots, v_{k-1}\}$ is connected.*

The following construction allows us to characterize vertex-connected graphs in terms of cut-vertices and maximal subgraphs that do not contain cut-vertices.

Definition 2.1 (Blocks)**.** *Let $G = (V, E)$ be a graph such that $|V| \geq 2$. A block of G is a maximal connected subgraph of G that has at least one edge and has no cut-vertex. Therefore if G has no self-loops, a block is either an induced connected subgraph on two vertices or it is a maximal 2-vertex-connected subgraph on at least three vertices.*

The following provides some useful properties of blocks.

Lemma 2.2 ([86]). *Let $G = (V, E)$ be a graph, and let $\{B_1, \ldots, B_k\}$ be the set of all blocks of G. The following properties hold*

1. *Two blocks share at most one vertex.*
2. *The blocks $B_1, \ldots, B_k$ of G partition E, that is $E = \cup_{i=1}^{k} E(B_i)$, and $E(B_i) \cap E(B_j) = \emptyset$, for $i \neq j$.*
3. *For two distinct edges e_1 and e_2, e_1 and e_2 belong to the same block B_i if and only if there is a cycle in B_i that contains e_1 and e_2.*
4. *If G is connected, G has at most $|V| - 1$ blocks.*

We also provide a similar theorem for edge-connectivity.

Theorem 2.2 ([86]). *Given a graph $G = (V, E)$, the following are equivalent*

1. *For all $u, v \in V$, $\lambda(u, v) \geq k$*
2. *For any $k-1$ edges $e_1, \ldots, e_{k-1} \in E$, the subgraph $G - \{e_1, \ldots, e_{k-1}\}$ is connected.*

2.2 Algorithms and Complexity

In the field of combinatorial optimization, we are interested in problems that have a goal of optimizing a specific objective function over a finite set of discrete configurations. These problems typically involve finding a best arrangement or combination of elements from a finite, structured set, subject to certain constraints. Although these problems are often straightforward to state, they can pose significant challenges in terms of creating algorithms for finding these solutions.

First, we define asymptotic relations between functions, which will be important for defining complexity classes.

Definition 2.2 ([6]). *Given functions $g : \mathbb{N} \to \mathbb{N}$, we define the following classes*

- $O(g(n)) := \{f : \mathbb{N} \to \mathbb{N} | \exists d \in \mathbb{R}_{\geq 0}, n_0 \in \mathbb{N}$, *such that* $f(n) \leq d g(n), \forall n \geq n_0\}$;
- $\Omega(g(n)) := \{f : \mathbb{N} \to \mathbb{N} | \exists c \in \mathbb{R}_{\geq 0}, n_0 \in \mathbb{N}$, *such that* $c g(n) \leq f(n) \forall n \geq n_0\}$, *and;*
- $\Theta(g(n)) := \Omega(g(n)) \bigcap O(g(n))$.

For the purposes of this thesis, we provide an informal description of an algorithm (for a more complete description, see Chapter 4 of [80]). An algorithm can be seen as a finite set of instructions that perform operations on a set of input data, often represented in some binary encoding. The order the instructions of an algorithm take depends both on the data that has been processed up to the current time as well as the most recent instruction performed. For algorithms that (can) terminate, there will be at least one instruction that determines when the algorithm terminates. When the algorithm terminates, the output will be found in a pre-determined location of the dataset.

We say that an algorithm runs in *polynomial time* if the maximum number of steps it takes to terminate on any input of size n, denoted $T(n)$, is bounded by a polynomial, where each step here consists of a single instruction. The *input size* of an algorithm is the length of a binary representation of the input. Thus, an algorithm runs in polynomial time if $T(n) \in O(n^c)$ for some constant $c \in \mathbb{R}_{\geq 0}$. We say that if $T(n) \in \Omega(n^c)$ and $T(n) \notin O(n^c)$, the algorithm runs in *superpolynomial time*.

We are interested in using algorithms to find solutions to *optimization problems*, where we find a solution that minimizes or maximizes a given objective function. To understand the complexity of these problems we first focus on so-called *decision problems*.

We are given a set Σ of characters with which we can describe a problem (note that Σ could simply be the set $\{0,1\}$). Let Σ^* denote the set of all finite strings (called *words*) of letters from Σ. When we refer to the *size* of a word $x \in \Sigma^*$, we mean the number of characters (counting multiplicities) in the word, which we denote by $size(x)$. Given a subset $X \subseteq \Sigma^*$, we define a *decision problem* in the following way: given a word $x \in \Sigma^*$, does x belong to X (call x a *candidate instance* of X)? If x is in X, we say that x is an *instance* of a given problem X.

A problem X is called *polynomial time solvable* if there exists a polynomial time algorithm that decides whether or not a given word $x \in \Sigma^*$ belongs to X. The collection of all polynomial time solvable problems $X \subseteq \Sigma^*$ is denoted by P.

A class of problems that are very important for this work is the class NP, which was informally defined in Chapter 1 as the set of problems for which a candidate solution x can be verified as being an instance of the problem in polynomial time. This class is defined formally here using the terminology of decision problems.

Definition 2.3 ([80]). *The class NP consists of the collection of problems $X \subseteq \Sigma^*$ for which there is a polynomial time problem Y, and a polynomial p such that, for every $x \in \Sigma^*$, one has the following:*

$x \in X$ if and only if there exists a word $y \in \Sigma^*$ of size at most $p(size(x))$ with the word $xy \in Y$.

An important tool for studying complexity is the notion of reductions, which allow one to make statements of the form "if one can solve Y in polynomial time, then one can solve X in polynomial time". Formally, we say that a problem $X \subseteq \Sigma^*$ is *reducible* to problem $Y \subseteq \Sigma^*$ if there is a polynomial time algorithm that, for any input $x \in \Sigma^*$, returns as output $y \in \Sigma^*$ with the following property: $x \in X$ if and only if $y \in Y$. With a reduction, one can show that if X is reducible to Y and if Y belongs to P (resp. NP), then X is also in P (resp. NP).

The class we are most interested in in this work is the following, again defined in Chapter 1 somewhat informally: the class NP-Complete, which contains the problems in NP for which can be reduced to by any other problem in NP.

Definition 2.4 ([80]). *The class NP-Complete a subset of Σ^* for which the following holds.*

1. *Every problem in NP-Complete is in NP, and;*
2. *For problem X in NP-Complete, every problem Y in NP is reducible to X.*

One important question about the class NP-Complete that needs to be answered is: does NP-Complete contain any problems? This is answered in the affirmative by Cook and Levin [30].

Theorem 2.3 ([30]). *The class NP-Complete is non-empty.*

The proof of this Theorem is somewhat involved, but with at least one NP-Complete problem, one can use polynomial time reductions to show that other problems are NP-Complete as well. An initial list of 21 NP-complete problems was provided by Karp [57], and since then many problems have been shown to be NP-Complete.

The question of whether $P = NP$ or not is a major open question in the field of computer science, among the famed Millennium Prize Problems. Since the question of $P = NP$ is an open question, we do not know of any efficient algorithm for NP-Complete problems. Before we address this, we first define optimization problems and show how they can relate to problems that are NP-Complete.

Definition 2.5 ([6]). *The class of minimization (resp. maximization) optimization problems is the set $(X, (S_x)_{x \in X}, c)$ with the following*

1. *X is a decision problem;*
2. *for each instance $x \in X$, there is a set $S_x \subseteq \Sigma^*$ called feasible solutions, and a polynomial p such that $size(y) \leq p(size(x))$ for all $x \in X$ and $y \in S_x$.*
3. *cost function $c : \{(x, y) : x \in X, y \in S_x\} \to \mathbb{Q}$, and;*

Given $x \in X$, our goal is to find $y^* \in S_x$ such that $c(x, y^*) = \min_{y \in S_x} c(x, y)$ for minimization problems (resp. $c(x, y^*) = \max_{y \in S_x} c(x, y)$ for maximization problems). We call y^* the optimal solution.

Note that one can rephrase an optimization problem as a decision problem by taking a rational number $r \in \mathbb{Q}$, and asking if there is a solution $y \in S_x$ such that $c(x, y) \leq r$. Many optimization problems, when framed as a decision problem in this way, are in NP, since the decision problem can be solved by finding a $y \in S_x$ satisfying the inequality. Thus, for optimization problems whose decision version is NP-Complete, we are then left with the question of how to approach these seemingly intractable problems.

2.3 Approximation Algorithms

When finding an exact solution is computationally intractable, often understood to mean that the problem does not admit a polynomial time algorithm that finds an exact solution (unless $P = NP$), approximation algorithms come into play. Roughly speaking, an approximation algorithm for an optimization problem is a polynomial time algorithm that produces a solution close to the optimal, and is the core subject of this thesis.

Definition 2.6 ([87]). *Given a minimization (resp. maximization) optimization problem $(X, (S_x)_{x \in X}, c)$, a polynomial time algorithm A is called an α-approximation algorithm for $\alpha \geq 1$ (resp. $\alpha \leq 1$), if for every instance $x \in X$, with optimal solution $OPT(x) \in S_x$, algorithm A has output $A(x) \in S_x$, such that $c(x, A(x)) \leq \alpha \cdot c(x, OPT(x))$ (resp. $c(x, A(x)) \geq \alpha \cdot c(x, OPT(x))$).*

To categorize the problems for which there exist an approximation algorithm, we define the following class.

Definition 2.7 ([6]). *The class APX is the set of optimization problems such that for any minimization (resp. maximization) problem $I \in APX$, for some $r \geq 1$ (resp. $0 \leq r \leq 1$), there is an r-approximation algorithm for I.*

We are interested in finding approximation algorithms for optimization problems whose decision version is NP-complete, and therefore, it is not known whether they admit a polynomial time algorithm. A natural question is how "good" of approximation algorithms can we obtain? In some cases, we can obtain very good approximation algorithms in the form of *polynomial time approximation schemes*.

Definition 2.8 ([6]). *Let I be a minimization (resp. maximization) optimization problem. An algorithm A is said to be a polynomial-time approximation scheme for I if, for any instance x of I, and any constant rational value $\varepsilon > 0$, A is a $(1+\varepsilon)$-approximation algorithm (resp. a $(1-\varepsilon)$-approximation algorithm).*

We define PTAS as the class of optimization problems that admit a polynomial-time approximation scheme.

We also note that the running time of a polynomial time approximation scheme may have a runtime that is exponential in the value of ε, since ε is taken as a constant value. Several problems have polynomial time approximation schemes. However, there exists a class of problems for which a PTAS does not exist, unless $P = NP$. To identify this class of problems, we require the following notion of reductions, which allow one to relate the approximation ratio of one problem in APX to the approximation ratio of another.

Definition 2.9 ([6]). *Consider optimization problems $P_1 = (X, (S_x)_{x\in X}, c_1)$ and $P_2 = (Y, (S'_y)_{y\in Y}, c_2)$. Say that P_1 is L-reducible to P_2, denoted $P_1 \leq_L P_2$, if there exist functions f and g, and positive constants β and $\gamma \in \mathbb{Q}^+$ such that:*

- *for any instance $x \in X$ of P_1, one can compute the instance $f(x) \in Y$ of P_2 in polynomial time;*
- *for any instance $x \in X$, if S_x is non-empty (there is a feasible solution to the instance x), then $S'_{f(x)}$ is non-empty (there is a feasible solution to the instance $f(x)$);*
- *for any instance $x \in X$, and any feasible solution $y \in S'_{f(x)}$, one can compute in polynomial time the feasible solution $g(x, y) \in S_x$;*
- *for any instance $x \in X$, with an optimal solution $OPT_{P_1}(x) \in S_x$, and optimal solution $OPT_{P_2}(f(x)) \in S'_{f(x)}$, we have*

$$c_2(f(x), OPT_{P_2}(f(x))) \leq \beta c_1(x, OPT_{P_1}(x)),$$

and;

- *for any instance* $x \in X$*, with an optimal solution* $OPT_{P_1}(x) \in S_x$*, for any solution* $y \in S'_{f(x)}$*, with optimal solution* $OPT_{P_2}(f(x))$ *we have*

$$|c_1(x, OPT_{P_1}(x)) - c_1(x, g(x, y))| \leq \gamma |c_2(f(x), OPT_{P_2}(f(x))) - c_2(f(x), y)|.$$

A problem I is called APX-Hard if, for every problem I' in APX, $I \leq_L I'$, and the class APX-Complete is the collection of problems that are both in APX and APX-Hard.

Chapter 3

Node Connectivity Augmentation via Iterative Randomized Rounding

Network connectivity problems play a central role in combinatorial optimization. As a general goal, one would like to design a cheap network able to satisfy some connectivity requirements among its nodes. Connectivity Augmentation asks to cheaply increase the connectivity of a given network from a value k to $k+1$. The most classical setting considers *edge*-connectivity: here the input consists of a k-edge-connected graph G and a set L of extra links, and the goal is to select the smallest subset of links $L' \subseteq L$ such that if we add L' to the graph G, then the connectivity of G increases by 1, i.e., it becomes $(k+1)$-edge-connected. This problem was defined in Chapter 1, but we now define it again for completeness.

Definition 3.1 (k-Edge-CAP)**.** *Let $G = (V, E)$ be a k-edge-connected graph, and $L \subseteq \binom{V}{2}$. The goal of k-Edge-CAP is to compute a minimum cardinality set $L' \subseteq L$ such that $G' = (V, E \cup L')$ is a $(k+1)$-edge-connected graph.*

The k-Edge-Connectivity Augmentation problem has a long history. It was observed long ago (see Dinitz et al. [31], as well as Cheriyan et al. [25] and Khuller and Thurimella [59]) that there is an approximation-preserving reduction from the Edge-Connectivity Augmentation problem for an arbitrary k to the

case where $k = 1$ (called Tree Augmentation problem or TAP), if k is odd, and to the case where $k = 2$ (called Cactus Augmentation problem or CacAP)[1], if k is even. It is easy to see that TAP is a special case of CacAP[2]. This, combined with the aforementioned reduction, implies that an α-approximation algorithm for CacAP yields an α-approximation algorithm for the general Edge-Connectivity Augmentation problem.

Both TAP and CacAP admit simple 2-approximation algorithms [46, 55, 59]. Approximation algorithms with approximation ratio better than 2 have been discovered for TAP in a long line of research spanning three decades [1, 19, 20, 23, 24, 26, 29, 33, 36, 37, 49, 59, 64, 65, 67, 70, 84], and more recently for CacAP [15, 19, 73, 84].

The above problems naturally extend to their node-connected variants. However, approximation results on Node-Connectivity Augmentation are more scarce. This problem is also defined in Chapter 1, again we state it here for completeness.

Definition 3.2 (k-Node-CAP)**.** *Let $G = (V, E)$ be a k-node-connected graph, and $L \subseteq \binom{V}{2}$. The goal of k-Node-CAP is to compute a minimum cardinality set $L' \subseteq L$ such that $G' = (V, E \cup L')$ is a $(k+1)$-node-connected graph.*

In this chapter, we are interested in k-Node-CAP, for $k = 1$. Unlike the case for k-Edge-CAP, even in the most basic case of the problem when the given input is a tree, there are fewer approximation results developed in the literature, and mostly concentrate on small values of k. A 2-approximation was known for 1-Node-CAP [37, 59]. A 2-approximation is also known for $k = 2$ [5], which can be improved in the unweighted setting when the input graph is a cycle [42]. For any fixed k, the weighted version of the problem admits a $(4+\varepsilon)$-approximation if n is large enough [72] (see also [28]). We call the most basic case, when $k = 1$, and the input is a tree *Block-TAP* and define it here.

Definition 3.3 (Block-TAP)**.** *Let $T = (V, E)$ be a tree, and $L \subseteq \binom{V}{2}$. The goal of Block-TAP is to compute a minimum cardinality set $L' \subseteq L$ such that $G = (V, E \cup L')$ is a 2-node-connected graph.*

Very recently, Nutov [71] observed that Block-TAP can be reduced to special instances of the (unweighted) Node-Steiner Tree problem, extending the techniques of Basavaraju et al. [9] and Byrka et al. [15] used for CacAP, thus

[1] A cactus is a connected graph where every edge is part of exactly one cycle.

[2] For that, we can simply replace each edge with two parallel copies in the input tree of a TAP instance and the resulting instance is equivalent to a CacAP instance.

resolving an open question whether an approximation algorithm with ratio better than 2 exists for this case.

We recall that the (unweighted) Node-Steiner Tree problem takes as input a graph G and a subset of nodes (called terminals), and asks to find a tree spanning the terminals which minimizes the number of non-terminal nodes included in the tree. These special Node-Steiner Tree instances exhibit some crucial properties, similar to the Steiner Tree instances constructed by [15] for CacAP, and hence the 1.91-approximation of [15] also yields a 1.91-approximation to Block-TAP. This is the first result breaking the barrier of 2 on its approximability, and the best bound known until the results presented in this thesis.

Our results and techniques. As stated, in this chapter we consider the problem of 1-Node-CAP and we give an improved 1.8596-approximation. Note that Block-TAP is the special case of 1-Node-CAP when the input graph is a tree[3].

Our first step will be to observe that instances of the more general 1-Node-CAP can be reduced to the previously mentioned Node-Steiner-Tree instances, which we call here *CA-Node-Steiner-Trees*. A defining characteristic of CA-Node-Steiner-Trees is that every Steiner node is adjacent to at most 2 terminals. We clarify that these instances, obtained by the reduction referred to above (which we lay out in detail in Section 3.1), can also be treated as Edge-Steiner Tree instances (as done by [15]) but we will instead view them as Node-Steiner Tree instances, since this allows for a more direct correspondence between "links to add" for 1-Node-CAP/Block-TAP/CacAP, and "Steiner nodes to select" for Steiner Tree. This view helps us to give a cleaner analysis of the iterative randomized rounding technique. Thus, the main task for finding an approximation for 1-Node-CAP/Block-TAP/CacAP is to design an approximation for CA-Node-Steiner-Tree.

Besides giving a 1.8596-approximation for CA-Node-Steiner-Tree (hence 1-Node-CAP), which improves upon the approximation of 1.91 by Nutov [71] for Block-TAP, our work gives some new insights on the iterative randomized rounding method introduced by Byrka et al. [16] that might be of independent interest. We give a few more details of this iterative rounding next.

The iterative randomized rounding technique, applied to CA-Node-Steiner-Tree, at each iteration uses an (approximate) LP-relaxation to sample a set of

[3] It is also stated in [71] that 1-Node-CAP and Block-TAP are equivalent by relying on constructing the so-called block-cut tree of a given graph. While this is immediate if one allows *weights* for the links (see also [37, 59]), the same reduction does not seem to hold in the unweighted setting. We refer to Section 3.1.4 for more details.

Steiner nodes connecting part of the terminals, contract them, and iterate until all terminals are connected. Roughly speaking, the heart of the analysis lies in bounding the expected number of iterations of the algorithm until a Steiner node of a given initial optimal solution is not needed anymore in the current (contracted) instance. This is achieved by a suitably chosen spanning tree on the set of terminals called the *witness tree*, which we define below. In the original Steiner Tree work given by [16], as well as in the work by [15], the witness tree is chosen (mostly) randomly. Later an edge-deletion process over this tree is mapped to an edge-deletion process over the edges of an optimal solution.

In contrast, we give a *purely deterministic* way to construct the witness tree, and then map an edge-deletion process over this tree to a node-deletion process over the Steiner nodes of an optimal solution. The deterministic method of constructing the witness tree that we introduce here relies on iteratively computing some *minimum weight paths* from the Steiner nodes in an optimal solution to terminals. More in detail, we first define the following problem that will be critical in our analysis.

Definition 3.4 (Node Witness Tree Problem)**.** *Given is a tree $T = (V, E)$. We denote by R the set of leaves of T, and $S = V \setminus R$. The goal of the Node Witness Tree Problem is to find a tree $W = (R, E_W)$, where $E_W \subseteq R \times R$, which minimizes the non-linear objective function $\nu_T(W) = \frac{1}{|S|} \sum_{v \in S} H_{w(v)}$, where $w : S \to \mathbb{Z}_{\geq 0}$ is defined as*

$$w(v) := |\{pq \in E_W : v \text{ is an internal node of the } p\text{-}q \text{ path in } T\}|$$

and H_ℓ denotes the ℓ^{th} harmonic number ($H_\ell = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\ell}$).

Given a CA-Node-Steiner-Tree instance $(G = (V, E), R)$, we can define the following:

$$\psi_{(G,R)} := \min_{\substack{T^* = (R \cup S^*, E^*):\ T^* \text{ is} \\ \text{optimal Steiner tree of } (G,R)}} \ \min_{\substack{W:\ W \text{ is a} \\ \text{witness tree} \\ \text{of } T^*}} \nu_{T^*}(W),$$

We also define the constant ψ:

$$\psi := \sup\{\psi_{(G,R)} : G \text{ is an instance of CA-Node-Steiner-Tree}\}.$$

An important tool of this chapter is the following theorem, that shows that the problem of finding an approximation for CA-Node-Steiner Tree (and hence, for 1-Node-CAP, Block-TAP, and CacAP), reduces to the problem of minimizing ψ.

Theorem 3.1. *For any $\varepsilon > 0$, there is a $(\psi + \varepsilon)$-approximation algorithm for CA-Node-Steiner Tree (and hence, for 1-Node-CAP, Block-TAP, and CacAP).*

Using Theorem 3.1, we can prove the following theorem, which is the main result of this chapter, which we prove in Section 3.3.

Theorem 3.2. $\psi \leq 1.8596$. *That is, there is a 1.8596-approximation algorithm for CA-Node-Steiner-Tree (and hence, for 1-Node-CAP, Block-TAP, and CacAP).*

Before we prove Theorem 3.2, we will first provide a warmup 1.8917-approximation algorithm in Section 3.2.

Finally, in Section 3.4, we also provide some concrete limits on how much our techniques can be further pushed. In Section 3.4.2, we show that our techniques can be refined and give a $(1.8\bar{3} + \varepsilon)$-approximation algorithm for what we call leaf-adjacent Block-TAP instances; these are Block-TAP instances where at least one endpoint of each link is a leaf. We note here that Nutov [73] recently gave a $1.6\bar{6}$-approximation for leaf-to-leaf Block-TAP instances; these are instances where both endpoints of each link are leaves. Thus, our $(1.8\bar{3} + \varepsilon)$-approximation algorithm deals with a strictly larger set of instances compared to [73], albeit with a worse approximation factor.

Theorem 3.3. *For any fixed $\varepsilon > 0$, there exists a $(1.8\bar{3}+\varepsilon)$-approximation algorithm for leaf-adjacent Block-TAP.*

In Section 3.4.1, we show that our approximation factor of $1.8\bar{3}$ for leaf-adjacent Block-TAP instances is tight. Then in Section 3.4.3 that there is a CA-Node-Steiner tree instance with an even larger lower bound.

Theorem 3.4. *For any $\varepsilon > 0$, there exists a CA-Node-Steiner-Tree instance $(G_\varepsilon, R_\varepsilon)$ such that $\psi_{(G_\varepsilon,R_\varepsilon)} > 1.841\bar{6} - \varepsilon$.*

This shows that any approximation bound better than $1.841\bar{6}$ for the general CA-Node-Steiner-Tree problem needs substantially different arguments.

3.1 Reduction from Connectivity Augmentation to Node-Steiner Tree

3.1.1 Reduction to CA-Node-Steiner-Tree instances

As a reminder, in the Node-Steiner Tree problem, we are given a graph $G = (V, E)$ and a subset $R \subseteq V$ of nodes, called *terminals*, and the goal is to compute a tree

T of G that contains all the terminals and minimizes the number of non-terminal nodes (so-called *Steiner nodes*) contained in T. We will refer to the number of Steiner nodes contained in a tree T as the *cost* of the solution, and indicate it with $cost(T)$.

In general, the Node-Steiner Tree problem is as hard to approximate as the Set Cover problem, and it admits a $O(\log|R|)$ approximation algorithm (which holds even in the more general weighted version [60]). However, the instances that arise from the reductions of [9, 15, 71] have special properties that allow for a constant factor approximation. We defined these instances in Chapter 2, and state them again here for completeness.

Definition 3.5 (CA-Node-Steiner-Tree)**.** *Let $G = (V, E)$ be an instance of Node-Steiner Tree, with $R \subseteq V$ being the set of terminals. The instance G is a **CA-Node-Steiner-Tree** instance if, for each Steiner node $\ell \in V \setminus R$, we have $|N_G(\ell) \cap R| \leq 2$.*

As already mentioned, the starting point of this work is the reduction from CacAP and Block-TAP to CA-Node-Steiner-Tree. An overview of the reduction from CacAP to CA-Node-Steiner-Tree by Byrka et al. [15] is found in the proof of Theorem 3.5. In Section 3.1.4 we extend the reduction from Block-TAP to CA-Node-Steiner-Tree by Nutov [71], to deal with 1-Node-CAP instances. The following two theorems make these connections formal.

Theorem 3.5 ([15])**.** *The existence of an α-approximation algorithm for CA-Node-Steiner-Tree implies the existence of an α-approximation algorithm for CacAP.*

The proof of this theorem comes from [15], we repeat it here for clarity and completeness, as the reduction is useful to have laid out.

Proof. Let $(G = (V, E), L)$ be a given CacAP instance, where $L \subseteq \binom{V}{2}$. For every link $\ell = (v_0, v_{n+1})$ let $v_1, \ldots, v_n$ be the sequence of nodes of degree at least 4 that belong to every simple path from v_0 to v_{n+1} in G, such that each pair $\ell_i = \{v_i, v_{i+1}\}$ lies on the same cycle C_i in G. We call each ℓ_i a *projection* of ℓ on C_i, and the set of projections of ℓ is denoted by $\text{proj}(\ell)$. Projections $\ell_i = (v, u)$ and $\ell'_{i'} = (v', u')$ of $\ell \in L$ and $\ell' \in L$, respectively, are said to *cross* if they satisfy one of the following conditions: they share an endpoint, or they have both endpoints on the same cycle C and one of the two simple v to u paths in C contain exactly one of u' or v'. Links ℓ and ℓ' are said to *cross* if they have projections $\ell_i \in \text{proj}(\ell)$ and $\ell'_{i'} \in \text{proj}(\ell')$ that cross.

From (G, L) we construct a Node-Steiner Tree instance $G_{ST} = (R \cup L, E_{ST})$ in the following way. The set of links L make up the Steiner nodes of G_{ST} and the

set of degree-2 nodes in G make up the set of terminals R. For each link $\ell \in L$ with endpoint $r \in R$, we have $(\ell, r) \in E_{ST}$, and for each pair of links ℓ and ℓ' that cross, we have $(\ell, \ell') \in E_{ST}$. Notice that G_{ST} is, in particular, a CA-Node-Steiner-Tree instance.

A key lemma that shows why the above reduction is useful is the following; its proof is given in [9, 15].

Lemma 3.1 ([9, 15])**.** *Let (G, L) be a CacAP instance, and let $G_{ST} = (R \cup L, E_{ST})$ be the corresponding CA-Node-Steiner-Tree instance. A subset of links $L' \subseteq L$ is a feasible solution of (G, L) if and only if $G_{ST}[R \cup L']$ is connected.*

Clearly, a feasible solution L' to a CacAP instance (G, L) has size $|L'|$ and implies a feasible solution to the corresponding CA-Node-Steiner-Tree instance $G_{ST} = (T \cup L, E_{ST})$ of size $|L'|$ and vice versa, so we can conclude Theorem 3.5. □

Following Nutov [71], the following theorem is proven in Section 3.1.4

Theorem 3.6 (Extending [71])**.** *The existence of an α-approximation algorithm for CA-Node-Steiner-Tree implies the existence of an α-approximation algorithm for 1-Node-CAP (and hence Block-TAP).*

The above two theorems imply that from now on we can therefore concentrate on the CA-Node-Steiner-Tree problem.

3.1.2 An approximate relaxation for CA-Node-Steiner-Tree

A crucial property we will use is that, similar to the Edge-Steiner Tree problem, one can show the *k-restricted* version of the problem provides a $(1+\varepsilon)$-approximate solution to the original problem, for an arbitrarily small $\varepsilon > 0$.

To explain this in more detail, we first recall that any Steiner tree can be seen as the union of components, where a component is a subtree whose leaves are all terminals, and whose internal nodes are all Steiner nodes. We note that two components of such a union are allowed to share nodes and edges. A component is called k-restricted if it has at most k terminals. A k-restricted Steiner tree is a collection of k-restricted components which induces a connected hypergraph when taking R as vertex set, and adding for each component the set of its terminals as a hyperedge. The key theorem of this section is the following.

Theorem 3.7. *Consider a CA-Node-Steiner-Tree instance and let OPT be its optimal value. For any integer $m \geq 1$, there exists a k-restricted Steiner tree $Q(k)$, for $k = 2^m$, whose cost satisfies $cost(Q(k)) \leq \left(1 + \frac{4}{\log k}\right) OPT$.*

We stress that $cost(Q(k))$ is equal to the number of Steiner nodes in $Q(k)$ counted with multiplicities; an example demonstrating this is given in Figure 3.1.

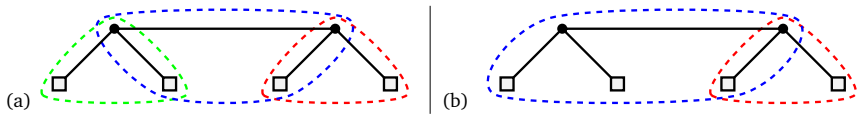


Figure 3.1: (a) An example of a 2-restricted CA-Node-Steiner-Tree, of cost 4. (b) An example of a 3-restricted CA-Node-Steiner-Tree, of cost 3. In both figures, the square nodes represent the terminals and the components are the nodes grouped together by the dashed lines.

The proof of the Theorem 3.7 mimics the result of Borchers and Du [13] regarding the Edge-Steiner Tree problem, and can be found in Section 3.1.5. In proving Theorem 3.7, we crucially use the properties stated in Definition 3.5, as the statement is not true for general Node-Steiner Tree instances.

The above theorem shows that CA-Node-Steiner-Tree can be approximated using k-restricted Steiner trees, with a small loss in the objective value. Based on this, we now present a linear programming relaxation for the k-restricted Node-Steiner Tree problem, which we call the Directed Components Relaxation (or, in short, DCR), as it mimics the DCR relaxation of Byrka et al. [16] for the Edge-Steiner Tree problem.

Let $G = (V, E)$ be a Node-Steiner Tree instance, where $R \subseteq V$ is the set of terminals. Let $k > 0$ be an integer parameter. For each subset $R' \subseteq R$ of terminals of size at most k, and for every $c' \in R'$, we define a directed component C' as the minimum Node Steiner tree on terminals R' with edges directed towards c'. More precisely, let $S(R') \subseteq V \setminus R$ be a minimum-cardinality set of Steiner nodes such that $G[R' \cup S(R')]$ is connected. We define C' by taking a spanning tree in $G[R' \cup S(R')]$ and directing the edges towards c', and we let $cost(C') = |S(R')|$. We call c' the *sink* of the component and all other terminals in $R' \setminus \{c'\}$ the *sources* of the component. Let $\mathbf{C}$ be the set of all such directed components C', with all possible sinks $c' \in C'$. Note that $\mathbf{C}$ has an element for each subset of terminals $R' \subseteq R$, $|R'| \leq k$, and each terminal $c' \in R'$. We introduce one variable $x(C')$ for each $C' \in \mathbf{C}$. Finally, we choose an arbitrary terminal $r \in R$ as the root. The

k-DCR relaxation is the following.

$$\begin{aligned}
\min\colon \quad & \sum_{C' \in \mathbf{C}} cost(C') \cdot x(C') \\
\text{s.t.:} \quad & \sum_{\substack{C' \in \mathbf{C}\colon \mathrm{sink}(C') \notin U \\ \text{and } \mathrm{sources}(C') \cap U \neq \emptyset}} x(C') \geq 1, \qquad \forall U \subseteq R \setminus \{r\}, U \neq \emptyset, \\
& x(C') \geq 0, \qquad \forall C' \in \mathbf{C}. \qquad (k\text{-DCR})
\end{aligned}$$

Let $OPT_{LP}(k)$ be the optimal value of the above relaxation for $k \in \mathbb{N}_{>0}$, and OPT be the value of an optimal (integral) solution to our original (unrestricted) CA-Node-Steiner-Tree instance. The next theorem states that the k-DCR LP can be solved in polynomial time for any fixed k and its optimal solution yields a $(1+\varepsilon)$-approximation of OPT.

Theorem 3.8. *For any fixed $\varepsilon > 0$, there exists a $k = k(\varepsilon) > 0$ such that $OPT_{LP}(k) \leq (1+\varepsilon)OPT$, and moreover, an optimal solution to the k-DCR LP can be computed in polynomial time.*

Proof. Given an $\varepsilon > 0$, we set $k = 2^{\lceil 4/\varepsilon \rceil}$. Since k is fixed, computing a minimum Node Steiner tree connecting k terminals can be done in polynomial time. To see this, we note that the Node-Steiner Tree problem can be reduced to the Directed Edge-Steiner Tree problem without modifying the number k of terminals (see [82]). Thus, one can use existing algorithms for optimally solving Directed Edge-Steiner Tree instances with a fixed number of terminals (see, e.g., [34]). We conclude that the set of components $\mathbf{C}$ can be computed in polynomial time, and thus we can generate the variables of k-DCR LP in polynomial time. Although the set of constraints is exponential in $|R|$, an optimal solution for it can be computed in polynomial time by standard flow techniques (see [Lemma 8, [15]]).

Regarding the LP value, it is easy to see that $OPT_{LP}(|R|) \leq OPT$, as the entire optimal tree can be viewed as an $|R|$-restricted Steiner Tree which can be directed towards r, and whose cost is equal to OPT. We now argue that $OPT_{LP}(k)$ is at most $\left(1 + \frac{4}{\log k}\right) OPT_{LP}(|R|)$. For this, let x be an arbitrary feasible solution of the $|R|$-DCR LP. We look at every component Q with terminals R' and sink $c' \in R'$ whose x-value is strictly positive, i.e., $x(Q) > 0$, and we do the following: if the component is a k-restricted component, we set $x'(Q) = x(Q)$. Otherwise, by Theorem 3.7, we compute a k-restricted Steiner Tree Q' with components $Q'_1, \ldots, Q'_j$, each containing at most k terminals, such

that $cost(Q') = \sum_{i=1}^{j} cost(Q'_i) \leq \left(1 + \frac{4}{\log k}\right) cost(Q)$. Moreover, we "direct" the components consistently towards the sink c' of Q, and increase the corresponding x'-variables by $x(Q')$ for each such directed component. The resulting vector x' is a feasible solution for k-DCR, whose objective function value is at most $\left(1 + \frac{4}{\log k}\right)$ times the objective function value of x. We therefore get $OPT_{LP}(k) \leq \left(1 + \frac{4}{\log k}\right) OPT_{LP}(|R|) \leq (1+\varepsilon)OPT$. □

Given the above theorems, all that remains to show is the following: given an optimal fractional solution to the k-DCR LP, design a rounding procedure that returns an integral CA-Node-Steiner-Tree solution whose cost is at most ψ times larger than the cost of an optimal k-restricted Steiner tree, for as small $\psi \geq 1$ as possible. Following the chain of reductions discussed in this section, such a rounding scheme would immediately imply a $(\psi + \varepsilon)$-approximation algorithm for 1-Node-CAP and CacAP, given that ψ is constant. In Section 3.2, we present the first such scheme that gives $\psi \leq 1.8917$. In Section 3.3, we build on these ideas and present a more complex scheme that gives $\psi \leq 1.8596$.

3.1.3 The iterative randomized rounding algorithm

Our rounding scheme for the k-DCR relaxation consists in applying the iterative randomized rounding technique of Byrka et al. [16], first applied to the Edge Steiner Tree problem, and is described in Algorithm 1.

Algorithm 1: The iterative randomized rounding scheme for the k-DCR LP

```
for i = 1,2,3,... do
    Compute an optimal solution x^i to the k-DCR LP w.r.t. the current instance;
    Sample a component C^i from the set C^i of current components, where C^i = C with probability x^i_C / Σ_{C'∈C^i} x^i_{C'}, and contract C^i into its sink node;
    If a single terminal remains, return the sampled components ⋃_{j=1}^{i} C^j;
end
```

We will now analyze the cost of the output solution. Throughout this section,

we follow the analysis of [16] and slightly modify it wherever needed. Let $G = (V, E)$ be a given CA-Node-Steiner-Tree instance, where $V = R \cup S$; R is the set of terminals and S is the set of Steiner nodes. Let $\varepsilon > 0$ be a fixed small constant and $k = k(\varepsilon)$ as in the proof of Theorem 3.8. For the sake of the analysis, we can assume that $\sum_{C' \in \mathbf{C}^i} x^i_{C'}$ is the same for all iterations i, as in [16][4]. So let $M = \sum_{C' \in \mathbf{C}^i} x^i_{C'}$.

Let $T = (R \cup S^*, E^*)$, $S^* \subseteq S$ and $E^* \subseteq E$, be an optimal solution to the given CA-Node-Steiner-Tree instance G of cost $cost(T) = |S^*|$. We will analyze the cost of the output of Algorithm 1 with respect to T. For that, we define a sequence of subgraphs of T, one for each iteration of the algorithm, in the following way: if, during the i^{th} iteration, we sample C^i, then we delete a subset of nodes of T and what remains is the subgraph T^i defined for that iteration. What must be specified is which nodes of T are deleted at each iteration, which will be explained shortly. Let $T = T^0 \supseteq T^1 \supseteq T^2 \supseteq \ldots$ be sequence of subgraphs remaining after each iteration, where the notation $T^{i+1} \sqsubseteq T^i$ means that T^{i+1} is a (not necessarily strict) subgraph of T^i. We show that there exists some universal constant $\psi \in \mathbb{R}_{\geq 1}$ and a choice of optimal solution T such that the following two properties are satisfied:

1. T^i plus the components sampled until iteration i form a connected subgraph spanning the terminals.

2. On average, a Steiner node in T is deleted after $M\psi$ iterations.

Similar to [16], we show that conditions (1) and (2) yield that the iterative randomized rounding algorithm is in fact a $(\psi + \varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$. This is achieved by relying on the construction of a *witness tree* W, which is a particular kind of spanning tree on the set of terminals. However, the main differences with respect to [16] are that (i) we delete *nodes* of T instead of edges, and (ii) we have a *purely deterministic* way to construct the witness tree, and thus we need an explicit averaging argument for (b).

We now discuss the details of our deletion process. As mentioned, given T we construct a witness tree $W = (R, E_W)$ that spans the set of terminals. For each component C, let $\mathscr{B}_W(C)$ be the family of maximal edge sets $B \subseteq E_W$ such that $(W \setminus B) \cup C$ forms a connected subgraph spanning the terminals. In each iteration

[4] Roughly speaking, in order to ensure that the sum is the same during all iterations, we can slightly modify the algorithm and add a dummy component of zero cost whose x-value is set in such a way so that the summation is equal to some $M > 0$. A coupling argument shows that all expected properties of the output are the same for both versions of the algorithm. See [page 12, [16]] for more details.

i, we "mark" a subset of edges of W that correspond to a randomly chosen set in $\mathscr{B}_W(C^i)$. For a positive integer t, let $H_t \coloneqq \sum_{j=1}^{t} \frac{1}{j}$ be the t^{th} harmonic number. The following lemma is proved in [16] (more precisely, see Lemmas 19 and 20 in [16]).

Lemma 3.2 ([16]). *For each component C, there exists a probability distribution over $\mathscr{B}_W(C)$ such that the following holds: for any $\widetilde{W} \subseteq E_W$, the expected number of iterations until all edges of $\widetilde{W}$ are marked is bounded by $H_{|\widetilde{W}|} \cdot M$, where the expectation is over the random choices of the algorithm and the distributions over $\{\mathscr{B}_W(C)\}_C$.*

We will delete Steiner nodes from T using the marked edges in W, as follows. For each Steiner node $v \in S^*$, we define

$$W(v) \coloneqq \{(p,q) \in E_W : v \text{ is internal node of the } p\text{-}q \text{ path in } T\},$$

and $w(v) \coloneqq |W(v)|$; less formally, $w(v)$ is equal to the number of edges e of W such that the path between the endpoints of e in T contains v. From now on, we will say that the vector $w : S^* \to \mathbb{N}_{\geq 0}$ is the vector *imposed* on S^* by W. Note that W combined with the vector w imposed on S^* correspond to a Node Witness Tree problem over the tree T.

A Steiner node $v \in S^*$ is deleted in the first iteration where all the edges in $W(v)$ become marked. Thus, each subgraph T^i, $i > 0$, defined in the beginning of this section, is the subgraph obtained from T^{i-1} after the i^{th} iteration according to this deletion process.

Lemma 3.3. *For every $i > 0$, $T^i \cup C^1 \cup \cdots \cup C^i$ spans the terminals.*

Proof. Let $E^i_W \subseteq E_W$ be the set of edges of W that have not been marked at iteration i. By construction, $E^i_W \cup C^1 \cup \cdots \cup C^i$ spans the terminals. For every $(p,q) \in E^i_W$, the unique path between p and q in T is still present in T^i, as none of the nodes in this path have been deleted. Hence p and q are connected in T^i. The result follows. □

We now recall the universal constant ψ. For a CA-Node-Steiner-Tree instance $G = (V, E)$, where $R \subseteq V$ is the set of terminals and $S = V \setminus R$ is the set of Steiner nodes, we define

$$\psi_G \coloneqq \min_{\substack{T=(R\cup S^*, E^*):\, T \text{ is} \\ \text{optimal Steiner tree}}} \; \min_{\substack{W:\, W \text{ is} \\ \text{witness tree}}} \nu_W(T)$$

where w is the vector imposed on S^* by each witness tree W considered. The constant ψ that will be used to pinpoint the approximation ratio of the algorithm is defined as

$$\psi := \sup\{\psi_G : G \text{ is an instance of CA-Node-Steiner-Tree}\}.$$

In Sections 3.2 we will prove the following weaker theorem as an introduction to the techniques employed in Section 3.3.

Theorem 3.9. $\psi \leq 1.8917$.

We will then prove the following theorem, in Section 3.3, which is the best known upper bound for ψ.

Theorem 3.10. $\psi \leq 1.8596$.

We are ready to prove the main theorem of this section; throughout its proof, we use the notation introduced in this section.

Theorem 3.11. *For any fixed $\varepsilon > 0$, the iterative randomized rounding algorithm (see Algorithm 1) yields a $(\psi+\varepsilon)$-approximation.*

Proof. Let $G = (R \cup S, E)$ be a CA-Node-Steiner-Tree instance. We run Algorithm 1 with $k = 2^{\lceil 4/\varepsilon' \rceil}$, as in the proof of Theorem 3.8, and $\varepsilon' := \frac{\varepsilon}{2}$. It is easy to see that the algorithm runs in polynomial time.

Let $T = (R \cup S^*)$ and W be the optimal Steiner tree and corresponding witness tree, respectively, such that $\psi_G = \nu_T(W)$. Clearly, $\psi_G \leq \psi$. For each Steiner node $v \in S^*$, let $D(v) := \max\{i : v \in T^i\}$. By Lemma 3.2, we have $\mathbf{E}[D(v)] \leq H_{w(v)} \cdot M$. We have

$$\begin{aligned}\sum_i \mathbf{E}[cost(C^i)] &= \sum_i \mathbf{E}\Big[\sum_C \frac{x_C^i}{M} cost(C)\Big] \leq \frac{1+\varepsilon'}{M} \sum_i \mathbf{E}[cost(T^i)] \\ &= \frac{1+\varepsilon'}{M} \sum_{v \in S^*} \mathbf{E}[D(v)] \leq (1+\varepsilon') \sum_{v \in S^*} H_{w(v)} \\ &= \psi_G \left(1 + \frac{\varepsilon}{2}\right) cost(T) \leq (\psi + \varepsilon) cost(T).\end{aligned}$$

The first inequality holds since (i) Lemma 3.3 implies that an optimal solution defined in iteration i to the CA-Node-Steiner-Tree instance defined in iteration i has cost at most $cost(T^i)$, and (ii) Theorem 3.8 shows that the k-DCR LP provides a $(1+\varepsilon')$-approximate solution to it, since it is a CA-Node-Steiner-Tree. The first inequality of the second line follows since $\sum_{v \in S^*} \mathbf{E}[D(v)] \leq M \cdot \sum_{v \in S^*} H_{w(v)}$, while the last inequality follows by the definition of ψ and Theorem 3.10. □

Putting everything together, we can now easily prove our main result, Theorem 3.2.

Proof of Theorem 3.2 . We first design a 1.8596-approximation algorithm for the CA-Node-Steiner-Tree problem. For that, we pick ε to be sufficiently small (e.g., $\varepsilon = 0.0003$ suffices) and by Theorems 3.10 and 3.11, we get that the iterative randomized rounding algorithm (Algorithm 1) is a 1.8596-approximation algorithm for CA-Node-Steiner-Tree. By Theorems 3.5 and 3.6, we also get a 1.8596-approximation algorithm for 1-Node-CAP and CacAP. This concludes the proof. □

3.1.4 Reduction from 1-Node-CAP to CA-Node-Steiner-Tree

In this section we slightly generalize the reduction from a Block-TAP instance to CA-Node-Steiner-Tree that was described by Nutov [71] (Section 3.1.4), and give a reduction from 1-Node-CAP to CA-Node-Steiner-Tree that shows that an α-approximation for CA-Node-Steiner-Tree implies an α-approximation for 1-Node-CAP (Section 3.1.4). The reduction has two steps. We first reduce the original problem to a weighted Block-TAP instance with specific properties, where each link has weight either 0 or 1. We then show that Nutov's reduction [71] still applies, and it gives a final CA-Node-Steiner-Tree instance that is unweighted.

The reduction to Block-TAP with $\{0,1\}$ weights

In this section, we use one of the standard ways to reduce a 1-Node-CAP instance to a Block-TAP instance. Let $G = (V, E)$ be a connected graph, and let $L \subseteq \binom{V}{2}$ be a set of links. Let $C \subseteq V$ be the set of cut nodes of G. As a reminder, a node $c \in V$ is a cut node of G if $G \setminus \{c\}$ is not connected. Note that if $C = \emptyset$, then G is already 2-node-connected, and so the problem is trivial. So, from now on we assume that $C \neq \emptyset$.

We start by computing the block-cut tree $T = (V_T, E_T)$ of G. It is well known that the block-cut tree of a graph can be computed in polynomial time. T contains one copy of each cut node of G and one node for each maximal 2-node connected component of G. Let $C = \{c_1, \ldots, c_n\} \subseteq V$ be the set of cut nodes of G, and let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be the collection of maximal 2-node connected components. We have $V_T = C \cup \mathcal{B}$. The edge set E_T is defined as follows: for every $i \in [n], j \in [m]$, $(c_i, B_j) \in E_T$ if and only if $c_i \in B_j$. It is easy to verify that the resulting graph T is indeed a tree.

Along with a block-cut tree T of a graph G, we also define a function f_G that maps the nodes of G to the nodes of T. More precisely, we define a map $f_G : V \to V_T$ as follows: $f_G(c_i) = c_i$, for every $i \in [n]$, and $f_G(v) = B_j$ for every $j \in [m]$, $v \in B_j \setminus C$. Note that f_G is well-defined, i.e., for every $v \in V \setminus C$, there exists exactly one block B_j such that $v \in B_j$. We also extend the map f_G and define $f_G(\ell) := (f_G(u), f_G(v))$, for any $\ell = (u, v) \in L$. Finally, the set of links $L_T \subseteq \binom{V_T}{2}$ is defined as follows:

- for each link $\ell = (u, v) \in L$ there is a link $\ell_T := f_G(\ell) \in L_T$ of weight 1. Let $f_G(L)$ denote the set of all such links.
- for each block B_j, $j \in [m]$, and every pair of cut-vertices $c, c' \in B_j \cap C$, where $c \neq c'$, there is a link $(c, c') \in L_T$ of weight 0. Let L_0 denote the set of all the links of this form.

Throughout the rest of this section, we will use the notation introduced in these last two paragraphs. The following observation is now easy to verify.

Observation 3.1. *Let $G = (V, E)$ be a connected graph and let $T = (V_T, E_T)$ be a block-cut tree of G. Let $f_G : V \to V_T$ be the associated map. Let c be a cut-vertex of G, and let $G_1, \ldots, G_k$ be the connected components of $G \setminus \{c\}$, where $k \in \mathbb{N}_{\geq 2}$. Let $T_1, \ldots, T_{k'}$ be the connected components of $T \setminus \{c\}$. Let $V_i \subseteq V$ be the set of nodes contained in the blocks that are nodes of T_i, for every $i \in [k']$. Then, the following hold:*

1. *$k = k'$,*
2. *there is a bijection $g_c : [k] \to [k]$, such that $V_{g_c(i)} = V(G_i) \cup \{c\}$ for every $i \in [k]$, where $V(G_i)$ is the set of nodes contained in G_i,*
3. *$V_i \cap V_j = \{c\}$ for every $i \neq j \in [k]$.*

We will now show that the two instances are equivalent. More precisely, we prove the following lemma.

Lemma 3.4. *Let $G = (V, E)$ be a connected graph, and L a set of links in G. Let $T = (V_T, E_T)$ be the corresponding block-cut tree of G, let f_G be the associated map between V and V_T, and let $L_T = f_G(L) \cup L_0$ be the corresponding set of links. Let $L' \subseteq L(G)$. Then, $G \cup L'$ is 2-node-connected if and only if $T \cup f_G(L') \cup L_0$ is 2-node-connected.*

Before proving the lemma, we note that we might have two (or more) links $\ell \neq \ell' \in L$ being mapped, via f_G, to the same $\bar{\ell}$ link in T; in other words, $f_G^{-1}(\bar{\ell})$ might be a set with more than one link. In such a case, we map back $\bar{\ell}$ to any link of $f_G^{-1}(\bar{\ell})$ in an arbitrary way.

Proof of Lemma 3.4. Let $L' \subseteq L$ be a set of links such that the graph $G \cup L'$ is 2-node-connected. We will show that $T \cup f_G(L') \cup L_0$ is 2-node-connected. Suppose otherwise. This means that there exists a node $u \in V_T$ that, if removed, disconnects the tree. We observe that u must necessarily be a cut node, i.e., $u = c \in C$. To see this, note that in the graph $T \cup L_0$, all nodes that are adjacent to a node B_j, for any $j \in [m]$, form a clique, and thus, the removal of B_j cannot disconnect the graph $T \cup L_0(T)$. Since $u = c$ is a cut node, it means that $G \setminus \{c\}$ is not connected. By Observation 3.1, the connected components $G_1, \ldots, G_k$ of $G \setminus \{c\}$ correspond, one by one, to the components $T_1, \ldots, T_k$ of $T \setminus \{c\}$. This implies that any link $\ell \in L'$ connecting component G_i with component G_j, $i \neq j \in [k]$, corresponds to a link $f_G(\ell)$ connecting T_i with T_j. Thus, since $(G \cup L') \setminus \{c\}$ is connected, this means that $(T \cup f_G(L')) \setminus \{c\}$ is connected, and so we get a contradiction. We conclude that $T \cup f_G(L') \cup L_0(T)$ is 2-node-connected.

Conversely, let $L' \subseteq L$ such that $T \cup f_G(L') \cup L_0$ is 2-node-connected. We will show that $G \cup L'$ is 2-node-connected. Suppose otherwise. This means that there exists a cut node $c \in C$ such that $(G \cup L') \setminus \{c\}$ is not connected. In particular, there are at least 2 connected components D_1 and D_2 in $(G \cup L') \setminus \{c\}$, and these components are simply unions of the connected components $G_1, \ldots, G_k$ of $G \setminus \{c\}$. Since there are no links with c as an endpoint in the graph $(T \cup f_G(L') \cup L_0) \setminus \{c\}$, this means that the connected components of $(T \cup f_G(L') \cup L_0) \setminus \{c\}$ are exactly the same as the connected components of $(T \cup f_G(L')) \setminus \{c\}$. Moreover, by Observation 3.1, the components $G_1, \ldots, G_k$ correspond, one by one, to the components $T_1, \ldots, T_k$ of $T \setminus \{c\}$. Since $(T \cup f_G(L')) \setminus \{c\}$ is connected, this means that the set $f_G(L')$ contains links that do not have c as an endpoint which connect all of the components $T_1, \ldots, T_k$ to a single component; in particular, if $u \in D_1$ and $v \in D_2$, then there is a path between $f_G(u)$ and $f_G(v)$ in $(T \cup f_G(L')) \setminus \{c\}$. This means that D_1 and D_2 must be connected via L', and so we get a contradiction. We conclude that $G \cup L'$ is 2-node-connected. □

The reduction to CA-Node-Steiner-Tree

Given the Block-TAP instance $T = (V_T, E_T)$, as constructed in the previous section, we now modify Nutov's reduction [71] in order to reduce our weighted

Block-TAP instance to an unweighted CA-Node-Steiner-Tree instance. To simplify notation, we denote the set of links in T as $L = L_0 \cup L_1$, where L_0 is the set of links of weight 0 and L_1 is the set of links of weight 1. Let $R \subseteq E_T$ be the set of edges of T that have a leaf as an endpoint; these edges will correspond to terminals in the resulting CA-Node-Steiner-Tree instance. We define three graphs, with the last one being the final resulting CA-Node-Steiner-Tree instance.

Definition 3.6. 1. *The* (L, E_T)-incidence graph *is the graph with node set* $L \cup E_T$ *and edge set defined as follows: there is an edge between* $\ell = (u, v) \in L$ *and* $e \in E_T$ *if* e *belongs to the unique path in* T *connecting* u *and* v.

2. *The* short-cut (L, E_T)-incidence graph *is obtained from the* (L, E_T)*-incidence graph by short-cutting the set* E_T*, where short-cutting a node* $e \in E_T$ *means adding a clique between the neighbors of* e.

3. *The* reduced (L, E_T)-incidence graph *is obtained from the short-cut* (L, E_T)*-incidence graph as follows. For every* $\ell \neq \ell' \in L_1$ *such that there is a path between* ℓ *and* ℓ' *in the short-cut* (L, E_T)*-incidence graph whose nodes, other than the endpoints, all belong to* L_0*, we add an edge between* ℓ *and* ℓ'*. We then delete the set* $E_T \setminus R$ *and the set* L_0*. In this resulting instance, let* R *be the set of terminals and* L_1 *be the set of Steiner nodes.*

We stress that the final reduced (L, E_T)-incidence graph is treated as an unweighted Node Steiner Tree instance, i.e, all Steiner nodes have weight 1. For $J = T \cup L$, we define $\kappa_J(s, t)$ to be the maximum number of internally node disjoint paths from s to t in J. Given a block-cut tree $T = (V_T, E_T)$ with a set of links $L = L_0 \cup L_1$, let H be the (L, E_T)-incidence graph, and let H_R be the reduced (L, E_T)-incidence graph. For distinct vertices $s, t \in V_T$, we say (s, t) are *H-reachable* if H has a path connecting $e^{(s)}$ with $e^{(t)}$, where $e^{(s)}$ and $e^{(t)}$ are the edges of the unique path from s to t in T that are incident to s and t, respectively. We are now ready to prove the key lemma of this section; its proof closely follows [71].

Lemma 3.5. *Let* $T = (V_T, E_T)$ *be a block-cut tree with a set of links* $L = L_0 \cup L_1$*. Let* H *be the* (L, E_T)*-incidence graph. Let* $J = T \cup L$*. Then for* $s, t \in V_T$ *such that* $(s, t) \notin E_T$*,* $\kappa_J(s, t) \geq 2$ *if and only if* (s, t) *are* H*-reachable.*

Proof. Let $s \neq t \in V_T$ such that $(s, t) \notin E_T$, and P be the unique path between s and t in T. We treat P both as a set of edges and a set of vertices. Let $e^{(s)}$ and $e^{(t)}$ be the edges of the unique path from s to t in T that are incident to s and t, respectively. We will prove the statement by using induction on the number of edges in P.

Base case. The base case is when P has 2 edges. Let $P = (s, u, t)$. If $u = B_j$, for some $j \in [m]$, then s and t are necessarily cut nodes that belong to the same block B_j. In this case, we have $\kappa_{T \cup L_0}(s, t) \geq 2$, and by construction, we also have the path $(e^{(s)}, \ell, e^{(t)})$ in H, where $\ell = (e^{(s)}, e^{(t)}) \in L_0$. Thus, in this case the statement trivially holds.

So, suppose now that $u = c \in C$. In this case, we have $s = B_i$ and $t = B_j$, $i \neq j$, such that $c \in B_i \cap B_j$. Suppose that $\kappa_J(s, t) \geq 2$. Consider $T \setminus \{c\}$ and let $T^{(s)}, T^{(t)}$ be the components of $T \setminus \{c\}$ that contain s and t, respectively. Then, $J \setminus \{c\}$ has a path from s to t. Consider such a path Q, and suppose that Q goes through each of the components $T^{(s)} = T_0, T_1, \ldots, T_{l-1}, T_l = T^{(t)} \subset T \setminus \{c\}$ exactly once, where $l \in \mathbb{N}_{\geq 1}$; without loss of generality the components are labelled in the order Q visits them. It is easy to see that we must have a link $\ell_x \in L_1$ between T_x and T_{x+1}, for every $x \in \{0, 1, \ldots, l-1\}$. Let e_x denote the edge in T that is adjacent to c and has an endpoint in T_x, for $x \in \{0, 1, \ldots, \ell\}$. Note that we have $e_0 = e^{(s)}$ and $e_l = e^{(t)}$. It is not hard to see that we have the following path between $e^{(s)}$ and $e^{(t)}$ in H: $e^{(s)}, \ell_0, e_1, \ell_1, \ldots, \ell_{l-1}, e^{(t)}$. We conclude that (s, t) are H-reachable.

Suppose now that $\kappa_J(s, t) = 1$. This implies that $J \setminus \{c\}$ has no path between s and t. Let $J^{(s)}$ be the component of $J \setminus \{c\}$ that contains s. Let $L^{(s)}$ and $L^{(t)}$ be the set of links that have both endpoints in $J^{(s)} \cup \{c\}$ and $V \setminus J^{(s)}$, respectively. Since there is no path between s and t in $J \setminus \{c\}$, it is clear that $L^{(s)}$ and $L^{(t)}$ partition L. In particular, this means that for any link $\ell \in L^{(s)}$, the path in T connecting the endpoints of ℓ shares no edges with the corresponding path for any link in $L^{(t)}$. Thus, there is no path between $e^{(s)}$ and $e^{(t)}$ in H, and the claim holds.

Induction step. We now suppose that P has at least 3 edges, and that the claim holds for all paths whose number of edges is strictly smaller than the number of edges in P. Let (u, v) and $(v, t) = e^{(t)}$ be the last two edges in P. In order to prove that the claim holds, we first prove several other facts we will need. From now on, let P_{xy} denote the subpath of P with endpoints $x \in V_T$ and $y \in V_T$.

Fact 3.1. *If $\kappa_J(s, t) \geq 2$, then $\kappa_J(s, v) \geq 2$.*

Proof. Suppose that $\kappa_J(s, t) \geq 2$, and let Z be the cycle in J formed by taking the union of two node disjoint paths from s to t in J. Obviously, if v is part of Z, we have two internally disjoint paths from s to v in J, and in this case the claim holds. If v is not part of Z, then let a be the node in P_{sv} such that the path P_{av} only shares a single node with the cycle Z; clearly, such a node a always exists, as s belongs both to Z and P_{sv}. We claim that the set of edges $Z \cup P_{av} \cup \{(v, t)\}$

contains two internally disjoint paths from s to v. In particular, the first path is the union of the shortest path P' in Z from s to a along with P_{av}, while the second path is a subset of the edges $Z \setminus P'$ along with the edge (v, t). □

Fact 3.2. *If (s, v) and (u, t) are both H-reachable, then (s, t) are H-reachable.*

Proof. By assumption, there is a path between $e^{(s)}$ and $(u, v) \in E_T$ in H, and a path between $(u, v) \in E_T$ and $(v, t) = e^{(t)} \in E_T$. Combining these paths gives the result. □

Fact 3.3. *If (s, t) are H-reachable, then (s, v) are H-reachable.*

Proof. Suppose Q is the path between $e^{(s)}$ and $e^{(t)}$ in H. First, suppose that $t \in C$. In this case, $v = B_j$ and $u = c' \in C$, which means that there exists a link $\ell' \in L_0$ that is adjacent to both $e^{(t)}$ and (u, v). Hence, $Q \cup \ell'$ connects $e^{(s)}$ and (u, v), meaning (s, v) are H-reachable.

Now, assume $t \notin C$. Thus, we must have $u = B_i$, $v = c \in C$ and $t = B_j$, for some $i \neq j \in [m]$. If there is a link $\ell \in L$ that is part of Q and is adjacent to (u, v), then the claim holds. So, suppose there is no such link. Let $L^{(s)}$ and $L^{(t)}$ be the set of links in L with both ends in the connected components of $T \setminus \{(u, v)\}$ containing s and t, respectively. The only links not included in $L^{(s)} \cup L^{(t)}$ are those where the path between their endpoints in T contains the edge $(u, v) \in E_T$. In particular, this implies that Q only contains links from $L^{(s)} \cup L^{(t)}$. However, there is no path between $e^{(s)}$ and $e^{(t)}$ using only edges from $L^{(s)}$ and $L^{(t)}$, which contradicts the existence of the path Q. Thus, Q must contain a link that is adjacent to (u, v) and we conclude that the claim holds. □

Resuming the induction, suppose that $\kappa_J(s, t) \geq 2$. Then by Fact 3.1, we know that $\kappa_J(s, v) \geq 2$ and $\kappa_J(u, t) \geq 2$. Using the induction hypothesis, this implies that (s, v) and (u, t) are H-reachable, and by Fact 3.2 we conclude that (s, t) are H-reachable.

Suppose now that (s, t) are H-reachable, and assume for contradiction that $\kappa_J(s, t) = 1$. By Fact 3.3, we know that (s, v) and (u, t) are H-reachable. Using the induction hypothesis, we get that $\kappa_J(s, v) \geq 2$ and $\kappa_J(u, t) \geq 2$. Since $\kappa_J(s, t) = 1$, there is a node $a \in P \setminus \{s, t\}$ such that s and t are in separate connected components of $J \setminus \{a\}$. If $a \in P_{su} \setminus \{s\}$, then this contradicts the fact that $\kappa_J(s, v) \geq 2$. If $a = v$, then this contradicts the fact that $\kappa_J(u, t) \geq 2$. Thus, we must have $\kappa_J(s, t) \geq 2$. This concludes the induction step, and the proof. □

Using the above lemma, we can show that we only need to focus on the reduced (L, E_T)-incidence graph.

Lemma 3.6. *J is 2-node-connected if and only if H_R has a path between every two terminals.*

Proof. We first observe that J is 2-node-connected if and only if $\kappa_J(s,t) \geq 2$ for every pair of leaves s, t of T. Combined with Lemma 3.5, this implies that J is 2-node-connected if and only if for any 2 leaves $s, t \in T$, (s,t) are H-reachable. To obtain the reduced (L, E_G)-incidence graph H_R, we perform the following steps. We first short-cut the set E_T of nodes of H; clearly this does not modify the connectivity of H. Afterwards, we extend the neighborhood of each link ℓ of weight 1, and make ℓ adjacent to any other link ℓ' of weight 1 that can be reached from ℓ in the incidence graph via a path that only uses links of weight 0.

We now make some observations. We observe that short-cutting ensures that if there is a path between two leaf-edges in the original incidence graph, then there is also a path in the incidence graph whose endpoints are leaf-edges and whose internal nodes are all links. Moreover, since no link of L_0 is adjacent to a leaf edge, this means that if there is a path between two leaf-edges in the original incidence graph, H, then there is also a path in H_R whose endpoints are leaf-edges and whose internal nodes are all links in L_1. This means that, by performing the final step where we remove the nodes of L_0 and $E_T \setminus R$, connectivity between leaf-edges is maintained. We conclude that for any 2 leaves $s, t \in T$, (s,t) are H-reachable if and only if they are H_R-reachable. Putting everything together, the claim holds. □

We now have all the ingredients necessary to prove Theorem 3.6.

Proof of Theorem 3.6. Given a connected graph $G = (V, E)$ with a set of links L, we construct the block-cut tree $T = (V_T, E_T)$ and the reduced $(f_G(L) \cup L_0, E_T)$-incidence graph H_R. The resulting instance is viewed as a Node Steiner Tree instance, where R, the set of leaf-edges of T, is the set of terminals and $f_G(L)$ is the set of links. Lemmas 3.6 and 3.4 imply that $J' = G \cup L'$ is 2-node-connected, for some $L' \subseteq L$, if and only if the set $f_G(L')$ is a feasible solution for the constructed Node Steiner Tree H_R, and moreover, the cost of L' in both instances is the same and equal to $|L'|$.

It remains to see that the resulting Node Steiner Tree instance is in fact a CA-Node-Steiner-Tree instance, which can be done by verifying that the requirements for a CA-Node-Steiner-Tree instance hold. By definition of the incidence graph it is clear that terminals are only adjacent to Steiner Nodes. By definition of the short-cut (L, E_G)-incidence graph we know that the neighbours of a

terminal form a clique. Finally, any link ℓ in H is adjacent only to the edges that appear on the path in T between its endpoints, and for any such path at most two edges are leaf-edges. Since we only have leaf-edges in the reduced incidence graph, we get that each Steiner node is adjacent to at most two terminals. This concludes the proof. □

3.1.5 CA-Node-Steiner-Tree to k-restricted CA-Node-Steiner-Tree

Before proving Theorem 3.7, we start with the following observations and lemmas.

Observation 3.2. *Let opt be the optimal cost of a CA-Node-Steiner-Tree instance with t terminals. Then, we have $opt \geq t/2$.*

Definition 3.7. *A rooted binary tree is called regular if every non-leaf node has exactly 2 children.*

As a clarification, a leaf in this rooted tree is a vertex with no children. For the rest of this section, we use the notation $P_T(u, v)$ to denote the unique path from u to v in a tree T.

Lemma 3.7 (see also Lemma 3.1 in [13])**.** *For any rooted regular binary tree $T = (V, E)$ with a set of leaves $C \subseteq V$, there exists a one-to-one mapping $f : V \setminus C \to C$ from non-leaf nodes to leaves, such that*

- *for every $u \in V \setminus C$, $f(u)$ is a descendent of u, and*
- *all paths $P_T(u, f(u))$ from $u \in V \setminus C$ to $f(u)$ are pairwise edge disjoint, and internally node disjoint (where internally node disjoint means that there is no node that belongs to both paths and is also internal for both paths).*

Proof. We will prove a slightly stronger statement which implies the lemma. We will show that for any regular binary tree $T = (V, E)$ rooted at a vertex $r \in V$ with a set of leaves $C \subseteq V$, there exists a one-to-one mapping $f : V \setminus C \to C$ from non-leaves to leaves such that:

1. for every $u \in V \setminus C$, $f(u)$ is a descendant of u,
2. all paths $P_T(u, f(u))$ from u to $f(u)$ are pairwise edge disjoint, and internally node disjoint,

3. there exists a leaf $c_0 \in C$ that is not the image of any non-leaf node, such that the path $P_T(r, c_0)$ from the root r of the tree to c_0 is edge disjoint and internally node disjoint from any other path $P_T(u, f(u))$, for any $u \in V \setminus C$.

To prove the above statement, we use induction on the height h of the tree, where we define the height of the tree as the maximum number of nodes that appear in any path from the root to a leaf, including the endpoints. The base case is $h = 1$, in which case, the tree consists of a single leaf, and thus the statement is trivially true. Suppose now that the statement holds for all regular binary trees of height at most $h \geq 1$. We will show that the statement also holds for all regular binary trees of height $h+1$.

Let $T = (V, E)$ be a tree of height $h+1$, let r be its root, and let u_1 and u_2 be its two children. Let $C \subseteq V$ be its set of leaves. The subtrees T_1 and T_2, rooted at u_1 and u_2, respectively, are both regular binary trees of height at most h. Let V_i denote the set of vertices of T_i, for $i \in \{1,2\}$, and $C_i \subseteq V_i$ denote the set of leaves of T_i, for $i \in \{1,2\}$. Note that $V = \{r\} \cup V_1 \cup V_2$ and $C = C_1 \cup C_2$. By the induction hypothesis, there exist maps $f_1 : V_1 \setminus C_1 \to C_1$ for T_1 and $f_2 : V_2 \setminus C_2 \to C_2$ for T_2 that satisfy the desired properties. Let $c_i \in C_i$ be the leaf of T_i that satisfies the third property of the statement, for $i \in \{1,2\}$. We now define the map $f : V \setminus C \to C$ for T as follows:

- $f(r) = c_1$,
- for every $i \in \{1,2\}$ and $u \in V_i \setminus C_i$, we set $f(u) = f_i(u)$,

Clearly, the first property of the statement is satisfied. By the induction hypothesis, and since T_1 and T_2 are disjoint, we get that all paths in the set $\{P_T(u, f(u))\}_{u \in V \setminus (C \cup \{r\})}$ are pairwise edge disjoint, and internally they are node disjoint. Thus, we only need to check the path $P_T(r, c_1)$. We observe that the path $P_T(r, c_1)$ from r to c_1 is clearly node disjoint from all paths of the set $\{P_T(u, f(u))\}_{u \in V_2 \setminus C_2}$ and by the induction hypothesis, it is also edge disjoint and internally node disjoint from any path $P_T(u, f(u))$ where $u \in V_1 \setminus C_1$. Again, we clarify here that internally node disjoint means that there is no node that belongs to both paths and is also internal for both paths. Thus, the first two properties are satisfied. In order to prove the third property, we set $c_0 \equiv c_2$. By the induction hypothesis, the path $P_T(v_2, c_2)$ is edge disjoint and internally node disjoint from any other path of the set $\{P_T(u, f(u))\}_{u \in V_2 \setminus C_2}$. Moreover, it only intersects the path from r to $f(r)$ at its endpoint. Thus, the third property is satisfied for T. This concludes the proof. □

Lemma 3.8. *Let $T = (V, E)$ be a tree with t leaves, and let $d(v)$ be the degree of $v \in V$. Then, we have $\sum_{v \in V: d(v) \geq 3} (d(v) - 2) \leq t$.*

Proof. Let n be the total number of vertices of T, which we decompose as $n = t + n_2 + n_{\geq 3}$, where n_2 is the number of vertices of degree exactly 2, and $n_{\geq 3}$ is the number of vertices of degree at least 3. We have $t + 2n_2 + \sum_{v \in V: d(v) \geq 3} d(v) = 2(n-1) = 2t + 2n_2 + 2n_{\geq 3} - 2$, which gives $\sum_{v \in V: d(v) \geq 3} (d(v) - 2) = t - 2 \leq t$. □

We are now ready to prove Theorem 3.7.

Proof of Theorem 3.7. We begin with an optimal solution Q of cost opt to the given CA-Node-Steiner-Tree instance, whose set of leaves is identical to the set of terminals. We modify Q as described in Xu et al. [89] to get a new tree Q^+ of equal cost. More precisely, we first create a dummy Steiner node of zero weight in the middle of an edge that serves as the root of the tree; thus, from now on Q will be a rooted tree. For each Steiner node v that has at least three children, we expand v into a binary tree by using duplicate nodes of zero weight such that every node in the resulting tree has at most two children. The following lemma, whose proof is straightforward and thus omitted, shows that the number of auxiliary nodes is proportional to the degree of the node that is modified.

Lemma 3.9. *Let v be a node of Q with at least three children. Then, if we apply the transformation described above to v, we get a new tree that contains $d(v) - 3$ extra auxiliary nodes, where $d(v)$ is the degree of v in Q.*

We repeatedly apply the transformation described in Lemma 3.9, and we get a new tree Q^+ where every Steiner node has at most two children. This new tree Q^+ has at most $n + \sum_{v \in Q: d(v) \geq 4} (d(v) - 3) \leq 3n$ vertices, where n is the number of vertices of the rooted tree Q. Finally, for any non-leaf node different from the root that has degree 2, we add a dummy terminal as a child, and we end up with a regular node weighted binary tree (where only Steiner nodes have weight), which, by slightly abusing notation, we call Q^+.

We now label the Steiner nodes of Q^+ with labels $\{0, \ldots, m-1\}$ as follows; we stress that the labeling and the arguments used closely follow the techniques of Borchers and Du [13]. We first clarify some terminology. We say that the root is at the 0^{th} layer of Q^+, the children of the root are at the 1^{st} layer, and so on. We label the root with 0, the 1^{st} layer of nodes with 1, and so on until the $(m-1)^{\text{th}}$ layer. We then repeat the labelling with label 0, starting from layer m. More precisely, the Steiner nodes at layer i all get the same label $i \mod m$. We obviously have the property that any m consecutive layers have different labels.

For each label $j \in \{0, \dots, m-1\}$, we will define a feasible k-restricted Steiner tree Q_j^+. We first define its components; the tree Q_j^+ is simply the union of these components and its cost is the sum of the costs of its components. For a fixed label j, each component of Q_j^+ is rooted at either the root of Q^+ or a Steiner node with label j. We define the component rooted at Steiner node v labelled with j as follows: the component rooted at v contains the paths in Q^+ to the first nodes below v that are also labelled with j; call these nodes *intermediate leaves*. More precisely, the component rooted at v connects to u if u is a descendant of v labelled with j, such that there is no other node of label j on the path from v to u besides v and u. Then, for an intermediate leaf u that is a Steiner node, the component contains the unique path $p(u)$, from u to $f(u)$, as defined in Lemma 3.7. If there is a path from v to a leaf of Q^+ that is a descendant of v such that the path does not contain a node with label j, then that path is also in the component. The component of Q_j^+ that is rooted at the root of Q^+ is constructed in a similar way. Finally, Q_j^+ is the union of all components rooted at a Steiner node with label j, along with the component rooted at the root of Q^+, which is always included, even if $j > 0$.

Lemma 3.10. *Each component of Q_j^+ has at most k terminals.*

Proof. It is clear that a component has at most 2^m intermediate leaves, and thus at most 2^m leaves connected by paths from these intermediate leaves. Moreover, if there is a direct path from the root of the component to a leaf that does not contain an intermediate leaf, then the leaf must be no more than m layers below the component root. So, the total number of leaves for any component is at most $2^m = k$. □

Lemma 3.11. *Let $j \in \{0, \dots, m-1\}$. The tree Q_j^+ is a feasible k-restricted CA-Node-Steiner-Tree.*

Proof. First, look at the part of each component that connects a node v labelled as j to its intermediate leaves: the union of all these components, along with the component rooted at the root of Q^+, contains all nodes of Q^+. Furthermore, take any two components such that one is rooted at v, and one is rooted at one of its intermediate leaves, say u: the components intersect at the terminal $f(u)$. It is not difficult to see then that Q_j^+ is indeed a feasible k-restricted Steiner tree. □

Given a k-restricted Steiner tree Q_j^+ of Q^+, we now explain how to construct a k-restricted Steiner tree Q_j of Q. For any component B of Q_j^+ we construct a component B' in Q by contracting every expanded node and removing every dummy terminal. Moreover, if B' contains the global root, we remove it as well and "restore" the edge that was split in order to create the root. B' clearly still has at most k terminals, and thus the union of these components of Q is a k-restricted Node Steiner Tree, denoted as Q_j. For the final step of our argument, we require the following observation about k-restricted trees that will be useful in bounding the optimal cost.

Observation 3.3. *Consider the k-restricted Steiner trees $Q_0^+,\dots,Q_{m-1}^+$ of Q^+ and their corresponding components. Any non-root Steiner node appears as an intermediate leaf in exactly one component of exactly one such Steiner tree.*

Using this observation we will now find a bound on the sum $\sum_{j=0}^{m-1} cost(Q_j)$. For that, we consider a Steiner node v of Q. We first note that v has $d(v)-2$ "copies" in Q^+. By Observation 3.3, each such copy appears as an intermediate leaf in exactly one component of exactly one Steiner tree among $Q_0^+,\dots,Q_{m-1}^+$, and thus it is counted twice, since it also appears once as the root of a component. Finally, each such copy can appear at most one more time as an internal node of a path from an intermediate leaf to a leaf (as implied by Lemma 3.7). Thus, v appears at most $2(d(v)-2)+m$ times in the sum $\sum_{j=0}^{m-1} cost(Q_j)$. Let S denote the set of Steiner nodes of Q, $S_2=\{s\in S: d(s)=2\}$ and $S_{\geq 3}=\{s\in S: d(s)\geq 3\}$. Note that $S=S_2\cup S_{\geq 3}$ and $opt=|S|$. Let t be the number of terminals. We have

$$\begin{aligned}
&\sum_{j=0}^{m-1} cost(Q_j) \leq m|S| + 2\sum_{s\in S}(d(s)-2) = (m-4)|S| + 2\sum_{s\in S_2} d(s) + 2\sum_{s\in S_{\geq 3}} d(s)\\
&\leq (m-4)|S| + 4\,(|S| - |S_{\geq 3}|) + 2\sum_{s\in S_{\geq 3}} d(s)\\
&= m\cdot opt + 2\sum_{s\in S_{\geq 3}} (d(s)-2) \leq m\cdot opt + 2t\\
&\leq m\cdot opt + 4opt = (m+4)opt,
\end{aligned}$$

where in the above derivations we used Observation 3.2 and Lemma 3.8. Thus, there exists a label $j\in\{0,1,\dots,m-1\}$ such that $cost(Q_j)\leq \left(1+\frac{4}{\log k}\right)opt$, where we recall that $k=2^m$. □

3.2 A deterministic construction of witness trees

In this section, for any given CA-Node-Steiner-Tree instance $G = (V, E)$, we consider a specific optimal Steiner tree and describe a deterministic construction of a witness tree W that shows that $\psi_G < 1.8917$. By the definition of ψ, this immediately implies Theorem 3.10.

Throughout this section, we use the following notation: given is a CA-Node-Steiner-Tree instance $G = (R \cup S, E)$ and an optimal solution $T = (R \cup S^*, E^*)$. The goal is to construct a witness tree $W = (R, E_W)$ spanning the terminals, such that $\nu_T(W)$ is minimized, where $w : S^* \to \mathbb{N}_{\geq 0}$ is the vector imposed by W. From now on, we will refer to $w(v)$ as the w-value of a node v. Since the proof of Theorem 3.10 consists of several steps, we will present them separately, and at the end of the section we will put everything together and formally prove Theorem 3.10.

We start by observing that every CA-Node-Steiner-Tree instance has an optimal solution in which the terminals are leaves. This follows by Definition 3.5, since (i) there are no edges between terminals and (ii) adjacent Steiner nodes of a terminal form a clique. Thus, from now on we assume that T is an optimal solution in which the terminals are leaves. Given this, the first step is to show that we can reduce the problem of constructing a witness tree spanning the terminals to the problem of constructing a witness tree spanning the Steiner nodes that are adjacent to terminals. This will allow us to remove the terminals from T, and only focus on $T[S^*]$, which is a (connected) tree.

3.2.1 Removing terminals from T

Let $F \subseteq S^*$ be the set of Steiner nodes contained in T with at least one adjacent terminal. We call F the set of *final* Steiner nodes. In order to simplify notation, we will remove the terminals from T, and compute a tree W spanning the final nodes of $T[S^*]$ (note that the set of final nodes contains all the leaves of $T[S^*]$). Any such tree W directly corresponds to a terminal spanning tree as follows: for each final Steiner node $f \in F$, we arbitrarily pick one of the (at most) two terminals adjacent to f, which we denote as $\texttt{rep}(f)$. Given a spanning tree W of the final Steiner nodes, we can now generate a terminal spanning tree W' as follows (see Figure 3.2a):

- we replace every edge (f, f') of W with the edge $(\texttt{rep}(f), \texttt{rep}(f'))$,
- in case a final Steiner node f has a second terminal t adjacent to it besides $\texttt{rep}(f)$, we also add the edge $(\texttt{rep}(f), t)$ to the tree.

It is easy to see that the vectors w and w' imposed by W and W', respectively, on the Steiner nodes are the same for all nodes except for the final Steiner nodes, where the w-values differ by 1 in case a final Steiner node has two adjacent terminals. Thus, from now on we focus on computing a tree W spanning the final Steiner nodes, and we simply increase the corresponding imposed w-value of each final Steiner node by 1. More formally, if $W = (S^*, E_W)$ is a tree spanning the final nodes of $T[S^*]$, we have for every $v \in S^*$:

$$w(v) \coloneqq |\{(p,q) \in E_W : v \text{ belongs to the } p\text{-}q \text{ path in } T[S^*]\}| + \mathbb{1}[v \in F], \quad (3.1)$$

where $\mathbb{1}[v \in F]$ denotes the indicator of the event "$v \in F$". This expression corresponds to the (worst-case) assumption that every final Steiner node is incident to two terminals; this is without loss of generality since, if the value $\frac{1}{|S|}\sum_{v\in S} H_{w(v)}$ computed in this way is less than a given value $\psi' > 0$, then clearly the value $\frac{1}{|S|}\sum_{v\in S} H_{w'(v)}$ computed by the terminal spanning tree W' is also bounded by ψ', as the harmonic function H_x is increasing in x.

The next step is to root $T[S^*]$ and then decompose it into a collection of rooted subtrees of $T[S^*]$ such that every final Steiner node appears as either a leaf or a root (or both); we call these subtrees *final-components*. This decomposition is nearly identical to the one used in Section 3.1.5 where we decomposed a tree into components where terminals appeared as leaves (here we use final Steiner nodes instead of terminals), with the added element that now these components are also rooted.

3.2.2 Decomposing T into final-components

To simplify notation, let $T = (V, E)$ be a tree where $F \subseteq V$ is the set of final Steiner nodes, again noting that F contains all the leaves of T. We start by rooting the tree T at an arbitrary leaf r; note that $r \in F$. We decompose T into a set of final-components $T_1, \ldots, T_\tau$, where a final-component is a rooted maximal subtree of T whose root and leaves are the only nodes in F. We clarify here that for a rooted tree, we call a vertex a leaf if it has no children (thus, although the root may have degree 1, it is not referred to as a leaf).

More precisely, we start with T and consider the maximal subtree T_1 of T that contains the root r, such that all leaves of T_1 are final Steiner nodes, and including the root, are the only final Steiner nodes. For each leaf $f \in F \cap T_1$, we do the following. If f has children $\{x_1, \ldots, x_p\}$ in T, for some $p \in \mathbb{N}_{p\geq 1}$, we first create p copies of f; this means that, along with the original node f, we now have $p+1$ copies of f, and one copy of it remains in T_1. We then remove T_1

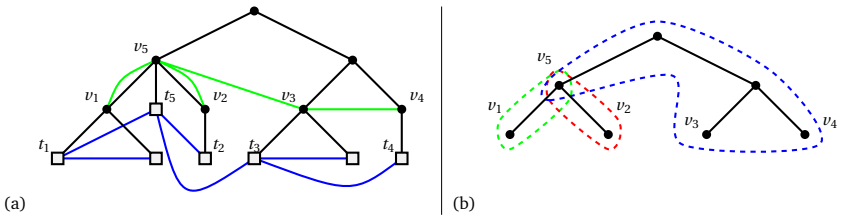


Figure 3.2: (a) Given is a Steiner tree, marked with black, where the terminals are marked as square nodes. The set of final nodes is $\{v_1, v_2, v_3, v_4, v_5\}$, and a tree spanning them is denoted with the green edges. In order to obtain a terminal spanning tree, each green edge (v_i, v_j) is replaced with the edge (t_i, t_j), marked with blue, where $t_i = \texttt{rep}(v_i)$ for every $i \in \{1,2,3,4,5\}$. Finally, every terminal with a "sibling" terminal has a corresponding edge added, marked again with blue. (b) An example of the the Steiner tree from (a) decomposed into its final-components as defined in Section 3.2.2, where the components are circled in green, red, and blue dashed lines.

from T, and for each tree $T(x_i)$, $i \in [p]$ where $[p] \coloneqq \{1,2,\ldots,p\}$, rooted at x_i, that is part of the forest $T \setminus T_1$, we connect x_i with one distinct copy of f, and set this copy to be the root of the modified tree $T(x_i)$. Doing this for every leaf of T_1, we end up with a forest. We repeat this procedure recursively for each tree of this forest, until all trees of the forest are final-components. A simple example of this decomposition is shown in Figure 3.2b. Let $T_1,\ldots,T_\tau$ be the resulting final-components. Without loss of generality, we assume that for every $i \in [\tau]$, $\bigcup_{j \le i} T_j$ is connected. We clarify that in this union of trees, duplicate copies of the same node are "merged" back into one node.

Given the final-components $T_1,\ldots,T_\tau$ of T, we will now show how to compute a tree W_i spanning the final Steiner nodes of each T_i, and then show how to combine all of them to generate a tree W that spans the final Steiner nodes of T, while bounding $\nu_T(W)$.

3.2.3 Computing a tree W spanning the final Steiner nodes of T

We start by describing how to compute a tree W_i that spans the final Steiner nodes of each final-component T_i, as well as how to join these trees $\{W_i\}_{i=1}^{\tau}$ to obtain a tree W spanning the whole set of final Steiner nodes of T. We construct W iteratively as follows. We start with $W = \emptyset$, process the final-components

$T_1, \ldots, T_\tau$ one by one according to the index-order, and for each i, compute a tree W_i for T_i and merge it with the current W; again, in this merging, multiple copies of the same node are merged back into one node.

To analyze this procedure, we fix a final-component T_i rooted at r_i, and let (r_i, v) be the unique edge incident to r_i inside T_i. Let $T' = \bigcup_{j<i} T_j$, and W' be the constructed witness tree for T'. Let w' be the vector imposed on the Steiner nodes of T' by W'. Note that for $i = 1$ we let $T' = \emptyset$, $W' = \emptyset$, and $w' = 0$.

We do some case analysis based on whether $v \in F$ or not; throughout this analysis, we use the notation $|T|$ to denote the number of nodes of a tree T.

Case 1: $v \in F$**.** In this case, the edge (r_i, v) is the only edge of T_i, so W_i simply consists of the nodes $\{r_i, v\}$ and the edge (r_i, v). The following lemma shows that merging W_i with W' (by simply merging the two copies of r_i and taking the union of the edges of W' and W_i, in the case where $W' \neq \emptyset$) would keep $\sum_{u \in T'} \frac{H_{w'(u)}}{|T'|}$ below our threshold. Let w'' be the vector imposed by $W'' = W' \cup W_i$.

Lemma 3.12. *Let $\psi' \geq H_3$, and suppose that $\sum_{u \in T'} \frac{H_{w'(u)}}{|T'|} < \psi'$.*

Then, $\sum_{u \in T' \cup T_i} \frac{H_{w''(u)}}{|T' \cup T_i|} < \psi'$.

Proof. We do some case analysis. If $i = 1$, then $W' = \emptyset$ and $r_i = r$. In this case we have $w''(r_i) = w''(v) = 2$, which gives $\sum_{u \in T' \cup T_i} \frac{H_{w''(u)}}{|T' \cup T_i|} = H_2 < \psi'$.

So, suppose now that $i > 1$, which implies that $W' \neq \emptyset$. Note that $w''(v) = 2$, $w''(r_i) = w'(r_i) + 1$, and for each $u \in T' \setminus \{r_i\}$, $w''(u) = w'(u)$. Moreover, we have $w'(r_i) \geq 2$; to see this, note that r_i is a final Steiner node of T' and has degree at least 1 in W'. Thus, $H_{w''(r_i)} \leq H_{w'(r_i)} + \frac{1}{3}$. Therefore,

$$\sum_{u \in T' \cup T_i} \frac{H_{w''(u)}}{|T' \cup T_i|} \leq \frac{H_{w''(v)} + 1/3 + \sum_{u \in T'} H_{w'(u)}}{|T' \cup T_i|}$$
$$= \frac{H_3 + \sum_{u \in T'} H_{w'(u)}}{|T'| + 1} < \psi'.$$

We conclude that the lemma holds. □

Case 2: $v \notin F$**.** In this case, the witness tree W_i of T_i is computed deterministically by a procedure described in Algorithm 2; we provide an example of the output of Algorithm 2 in Figure 3.3.

In order to bound the value of $\sum_{u \in T'} \frac{H_{w'(u)}}{|T'|}$ that we get after adding W_i to the current W', we first introduce some notation. For any node $u \neq r_i$ of T_i, let

Algorithm 2: Computing the tree W_i

1 Assign an arbitrary ordering to the leaves of T_i;

2 Assign a weight to each node $u \in T_i$ that is equal to the inverse of its degree in T_i;

3 For each non-final Steiner node $u \in T_i$, compute a minimum-weight path $P(u)$ from u to a leaf that is a descendant of u in T_i, where the weight of the path is given by the sum of weights of its nodes. Break ties in favor of the smallest leaf according to the ordering computed at step (1);

4 For each non-final Steiner node $u \in T_i$, contract the path $P(u)$ computed in step (3) into its leaf endpoint, and obtain a tree spanning the final Steiner nodes of T_i; use this as the witness tree W_i;

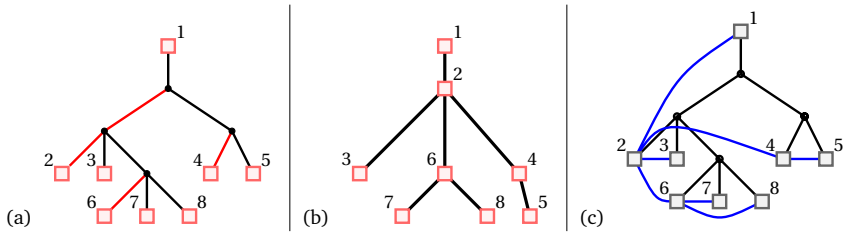


Figure 3.3: (a) Given is a graph $G = (V, E)$, with final Steiner nodes depicted as red squares. At each interior node u we mark in red the u-to-leaf-path $P(u)$ with the least weight (sum of the inverse of the node degrees on the path); this corresponds to step (3) of Algorithm 2. (b) At step (4) of Algorithm 2, we contract the red edges and label the new contracted nodes with the name of the original (marked) leaf node. (c) A pictorial description of the witness tree obtained for the original graph.

$\texttt{parent}(u)$ denote the parent of u in T_i. Let Q_u denote the subtree of T_i rooted at u. For any non-final Steiner node u, $P(u)$ denotes the minimum-weight path from u to a leaf, as computed at step (3) of Algorithm 2. In the case where u is a final Steiner node of T_i, we define $P(u) := \{u\}$; in particular $P(r_i) := \{r_i\}$. For every $u \in T_i$, let q_u be the number of nodes of $P(u)$ (including the endpoints) and $l(u)$ denote the (unique) endpoint of this path that is a final Steiner node. For every non-final Steiner node u, exactly one child of u is in this path, which we call the *marked* child of u. For every $u \neq r_i$, we let $a(u)$ denote the node

closest to u on the path from u to r_i such that $l(u) \neq l(a(u))$. Examples of these definitions are given in Figure 3.4.

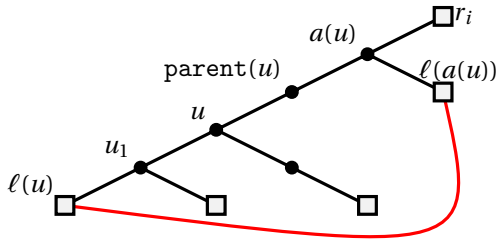


Figure 3.4: Given is an example of a node u; the minimum-weight path from u to the leaf $\ell(u)$, $P(u) = \{u, u_1, \ell(u)\}$; the parent of u, parent(u); the first node on the u to r_i path, $a(u)$, such that $\ell(u) \neq \ell(a(u))$; and finally, the edge $e_u = \{\ell(u), \ell(a(u))\}$.

To begin the analysis, we first observe that for every node $u \neq r_i$ of T_i, W_i contains an edge with endpoints $l(u)$ and $l(a(u))$; we call this edge e_u. Note that we can have $e_u = e_v$ for $u \neq v$. For every $u \neq r_i$, we consider the subtree W^u of W_i defined with respect to Q_u, defined as a set of edges as

$$W^u = \{e \in W_i : \text{ both the endpoints of } e \text{ are in } Q_u\} \cup e_u.$$

We also define $W^{r_i} = W_i$. Observe that for every $u \in T_i$, W^u indeed corresponds to a connected subtree of W_i. An example of W^u is provided in Figure 3.5, where we also clarify several properties of Lemma 3.13, proved below. We let w^u be the vector imposed by W^u, where for every $\ell \in Q_u$, we have

$$w^u(\ell) := |\{(p,q) \in W^u : \ell \text{ belongs to the } p\text{-}q \text{ path in } T_i\}| + \mathbb{1}[\ell \text{ is a leaf of } Q_u].$$

Let $h(Q_u) := \sum_{\ell \in Q_u} H_{w^u(\ell)}$. We note here that W^u might involve one final Steiner node, namely $l(a(u))$, that is not contained in Q_u. Finally, let d_u be equal to 3 if u is a leaf, otherwise, let d_u be equal to the degree of u in T_i.

Lemma 3.13. *Let $u \in T_i$ be a non-final Steiner node and $u_1, \ldots, u_p$ be the children of u, with u_1 being the marked child of u. Then:*

1. $w^u(u) = p$.
2. *For every $j \in \{2, \ldots, p\}$ and every Steiner node $\ell \in Q_{u_j}$, $w^u(\ell) = w^{u_j}(\ell)$.*

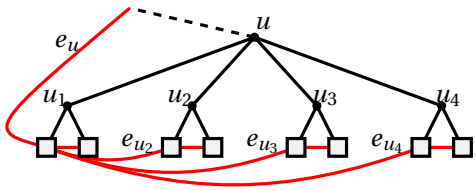


Figure 3.5: Given is the tree Q_u of T_i. W^u contains all of the red edges. Note that each W^{u_i} is equal to the red edge with both endpoints below u_i plus e_{u_i}, for $i = 1,2,3,4$. We can also see an example of the first three parts of Lemma 3.13: (a) $w^u(u) = 4$; (b) for $j = 2,3,4$, and every $\ell \in Q_{u_j}$, $w^u(\ell) = w^{u_j}(\ell)$, and; (c) for each $\ell \in Q_{u_1} \setminus P(u_1)$, $w^u(\ell) = w^{u_1}(\ell)$.

3. *For every* $\ell \in Q_{u_1} \setminus P(u_1)$, $w^u(\ell) = w^{u_1}(\ell)$.

4. *For every* $\ell \in P(u_1)$, $H_{w^u(\ell)} - H_{w^{u_1}(\ell)} \leq \sum_{j=2}^{p} \frac{1}{d_\ell + j - 2}$.

Proof. We first observe that W^u is given by the disjoint union of $W^{u_1} \cup W^{u_2} \cup \ldots \cup W^{u_p}$. In fact, for $j = 2,\ldots,p$, the edge $e_{u_j} \in W^{u_j}$ is precisely the edge with endpoints $l(u_1)$ and $l(u_j)$, and the edge $e_{u_1} \in W^{u_1}$ corresponds to e_u. Given these observations, $w^u(u) = p$ as the edges of W^u which contribute to $w^u(u)$ are precisely $e_{u_1},\ldots,e_{u_p}$. Hence (a) holds.

Statements (b) and (c) follow by observing that the only nodes whose w^u-value differs from the corresponding w^{u_j}-value are in Q_{u_1}, and are precisely the nodes of the path $P(u_1)$. It is not hard to see that for every $\ell \in P(u_1)$, we have $w^u(\ell) = w^{u_1}(\ell) + p - 1$. Thus, we get that $H_{w^u(\ell)} - H_{w^{u_1}(\ell)} = \sum_{j=2}^{p} \frac{1}{w^{u_1}(\ell)+j-1}$. We now compute a lower bound for $w^{u_1}(\ell)$. If $\ell \neq l(u_1)$, then we can apply statement (a), which we just proved, to Q_{u_ℓ}, and get that $d_\ell - 1 = w^\ell(\ell) \leq w^{u_1}(\ell)$. Hence (d) holds in this case.

We now consider the case $\ell = l(u_1)$, i.e., ℓ is a leaf of Q_{u_1}. In this case, we have $d_\ell = 3$ and $w^\ell(\ell) = 2$, where the latter equality holds since there is a unique edge $e_\ell \in W^\ell$ and ℓ is a leaf, which means that its w-value is increased by 1. Thus, statement (d) holds in this case as well. This concludes the proof. □

The following lemma is the main technical part of our analysis.

Lemma 3.14. *Let* $\delta = 7/120$. *For each node* $u \in T_i \setminus r_i$, *we have*

$$h(Q_u) + \sum_{\ell \in P(u)} \frac{1}{d_\ell} + \delta < 1.8917 \cdot |Q_u|.$$

Proof. The proof is by induction on $|Q_u|$. The base case is $|Q_u| = 1$, in which case u is leaf of T_i. In this case, we have $d_u = 3$, $w^u(u) = 2$, and $P(u) = \{u\}$, giving

$$h(Q_u) + \frac{1}{3} + \delta = H_2 + \frac{1}{3} + \delta = 1.891\bar{6} < 1.8917.$$

For the induction step, let u be a non-final Steiner node of T_i, let $u_1, \ldots, u_p$ be its children, and let u_1 be its marked child. By Lemma 3.13 we have that

$$h(Q_u) = \sum_{j=1}^{p} h(Q_{u_i}) + \sum_{\ell \in P(u_1)} \left(H_{w^u(\ell)} - H_{w^{u_1}(\ell)}\right) + H_p.$$

By the induction hypothesis, for each $j \in [p]$ we have $h(Q_{u_j}) + \sum_{\ell \in P(u_j)} \frac{1}{d_\ell} + \delta < 1.8917 \cdot |Q_{u_j}|$. Combining these we get

$$\begin{aligned} h(Q_u) &< 1.8917 \sum_{j=1}^{p} |Q_{u_j}| - \delta p - \sum_{j=1}^{p} \sum_{\ell \in P(u_j)} \frac{1}{d_\ell} + \sum_{\ell \in P(u_1)} (H_{w^u(\ell)} - H_{w^{u_1}(\ell)}) + H_p \\ &\leq 1.8917 \sum_{j=1}^{p} |Q_{u_j}| - \delta p - \sum_{j=1}^{p} \sum_{\ell \in P(u_j)} \frac{1}{d_\ell} + \sum_{\ell \in P(u_1)} \sum_{j=2}^{p} \frac{1}{d_\ell + j - 2} + H_p, \end{aligned}$$

where the last inequality follows by statement (d) of Lemma 3.13. We now observe that due to the greedy choice of the path $P(u)$ at step (3) of Algorithm 2, for each $j \in [p]$ we have $\sum_{\ell \in P(u_j) \setminus F} \frac{1}{d_\ell} + 1 \geq \sum_{\ell \in P(u_1) \setminus F} \frac{1}{d_\ell} + 1$, since there is exactly one final Steiner node in each path $P(u_j)$, $j \in [p]$. Thus, we get $\sum_{\ell \in P(u_j)} \frac{1}{d_\ell} \geq \sum_{\ell \in P(u_1)} \frac{1}{d_\ell}$ for every $j \in [p]$, and so $\sum_{j=1}^{p} \sum_{\ell \in P(u_j)} \frac{1}{d_\ell} \geq p \sum_{\ell \in P(u_1)} \frac{1}{d_\ell}$. Moreover, for every $\ell \in P(u_1)$ we have $\sum_{j=2}^{p} \frac{1}{d_\ell + j - 2} \leq \frac{p-1}{d_\ell}$. Combining these we get $h(Q_u)$ is less than

$$\begin{aligned} &< 1.8917 \sum_{j=1}^{p} |Q_{u_j}| - \delta p - \sum_{\ell \in P(u_1)} \frac{p}{d_\ell} + \sum_{\ell \in P(u_1) \setminus \{l(u)\}} \frac{p-1}{d_\ell} + \sum_{j=2}^{p} \frac{1}{d_{l(u)} + j - 2} + H_p \\ &\leq 1.8917 \sum_{j=1}^{p} |Q_{u_j}| - \delta p - \sum_{\ell \in P(u_1)} \frac{1}{d_\ell} - \frac{p-1}{d_{l(u)}} + \sum_{j=2}^{p} \frac{1}{d_{l(u)} + j - 2} + H_p. \end{aligned}$$

Substituting $d_{l(u)} = 3$ into this equation, we get

$$h(Q_u) < 1.8917 \sum_{j=1}^{p} |Q_{u_j}| - \delta p - \sum_{\ell \in P(u_1)} \frac{1}{d_\ell} - \frac{p-1}{3} + \sum_{j=2}^{p} \frac{1}{j+1} + H_p.$$

Rearranging, and using $\sum_{\ell\in P(u)}\frac{1}{d_\ell}=\sum_{\ell\in P(u_1)}\frac{1}{d_\ell}+\frac{1}{d_u}$ and $d_u=p+1$, we get

$$h(Q_u)+\sum_{\ell\in P(u)}\frac{1}{d_\ell}+\delta<1.8917\sum_{j=1}^{p}|Q_{u_j}|-\delta(p-1)-\frac{p-1}{3}+\sum_{j=2}^{p}\frac{1}{j+1}+H_p+\frac{1}{d_u}$$
$$=1.8917\sum_{j=1}^{p}|Q_{u_j}|-\delta(p-1)-\frac{p-1}{3}+\sum_{j=2}^{p}\frac{1}{j+1}+H_{p+1}.$$

We will now show that $-\delta(p-1)-\frac{p-1}{3}+\sum_{j=2}^{p}\frac{1}{j+1}+H_{p+1}\le 1.8917$. For that, we define $\psi(p):=2H_{p+1}-(p-1)\left(\frac{1}{3}+\delta\right)-H_2$. Note that the left-hand side of the inequality above is equal to $\psi(p)$. We now show that $\psi(p)\le\psi(4)$ for every $p\in\mathbb{N}_{>0}$. For that, we have $\psi(p+1)-\psi(p)$ is equal to

$$=2H_{p+2}-p\left(\frac{1}{3}+\delta\right)-H_2-\left(2H_{p+1}-(p-1)\left(\frac{1}{3}+\delta\right)-H_2\right)=\frac{2}{p+2}-\frac{1}{3}-\delta.$$

We now observe that $\psi(p+1)-\psi(p)\ge 0$ for $p\in\{1,2,3\}$ and $\psi(p+1)-\psi(p)<0$ for $p\ge 4$. Thus, $\psi(p)$ is increasing for $p\le 4$ and decreasing for $p\ge 4$. This means that $\max_{p\in\mathbb{N}_{>0}}\psi(p)=\psi(4)$. It is easy to verify that $\psi(4)=1.891\bar{6}<1.8917$. Putting everything together, we conclude $h(Q_u)+\sum_{\ell\in P(u)}\frac{1}{d_\ell}+\delta<1.8917\cdot|Q_u|$. □

Finally, we observe that $W_i=W^{r_i}$. With the above lemma at hand, it is not difficult to show that merging W_i with the current W keeps the value below our targeted threshold. Once again, let T' be the union of $T_1,\dots,T_{i-1}$, let w' be the vector imposed by the current W' before adding W_i, and w'' be the vector imposed by $W'\cup W_i$.

Lemma 3.15. *If* $\sum_{u\in T'}\frac{H_{w'(u)}}{|T'|}<1.8917$, *then* $\sum_{u\in T'\cup T_i}\frac{H_{w''(u)}}{|T'\cup T_i|}<1.8917$.

Proof. We do some case analysis. We first consider the case of $i=1$, which means that $r_i=r$ and $W'=\emptyset$. In this case, since $T_i\setminus\{r_i\}=Q_v$, we can apply Lemma 3.14 and get that $\sum_{\ell\in T_i\setminus\{r_i\}}H_{w''(\ell)}<1.8917(|T_i|-1)-\sum_{\ell\in P(v)}\frac{1}{d_\ell}-\delta$. Moreover, we have $w''(r_i)=2$. Thus, we get

$$\sum_{u\in T'\cup T_i}\frac{H_{w''(u)}}{|T'\cup T_i|}=\frac{\sum_{\ell\in T_i\setminus\{r_i\}}H_{w''(\ell)}+H_2}{|T'\cup T_i|}$$
$$<\frac{1.8917(|T_i|-1)+H_2}{|T'\cup T_i|}<1.8917.$$

We now turn to the case of $i>1$, which means that $W'\neq\emptyset$. In this case, we note the following:

1. $w''(\ell) = w'(\ell)$ for all $\ell \in T_i \setminus r_i$,
2. $w''(r_i) = w'(r_i) + 1$, and
3. $w'(u) = w''(u)$ for all $u \in T' \setminus \{r_i\}$.

Since r_i is a leaf of T', we have $w'(r_i) \geq 2$, which implies that $H_{w''(r_i)} \leq H_{w'(r_i)} + \frac{1}{3}$. Moreover, since $T_i \setminus \{r_i\} = Q_v$, we can apply Lemma 3.14 to get $\sum_{\ell \in T_i \setminus \{r_i\}} H_{w''(\ell)} < 1.8917(|T_i| - 1) - \sum_{\ell \in P(v)} \frac{1}{d_\ell} - \delta$. Furthermore, $\sum_{\ell \in P(v)} \frac{1}{d_\ell} + \delta \geq \frac{1}{3}$, as $d_{l(v)} = 3$ and $l(v) \in P(v)$. Hence we get that

$$\sum_{u \in T' \cup T_i} \frac{H_{w''(u)}}{|T' \cup T_i|} \leq \frac{\sum_{\ell \in T_i \setminus \{r_i\}} H_{w''(\ell)} + 1/3 + \sum_{u \in T'} H_{w'(u)}}{|T' \cup T_i|}$$
$$< \frac{1.8917(|T_i| - 1) + 1.8917|T'|}{|T' \cup T_i|} = 1.8917.$$

□

We are now ready to put everything together and prove Theorem 3.9.

Proof of Theorem 3.9. Let $G = (R \cup S, E)$ be a CA-Node-Steiner-Tree instance, where R is the set of terminals. Let $T = (R \cup S^*, E^*)$ be an optimal solution in which the terminals are leaves. By the discussion of Section 3.2.1, it suffices to construct a tree $W = (F, E_W)$ spanning the set of final Steiner nodes of T, whose imposed w-value on $v \in S^*$ is defined as

$$w(v) \coloneqq |\{(p, q) \in E_W : v \text{ is a node of the } p - q \text{ path in } T[S^*]\}| + \mathbb{1}[v \in F].$$

Following the discussion of Section 3.2.2, we root T at an arbitrary leaf r and decompose it into a set of final-components $T_1, \ldots, T_\tau$, with the property that $r \in T_1$ and for every $i \in [\tau]$, $\bigcup_{j \leq i} T_j$ is connected. We process the trees in increasing index order and do the following. We maintain a tree W' spanning the final Steiner nodes of $T' \coloneqq \bigcup_{j<i} T_j$, and during iteration i, we compute a tree W_i spanning the final Steiner nodes of T_i and then merge it with W', as discussed in Section 3.2.3, to find W''. The resulting tree is a tree spanning the final Steiner nodes of $T'' \coloneqq \bigcup_{j \leq i} T_j$, as demonstrated in Section 3.2.3. The analysis we did now shows that after each iteration i, the resulting tree spanning the final Steiner nodes of $\bigcup_{j \leq i} T_j$ satisfies the desired bound of 1.8917. More precisely, by setting $\psi' = 1.8917$ in Lemma 3.12 and by using Lemma 3.15, we get that after each iteration, the resulting tree has $\nu_{T''}(W'') < 1.8917$. Thus, a straightforward induction shows that the final witness tree W will $\nu_T(W) < 1.8917$ and will span the final Steiner nodes of T. This means that $\psi_G \leq \nu_T(W) < 1.8917$, which implies that $\psi \leq 1.8917$. □

3.2.4 Lower bound for the witness tree generated by Algorithm 2

In addition to an upper bound on ψ, we also give an explicit example of a CA-Node-Steiner-Tree instance for which Algorithm 2 finds a witness tree W that gives $\nu_W(T) > 1.8504$. This shows that our analysis of the approximation factor of Algorithm 1 with the witness tree generated by Algorithm 2 is off by at most 0.0416.

We now describe the CA-Node-Steiner-Tree instance $T = (R \cup S^*, E^*)$, which is a tree; thus the optimal solution is T itself. We will show that Algorithm 2 finds a witness tree spanning the terminals of T that gives $\nu_W(T) > 1.8504$. We now describe the tree T, which consists of five layers:

- the 1^{st} layer contains a single node r.
- the 2^{nd} layer consists of 9 nodes $\{x_1, \ldots, x_9\}$. For each $i \in [9]$, there is an edge (r, x_i).
- the 3^{rd} layer is the set $\bigcup_{i=1}^{9} \{y_{i1}, \ldots, y_{i5}\}$, which consists of 9 groups of 5 nodes. For each $i \in [9]$ and $j \in [5]$, there is an edge (x_i, y_{ij}).
- the 4^{th} layer is the set $\bigcup_{i=1}^{9} \bigcup_{j=1}^{5} \{z_{ij1}, \ldots, z_{ij4}\}$, which consists of 9×5 groups of 4 nodes. For each $i \in [9]$, $j \in [5]$ and $k \in [4]$, there is an edge (y_{ij}, z_{ijk}).
- the 5^{th} layer is the set $R = \bigcup_{i=1}^{9} \bigcup_{j=1}^{5} \bigcup_{k=1}^{4} \{q_{ijk}^{(1)}, q_{ijk}^{(2)}\}$, which consists of $9 \times 5 \times 4 \times 2$ terminals. For each $i \in [9]$, $j \in [5]$ and $k \in [4]$, there is an edge $(z_{ijk}, q_{ijk}^{(1)})$ and an edge $(z_{ijk}, q_{ijk}^{(2)})$.

There are no other edges contained in T. As in Section 3.2.1, the set of final Steiner nodes is the set of vertices in the 4^{th} layer, and since there are exactly two distinct terminals adjacent to each final Steiner node, we remove all the terminals and simply increase the w-value of the each final Steiner node by 1. A pictorial representation of $T \setminus R$ is provided in Figure 3.6.

From now on, we denote the set of vertices of the subtree rooted at x_i as Q_{x_i}. To use Algorithm 2, we root T at a final Steiner node, and without loss of generality, by the symmetry of T, we can root T at z_{222}. Similarly, without loss of generality we can let the ordering of the leaves of T selected at step (1) of Algorithm 2 coincide with the lexicographic ordering of the leaf indices. With this ordering, for each non-final Steiner node $u \in T$, we find the minimum-weight path $P(u)$ described at step (3) of Algorithm 2, which we highlight in Figure 3.7, along with the corresponding w-values.

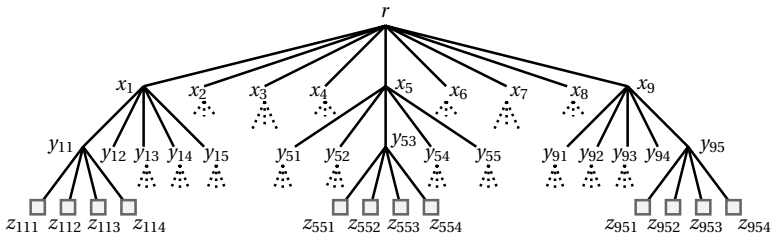


Figure 3.6: The CA-Node-Steiner-Tree instance T with its first four layers of Steiner nodes; the set of terminals is absent while final Steiner nodes are depicted with squares, and many edges are depicted with dotted lines to simplify the figure. Since T is a tree, the solution to the CA-Node-Steiner-Tree problem is trivially T itself.

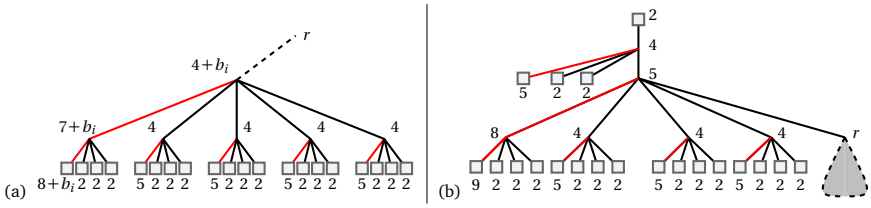


Figure 3.7: (a) The w-values for the nodes of Q_{x_i} are described pictorially. The term b_i is equal to 8 if $i = 1$ and equal to 1 otherwise. The red edges indicate the paths found by Algorithm 2. (b) A clearer explanation of the w-values for every node in Q_{x_2} is given here, as well as the shape of Q_{x_2} when T is rooted at the node z_{222}.

For each Q_{x_i}, $i \in \{3,\dots,9\}$, we have

$$\sum_{v \in Q_{x_i}} H_{w(v)} = 15H_2 + 4H_4 + 5H_5 + H_8 + H_9,$$

and for Q_{x_1} we have

$$\sum_{v \in Q_{x_1}} H_{w(v)} = 15H_2 + 4H_4 + 4H_5 + H_{12} + H_{15} + H_{16}.$$

We now turn to the nodes in the set $T \setminus \bigcup_{i \in [9]\setminus\{2\}} Q_{x_i}$. It is not hard to see that $w(r) = 8$, and

$$\sum_{v \in T \setminus \cup_{i \in [9]\setminus\{2\}} Q_{x_i}} H_{w(v)} = 15H_2 + 4H_4 + 5H_5 + 2H_8 + H_9.$$

Putting everything together, we get that $\nu_W(T)$ is

$$\frac{\sum_{v\in S^*} H_{w(v)}}{|S^*|} = \frac{\sum_{v\in T\setminus\cup_{i\in[9]\setminus\{2\}} Q_{x_i}} H_{w(v)} + \sum_{i=3}^{9}\sum_{v\in Q_{x_i}} H_{w(v)} + \sum_{v\in Q_{x_1}} H_{w(v)}}{235}$$
$$= \frac{135H_2 + 36H_4 + 44H_5 + 9H_8 + 8H_9 + H_{12} + H_{15} + H_{16}}{235} > 1.8504.$$

We conclude that for this particular instance, the analysis of of Algorithm 1 with the witness tree generated by Algorithm 2 will necessarily give an approximation factor strictly larger than 1.8504.

3.3 Improved approximation for CA-Node-Steiner-Tree

Our result is based on a better construction of witness trees. At a very high level, the witness tree constructions used previously in the literature use a *marking-and-contraction* approach, that can be summarized as follows. First, root the given tree T at some internal Steiner node. Then, every Steiner node v chooses (*marks*) an edge which connects to one of its children: this identifies a path from v to a terminal. Contracting the edges along this path yields a witness tree W. The way this marking choice is made varies: it is random in [16], it is biased depending on the nature of the children in [15], and lastly, it is deterministic and taking into account the structure of T in Section 3.2. However, all such constructions share the fact that decisions can be thought of as being taken "in one shot", at the same time for all Steiner nodes.

Instead, here we consider a bottom-up approach for the construction of our witness tree, where a node takes a marking decision only after the decisions of its children have been made. A sequential approach of this kind allows a node to have a more precise estimate on the impact of its own decision to the overall non-linear objective function cost, but it becomes more challenging to analyze. Overcoming this challenge is the main technical contribution of this section, and the insight behind our improved upper-bound on ψ.

The goal of this section is to prove that $\psi < 1.8596$, and thus Theorem 3.10. From now on, we assume we are given a tree $T = (R \cup S^*, E^*)$, where each Steiner node is adjacent to at most two terminals. We split the tree T into final-components $T_1, \ldots, T_\tau$ just as in Section 3.2.2.

3.3.1 Computing a witness tree W_i for a component T_i.

Here we deal with a component T_i rooted at r_i, and describe how to construct a witness tree W_i. If T_i is a single edge $e = r_i v$, we simply let $W_i = (\{r_i, v\}, \{r_i v\})$.

Now we assume that T_i is not a single edge. We will construct a witness tree with a bottom-up procedure. At a high level, each node $u \in T_i \setminus r_i$ looks at the subtree Q_u of T_i rooted at u, and constructs a portion of the witness tree: namely, a subtree $\overline{W}^u$ spanning the leaves of Q_u (note that, in case the degree of u is 1 in Q_u, we do not consider u to be a leaf of Q_u but just its root). Assume u has children $u_1, \ldots, u_k$. Because of the bottom-up procedure, each child u_j has already constructed a subtree $\overline{W}^{u_j}$. That is, u has to decide how to join these subtrees to get $\overline{W}^u$.

To describe how this is done formally, we first need to introduce some more notation. For every node $u \in T_i \setminus F$, we select one of its children as the "marked child" of u (according to some rule that we will define later). In this way, for every $u \in T_i$ there is a unique path along these marked children to a leaf. We denote this path by $P(u)$, and we let $\ell(u)$ denote the leaf descendent of this path. For final nodes $u \in F$, we define $\ell(u) := u$ and $P(u) := u$. For a subtree Q_u of T_i rooted at u and a witness tree $\overline{W}^u$ over the leaves of Q_u, let $\overline{w}^u$ be the vector imposed on the nodes of Q_u by $\overline{W}^u$ according to (3.1). Next, we define the following quantity (which, roughly speaking, represents the cost-increase incurred after increasing $\overline{w}^u(v)$ for each $v \in P(u) \setminus \ell(u)$ for the $(j+1)^{th}$ time):

$$C_j^u := \sum_{v \in P(u) \setminus \ell(u)} \left(H_{\overline{w}^u(v)+j+1} - H_{\overline{w}^u(v)+j} \right) = \sum_{v \in P(u) \setminus \ell(u)} \frac{1}{\overline{w}^u(v) + j + 1}$$

We can now describe the construction of the witness tree more formally. We begin by considering the leaves of T_i; for a final node (leaf) u, we define a witness tree on the (single) leaf of Q_u as $\overline{W}^u = (\{u\}, \emptyset)$. For a non-final node u, with children $u_1, \ldots, u_k$ and corresponding witness trees $\overline{W}^{u_1}, \ldots, \overline{W}^{u_k}$, we select a marked child u_m for u as outlined in Algorithm 3, setting $\phi = 1.86 - \frac{1}{2100}$ and $\delta = \frac{97}{420}$. With this choice, we compute $\overline{W}^u$ by joining the subtrees $\overline{W}^{u_1}, \ldots, \overline{W}^{u_k}$ via the edges $\ell(u_m)\ell(u_j)$ for $j \neq m$. Finally, let v be the unique child of r_i. We let W_i be equal to the tree $\overline{W}^v$ plus the extra edge $\ell(v) r_i$, to account for the fact that r_i is also a final node.

Algorithm 3: Computing the tree $\overline{W}^u$

u has Steiner node children $u_1, u_2, \ldots, u_k$, and $\overline{W}^{u_j}$ have been defined
if *$u_1, \ldots, u_k$ are all non-final,* **then**
  The *marked* child is u_m, minimizing $C_1^{u_m}$
else
  Assume $\{u_1, \ldots, u_{k_1}\}$, $1 \le k_1 \le k$, are final node children of u
  if $k_1 = k$, *or, for all* $j \in \{k_1+1, \ldots, k\}$, $C_1^{u_j} \ge \phi - \delta - H_2$ **then**
    The *marked* child of u is u_m for $1 \le m \le k_1$ such that $C_1^{u_m}$ is minimized.
  end
  if *There is a* $j \in \{k_1+1, \ldots, k\}$ *such that* $C_1^{u_j} < \phi - \delta - H_2$ **then**
    The *marked* child of u is u_m for $k_1 < m \le k$ such that $C_1^{u_m}$ is minimized.
  end
end
$\overline{W}^u \leftarrow \left(\bigcup_{j=1}^k V[Q_{u_j}], \bigcup_{j=1}^k \overline{W}^{u_j} \bigcup_{j \ne m} \{\ell(u_m)\ell(u_j)\}\right)$
Return $\overline{W}^u$

3.3.2 Bounding the cost of W_i

It will be convenient to introduce the following definitions. For a component T_i and a node $u \in T_i \setminus r_i$, we let W^u be the tree $\overline{W}^u$ plus one extra edge e^u, defined as follows. Let $a(u)$ be the first ancestor node of u with $\ell(a(u)) \ne \ell(u)$ (recall $\ell(r_i) = r_i$). We then let the edge $e^u := \ell(u)\ell(a(u))$. We denote by w^u the vector imposed on the nodes of Q_u by $W^u := \overline{W}^u + e^u$. Note that, with this definition, $W_i = W^v$ for v being the unique child of r_i.

We now state two useful lemmas. The first one relates the functions w^u and w^{u_j} for a child u_j of u. The statements (1)-(3) below can be proved similarly to Lemma 3.13.

Lemma 3.16. *Let $u \in T_i \setminus r_i$ have children $u_1, \ldots, u_k$, and u_1 be its marked child. Then:*

1. $w^u(u) = k$.
2. *For every $j \in \{2, \ldots, k\}$ and every node $v \in Q_{u_j}$, $w^u(v) = w^{u_j}(v)$.*
3. *For every $v \in Q_{u_1} \setminus P(u_1)$, $w^u(v) = w^{u_1}(v)$.*

4. *For every* $v \in P(u_1)$, $w^u(v) = w^{u_1}(v) + k - 1$.

5. $\sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^u(v)} = \sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^{u_1}(v)} + \sum_{j=1}^{k-1} C_j^{u_1}$.

Proof. Recall W^u is the union of $\overline{W}^{u_1}, \ldots, \overline{W}^{u_k}$ plus the edges e^u, and $\ell(u_1)\ell(u_j)$ for $j = 2, \ldots, k$.

4. To see this, recall that if u_1 is the marked child of u, then W^u is equal to $\overline{W}^{u_1}, \ldots, \overline{W}^{u_k}$ plus the edges $\ell(u_1)\ell(u_j)$ $j \neq 1$, and e^u.

5. Additionally, we can see

$$\begin{aligned}
&\sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^{u_1}(v)} + \sum_{j=1}^{k-1} C_j^{u_1} \\
&= \sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^{u_1}(v)} + \sum_{j=1}^{k-1} \sum_{v \in P(u_1) \setminus \ell(u_1)} \left(H_{w^{u_1}(v)+j} - H_{w^{u_1}(v)+j-1} \right) \\
&= \sum_{v \in P(u_1) \setminus \ell(u_1)} \left(H_{w^{u_1}(v)} + \sum_{j=1}^{k-1} \left(H_{w^{u_1}(v)+j} - H_{w^{u_1}(v)+j-1} \right) \right) \\
&= \sum_{v \in P(u_1) \setminus \ell(u_1)} \left(H_{w^{u_1}(v)} + H_{w^{u_1}(v)+k-1} - H_{w^{u_1}(v)} \right) \\
&= \sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^{u_1}(v)+k-1} = \sum_{v \in P(u_1) \setminus \ell(u_1)} H_{w^u(v)}
\end{aligned}$$

Where the last equality above follows from (d). □

Next lemma relates the "increase" of cost C_j^u to the *degree* of some nodes in T_i.

Lemma 3.17. *Let* $u \in T_i \setminus r_i$ *have children* $u_1, \ldots, u_k$, *and* u_1 *be its marked child. Then,* $C_1^u = C_k^{u_1} + \frac{1}{k+1}$. *Furthermore, if* u_1 *is non-final and has degree* d *in* T_i, *then: 1)* $\sum_{j=1}^{k} (C_j^{u_1} - C_1^{u_j}) \leq \sum_{j=1}^{k-1} \left(\frac{1}{d+j} - \frac{1}{d} \right)$; *2)* $H_{w^u(\ell(u_1))} - H_{w^{u_1}(\ell(u_1))} \leq \sum_{j=1}^{k-1} \frac{1}{d+j}$.

Proof. 1. First observe that since $C_1^{u_1} = \min_{j \in [k]} C_1^{u_j}$, we have $C_j^{u_1} - C_1^{u_j} \leq C_j^{u_1} - C_1^{u_1}$. Consider $j \geq 1$, $C_j^{u_1} - C_1^{u_1}$ is equal to

$$\begin{aligned}
&= \sum_{v \in P(u_1) \setminus \ell(u)} \left(H_{w^{u_1}(v)+j} - H_{w^{u_1}(v)+j-1} - H_{w^{u_1}(v)+1} + H_{w^{u_1}(v)} \right) \\
&= \sum_{v \in P(u_1) \setminus \ell(u)} \left(\frac{1}{w^{u_1}(v)+j} - \frac{1}{w^{u_1}(v)+1} \right) \leq \frac{1}{w^{u_1}(u_1)+j} - \frac{1}{w^{u_1}(u_1)+1}
\end{aligned}$$

Where the inequality follows since every term in the sum is negative. We know that $w^{u_1}(u_1) = d-1$ by Lemma 3.16.(a), therefore, $C_j^{u_1} - C_1^{u_1} \leq \frac{1}{d+j-1} - \frac{1}{d}$, and the claim is proven by summing over $j = 1, \ldots, k$.

2. To prove the second inequality, first observe that $w^u(\ell(u_1)) = w^{u_1}(\ell(u_1)) + k - 1$. This follows by recalling that W^u is equal to $\overline{W}^{u_1}, \ldots, \overline{W}^{u_k}$ plus the edges $\ell(u_1)\ell(u_j)$ for $j \neq 1$, and e^u. Thus, $H_{w^u(\ell(u_1))} - H_{w^{u_1}(\ell(u_1))} = H_{w^{u_1}(\ell(u_1))+k-1} - H_{w^{u_1}(\ell(u_1))} = \sum_{i=1}^{k-1} \frac{1}{w^{u_1}(\ell(u_1))+i}$. Recall u_1 is not a final node, so $w^{u_1}(\ell(u_1)) > d$. Therefore,

$$\sum_{i=1}^{k-1} \frac{1}{w^{u_1}(\ell(u_1)) + i} \leq \sum_{i=1}^{k-1} \frac{1}{d+i}.$$

□

Finally, the following lemmas will be useful tools.

Lemma 3.18. *Let $k_1 \leq k \in \mathbb{Z}_{>0}$. Let $\delta = \frac{97}{420}$, and $\phi = 1.86 - \frac{1}{2100}$. Let $\beta(k)$ be equal to 0 for $k = 0, \ldots, 8$, and $\frac{1}{3} - \delta$ for $k \geq 9$. Then the following inequalities hold:*

1. $-(k-1)\delta + \sum_{j=1}^{k-1} \left(\frac{2}{d+j} - \frac{1}{d}\right) + H_{k+1} \leq \phi - \beta(k)$
2. $(k-1)H_2 + 2H_{k+1} \leq (k+1)\phi - \beta(k) - \delta$;
3. $-(k-k_1-1)\delta + k_1 H_2 + H_{k+1} + \sum_{j=1}^{k-1} \left(\frac{2}{8+j} - \frac{1}{8}\right) + \frac{k_1}{8} < (k_1+1)\phi - \beta(k)$;
4. $-(k-1)\delta + H_{k+1} + k_1\phi + \sum_{j=1}^{k-1} \frac{1}{16+j} < (k_1+1)\phi - \beta(k)$.

Proof. 1. We can define the terms of the desired inequality to be equal to

$$f(k) = \sum_{i=1}^{k-1} \left(\frac{2}{d+i} - \frac{1}{d}\right) + H_{k+1} - (k-1)\delta - \phi + \beta(k).$$

Thus, if we show that $f(k) \leq 0$ we have proven the claim. Observe that

$$f(k+1) - f(k) = \frac{2}{d+k} - \frac{1}{d} + \frac{1}{k+2} - \delta + \beta(k+1) - \beta(k).$$

We wish to show that $\frac{2}{d+k} - \frac{1}{d} < \frac{1}{5k}$. To see this, first note the following

$$\frac{2}{d+k} - \frac{1}{d} = \frac{d-k}{d(d+k)}$$

Thus, we can consider

$$\begin{aligned} d(d+k)-5k(d-k) = d^2+(2k)^2-4dk+k^2 = (d-2k)^2+k^2 > 0 \\ \Rightarrow d(d+k) > 5k(d-k) \\ \Rightarrow \frac{1}{5k} > \frac{d-k}{d(d+k)} = \frac{2}{d+k} - \frac{1}{d} \end{aligned}$$

And thus $f(k+1)-f(k) < \frac{1}{5k} + \frac{1}{k+2} - \delta + \beta(k+1) - \beta(k)$. Therefore for $k \geq 4$, $f(k+1)-f(k) < 0$.

Furthermore observe that if $k \in \{1,2\}$, then $f(k+1)-f(k) = \frac{2}{d+k} - \frac{1}{d} + \frac{1}{k+2} - \delta + \geq \frac{1}{k+2} - \delta > 0$. Therefore, it suffices to prove $f(k) \leq 0$ only for $k \in \{3,4\}$.

- if $k=3$, then first define $g(d) := \frac{2}{d+1} + \frac{2}{d+2} - \frac{2}{d}$. Then

  $$g(d+1)-g(d) = \frac{2}{d+3} - \frac{4}{d+1} + \frac{2}{d}$$

  One can easily compute that for $d = 1,2$, we have $g(d+1)-g(d) > 0$, and for $d \geq 3$, we have $g(d+1)-g(d) \leq 0$. Therefore $g(d) \leq g(3) = \frac{7}{30}$. By the inequality shown above we can see

  $$f(3) = \frac{2}{d+1} + \frac{2}{d+2} - \frac{2}{d} + H_4 - 2\delta \leq \frac{7}{30} + H_4 - 2\delta < 1.855 < \phi$$

- if $k=4$, the first define $g(d) := \frac{2}{d+1} + \frac{2}{d+2} + \frac{2}{d+3} - \frac{3}{d}$, then we have

  $$g(d+1)-g(d) = \frac{2}{d+4} + \frac{3}{d} - \frac{5}{d+1}$$

  One can easily compute that for $d < 4$, we have $g(d+1)-g(d) > 0$, and for $d \geq 4$ we have $g(d+1)-g(d) \leq 0$. Therefore $g(d) \leq g(4) = \frac{113}{420}$. By the inequality shown above we can see

  $$f(4) = \frac{2}{d+1} + \frac{2}{d+2} + \frac{2}{d+3} - \frac{3}{d} + H_5 - 3\delta \leq \frac{113}{420} + H_5 - 3\delta = \phi$$

2. We reorganize the terms of the inequality so all are on the left side, and define $f(k) := (k-1)H_2 + 2H_{k+1} - (k+1)\phi + \beta(k) + \delta$. We will show that $f(k) \leq 0$. First, note that $f(k+1)-f(k) = (H_2 - \phi) + \frac{2}{k+2} + \beta(k+1) - \beta(k)$ from which it is clear that $f(k+1)-f(k) > 0$ if and only if $k < 4$. Therefore $f(k) \leq f(4) = 0$, and the claim is proven.

3. Observe that

$$k_1 H_2 + H_{k+1} + \sum_{i=1}^{k-1}\left(\frac{2}{8+i} - \frac{1}{8}\right) + \frac{k_1}{8} - (k-k_1-1)\delta - (k_1+1)\phi + \beta(k)$$
$$= k_1(H_2 + \frac{1}{8} + \delta - \phi) + H_{k+1} + \sum_{i=1}^{k-1}\left(\frac{2}{8+i} - \frac{1}{8}\right) - (k-1)\delta - \phi + \beta(k)$$

Since $H_2 + \frac{1}{8} + \delta < \phi$ we have:

$$< H_{k+1} + \sum_{i=1}^{k-1}\left(\frac{2}{8+i} - \frac{1}{8}\right) - (k-1)\delta - \phi + \beta(k)$$

We show that $\sum_{i=1}^{k-1}\left(\frac{2}{8+i} - \frac{1}{8}\right) + H_{k+1} - (k-1)\delta + \beta(k) - \phi < 0$.

Now let $f(k) \coloneqq \sum_{i=1}^{k-1}\left(\frac{2}{8+i} - \frac{1}{8}\right) + H_{k+1} - (k-1)\delta - \phi + \beta(k)$, and consider $f(k+1) - f(k) = \frac{1}{k+2} + \frac{2}{8+k} - \frac{1}{8} - \delta + \beta(k+1) - \beta(k)$. Observe that $f(k+1) - f(k)$ is positive if and only if $k < 4$. Therefore

$$f(k) \leq f(4) = -\frac{1109}{27720} < 0$$

4. We can see

$$-(k-1)\delta + H_{k+1} + k_1\phi + \sum_{j=1}^{k-1}\frac{1}{16+j} - (k_1+1)\phi + \beta(k)$$
$$= -(k-1)\delta + H_{k+1} + \sum_{j=1}^{k-1}\frac{1}{16+j} - \phi + \beta(k)$$

We show that $f(k) = -(k-1)\delta + H_{k+1} + \sum_{j=1}^{k-1}\frac{1}{16+j} - \phi + \beta(k) < 0$. Note that $f(k+1) - f(k) = \frac{1}{k+2} + \frac{1}{16+k} - \delta + \beta(k+1) - \beta(k)$ is negative if and only if $k \geq 4$ and $k \neq 8$. Therefore $f(k)$ is upper-bounded by

$$\max\{f(4), f(9)\}$$
$$= \max\{H_5 + \sum_{j=1}^{3}\frac{1}{16+j} - 3\delta - \phi, H_{10} + \sum_{j=1}^{8}\frac{1}{16+j} - 8\delta - \phi + \beta(9)\}$$
$$\approx -0.102036 < 0.$$

□

Key Lemma To simplify our analysis, we define $h_{W^u}(Q_u) := \sum_{\ell \in Q_u} H_{w^u(\ell)}$, and we let $|Q_u|$ be the number of nodes in Q_u. The next lemma is the key ingredient to prove Theorem 3.10.

Lemma 3.19. *Let $\delta = \frac{97}{420}$ and $\phi = 1.86 - \frac{1}{2100}$. Let $u \in T_i \setminus r_i$ and k be the number of its children. Let $\beta(k)$ be equal to 0 for $k = 0, \dots, 8$ and $\frac{1}{3} - \delta$ for $k \geq 9$. Then*

$$h_{W^u}(Q_u) + C_1^u + \delta + \beta(k) \leq \phi \cdot |Q_u|$$

Proof. The proof of Lemma 3.19 will be by induction on $|Q_u|$. The base case is when $|Q_u| = 1$, and hence u is a leaf of T_i. Therefore, W^u is just the edge e^u, and so by definition of w^u we have $w^u(u) = 2$. We get $h_{W^u}(Q_u) = 1.5$, $C_1^u = 0$, $\beta(k) = 0$ and the claim is clear in the base case.

For the induction step: suppose that u has children $u_1, \dots, u_k$. We will initially distinguish 2 cases: (i) u has no children that are final nodes; (ii) u has some child that is a final node (which is then again broken into subcases).

Case (i): No children of u are final. According to Algorithm 3, we mark the child u_m of u that minimizes $C_1^{u_j}$. Without loss of generality, let $u_m = u_1$. Furthermore, let $\ell := \ell(u_1)$. We note the following.

$$h_{W^u}(Q_u) = \sum_{j=1}^{k} h_{W^u}(Q_{u_j}) + H_{w^u(u)}$$

By applying Lemma 3.16.(a) we have $H_{w^u(u)} = H_k$. By Lemma 3.16.(b) we see $h_{W^u}(Q_{u_j}) = h_{W^{u_j}}(Q_{u_j})$ for $j \geq 2$. Using Lemma 3.16.(c) and (d) we get $h_{W^u}(Q_{u_1}) = h_{W^{u_1}}(Q_{u_1}) + \sum_{j=1}^{k-1} C_j^{u_1} + H_{w^u(\ell)} - H_{w^{u_1}(\ell)}$. Therefore:

$$h_{W^u}(Q_u) = \sum_{j=1}^{k} h_{W^{u_j}}(Q_{u_j}) + \sum_{j=1}^{k-1} C_j^{u_1} + H_k + H_{w^u(\ell)} - H_{w^{u_1}(\ell)}$$

We apply our inductive hypothesis on $Q_{u_1}, \dots, Q_{u_k}$, and use $\beta(j) \geq 0$ for all j:

$$\begin{aligned} h_{W^u}(Q_u) &\leq \sum_{j=1}^{k} \left(\phi |Q_{u_j}| - \delta - C_1^{u_j} \right) + \sum_{j=1}^{k-1} C_j^{u_1} + H_k + H_{w^u(\ell)} - H_{w^{u_1}(\ell)} \\ &= \phi(|Q_u| - 1) - k\delta - C_k^{u_1} + \sum_{j=1}^{k} \left(C_j^{u_1} - C_1^{u_j} \right) + H_k + H_{w^u(\ell)} - H_{w^{u_1}(\ell)} \end{aligned}$$

Using Lemma 3.17, we get

$$\leq \phi(|Q_u|-1) - k\delta - C_1^u + \sum_{j=1}^{k-1}\left(\frac{1}{d+j} - \frac{1}{d}\right) + H_{k+1} + \sum_{j=1}^{k-1}\frac{1}{d+j}$$
$$\leq \phi|Q_u| - \delta - C_1^u - \beta(k)$$

where the last inequality follows since one checks that for any $k \geq 1$ and $d \geq 2$ we have $-\phi - (k-1)\delta + \sum_{j=1}^{k-1}\left(\frac{1}{d+j} - \frac{1}{d}\right) + H_{k+1} + \sum_{j=1}^{k-1}\frac{1}{d+j} \leq -\beta(k)$.

We show this inequality in Lemma 3.18.(a).

Case(ii): u has a final child. We note the following.

$$h_{W^u}(Q_u) = \sum_{j=1}^{k} h_{W^u}(Q_{u_j}) + H_{w^u(u)}$$

Let $\ell := \ell(u_m)$. By Lemma 3.16.(a) we can see $H_{w^u(u)} = H_k$, and $H_{w^{u_j}(u_j)} = H_2$ for $j = 1, \ldots, k_1$. By Lemma 3.16.(b), we can see $h_{W^u}(Q_{u_j}) = h_{W^{u_j}}(Q_{u_j})$ for $j \neq m$, and by Lemma 3.16.(c) and (e) we can see $h_{W^u}(Q_{u_m}) = h_{W^{u_m}}(Q_{u_m}) + \sum_{j=1}^{k-1} C_j^{u_m} + H_{w^u(\ell)} - H_{w^{u_m}(\ell)}$. Therefore:

$$h_{W^u}(Q_u) = \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + \sum_{j=1}^{k-1} C_j^{u_m} + H_k + H_{w^u(\ell)} - H_{w^{u_m}(\ell)}$$

By Algorithm 3 we mark a final child u_m of u depending on the value of $\min_{j\in\{k_1+1,\ldots,k\}} C_1^{u_j}$. We consider these cases.

Case (ii).(a) If $k_1 = k$ or if $\min_{j\in\{k_1+1,\ldots,k\}} C_1^{u_j} \geq \phi - \delta - H_2$, final node $u_1 = u_m$ is the marked child of u according to Algorithm 3. Since u_1 is final, $C_j^{u_1} = 0$ and, $h_{W^{u_1}}(Q_{u_1}) = H_2$. Finally, applying Lemma 3.16.(d) to Q_{u_1}, we see $h_{W^u}(Q_{u_1}) = H_{k+1}$. Therefore:

$$h_{W^u}(Q_u) = \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + (k_1-1)H_2 + H_k + H_{k+1}$$

We apply our inductive hypothesis on $Q_{u_{k_1+1}}, \ldots, Q_{u_k}$, and use $\beta(j) \geq 0$ for all $j \geq 0$:

$$h_{W^u}(Q_u) \leq \sum_{j>k_1}\left(\phi|Q_{u_j}| - C_1^{u_j} - \delta\right) + (k_1-1)H_2 + H_k + H_{k+1}$$
$$= \phi(|Q_u| - k_1 - 1) - \sum_{j>k_1} C_1^{u_j} - (k-k_1)\delta + (k_1-1)H_2 + H_k + H_{k+1}$$

Applying the assumption that $\min_{j\in\{k_1+1,\dots,k\}} C_1^{u_j} \geq \phi - \delta - H_2$:

$$\leq \phi \sum_{j>k_1} |Q_{u_j}| - (k-k_1)(\phi-\delta-H_2) - (k-k_1)\delta + (k_1-1)H_2 + H_k + H_{k+1}$$

$$= \phi(|Q_u| - k - 1) + (k-1)H_2 + 2H_{k+1} - \frac{1}{k+1}$$

Using Lemma 3.18.(b), and the fact that $C_1^u = \frac{1}{k+1}$, we have

$$\leq \phi(|Q_u| - k - 1) - \delta + (k+1)\phi - \beta(k) - \frac{1}{k+1} = \phi|Q_u| - C_1^u - \delta - \beta(k).$$

Case (ii).(b) In this case we assume $\min_{j\in\{k_1+1,\dots,k\}} C_1^{u_j} < \phi - H_2 - \delta$ and, by Algorithm 3, we mark some child u_m for $k_1+1 \leq m \leq k$. Without loss of generality we will assume that $m = k$. We let d_x denote the degree of a non-final node x in T_i. Let $d_{u_k} := d$.

We now consider by cases if the marked child of u_k, denoted v, is a final node.

Case (ii).(b).i: v is a final node. Since v is final, we have $\ell = v$. By Lemma 3.16.(a), $H_{w^{u_k}(u_k)} = H_{d-1}$. By Lemma 3.16.(d) we have $H_{w^u(u_k)} = H_{k+d-2}$. Since v is a final node we know $C_j^{u_k} = \frac{1}{w^{u_k}(u_k)+j} = \frac{1}{d+j-1}$. Therefore,

$$h_{W^u}(Q_u) = \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + \sum_{j=1}^{k-1} \frac{1}{d+j-1} + H_k + H_{w^u(v)} - H_{w^{u_k}(v)}$$

Since v is final, $H_{w^v(v)} = H_2$. By Lemma 3.16.(d) we have, $H_{w^{u_k}(v)} = H_d$, and $H_{w^u(v)} = H_{d+k-1}$.Therefore,

$$h_{W^u}(Q_u) = \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \frac{1}{d+j-1} + \sum_{j=1}^{k-1} \frac{1}{d+j}$$

$$= \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \frac{2}{d+j} - \frac{1}{d+k-1} + \frac{1}{d}$$

Observe that since $C_1^{u_k} = \frac{1}{d} < \phi - H_2 - \delta < 0.1286 < \frac{1}{7}$, we have $d \geq 8$.

$$h_{W^u}(Q_u) \leq \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \frac{2}{8+j} - \frac{1}{d+k-1} + \frac{1}{8}$$

We apply our inductive hypothesis on $Q_{u_{k_1+1}}, \ldots, Q_{u_k}$, and use $\beta(j) \geq 0$ for all $j \geq 0$.

$$\leq \sum_{j>k_1} (\phi|Q_{u_j}| - \delta - C_1^{u_j}) + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \frac{2}{8+j} - \frac{1}{d+k-1} + \frac{1}{8}$$

We assumed that $C_1^{u_k} = \min_{j\in\{k_1+1,\ldots,k\}} C_1^{u_j}$. Since the marked child of u_k is a final node we know $C_1^{u_k} = \frac{1}{w^{u_k}(u_k)+1} = \frac{1}{d}$. Therefore:

$$\leq \sum_{j>k_1} (\phi|Q_{u_j}| - \delta - \frac{1}{8}) + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \frac{2}{8+j} - \frac{1}{d+k-1} + \frac{1}{8}$$

$$= \sum_{j>k_1} \phi|Q_{u_j}| - (k-k_1)\delta + k_1 H_2 + H_k + \sum_{j=1}^{k-1} \left(\frac{2}{8+j} - \frac{1}{8}\right) - \frac{1}{d+k-1} + \frac{k_1}{8}$$

Using Lemma 3.18.(c), we see

$$< \sum_{j>k_1} \phi|Q_{u_j}| - \frac{1}{k+1} - \frac{1}{d+k-1} + (k_1+1)\phi - \delta - \beta(k)$$

$$= \phi|Q_u| - \frac{1}{k+1} - \frac{1}{d+k-1} - \delta - \beta(k)$$

$$= \phi|Q_u| - \frac{1}{w^u(u)+1} - \frac{1}{w^u(u_k)+1} - \delta - \beta(k) = \phi|Q_u| - C_1^u - \delta - \beta(k)$$

Where the second equality follows from Lemma 3.16.(a) and (e). And the claim is proven.

Case: (ii).(b).ii: v is not a final node. In order to complete the proof in this case we make use of the following lemma.

Claim 3.1. *, Let $2 \leq x, y \in \mathbb{Z}_{>0}$. Let $\delta = \frac{97}{420}$, and $\phi = 1.86 - \frac{1}{2100}$.*
If $\frac{1}{x} + \frac{1}{x+y-2} < \phi - H_2 - \delta$. Then $x + y \geq 18$.

Proof. Assume that $x + y < 18$. Since $x, y \geq 2$, then

$$\frac{1}{x} + \frac{1}{x+y-2} \geq \frac{2}{x+y-2} \geq \frac{2}{15} = 0.1\bar{3} > 0.1286 > \phi - H_2 - \delta$$

which is a contradiction. □

Since v is not final, we know $C_j^{u_k} \geq \frac{1}{w^{u_k}(u_k)+j} + \frac{1}{w^{u_k}(v)+j}$. Therefore, $\phi - H_2 - \delta > C_1^{u_k} \geq \frac{1}{d_{u_k}} + \frac{1}{d_{u_k}+d_v-2}$. Applying Claim 3.1 we see $d_{u_k} + d_v \geq 18$, and by Lemma 3.16.(d) we see $w^{u_k}(\ell) \geq d_{u_k} + d_v - 2 \geq 16$. Therefore, $H_{w^u(\ell)} - H_{w^{u_k}(\ell)} = \sum_{j=1}^{k-1} \frac{1}{w^{u_k}(\ell)+j} \leq \sum_{j=1}^{k-1} \frac{1}{16+j}$.

$$h_{W^u}(Q_u) \leq \sum_{j>k_1} h_{W^{u_j}}(Q_{u_j}) + k_1 H_2 + \sum_{j=1}^{k-1} C_j^{u_k} + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

We apply the inductive hypothesis to $Q_{u_{k_1+1}}, \ldots, Q_{u_k}$, and that $\beta(j) \geq 0$ for all $j \geq 0$:

$$\leq \sum_{j>k_1} \left(\phi|Q_{u_j}| - C_1^{u_j} - \delta\right) + k_1 H_2 + \sum_{j=1}^{k-1} C_j^{u_k} + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

We apply the assumption $C_1^{u_k} = \min_{j\in\{k_1+1,\ldots,k\}} C_1^{u_j}$:

$$\leq \sum_{j>k_1} \left(\phi|Q_{u_j}| - \delta\right) - (k-k_1)C_1^{u_k} + k_1 H_2 + \sum_{j=1}^{k-1} C_j^{u_k} + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

$$= \sum_{j>k_1} (\phi|Q_{u_j}| - \delta) + C_1^{u_k}(k_1 - 1) + k_1 H_2 + \sum_{j=1}^{k-1} \left(C_j^{u_k} - C_1^{u_k}\right) + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

Using $C_1^{u_k} < \phi - H_2 - \delta$:

$$< \sum_{j>k_1} (\phi|Q_{u_j}| - \delta) + k_1(\phi - H_2 - \delta) - C_1^{u_k} + k_1 H_2 + \sum_{j=1}^{k-1} \left(C_j^{u_k} - C_1^{u_k}\right) + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

$$= \sum_{j>k_1} \phi|Q_{u_j}| + k_1\phi - k\delta - C_1^{u_k} + \sum_{j=1}^{k-1} \left(C_j^{u_k} - C_1^{u_k}\right) + H_k + \sum_{j=1}^{k-1} \frac{1}{16+j}$$

Therefore, we can apply Lemma 3.18.(e) to see the following

$$\leq \phi(k_1 + 1 + \sum_{j>k_1} |Q_{u_j}|) - \delta - C_1^{u_k} + \sum_{j=1}^{k-1} \left(C_j^{u_k} - C_1^{u_k}\right) - \frac{1}{k+1} - \beta(k)$$

$$= \phi|Q_u| - \delta - C_1^{u_k} + \sum_{j=1}^{k-1} \left(C_j^{u_k} - C_1^{u_k}\right) - \frac{1}{k+1} - \beta(k)$$

Where the equality above follows since $\sum_{j>k_1}|Q_{u_j}| = |Q_u| - k_1 - 1$. We can apply $C_j^{u_k} \le C_1^{u_k}$ to see the claim

$$\le \phi|Q_u| - \delta - C_k^{u_k} + \sum_{j=1}^{k}\left(C_j^{u_k} - C_1^{u_k}\right) - \frac{1}{k+1} - \beta(k)$$
$$\le \phi|Q_u| - \delta - C_k^{u_k} - \frac{1}{k+1} - \beta(k) = \phi|Q_u| - \delta - C_1^{u} - \beta(k).$$

□

3.3.3 Merging and bounding the cost of W

Once the $\{W_i\}_{i\ge 1}$ are computed for each component T_i, we let the final witness tree be simply the union $W = \cup_i W_i$. Our goal now is to prove the following.

Lemma 3.20. $\nu_T(W) \le \phi = 1.86 - \frac{1}{2100}$.

Proof. Recall that we decomposed T into components $\{T_i\}_{i=1}^{\tau}$, such that $\cup_{j\le i} T_j$ is connected for all $i \in [\tau]$. For a given i, define $T' = \cup_{j<i} T_j$, $W' = \cup_{j<i} W_i$, and let w' be the vector imposed on the nodes of T' by W' (for $i = 1$, set $T' = \emptyset$, $W' = \emptyset$, and $w' = 0$). Finally, define $W'' = W_i \cup W'$ and let w'' be the vector imposed on the nodes of $T'' := T' \cup T_i$. By induction on i, we will show that $\nu_{T''}(W'') \le \phi$. The statement will then follow by taking $i = \tau$. Recall that, for any i, r_i is adjacent to a single node v in T_i, and $W_i = W^v$.

First consider $i = 1$. Hence, $W'' = W_1 = W^v$ and $w''(r_1) = 2$. By applying Lemma 3.19 to the subtree Q_v we get

$$\sum_{u\in T''} H_{w''(u)} = h_{W^v}(Q_v) + H_{w''(r_i)} \le \phi(|Q_v|) + H_2 \le \phi(|Q_v|+1) \Rightarrow \nu_{T''}(W'') \le \phi$$

Now consider $i > 1$. In this case, $w''(r_i) = w'(r_i) + 1 \ge 3$. Therefore:

$$\sum_{u\in T''} H_{w''(u)} = \sum_{u\in T_i\setminus r_i} H_{w^v(u)} + \sum_{u\in T'} H_{w'(u)} - H_{w'(r_i)} + H_{w'(r_i)+1}$$
$$= \sum_{u\in T_i\setminus r_i} H_{w^v(u)} + \sum_{u\in T'} H_{w'(u)} + \frac{1}{w'(r_i)+1} \le \sum_{u\in T_i\setminus r_i} H_{w^v(u)} + \sum_{u\in T'} H_{w'(u)} + \frac{1}{3}$$

If v is a final node, then $\sum_{u\in T_i\setminus r_i} H_{w^v(u)} = H_{w^v(v)} = H_2$ and by induction

$$\sum_{u\in T''} H_{w''(u)} \le H_3 + \sum_{u\in T'} H_{w'(u)} \le \phi|T''| \Rightarrow \nu_{T''}(W'') \le \phi$$

If v is not a final node, then by induction on T' and by applying Lemma 3.19 to the subtree Q_v, assuming that v has k children, we can see

$$\sum_{u \in T''} H_{w''(u)} \leq \phi |T''| - C_1^v - \delta - \beta(k) + \frac{1}{3} \leq \phi |T''| - \frac{1}{k+1} - \delta - \beta(k) + \frac{1}{3}$$

If $1 \leq k \leq 8$, then $\beta(k) = 0$, but we have $\frac{1}{3} < 431/1260 = \frac{1}{9} + \delta \leq \frac{1}{k+1} + \delta$. If $k \geq 9$, $\beta(k) = \frac{1}{3} - \delta$ and $\frac{1}{3} - \delta - \beta(k) = 0$. In both cases, $\nu_{T''}(W'') \leq \phi$. □

Note that we did not make any assumption on T, other than being a CA-Node-Steiner-Tree. Hence, Lemma 3.20 yields the following corollary.

Corollary 3.1. $\psi \leq 1.86 - \frac{1}{2100} < 1.8596$.

Combining Corollary 3.1 with Theorem 3.11 yields a proof of Theorem 3.10.

3.4 Limitations of the witness tree analysis

The goal of this section is to prove lower bounds on the witness tree analysis, and ultimately, prove Theorem 3.4. To prove this theorem, we will provide an example instance of a CA-Node-Steiner-Tree instance with a trivial optimal solution T, and show that any witness tree defined on T has the desired upper bound.

We will first provide a warmup example in Section 3.4.1 to find a bound of $1.8\bar{3}$, in fact, we show this bound is tight by providing an approximation algorithm in Section 3.4.2 with a matching approximation ratio. Then in Section 3.4.3 we will provide the proof for Theorem 3.4.

3.4.1 Lower bound for ψ

In this section, we show that $\psi \geq H_3$. To prove this, we construct a family of leaf-adjacent CA-Node-Steiner-Tree instances that are trees, i.e., the optimal Steiner tree is the input graph itself, along with a specific witness tree that we prove to be optimal, such that $\nu_T(W) \geq H_3 - \varepsilon$, for any fixed $\varepsilon > 0$. More formally, we prove the following theorem.

Theorem 3.12. *For any $\varepsilon > 0$, there exists a leaf-adjacent CA-Node-Steiner-Tree instance G_ε such that $\psi_{G_\varepsilon, R_\varepsilon} > H_3 - \varepsilon$.*

Proof. We give an explicit construction that corresponds to a leaf-to-leaf Block-TAP instance, i.e., every Steiner node is adjacent to exactly two terminals. Recall that leaf-to-leaf instances are a specific case of leaf-adjacent instances. Consider the following graph. Let $t = \lceil \frac{2}{3\varepsilon} \rceil + 1$ and let $S = \{s_1, \ldots, s_t\}$. Let $R = \bigcup_{i=1}^{t} \{r_i^{(1)}, r_i^{(2)}\}$. For every $i \in [t-1]$, the node s_i is adjacent to s_{i+1}. In other words, the nodes $s_1, \ldots, s_t$ form a path whose endpoints are s_1 and s_t. Finally, every Steiner node s_i is adjacent to terminals $r_i^{(1)}$ and $r_i^{(2)}$. Let $G_\varepsilon = (R \cup S, E)$ be the resulting CA-Node-Steiner-Tree instance. Observe that there is a unique optimal Steiner tree, which is the graph G_ε itself. This implies that

$$\psi_{G_\varepsilon} = \frac{1}{t} \min_{\substack{W: W \text{ is} \\ \text{witness tree}}} \left(\sum_{i=1}^{t} H(w(s_i)) \right).$$

We now consider the following witness tree W, that simply "follows" the path $s_1, \ldots, s_t$. More precisely, we define $W = (R, E_W)$ as follows:

- There is an edge between $r_i^{(1)}$ and $r_i^{(2)}$ for every $i \in [t]$.
- There is an edge between $r_i^{(1)}$ and $r_{i+1}^{(1)}$ for every $i \in [t-1]$.

It is easy to see that we have $w(s_1) = w(s_t) = 2$, and $w(s_i) = 3$ for every $i \in \{2, \ldots, t-1\}$, and so we get that

$$\frac{1}{t} \sum_{i=1}^{t} H_{w(s_i)} = \frac{2H_2 + (t-2)H_3}{t} = H_3 - \frac{2}{3t} > H_3 - \varepsilon.$$

Thus, the only thing remaining to show is that W is an optimal witness tree with respect to minimizing $\sum_{i=1}^{t} H_{w(s_i)}$. For that, let W^* be an optimal witness tree with respect to minimizing $\sum_{i=1}^{t} H_{w(s_i)}$, and let w^* be the vector imposed on S by W^*. We first check if $(r_i^{(1)}, r_i^{(2)}) \in W^*$ for every $i \in [t]$. Suppose that there is an $i \in [t]$ such that $(r_i^{(1)}, r_i^{(2)}) \notin W^*$. We now add the edge $(r_i^{(1)}, r_i^{(2)})$ to W^* and a cycle is created. Clearly, there exists an edge $e' \in W^*$ adjacent to $r_i^{(2)}$ such that $W' = (W^* \setminus \{e'\}) \cup \{(r_i^{(1)}, r_i^{(2)})\}$ is a terminal spanning tree. It is also easy to see that $w'(s_j) \leq w^*(s_j)$ for every $j \in [t]$, where w' is the vector imposed on S by W'. We conclude that $\sum_{i=1}^{t} H_{w'(s_i)} \leq \sum_{i=1}^{t} H_{w^*(s_i)}$, and so from now on we assume without loss of generality that $(r_i^{(1)}, r_i^{(2)}) \in W^*$ for every $i \in [t]$.

We now impose some more structure on W^*. In particular, we process the terminals in increasing order of index, and for each $i \in [t-1]$, we replace any edge of the form $(r_i^{(2)}, r_j^{(x)})$, $j > i$ and $x \in \{1,2\}$, with the edge $(r_i^{(1)}, r_j^{(1)})$. It is easy

to see that no w-value changes, and so from now on we assume without loss of generality that for every $i \in [t]$, the terminal $r_i^{(2)}$ is a leaf of W^* that is connected with $r_i^{(1)}$.

Finally, we turn to the edges $(r_i^{(1)}, r_{i+1}^{(1)})$ that are in W for every $i \in [t-1]$. If $W^* \neq W$, then there exist $1 \leq i < j \leq t$, with $j - i > 1$ such that $e = (r_i^{(1)}, r_j^{(1)}) \in W^*$. We remove e from W^* and get two subtrees W_1^* and W_2^*, where $r_i^{(1)} \in W_1^*$ and $r_j^{(1)} \in W_2^*$. Consider any $k \in \{i+1, \ldots, j-1\}$. Assume $r_k^{(1)} \in W_1^*$. We now add the edge $e' = (r_k^{(1)}, r_j^{(1)})$ and obtain a new witness tree $W' = W_1^* \cup W_2^* \cup \{e'\}$. Note that the removal of e causes $w(s_l)$ to decrease by 1 for every Steiner node s_l with $i \leq l \leq j$, while the addition of e' increases $w(s_l)$ by 1 for every Steiner node s_l with $k \leq l \leq j$. The case when $r_k^{(1)} \in W_2^*$ is symmetric, simply adding $(r_i^{(1)}, r_k^{(1)})$ instead of $(r_k^{(1)}, r_j^{(1)})$. We conclude that $\sum_{i=1}^{t} w'(s_i) \leq \sum_{i=1}^{t} w^*(s_i)$, where w' is the vector imposed on S by W'. Putting everything together, we get that W is an optimal witness tree with respect to minimizing $\sum_{i=1}^{t} H_{w(s_i)}$, and so $\psi_{G_\varepsilon, R_\varepsilon} > H_3 - \varepsilon$.

□

An immediate corollary of the Theorem 3.12 is the following.

Corollary 3.2. $\psi \geq H_3 = 1.8\bar{3}$.

The above lower bound shows a limitation of our techniques, as it demonstrates that one cannot hope to obtain a better than $1.8\bar{3}$-approximation by selecting a different (deterministic/randomized) witness tree.

3.4.2 Approximation algorithm for leaf-adjacent Block-TAP

In this section, we consider Block-TAP instances where at least one endpoint of every link is a leaf; we call such instances *leaf-adjacent* Block-TAP instances. We note that leaf-adjacent Block-TAP is a generalization of the case where both endpoints are leaves, often called *leaf-to-leaf* Block-TAP. For leaf-to-leaf Block-TAP, Nutov gave a $1.6\bar{6}$-approximation [73]. Here, we give a $(1.8\bar{3}+\varepsilon)$-approximation for the more general setting of leaf-adjacent Block-TAP.

Throughout this section, we assume that we are given a Block-TAP instance, where $T = (V_T, E_T)$ is the input tree, $R_T \subseteq V_T$ is its set of leaves and $L \subseteq \binom{V_T}{2}$ is the set of links, such that $\ell \cap R_T \neq \emptyset$ for every $\ell \in L$. Using the reduction to CA-Node-Steiner-Tree (see Theorem 3.6), we get a CA-Node-Steiner-Tree instance $G = (R \cup S, E)$ that satisfies the following property: every Steiner node in G is

adjacent to at least one terminal. We call such instances *leaf-adjacent* CA-Node-Steiner-Tree instances.

We start by observing that any leaf-adjacent CA-Node-Steiner-Tree instance $G = (R \cup S, E)$ has an optimal solution $T^* = (R \cup S^*, E^*)$ such that every Steiner node $s \in S^*$ is adjacent to at least one terminal in T^*. To see this, suppose that there is a Steiner node $s \in S^*$ that is not adjacent to any terminal in T^*. Since s is adjacent to some terminal $r \in R$ in G, we add the edge (s, r) to T^*. This creates a cycle that goes through s. By removing the edge of the cycle that is adjacent to s and is not equal to (s, r), we end up with a different optimal solution where s is adjacent to a terminal. This shows that we can transform any optimal solution to a solution that satisfies the desired property, namely, that every Steiner node is adjacent to a terminal. Thus, from now on we assume that T^* satisfies this property.

Our analysis consists of demonstrating that the procedure laid out in Section 3.2.3 finds a witness tree W for T^* such that if $w : S^* \to \mathbb{N}$ is the vector imposed on S^* by W, then we have $\frac{1}{|S^*|} \sum_{v \in S^*} H_{w(v)} \leq H_3$. To see this, we first transform T^* to a forest as follows. In case there is a terminal $r \in R$ of T^* that is not a leaf, we split the tree T^* at r by first removing r, and then adding back a copy of r as a leaf to the $d(r)$ trees of the forest $T^* \setminus \{r\}$, where $d(r)$ is the degree of r in T^*. By performing this operation for all terminals whose degree is larger than 1, we end up with a forest, where each terminal $r \in R$ appears in $d(r)$ trees, in total, such that for each tree of the forest, every final Steiner node is adjacent to a terminal and moreover, all terminals are leaves. Since each Steiner node belongs to exactly one tree of this forest, it is easy to see that if we compute a terminal spanning tree for each tree of the resulting forest, and take the union of these trees, then the w-value of each Steiner node imposed by the final terminal spanning tree is the same as the w-value imposed by the terminal spanning tree computed for the particular tree of the forest which the Steiner node belongs to. Thus, without loss of generality, we assume from now on that T^* satisfies both desired properties, namely that every Steiner node is adjacent to a terminal and every terminal is a leaf.

We now remove the terminals from T^* as in Section 3.2.1 and decompose the remaining tree $T^*[S^*]$ into final-components as in Section 3.2.2, which we denote $T_1^*, \ldots, T_\tau^*$. It then remains to see that when we construct the witness trees $\{W_i\}_{i=1}^\tau$ and merge them together to create the witness tree W, every tree in $\{T_i^*\}_{i=1}^\tau$ falls under Case 1 of the analysis. This implies that we only apply Lemma 3.12 when merging witness trees. Since the guarantee of Lemma 3.12 holds for any $\gamma' \geq H_3$, we use it with $\gamma' = H_3$ and get the desired bound.

More precisely, observe that each node of $T^*[S^*]$ is a final node by the

construction of T^*, so the final-components $T_1^*, \ldots, T_\tau^*$ are in fact the edges of $T^*[S^*]$. When the analysis of Section 3.2 considers a fixed final-component T_i^* rooted at r_i with (r_i, v) being the unique edge incident to r_i inside T_i^*, we know that v is a final node by construction, so we only consider Case 1 when analyzing the change of $\nu_{T^*}(W)$. Thus, we obtain the following theorem.

Theorem 3.13. *Given is a leaf-adjacent CA-Node-Steiner-Tree instance $G = (R \cup S, E)$, and let $T^* = (R \cup S^*, E^*)$ be an optimal Steiner tree such that each Steiner node $s \in S^*$ is adjacent to at least one terminal in T^*. Then, there exists a witness tree W such that $\nu_{T^*}(W) \leq H_3$, where w is the vector imposed on S^* by W.*

The witness tree that achieves the stated value of $\nu_{T^*}(W)$ is in fact a very natural one. We perform the operation above to transform T^* into a forest where the terminals are exactly the leaves of T^*. For every component, and for every edge (u, v) of that component, there is a corresponding edge in the witness tree between one of the terminals adjacent to u and one adjacent to v. Furthermore, if two terminals have the same adjacent Steiner node, then the witness tree has an edge between those two terminals as well.

The above theorem, along with Theorems 3.6 and 3.11, immediately imply Theorem 3.3.

3.4.3 Improved Lower Bound on ψ

We now prove Theorem 3.4. To do this, we first prove some key structural properties of witness trees. We assume we are given a Node Witness Tree instance $T = (V, E)$ with leaves R (and edge costs $c : E \to \mathbb{R}_{\geq 0}$), where R denotes the leaves of T, we will show that we can characterize witness trees minimizing $\nu_T(W)$ ($\bar{\nu}_T(W)$) using the following notion of *laminarity*. Given a witness tree $W = (R, E_W)$, we say edges $f_1 f_2, f_3 f_4 \in E_W$ *cross* if the f_1-f_2 and f_3-f_4 paths in T share an internal node but not an endpoint. We say that W is *laminar* if it has no crossing edges. For nodes $u, v \in V$, we denote by T_{uv} the path in T between the nodes u and v. Similarly, for $e \in E_W$, we denote by T_e the path in T between the endpoints of e.

The following Theorem shows that there is always a witness tree minimizing $\nu_T(W)$ that is laminar.

Theorem 3.14. *Given an instance of the Node Witness Tree problem $T = (V, E)$, let $\mathcal{W}$ be the family of all witness trees for T. Then there exists a laminar witness tree W such that $\nu_T(W) = \min_{W' \in \mathcal{W}} \nu_T(W')$.*

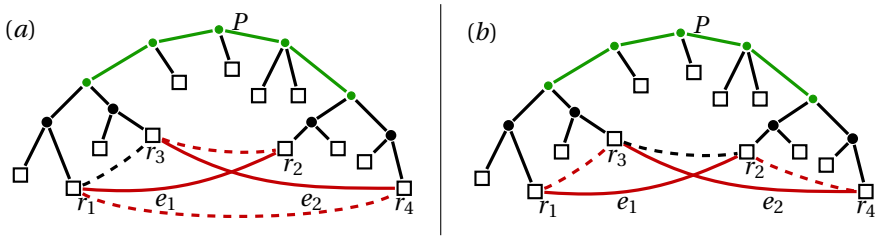


Figure 3.8: In both figures we have a tree, T, shown with black edges and green edges, with leaves, R, denoted by squares. Crossing edges e_1 and e_2 are shown with solid red edges. The green edges denote the path P. Figure (a): In this case, r_1 and r_3 are in the same component of $W \backslash \{e_1, e_2\}$, represented by the dashed black edge. We can replace e_1 with $r_2 r_3$ or replace e_2 with $r_1 r_4$ (red dashed edges). Figure (b): In this case, r_3 and r_2 are in the same component, denoted by the black dashed edge. We can replace e_1 and e_2 with $r_1 r_3$ and $r_2 r_4$ (red dashed edges).

Proof. We first show that there is a witness tree W minimizing $\nu_T(W)$ such that the induced subgraph of W on any maximal set of terminals that share a neighbour in $V \backslash R$ is a star. We assume for the sake of contradiction that there is a maximal set of terminals $S \subseteq R$ sharing a neighbour $v \in V \backslash R$, such that the induced subgraph of W on S is a set of connected components $W_1, \ldots, W_i$ for $i > 1$. Without loss of generality, suppose the shortest path between two components is from W_1 to W_2, and let e denote the edge of this path incident to W_2. We define $W' := W \cup \{f\} \backslash \{e\}$, where f is an arbitrary edge between W_1 and W_2. Since $\{v\} = T_f \backslash R \subsetneq T_e \backslash R$, we have $\nu_T(W') < \nu_T(W)$, contradicting the minimality of W. Therefore, the induced subgraph on S is connected. We can rearrange the edges of this subgraph to be a star as this will not affect $\nu_T(W)$, so we assume this holds on W for any such S.

For a maximal set of terminals $S \subseteq R$ that share a neighbour, by a slight abuse of notation, we denote by S the induced star subgraph of W on S, and denote its center by $s \in S$. We will assume without loss of generality that edges of W incident to S have endpoint s. To see this, as S is a connected subgraph of W, any pair of edges incident to S cannot share an endpoint outside of S, otherwise we have found a cycle in W. Furthermore, for any edge of W incident to S where s is not an endpoint, we can change the endpoint in S of that edge to be s and maintain the connectivity of W since S is connected. Edges changed in this way will have the same interior nodes between their endpoints, so this does not

increase $\nu_T(W)$.

We assume for the sake of contradiction that the witness tree W minimizing $\nu_T(W)$ is not a laminar witness tree. As W is not laminar, there exist distinct leaves $r_1, r_2, r_3, r_4 \in R$ such that $e_1 = r_1 r_2, e_2 = r_3 r_4 \in E_W$ are crossing. We denote the path $T_{e_1} \cap T_{e_2}$ by P. We denote by P_i the (potentially empty) set of internal nodes of the shortest path from P to r_i in T.

Since e_1 and e_2 are crossing edges, one of $T_{r_1 r_3}$ or $T_{r_1 r_4}$ contains exactly one node of P. The same is true for r_2. Without loss of generality, let us assume that the paths $T_{r_1 r_3}$ and $T_{r_2 r_4}$ contain exactly one node of P. We consider by cases which component of $W \setminus \{e_1, e_2\}$ contains two nodes among r_1, r_2, r_3 and r_4. See Figure 3.8 for an example.

- Case: r_1 and r_3 (or similarly, r_2 and r_4) are in the same component of $W \setminus \{e_1, e_2\}$. If $P_1 = P_3 = \emptyset$, then r_1 and r_3 share a neighbour and thus, as shown above, e_1 and e_2 are assumed to share an endpoint, and are thus not crossing.

  Consider $W' := W \cup \{r_2 r_3\} \setminus \{e_1\}$ and $W'' := W \cup \{r_1 r_4\} \setminus \{e_2\}$. If $\nu_T(W) - \nu_T(W') > 0$, this contradicts the minimality of $\nu_T(W)$. Therefore, we can see

  $$0 \le |V \setminus R|(\nu_T(W') - \nu_T(W)) = \sum_{u \in P_3} \frac{1}{w(u)+1} - \sum_{u \in P_1} \frac{1}{w(u)}$$
  $$< \sum_{u \in P_3} \frac{1}{w(u)} - \sum_{u \in P_1} \frac{1}{w(u)+1} = |V \setminus R|(\nu_T(W) - \nu_T(W''))$$

  Clearly, we have $\nu_T(W'') < \nu_T(W)$, contradicting minimality of $\nu_T(W)$.

- Case: r_2 and r_3 (or similarly, r_1 and r_4) are in the same component of $W \setminus \{e_1, e_2\}$. Without loss of generality we can assume that $|V(P)| > 1$, because if $|V(P)| = 1$ then we can reduce to the previous case by relabelling the nodes r_1, r_2, r_3 and r_4. In this case, consider $W' := W \cup \{r_1 r_3, r_2 r_4\} \setminus \{e_1, e_2\}$. Therefore, we can see

  $$|V \setminus R|\left(\nu_T(W') - \nu_T(W)\right) \le -\sum_{u \in P} \frac{1}{w(u)} < 0$$

Thus, we have $\nu_T(W') < \nu_T(W)$, contradicting the minimality of $\nu_T(W)$. □

Recall that we define the method of "mark-and-contraction" as follows. First, root the given tree T at an internal Steiner node. Then, exactly one child edge of each Steiner node $v \in S$ is "marked": this identifies a path of marked edges

from v to a terminal. Contracting the edges along these paths yields a terminal spanning tree, which is our resulting witness tree. The following Lemma shows that the set of laminar witness trees coincide precisely with the set of witness trees one can obtain with marking-and-contraction.

Theorem 3.15. *Given a tree $T = (V, E)$ with leaves R, a witness tree $W = (R, E_W)$ for T can be found by marking-and-contraction if and only if W is laminar.*

Proof. Let $M \subseteq E$, be the edges marked by a method of marking-and-contraction. And let $\bar{M} := E \setminus M$.

$\Rightarrow$) Consider a witness tree $W = (R, E_W)$ of T found by a method of marking-and-contraction. Assume for the sake of contradiction that W is not laminar, and that edges $e_1, e_2 \in E_W$ are crossing. By the method of marking-and-contraction, for $i = 1, 2$, we know that the nodes of T_{e_i} are contained precisely in two separate connected regions of M, denoted $M_{i,1}$ and $M_{i,2}$, and furthermore, are the unique leaves belonging to $M_{i,1}$ and $M_{i,2}$ respectively.

Therefore, if $T_{e_1} \cap T_{e_2} \neq \emptyset$, then e_1 and e_2 share an endpoint, contradicting our assumption.

$\Leftarrow$) Let $W = (R, E_W)$ be a laminar witness tree. Our goal is to find a set of edges $M \subseteq E$, such that W by marking-and-contraction on the edges M. As a simplifying step we contract any node of T that has degree 2, as any two edges in E_W that share a degree 2 node between their endpoints must share another node between their endpoints. For edge $f \in E$, we mark f and add f to M if there are distinct edges $e, e' \in E_W$ such that $f \in E[T_e \cap T_{e'}]$.

First, we want to show that for any edge $e \in E_W$, there is at most one edge of T_e that is in $\bar{M}$. Assume for contradiction that there are distinct edges $f_1, f_2 \in T_e \setminus M$. Of the three connected components of $T \setminus \{f_1, f_2\}$, we denote the component that is incident to both f_1 and f_2 by T'. T is assumed to have no degree 2 vertices, so there is at least one leaf $r \in R(T')$. There is a path in W from r to an endpoint of e. So there is an edge $e' \neq e \in W$, with at least one of f_1 or f_2 in $T_{e'}$, and thus that edge is in M, contradicting our assumption. See Figure 3.9.(a) for an example.

We now show that for every edge $e = uv \in E_W$, there is at least one edge on the T_e path that is in $\bar{M}$. We assume for contradiction that that every edge in E_W on the path T_e is in M, and we enumerate the nodes of T_e as $u = v_0, v_1, \ldots, v_k = v$ in order. Since $v_0 v_1 \in M$, there is an edge with endpoint at v_0 not equal to e. Pick $e_i \in \binom{R}{2}$ to be the edge that maximizes $\{v_0, v_1, \ldots, v_i\} \subset T_{e_i}$, denote its endpoints by u and r_i. Since $v_i v_{i+1} \in M$, there is an edge $e' \in E_W$ such that $v_i v_{i+1} \in E[T_{e'}]$, where $e' \neq e, e_i$. Since W is laminar, we know that e' must share an endpoint with e and with e_i. We picked e_i to be the edge in E_W that maximizes the set

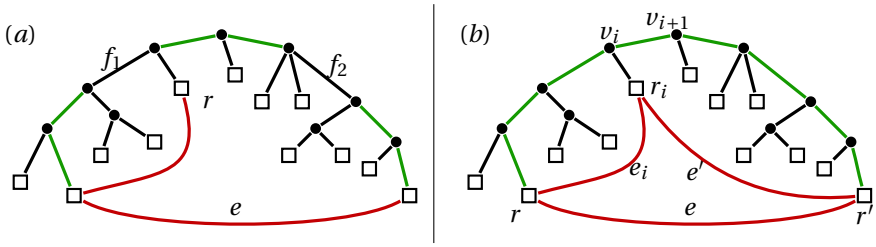


Figure 3.9: Returning to the tree T from Figure 3.8 the green edges denote the edges marked on T_e, and the red edges denote edges of W. Figure (a): $f_1, f_2 \in T_e$ are unmarked. The component of $T \backslash \{f_1, f_2\}$ has a terminal r and there must be a path from r to an endpoint of e in W. So f_1 must be marked by definition. Figure (b): Every edge of T_e is marked. So taking e_i the edge with endpoint r that maximally intersects T_e, has endpoint at r_i. The edge v_{i+1} is marked, so e' must have endpoints at r_i and r' by laminarity since it shares v_i with e_i and e.

$\{v_0, v_1, \ldots, v_i\} \subset T_{e_i}$, so e' cannot have u as an endpoint, and therefore must have v as an endpoint. Similarly, the second endpoint of e' must be r_i so e' does not cross with e_i. Therefore, the edges $e, e_i, e' \in E_W$ form a cycle in W, contradicting the assumption that W is a tree. See Figure 3.9.(b) for an example.

By construction, the edges $e \in E_W$ have a one to one correspondence to $M \subseteq E$, by the unique edge T_e that is unmarked.

It remains to show that the connected regions of M contain exactly one leaf, and thus, by contracting each regions, the resulting tree found by $\bar{M}$ will be exactly W. By the one to one correspondence between $\bar{M}$ and E_W, we have $|R| - 1 = |\bar{M}|$, and thus $|R|$ connected regions of M.

We assume for contradiction that there is a maximal connected region of M that does not contain a leaf, which we denote by C. Note that the subgraph induced by the edges of C is itself a tree. Consider a leaf v of C. By our assumption, v cannot be a leaf of T, and since we assume T has no degree 2 nodes, v is incident to at least two edges $f_1, f_2 \in E \backslash C$. Moreover, since C is a maximal connected region, $f_1, f_2 \in \bar{M}$. By definition of M, and thus $\bar{M}$, there are distinct (and unique) edges $e_1, e_2 \in E_W$ such that $f_1 \in E[T_{e_1}]$ and $f_2 \in E[T_{e_2}]$. By the laminarity of W, since $v \in T_{e_1} \cap T_{e_2}$, e_1 and e_2 must share an endpoint, and the path $T_{e_1} \cap T_{e_2} \subseteq M$. Therefore, there is a path in M from v to a leaf in R. Therefore, C contains a terminal, contradicting our assumption. □

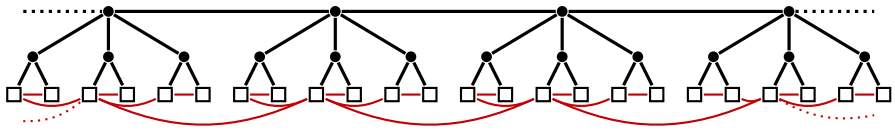


Figure 3.10: Lower bound instance shown in black. The white squares are terminals and black circles are Steiner nodes. Red edges form the laminar witness tree W^*.

Incidentally, Theorem 3.15 has the following side implication. The authors of [48] gave a dynamic program (that is also a bottom-up approach) to compute the best possible witness tree obtainable with a marking-and-contraction scheme. Our structural results imply that their dynamic program computes an optimal solution for the EWT problem (though for the purpose of the approximation analysis, being able to compute the best witness tree is not that relevant: being able to bound ψ is what matters).

Given the notion of laminarity, we consider a CA-Node-Steiner-Tree instance (G,R), where G consists of a path of Steiner nodes $s_1,\dots,s_q$ such that, for all $i \in [q]$, s_i is adjacent to Steiner nodes t_{i1}, t_{i2}, t_{i3}, and each t_{ij} is adjacent to two terminals r^1_{ij} and r^2_{ij}. See the black edges of Figure 3.10. We will refer to the subgraph induced by $s_i, t_{ij}, r^1_{ij}, r^2_{ij}$ $(j = 1,2,3)$ as B_i. Since G is a tree connecting the terminals, clearly the optimal Steiner tree for this instance is $T = G$.

Let W^* be a witness tree that minimizes $\nu_T(W^*)$. Recall that we can assume W^* to be laminar by Theorem 3.14. We provide here a characterization of W^* that will be useful for explicitly determining W^*.

Property (a): we observe that, without loss of generality, we can assume that every pair of terminals r^1_{ij} and r^2_{ij} are adjacent in W^* and that r^2_{ij} is a leaf of W^*.

Property (b): using the latter of these observations and laminarity, we show that for all i, the subgraph of W induced by $r^1_{i1}, r^1_{i2}, r^1_{i3}$ can only be either a star, or three singletons, adjacent to a unique terminal $r \notin B_i$.

For any witness tree W with properties (a) and (b), we provide some notation that will be useful. We say that B_i is a *center* in W the subgraph of W induced by $r^1_{i1}, r^1_{i2}, r^1_{i3}$ is a star, and without loss of generality, we assume that r^1_{i2} is the center of the star. For a center B_i in W, we denote by x^i_L (resp. x^i_R) the maximum cardinality of the subset $\{B_{i-x^i_L},\dots,B_{i-1}\} \subseteq \{B_1,\dots,B_q\}$,

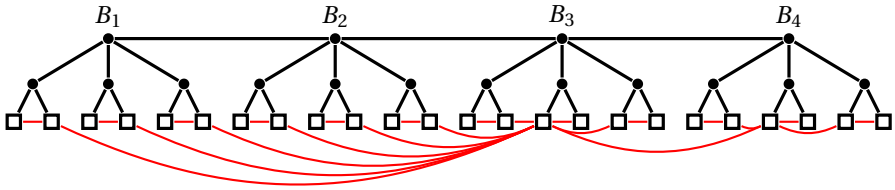


Figure 3.11: Red edges form a laminar witness tree W that is not optimal. In this case we have centers B_3 and B_4. Where B_3 has $x_L^3 = 2$, $x_R^3 = 0$, $L_3 = 0$, and $R_3 = 1$, and B_4 has $x_L^4 = 0$, $x_R^4 = 0$, $L_4 = 1$, and $R_4 = 0$.

(resp. $\{B_{i+1}, \dots, B_{i+x_R^i}\} \subseteq \{B_1, \dots, B_q\}$) such that for each $B_j \in \{B_{i-x_L^i}, \dots, B_{i-1}\}$ (resp. $B_j \in \{B_{i+1}, \dots, B_{i+x_R^i}\}$), the subgraph of W induced by $r_{j1}^1, r_{j2}^1, r_{j3}^1$ is three singletons and the unique terminal adjacent to these is in B_i. Furthermore, we let $L_i = 1$ (resp. $R_i = 1$) if there is a center B_k in W with $k < i$ (resp. $k > i$) and equal to 0 if not. See Figure 3.11.

For a given center B_i, there are exactly $3x_L^i$ edges between B_i and the subtrees $\{B_{i-x_L^i}, \dots, B_{i-1}\}$. Similarly, there are exactly $3x_R^i$ edges between B_i and the subtrees $\{B_{i+1}, \dots, B_{i+x_R^i}\}$. Furthermore, if $L_i = 1$ (resp. $R_i = 1$) then there is a single edge between B_i and a center B_j with $k < i - x_L^i$ (resp. $k > i + x_R^i$). Therefore, there are a total of $3x_L^i + 3x_R^i + L_i + R_i$ edges of E_W incident to the center B_i.

If w is the vector imposed on the Steiner nodes of T by W, then it is clear that $w(s_i) = 3x_L^i + 3x_R^i + L_i + R_i + 2$, $w(t_{i2}) = 3x_L^i + 3x_R^i + L_i + R_i + 3$, and $w(t_{i1}) = w(t_{i3}) = 2$. Furthermore, since W is laminar, it is clear that for every $j = 1, \dots, x_R^i$, we have $w(s_{i+j}) = 3(x_R^i - j + 1) + R_i$ and $w(t_{(i+j)1}), w(t_{(i+j)2}), w(t_{(i+j)3}) = 2$. Similarly, for every $j = 1, \dots, x_L^i$, we have $w(s_{i-j}) = 3(x_L^i - j + 1) + L_i$ and $w(t_{(i-j)1}), w(t_{(i-j)2}), w(t_{(i-j)3}) = 2$ for $k \in \{1, 2, 3\}$.

The following Lemma provides an explicit description of W^*.

Lemma 3.21. *Let $\mathcal{W}$ be the family of all laminar witness trees over T, and let W^* be a laminar witness tree such that for every $i \in [q]$, B_i is a center in W^*. Then $\nu_T(W^*) = \min_{W \in \mathcal{W}} \nu_T(W)$.*

Proof. To prove this lemma, we consider an arbitrary laminar witness W that satisfies properties (a) and (b) above. We show that if W contains a center $B_i \in \{B_1, \dots, B_q\}$ with x_L^i, x_R^i, R_i and L_i as defined above such that $x_L^i + x_R^i > 0$, then we

can find a witness tree W' such that $\nu_T(W') < \nu_T(W)$. Thus, let $B_i \in \{B_1, \ldots, B_q\}$ be a center of W with x_L^i, x_R^i, R_i and L_i.

To prove our claim, first observe that $|V[T]\backslash R|\nu_T(W^*) = \sum_{k=1}^{q} \sum_{v \in B_k} H_{w(v)}$. To prove our claim, we simply must find a if we can find a witness tree W' with vector w' imposed on $V[T]\backslash R$ such that

$$\sum_{k=1}^{i-x_L^i-1} \sum_{v \in B_k} H_{w'(v)} + \sum_{k=1}^{i+x_R^i+1} \sum_{v \in B_k} H_{w'(v)} = \sum_{k=1}^{i-x_L^i-1} \sum_{v \in B_k} H_{w(v)} + \sum_{k=1}^{i+x_R^i+1} \sum_{v \in B_k} H_{w(v)}$$

and

$$\sum_{k=i-x_L^i}^{i+x_R^i} \sum_{v \in B_k} H_{w(v)} > \sum_{k=i-x_L^i}^{i+x_R^i} \sum_{v \in B_k} H_{w'(v)}$$

then we will have proven our claim.

Consider laminar witness tree W' that is equal to W^* except for edges with endpoints in $\{B_{i-x_l^i}, \ldots, B_{i+x_r^i}\}$, which are instead centers in W' with $x_L^k = x_R^k = 0$ for $k = i - x_l^i, \ldots, i + x_r^i$. Furthermore, we can clearly see $L_k = R_k = 1$ for $k = i - x_l^i + 1, \ldots, i + x_r^i - 1$, while $L_{i-x_L^i} = L_i$, and $R_{i+x_R^i} = R_i$. It is not hard to see that W' is indeed a laminar witness tree, as the connectivity is guaranteed since $L_{i-x_L^i} = L_i$, and $R_{i+x_R^i} = R_i$, and the choice of edges is also clearly laminar.

To see the difference between $\sum_{k=1}^{q} \sum_{v \in B_k} H_{w(v)}$ and $\sum_{k=1}^{q} \sum_{v \in B_k} H_{w'(v)}$, we first consider the value of $\sum_{k=i-x_L^i}^{i+x_R^i} \sum_{v \in B_j} H_{w(v)}$ which is equal to

$$\begin{aligned}
&\sum_{k=i-x_L^i}^{i-1} \sum_{v \in B_k} H_{w(v)} + \sum_{k=i+1}^{i+x_R^i} \sum_{v \in B_k} H_{w(v)} + \sum_{v \in B_i} H_{w(v)} \\
&= \sum_{k=1}^{x_L^i} (3H_2 + H_{3j+L_i}) + \sum_{k=1}^{x_R^i} (3H_2 + H_{3jR_i}) + \\
&\qquad + (2H_2 + H_{3x_L^i+3x_R^i+R_i+L_i+2} + H_{3x_L^i+3x_R^i+R_i+L_i+3})
\end{aligned}$$

We can similarly see that the value of $\sum_{k=i-x_L^i}^{i+x_R^i} \sum_{v\in B_k} H_{w'(v)}$ which is equal to

$$\begin{aligned}
&\sum_{v\in B_{i-x_L^i}} H_{w'(v)} + \sum_{k=i-x_L^i+1}^{i+x_R^i-1} \sum_{v\in B_k} H_{w'(v)} + \sum_{v\in B_{i+x_R^i}} H_{w'(v)} \\
=&(2H_2 + H_{3+L_i} + H_{4+L_i}) + \sum_{k=i-x_L^i+1}^{i+x_R^i-1} (2H_2 + H_4 + H_5) + (2H_2 + H_{3+R_i} + H_{4+R_i}) \\
=&(2x_L^i + 2x_R^i + 2)H_2 + (x_L^i + x_R^i - 1)(H_4 + H_5) + H_{3+L_i} + H_{4+L_i} + H_{3+R_i} + H_{4+R_i}
\end{aligned}$$

Therefore, the difference between $\sum_{v\in T} H_{w(v)}$ and $\sum_{v\in T} H_{w'(v)}$ is equal to

$$\begin{aligned}
&\sum_{k=1}^{x_L^i} (3H_2 + H_{3k+L_i}) + \sum_{k=1}^{x_R^i} (3H_2 + H_{3k+R_i}) \\
&+ 2H_2 + H_{3x_L^i+3x_R^i+R_i+L_i+2} + H_{3x_L^i+3x_R^i+R_i+L_i+3} \\
&- (2x_L^i + 2x_R^i + 2)H_2 - (x_L^i + x_R^i - 1)(H_4 + H_5) - H_{3+L_i} - H_{4+L_i} - H_{3+R_i} - H_{4+R_i}
\end{aligned}$$

We denote this difference by $P(x_R^i, x_L^i, L_i, R_i)$. We will show that $P(x_R^i, x_L^i, L_i, R_i) > 0$, for every $(x_L^i, x_R^i, L_i, R_i) \in \mathbb{Z}^4$ such that 1) $x_L^i, x_R^i \geq 0$; 2) $x_L^i + x_R^i \geq 1$, and; 3) $L_i, R_i \in \{0,1\}$. Contradicting the assumption that $\nu_T(W^*) = \min_{W\in\mathcal{W}} \nu_T(W)$. We proceed by induction on $x_R^i + x_L^i$.

For our base case, we assume $x_R^i = 1 \geq x_L^i$. We have the following cases for the values of x_L^i:

1. Case: $x_L^i = 0$. Then $P(0, 1, L_i, R_i)$ is equal to

$$\begin{aligned}
&5H_2 + H_{3+R_i} + H_{5+R_i+L_i} + H_{6+R_i+L_i} \\
&- 4H_2 - H_{3+L_i} - H_{4+L_i} - H_{3+R_i} - H_{4+R_i} \\
=&H_2 + H_{5+R_i+L_i} + H_{6+R_i+L_i} - H_{3+L_i} - H_{4+L_i} - H_{4+R_i} \\
\geq&H_2 + H_{6+R_i} + H_{7+R_i} - H_4 - H_5 - H_{4+R_i} \\
\geq&H_2 + H_6 + H_7 - H_4 - H_5 - H_4 \\
=&13/140 > 0
\end{aligned}$$

   Where the first inequality follows since it is not hard to see that $H_{5+R_i} + H_{6+R_i} - H_3 - H_4 > H_{6+R_i} + H_{7+R_i} - H_4 - H_5 > 0$. The second inequality follows for a similar reason.

2. Case: $x_L^i = 1$. Then $P(1,1,L_i,R_i)$ is equal to

$$\begin{aligned}
&8H_2 + H_{3+L_i} + H_{3+R_i} + H_{8+L_i+R_i} + H_{9+L_i+R_i} - 6H_2 - H_4 - H_5 - H_{3+L_i} \\
&- H_{4+L_i} - H_{3+R_i} - H_{4+R_i} \\
=&2H_2 + H_{8+L_i+R_i} + H_{9+L_i+R_i} - H_4 - H_5 - H_{4+L_i} - H_{4+R_i} \\
\geq&2H_2 + H_8 + H_9 - 3H_4 - H_5 = 17/2160 > 0
\end{aligned}$$

Where the first inequality above follow easily by checking the values of $L_i, R_i \in \{0,1\}$.

Our inductive hypothesis is to assume the inequality holds for $x_L^i + x_R^i = k \geq 1$. We will show the claim holds when $x_L^i + x_R^i = k+1$. Since we showed the base case for $x_R^i = 1$ and $x_L^i \in \{0,1\}$, we can assume $\max\{x_R^i, x_L^i\} \geq 2$. Furthermore, we can assume without loss of generality that $3x_R^i + R_i \geq 3x_L^i + L_i$, which implies $x_R^i \geq x_L^i$. We will show that $P(x_L^i, x_R^i, L_i, R_i) > 0$, by first showing that $P(x_L^i, x_R^i, L_i, R_i) > P(x_L^i, x_R^i - 1, L_i, R_i)$ and applying the inductive hypothesis to $P(x_L^i, x_R^i - 1, L_i, R_i)$. We can see

$$\begin{aligned}
&P(x_L^i, x_R^i, L_i, R_i) - P(x_L^i, x_R^i - 1, L_i, R_i) \\
=&H_2 + H_{3x_R^i + R_i} - H_4 - H_5 + H_{3x_L^i + 3x_R^i + L_i + R_i + 2} - H_{3x_L^i + 3x_R^i + L_i + R_i - 1} \\
&+ H_{3x_L^i + 3x_R^i + L_i + R_i + 3} - H_{3x_L^i + 3x_R^i + L_i + R_i} \\
\geq&H_2 + H_{3x_R^i + R_i} - H_4 - H_5 + H_{6x_R^i + 2R_i + 2} - H_{6x_R^i + 2R_i - 1} + H_{6x_R^i + 2R_i + 3} - H_{6x_R^i + 2R_i}
\end{aligned}$$

Where the inequality above follows since we can see that the following inequalities hold since $3x_R^i + R_i \geq 3x_L^i + L_i$

$$\begin{aligned}
H_{3x_L^i + 3x_R^i + L_i + R_i + 2} - H_{3x_L^i + 3x_R^i + L_i + R_i - 1} &\geq H_{6x_R^i + 2R_i + 2} - H_{6x_R^i + 2R_i - 1} \\
H_{3x_L^i + 3x_R^i + L_i + R_i + 3} - H_{3x_L^i + 3x_R^i + L_i + R_i} &\geq H_{6x_R^i + 2R_i + 3} - H_{6x_R^i + 2R_i}
\end{aligned}$$

Similarly, since $R_i \leq 1$, we have

$$\begin{aligned}
&H_2 + H_{3x_R^i + R_i} - H_4 - H_5 + H_{6x_R^i + 2R_i + 2} - H_{6x_R^i + 2R_i - 1} + H_{6x_R^i + 2R_i + 3} - H_{6x_R^i + 2R_i} \\
\geq&H_2 + H_{3x_R^i + 1} - H_4 - H_5 + H_{6x_R^i + 4} - H_{6x_R^i + 1} + H_{6x_R^i + 5} - H_{6x_R^i + 2}
\end{aligned}$$

Collecting the terms with x_R^i we have

$$\begin{aligned}
&H_2 - H_4 - H_5 + H_{3x_R^i+1} + H_{6x_R^i+5} - H_{6x_R^i+2} + H_{6x_R^i+4} - H_{6x_R^i+1} \\
=&H_2 - H_4 - H_5 + \frac{1}{6x_R^i+5} + \frac{1}{6x_R^i+4} + \frac{1}{6x_R^i+3} + \frac{1}{6x_R^i+4} + \frac{1}{6x_R^i+3} + \frac{1}{6x_R^i+2} \\
\geq&H_2 - H_4 - H_5 + \frac{2}{6x_R^i+5} + \frac{2}{6x_R^i+4} + \frac{2}{6x_R^i+3} \\
\geq&H_2 - H_4 - H_5 + \frac{1}{3x_R^i+4} + \frac{1}{3x_R^i+3} + \frac{1}{3x_R^i+2} = H_2 - H_4 - H_5 + H_{3x_R^i+4} \\
\geq&H_2 - H_4 - H_5 + H_{10} = 157/2520 > 0
\end{aligned}$$

which completes the proof of Lemma 3.21. □

We are now ready to prove Theorem 3.4.

Proof of Theorem 3.4. Once we impose the condition that all B_i are centers, one notes that the tree W^* essentially must look like the one shown in Figure 3.10. So it only remains to compute $\nu_T(W^*)$. For every B_i, we can compute $\sum_{v\in B_i} H_{w^*(v)}$, where w^* is the vector imposed on the set S of Steiner nodes by W^*. For $i \in \{2,\dots,q-1\}$, one notes that $\frac{1}{4}\sum_{v\in B_i} H_{w^*(v)} = \frac{1}{4}(2H_2 + H_4 + H_5) = 221/120 = 1.841\bar{6}$. Similarly, for $i = 1$ and q we have $\frac{1}{4}\sum_{v\in B_1} H_{w^*(v)} = \frac{1}{4}\sum_{v\in B_q} H_{w^*(v)} = \frac{1}{4}(2H_2 + H_3 + H_4) = \frac{83}{48} = 1.7291\bar{6}$. Therefore, we can see that $\nu_T(W^*) = \sum_{v\in S} \frac{H_{w^*(v)}}{|S|} = \frac{1.841\bar{6}q - 2(1.841\bar{6} - 1.7291\bar{6})}{q}$. Thus, for $q > \frac{1}{\varepsilon}$ we have $\nu_T(W^*) > 1.841\bar{6} - \frac{1}{q}$. □

Chapter 4

Steiner Claw Free

The Steiner Tree problem, is one of the most fundamental and classic optimization problems in algorithm design. It has always attracted a lot of interest, both from a theoretic perspective due its algorithmic challenges, and from a practical perspective, because of its wide array of applications. We repeat here the definition.

Definition 4.1 (Steiner Tree Problem)**.** *Let $G = (V, E)$ by a connected graph, with edge costs $c : E \to \mathbb{R}_{\geq 0}$, and subset of vertices $R \subseteq V$ which we call* terminals*. The goal of the Steiner Tree Problem is to compute a minimum cost set of edges $F \subseteq E$ such that F induces a tree containing every terminal.*

Note that S might contain some other nodes, besides the terminals, which we call *Steiner nodes*. The Steiner Tree problem is known to be NP-Hard from Garey and Johnson [43]. In fact it is NP-hard to find an approximation better than $\frac{96}{95}$ [12]. The Steiner Tree problem has a long history of approximation see, e.g., [45,58,78,79,85,90], culminating in a $\ln(4)+\varepsilon$-approximation [16,48,83].

All three of the works that currently provided the best Steiner Tree approximation, [16,48,83], rely on the notion of *k-restricted Steiner trees*. A k-restricted Steiner tree S is a collection of components (trees whose leaves coincide with a subset of terminals), where each component has at most k terminals each, and whose union induces a Steiner tree. Furthermore, in all three of these algorithms, the analysis at some point relies on constructing *witness trees*, and in particular to solutions of the following problem.

Definition 4.2 (Edge Witness Tree (EWT) Problem)**.** *Given is a tree $T = (V, E)$*

with edge costs $c : E \to \mathbb{R}_{\geq 0}$. We denote by R the set of leaves of T. The goal of the Edge Witness Tree Problem is to find a tree $W = (R, E_W)$, where $E_W \subseteq R \times R$, which minimizes the non-linear objective function $\bar{v}_T(W) = \frac{1}{c(E)} \sum_{e \in E} c(e) H_{\bar{w}(e)}$, where $c(E) = \sum_{e \in E} c(e)$, the function $\bar{w} : E \to \mathbb{Z}_{\geq 0}$ is defined as

$$\bar{w}(e) := |\{pq \in E_W : e \text{ is an internal edge of the } p\text{-}q \text{ path in } T\}|$$

and H_ℓ denotes the ℓ^{th} harmonic number ($H_\ell = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\ell}$).

The Edge Witness Tree problem is related to Steiner Trees in the following way. Suppose we are given a Steiner Tree instance $(G = (V, E), R, c)$ where $c : E \to \mathbb{R}_{\geq 0}$ gives the edge costs. We define the following:

$$\gamma_{(G,R,c)} := \min_{\substack{T^* = (R \cup S^*, E^*): T^* \text{ is} \\ \text{optimal Steiner tree of } (G,R,c)}} \; \min_{\substack{W: W \text{ is a} \\ \text{witness tree} \\ \text{of } T^*}} \bar{v}_{T^*}(W)$$

We also define the following constant γ:

$$\gamma := \sup\{\gamma_{(G,R,c)} : (G, R, c) \text{ is an instance of Steiner Tree}\}.$$

Byrka et al. [16] were the first to essentially prove the following.

Theorem 4.1. *For any $\varepsilon > 0$, there is a $(\gamma + \varepsilon)$-approximation algorithm for Steiner Tree.*

Furthermore, the authors in [16] showed that $\gamma \leq \ln(4)$, and hence they obtained the previously mentioned $(\ln(4) + \varepsilon)$-approximation for Steiner Tree.

The algorithm in [16] is LP based, and relies on an LP-relaxation, called the hypergraphic relaxation, whose integrality gap is $\ln(4)$, provided in [46, 82]. We also utilize this relaxation. However, hypergraphic relaxations are large and as computation problems, hard to solve exactly. A different, efficiently solvable relaxation called the so-called *Bidirected Cut Relaxation* [16, 21, 32, 35, 38, 47, 48, 88]. The integrality gap of the bidirected cut relaxation for the Steiner Tree problem has been long conjectured to be strictly smaller than 2. The authors of [35] then show that the hypergraphic relaxation and the Bidirected Cut Relaxation are polyhedrally equivalent for a certain set of instances called *Steiner-Claw Free*. In Steiner-Claw Free instances, the subgraph induced on the Steiner vertices does not contain a claw. The focus of this chapter is on these instances.

Our Results and Techniques The primary result of this chapter, proven in Section 4.1 is the following approximation result for Steiner-Claw Free instances.

Theorem 4.2. *There is a* $(\frac{991}{732}+\varepsilon < 1.354)$*-approximation for Steiner Tree on Steiner-Claw Free instances.*

We prove this theorem by showing that, for any Steiner-Claw Free instance (G,R,c), $\gamma_{(G,R,c)} \leq \frac{991}{732}$. The observation we use here is that an optimal Steiner Tree solution T in this case is the union of components that are caterpillar graphs[1]: this knowledge can be exploited to design ad-hoc witness trees. Interestingly, we can also show that this bound is tight: once again, the proof of this lower-bound result relies on showing laminarity for optimal witness trees, and is proven in Section 4.2.

Theorem 4.3. *For any* $\varepsilon > 0$*, there exists Steiner-Claw Free instance* $(G_\varepsilon, R_\varepsilon, c_\varepsilon)$ *such that* $\gamma_{(G_\varepsilon,R_\varepsilon,c_\varepsilon)} > \frac{991}{732} - \varepsilon$.

As a corollary of our results, we also get an improved bound on the integrality gap of the bidirected cut relaxation for Steiner-Claw Free instances (this follows directly from combining our upper bound with the results in [35]). Though these instances are quite specialized, they serve the purpose of passing the message: exploiting the structure of optimal solutions helps in choosing better witnesses, hopefully arriving at tight (upper and lower) bounds on γ.

4.1 Tight Approximation for Steiner-Claw Free

We here prove Theorem 4.2. Our goal is to show that for any Steiner-Claw Free instance (G,R,c), $\gamma_{(G,R,c)} \leq \frac{991}{732}$, improving over the known $\ln(4)$ bound that holds in general. From now on, we assume that we are given an optimal solution $T = (R \cup S^*, E^*)$ to (G,R,c).

Simplifying Assumptions. As standard, note that T can be decomposed into components $T_1,\ldots,T_\tau$, where each component is a maximal subtree of T whose leaves are terminals and internal nodes are Steiner nodes. Since components do not share edges of T, it is not difficult to see that one can compute a witness tree W_i for each component T_i separately, and then take the union of the $\{W_i\}_{i\geq 1}$ to get a witness tree W whose objective function $\bar{v}_T(W)$ will be bounded by the

[1] A caterpillar graph is defined as a tree in which every leaf is of distance 1 from a central path.

maximum among $\bar{v}_{T_i}(W_i)$. Hence, from now on we assume that T is made by one single component. Since T is a solution to a Steiner-claw free instance, each Steiner node is adjacent to at most 2 Steiner nodes. In particular, the Steiner nodes induce a path in T, which we enumerate as $s_1, \ldots, s_q$. We will assume without loss of generality that each s_j is adjacent to exactly one terminal $r_j \in R$: this can be achieved by replacing a Steiner node incident to p terminals, with a path of length p made of 0-cost edges, if $p > 1$, and with an edge of appropriate cost connecting its 2 Steiner neighbors, if $p = 0$.

Algorithm 4: Computing the witness tree W

Initialize $W = (R, E_W = \emptyset)$
Sample uniformly at random σ from $\{1, \ldots, t_\alpha\}$.
$E_W \leftarrow \{r_\sigma r_{\sigma+k} | 1 \le |k| \le \lfloor \frac{t_\alpha}{2} \rfloor, 1 \le \sigma + k \le q\}$
Initialize $j = 1$
while $j \le \frac{q-\sigma}{t_\alpha}$ **do**
  $\ell := \sigma + t_\alpha j$
  $E_W \leftarrow E_W \cup \{r_\ell r_{\ell+k} | 1 \le |k| \le \lfloor \frac{t_\alpha}{2} \rfloor, 1 \le \ell + k \le q\}$
  $E_W \leftarrow E_W \cup \{r_{\sigma+t_\alpha(j-1)} r_{\sigma+t_\alpha j}\}$
  $j \leftarrow j + t_\alpha$
end
if $\sigma > \lceil \frac{t_\alpha}{2} \rceil$ **then**
  $E_W \leftarrow E_W \cup \left\{r_1 r_k | 2 \le k \le \sigma - \lceil \frac{t_\alpha}{2} \rceil\right\} \cup \{r_1 r_\sigma\}$
end
$j \leftarrow \lfloor \frac{q-\sigma}{t_\alpha} \rfloor$
if $\sigma + t_\alpha j \le q - \lceil \frac{t_\alpha}{2} \rceil$ **then**
  $E_W \leftarrow E_W \cup \{r_k r_q | \sigma + t_\alpha j + \lceil \frac{t_\alpha}{2} \rceil \le k \le q - 1\} \cup \{r_{\sigma+t_\alpha j} r_q\}$
end
Return W

Witness tree computation and analysis. We denote by $L \subseteq E^*$ the edges of T incident to a terminal, and by $O = E^* \setminus L$ the edges of the path $s_1, \ldots, s_q$. Let $\alpha := c(O)/c(L)$.

We first show that if $q < 5$, the claim holds

Lemma 4.1. *If $q < 5$, then $\gamma_{(G,R,c)} < \frac{991}{732}$.*

Proof. Note that $c(E^*) = (1+\alpha)c(L) = \frac{1+\alpha}{\alpha}c(O)$. We distinguish two cases for the values of α:

- First assume $\alpha \geq \frac{1}{2}$. In this case we define E_W as $\{r_i r_{i+1} | 1 \leq i < q\}$. Observe that $w(e)$ is 1 if $e = s_i s_{i+1}$ for $1 \leq i < q$ and is at most 2 if $e = s_i r_i$ $1 \leq i \leq q$. Therefore:

$$\sum_{e \in E^*} c(e) H_{\tilde{w}(e)} \leq \frac{c(E^*)}{1+\alpha} H_2 + \frac{\alpha c(E^*)}{1+\alpha} H_1 = \frac{H_2 + \alpha}{1+\alpha} c(E^*) \leq \frac{4}{3} c(E^*).$$

  Therefore $\bar{\nu}_T(W) \leq \frac{4}{3} < \frac{991}{732}$.

- Now assume $\alpha < \frac{1}{2}$. We uniformly at random select $1 \leq \sigma \leq q$ and then we define E_W as $\{r_\sigma r_i | 1 \leq i \leq q, i \neq \sigma\}$. If $e = s_i s_{i+1}$ for $1 \leq i < q$ then it's not hard to see $\mathbb{E}[H_{w(e)}] \leq H_2$ since $q < 5$. For $e = s_i r_i$, $\mathbb{E}[H_{w(e)}] = \frac{1}{q} H_{q-1} + \frac{q-1}{q} H_1 \leq \frac{H_3+3}{4} = 29/24$. Therefore:

$$\sum_{e \in E^*} c(e) H_{\tilde{w}(e)} \leq \frac{\frac{29}{24} c(E^*)}{1+\alpha} + \frac{\alpha c(E^*)}{1+\alpha} H_2 = \frac{\frac{29}{24} + \alpha H_2}{1+\alpha} c(E^*) < \frac{47}{36} c(E^*).$$

Thus $\mathbb{E}[\bar{\nu}_T(W)] \leq \frac{47}{36} < \frac{991}{732}$, which implies $\gamma_{(G,R,c)} < \frac{991}{732}$. □

We now assume that $q \geq 5$. For a fixed value of $\alpha \geq 0$, we will fix a constant t_α as follows: if $\alpha \in [0, 32/90]$, then $t_\alpha = 5$; if $\alpha \in (32/90, 1)$, then $t_\alpha = 3$, and; if $\alpha \geq 1$, then $t_\alpha = 1$. Given α (and thus t_α), we construct W using the randomized process outlined in Algorithm 4. At a high level, starting from a random offset, Algorithm 4 adds sequential stars of t_α terminals to W, connecting the centers of these stars together in this sequence. See Figure 4.1 for an example. Under this random scheme, we define $\lambda_L(t_\alpha) := \max_{e \in L} \mathbb{E}[H_{\tilde{w}(e)}]$, and $\lambda_O(t_\alpha) := \max_{e \in O} \mathbb{E}[H_{\tilde{w}(e)}]$.

Lemma 4.2. *For any $\alpha \geq 0$, $\lambda_L(t_\alpha) \leq \frac{1}{t_\alpha} H_{t_\alpha+1} + \frac{t_\alpha - 1}{t_\alpha}$, and $\lambda_O(t_\alpha) \leq \frac{1}{t_\alpha} + \frac{2}{t_\alpha} \sum_{i=2}^{\lceil \frac{t_\alpha}{2} \rceil} H_i$. Moreover, for any $\alpha \geq 0$, the following bounds holds:*

$$\frac{1}{\alpha+1} \left(\frac{1}{t_\alpha} H_{t_\alpha+1} + \frac{t_\alpha - 1}{t_\alpha} + \alpha \left(\frac{1}{t_\alpha} + \frac{2}{t_\alpha} \sum_{i=2}^{\lceil \frac{t_\alpha}{2} \rceil} H_i \right) \right) \leq \frac{991}{732}$$

Proof. Let $W = (R, E_W)$ be a witness tree returned from running Algorithm 4 with α and $t := t_\alpha$, and let w be the vector imposed on E^* by W. If Algorithm 4

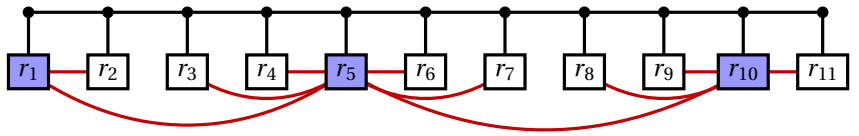


Figure 4.1: Edges of T are shown in black. Red edges show W. Here, $q = 11$, $t_\alpha = 5$ and $\sigma = 5$. Initially r_5 and r_{10} are picked as the centers of stars in W. Since $\sigma > \lceil \frac{t_\alpha}{2} \rceil$, r_1 is also the center of a star. Since $\sigma + t_\alpha \lfloor \frac{q-\sigma}{t_\alpha} \rfloor > q - \lceil \frac{t_\alpha}{2} \rceil$, r_q is not the center of a star.

samples $\sigma \in \{1, \ldots, t\}$, then we say that the terminals $r_{\sigma+tj}$ are *marked* by the algorithm. Moreover, if $\sigma > \lceil \frac{t_\alpha}{2} \rceil$ (resp. $\sigma + t_\alpha \lfloor \frac{q-\sigma}{t_\alpha} \rfloor \leq q - \lceil \frac{t_\alpha}{2} \rceil$) then r_1 (resp. r_q) is also considered marked.

1. Consider edge $e = s_j s_{j+1} \in O$, with $j \in \{\lceil \frac{t}{2} \rceil, \ldots, q - \lceil \frac{t}{2} \rceil\}$. Let $m \in \{j - \lfloor \frac{t}{2} \rfloor, \ldots, j + \lfloor \frac{t}{2} \rfloor\}$, such that $\sigma \mod t = m \mod t$. Observe that in this case r_m is marked. If $m = j - x$ for $x \in \{0, \ldots, \lfloor \frac{t}{2} \rfloor\}$, then $w(s_j s_{j+1}) = \lceil \frac{t}{2} \rceil - x$. Similarly if $m = j + x$ for $x \in \{1, \ldots, \lfloor \frac{t}{2} \rfloor\}$, then $w(s_j s_{j+1}) = \lceil \frac{t}{2} \rceil - x + 1$. Since $m \mod t = \sigma \mod t$ with probability $\frac{1}{t}$, we have $\mathbb{E}[H_{w(s_j s_{j+1})}] = \frac{1}{t} + \frac{2}{t} \sum_{k=2}^{\lceil \frac{t}{2} \rceil} H_k$.

   Now assume $j < \lceil \frac{t}{2} \rceil$ (the case $j > q - \lceil \frac{t}{2} \rceil$ can be handled similarly). Recalling that since t is odd it is not hard to determine the value of $w(s_j s_{j+1})$ by cases, depending on the value of σ.

   (a) $1 \leq \sigma \leq j$: Then $w(s_j s_{j+1}) = \lceil \frac{t}{2} \rceil + \sigma - j$.

   (b) $j + 1 \leq \sigma \leq \lceil \frac{t}{2} \rceil$: Then $w(s_j s_{j+1}) = j$.

   (c) $\lceil \frac{t}{2} \rceil + 1 \leq \sigma \leq j + \lfloor \frac{t}{2} \rfloor$: Then $w(s_j s_{j+1}) = \lceil \frac{t}{2} \rceil - \sigma + j + 1$.

   (d) $j + \lceil \frac{t}{2} \rceil \leq \sigma \leq t$: Then $w(s_j s_{j+1}) = \sigma - j - \lceil \frac{t}{2} \rceil + 1$.

$$
\begin{aligned}
&\mathbb{E}[H_{w(s_j s_{j+1})}] = \\
&= \frac{1}{t} \left(\sum_{\sigma=1}^{j} H_{\lceil \frac{t}{2} \rceil + \sigma - j} + \sum_{\sigma=j+1}^{\lceil \frac{t}{2} \rceil} H_j + \sum_{\sigma=\lceil \frac{t}{2} \rceil + 1}^{j + \lfloor \frac{t}{2} \rfloor} H_{\lceil \frac{t}{2} \rceil - \sigma + j + 1} + \sum_{\sigma=j+\lceil \frac{t}{2} \rceil}^{t} H_{\sigma - j - \lceil \frac{t}{2} \rceil + 1} \right) \\
&= \frac{1}{t} \left(\sum_{i=\lceil \frac{t}{2} \rceil - j + 1}^{\lceil \frac{t}{2} \rceil} H_i + \left(\left\lceil \frac{t}{2} \right\rceil - j \right) H_j + \sum_{i=2}^{j} H_i + \sum_{i=1}^{\lceil \frac{t}{2} \rceil - j} H_i \right)
\end{aligned}
$$

$$= \frac{1}{t}\left(\sum_{i=1}^{\lceil \frac{t}{2} \rceil} H_i + \left(\left\lceil \frac{t}{2} \right\rceil - j\right) H_j + \sum_{i=2}^{j} H_i\right) < \frac{1}{t}\left(1 + 2\sum_{i=2}^{\lceil \frac{t}{2} \rceil} H_i\right).$$

2. Consider edge $e = s_j r_j \in L$. We first show the bound for $j \in \{1, \ldots, q\}$. Algorithm 4 marks terminal r_i with probability $\frac{1}{t}$. If r_i is marked, then $w(e) \leq t$. If r_i is not marked, then $w(e) = 1$. Therefore, $\mathbb{E}[H_{w(e)}] \leq \frac{1}{t} H_{t+1} + \frac{t-1}{t}$

   Now consider edge $e = s_1 r_1$ (the case $e = s_q r_q$ can be handled similarly). We consider specific values of $\sigma \in \{1, \ldots, t\}$ sampled by Algorithm 4. With probability $\frac{1}{t}$, we have $\sigma = 1$, so r_1 is marked initially and $w(e) = \lceil t/2 \rceil$. For $\sigma = 2, \ldots, \lceil t/2 \rceil$, r_1 is unmarked and $w(e) = 1$. If $\sigma > \lceil t/2 \rceil$, then r_1 is marked by the algorithm and $w(e) = \sigma - \lceil t/2 \rceil$. Therefore, we can see

   $$\mathbb{E}[H_{w(r_1 s_1)}] = \frac{1}{t}\left(H_{\lceil t/2 \rceil} + \left\lfloor \frac{t}{2} \right\rfloor + \sum_{k=1}^{t - \lceil t/2 \rceil} H_k\right)$$

   We let $g(t)$ be equal to the equality above. It remains to show that $g(t) \leq \frac{1}{t} H_{t+1} + \frac{t-1}{t} := f(t)$ for $t \in \{1, 3, 5\}$.

   $$g(1) = H_1 = 1 < H_2 = f(1)$$
   $$g(3) = \frac{1}{3}(H_2 + 1 + H_1) = 1.1\bar{6} < 1.36\bar{1} = \frac{1}{3}(H_4 + 2) = f(3)$$
   $$g(5) = \frac{1}{5}(H_3 + 2 + H_1 + H_2) = 1.2\bar{6} < 1.29 = \frac{1}{5}(H_6 + 4) = f(5)$$

Combining these two facts gives us the bound on $\lambda_{L_i}(t)$, for $t \in \{1, 3, 5\}$. To prove the second part of the statement, we denote

$$f(\alpha) := \frac{1}{\alpha + 1}\left(\frac{1}{t_\alpha} H_{t_\alpha + 1} + \frac{t_\alpha - 1}{t_\alpha} + \alpha\left(\frac{1}{t_\alpha} + \frac{2}{t_\alpha}\sum_{i=2}^{\lceil \frac{t_\alpha}{2} \rceil} H_i\right)\right).$$

Suppose $\alpha \in [0, 0.3\bar{5}]$. Then by definition $t_\alpha = 5$ and therefore we have $f(\alpha) = 1.5\bar{3} - \frac{1.5\bar{3} - 1.29}{\alpha + 1} \leq \frac{991}{732}$. Suppose $\alpha \in (0.3\bar{5}, 1)$. In this case $t_\alpha = 3$, thus $f(\alpha) = 1.\bar{3} + \frac{1.36\bar{1} - 1.\bar{3}}{\alpha + 1} < \frac{991}{732}$. Furthermore for $\alpha \geq 1$, $t_\alpha = 1$; so $f(\alpha) = 1 + \frac{0.5}{\alpha + 1} \leq 1.25$.

□

We are now ready to prove the following:

Lemma 4.3. $\mathbb{E}[\bar{\nu}_T(W)] \leq \frac{991}{732}$.

Proof. One observes:

$$\sum_{e \in L \cup O} c(e)\mathbb{E}[H_{\bar{w}(e)}] \leq \sum_{e \in L} c(e)\lambda_L(t_\alpha) + \sum_{e \in O} c(e)\lambda_O(t_\alpha) = (\lambda_L(t_\alpha) + \alpha\lambda_O(t_\alpha)) \sum_{e \in L} c(e)$$

Therefore $\mathbb{E}[\nu_T(W)]$ is bounded by:

$$\frac{\sum_{e \in L \cup O} c(e)\mathbb{E}[H_{\bar{w}(e)}]}{\sum_{e \in L \cup O} c(e)} \leq \frac{(\lambda_L(t_\alpha) + \alpha\lambda_O(t_\alpha)) \sum_{e \in L} c(e)}{(\alpha+1)\sum_{e \in L} c(e)} = \frac{\lambda_L(t_\alpha) + \alpha\lambda_O(t_\alpha)}{\alpha+1} \leq \frac{991}{732}.$$

where the last inequality follows using Lemma 4.2. □

We now have all the ingredients necessary to prove Theorem 4.2.

Proof of Theorem 4.2. We assume we are given a Steiner-Claw Free instance (G, R, c) with optimal solution $T = (R \cup S^*, E^*)$. As we have shown, we can decompose T into components $T_1, \ldots, T_\tau$ where each component is a maximal subtree of T whose leaves are terminals and the internal nodes are Steiner nodes. We show how to compute a witness tree W_i for each component T_i. It is not difficult to see that the union $W := W_1 \cup \cdots \cup W_\tau$ is a feasible witness tree for T whose objective function $\bar{\nu}_T(W)$ is bounded by the maximum among $\bar{\nu}_{T_i}(W_i)$.

Hence, we assume that T is a single component with Steiner nodes $s_1, \ldots, s_q$, whose induced subgraph is a path. Moreover, as we have previously described, we can assume without loss of generality that each s_j is adjacent to exactly one terminal $r_j \in R$. Where this can be achieved by replacing any Steiner node adjacent to $p > 1$ terminals with a path of length p 0-cost edges each adajcent to exactly one terminal, and replacing any Steiner node adjacent to 0 terminals with a single edge of appropriate cost connecting its 2 Steiner neighbours.

By Lemma 4.1, if $q < 5$ then the claim holds. If $q \geq 5$, we apply Algorithm 4 and by combining Lemma 4.3 with Theorem 4.1 in which γ is replaced by the supremum taken over all Steiner-claw free instances (rather than over all Steiner Tree instances) we complete the proof. □

4.2 Steiner-claw Free Lower Bound

The goal of this section is to prove Theorem 4.3. The structure of this section is similar to that of Section 3.4.3. We will consider a Steiner-Claw Free instance $(G = (R \cup S, E), c)$, where the Steiner nodes S consist of a path $s_0, s_1, \ldots, s_{q+1}$ for

$q > \frac{2}{\varepsilon}$, with $q \in \mathbb{Z}$, and each $s_i \in S$ is adjacent to exactly one terminal $r_i \in R$. See Figure 4.4.

To prove Theorem 4.3, we first show in Section 4.2.1 that there is a witness tree of least cost that is laminar in a theorem closely reminiscent of Theorem 3.14.

With Theorem 4.3, we show in Subsection 4.2.2 that any laminar witness tree, W, can be represented as a family of stars whose centres form a path in W. We will show that each we can find W^* by replacing these stars with bigger or smaller ones to find a new with minimum cost, and then applying Theorem 4.4 to see that this is witness tree of least cost.

4.2.1 Laminar Witness Trees

In this section, we prove some key structural properties of witness trees. We assume we are given an Edge Witness Tree instance $T = (V, E)$ with leaves R (and edge costs $c : E \to \mathbb{R}_{\geq 0}$), where R denotes the leaves of T, we will show that we can characterize witness trees minimizing $\nu_T(W)$ ($\bar{\nu}_T(W)$) using the following notion of *laminarity*. Given a witness tree $W = (R, E_W)$, we say edges $f_1 f_2, f_3 f_4 \in E_W$ *cross* if the f_1-f_2 and f_3-f_4 paths in T share an internal node but not an endpoint. We say that W is *laminar* if it has no crossing edges. For nodes $u, v \in V$, we denote by T_{uv} the path in T between the nodes u and v. Similarly, for $e \in E_W$, we denote by T_e the path in T between the endpoints of e.

Theorem 4.4. *Given an instance of the Edge Witness Tree problem $T = (V, E)$ with edge costs c, let $\mathcal{W}$ be the family of all witness trees for T. Then there exists a laminar witness tree W such that $\bar{\nu}_T(W) = \min_{W' \in \mathcal{W}} \bar{\nu}_T(W')$.*

Proof. To prove this Theorem, we first show that there is a witness tree W minimizing $\bar{\nu}_T(W)$ such that the subgraph of W induced on the terminals of any maximally connected region $F \subseteq E$ of 0 cost edges is a star. To show this, we will first show that for any edge $e \in E_W$ with $T_e \cap F \neq \emptyset$, e must have an endpoint in $R(F)$. We will then show that the subgraph of W induced on $R(F)$ must be connected, and combined with the fact that $c(F) = 0$, can be assumed to be a star.

First, we show that for any edge $e \in E_W$ with $T_e \cap F \neq \emptyset$, e has an endpoint in $R(F)$. Assume for sake of contradiction there is edge $uv \in E_W$, with $u, v \notin R(F)$ but $T_{uv} \cap F \neq \emptyset$. Consider connected components W_u and W_V of $W \backslash \{uv\}$, that each contain u and v respectively. Without loss of generality we assume that $|W_u| > 1$, and consider terminal $r \in R(W_u \cap F)$ such that T_{ur} can be decomposed

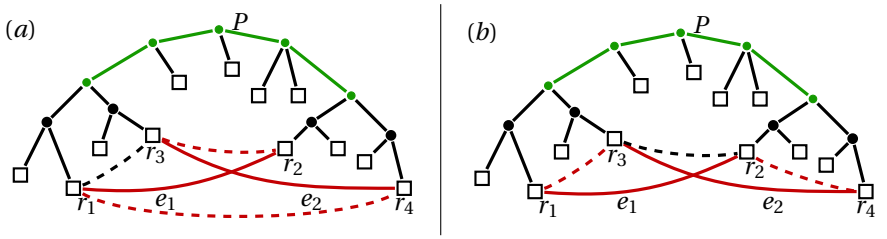


Figure 4.2: In both figures we have a tree, T, shown with black edges and green edges, with leaves, R, denoted by squares. Crossing edges e_1 and e_2 are shown with solid red edges. The green edges denote the path P. Figure (a): In this case, r_1 and r_3 are in the same component of $W\backslash\{e_1, e_2\}$, represented by the dashed black edge. We can replace e_1 with r_2r_3 or replace e_2 with r_1r_4 (red dashed edges). Figure (b): In this case, r_3 and r_2 are in the same component, denoted by the black dashed edge. We can replace e_1 and e_2 with r_1r_3 and r_2r_4 (red dashed edges).

into a path in F, and a path containing u. Since T_{uv} intersects F it must contain the subpath $T_{ur}\backslash F$, as T is a tree, and similarly T_{vr} must contain the subpath $Tur \cap F$. By construction $c(T_{ur}\backslash F) > 0$ and $c(Tur \cap F) = 0$, and thus by replacing uv with rv in W, we reduce the cost of W, contradicting the assumption of minimality. See Figure 4.3. Therefore, we assume that every edge $e \in E_W$ with $E[T_e] \cap F \neq \emptyset$, e has an endpoint in F.

We assume for the sake of contradiction that a maximally connected region $F \subseteq E$ of 0 cost edges exists where the subgraph of W induced on $R(F)$ is a family of connected components $W_1, \ldots, W_i$, for $i > 1$. Consider components W_1 and W_2, and let $e \in E_W$ be the edge on the unique path between these components that is incident to W_2. By construction we have $\sum_{e' \in E[T_e]} c(e') > 0$. Let $f \in \binom{R}{2}$ be an edge with one endpoint in W_1 and the other endpoint in W_2. Clearly, $\sum_{e' \in E[T_f]} c(e') = 0$, since the terminals of W_1 and W_2 are contained in F in T. We define $W' := W \cup \{f\}\backslash\{e\}$. Since $\sum_{e' \in E[T_f]} c(e') = 0 < \sum_{e' \in E[T_e]} c(e')$, it is not hard to see that $\bar{v}_T(W') < \bar{v}_T(W)$, contradicting the minimality of W. Therefore, the subgraph of W induced on the terminals of F is a subtree of W. We can replace the edges of this subtree to be a star for no change in the value of $\bar{v}_T(W)$. Therefore, we can assume that for every 0 cost region of T this claim holds. Furthermore, for each such star, we can assume that the edges of W incident to this star have endpoint at the star's centre.

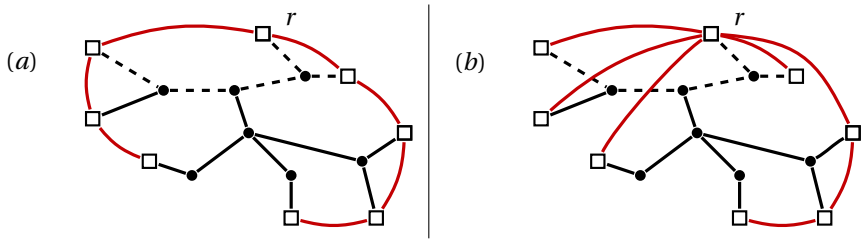


Figure 4.3: In both figures we have a tree, T, shown with solid and dashed black edges, with leaves, R, denoted by squares. The edge cost of the solid edges is greater than 0, and the edge cost of the dashed edges is 0. Witness trees are represented by red edges. Figure (a): In this case, W has two edges whose paths between their endpoints contain 0 cost edges but whose endpoint are not within a region of zero cost edges. Furthermore, the edges of W whose endpoints are both in the zero cost region are not in a star. Figure (b): In this case, the edges described previously are replaced with edges that have endpoint at terminal r.

We assume for the sake of contradiction that the witness tree W minimizing $\bar{\nu}_T(W)$ is not a laminar witness tree, and that it has the minimum number of pairs of crossing edges. That is, there exist distinct leaves $r_1, r_2, r_3, r_4 \in R$ such that $e_1 = r_1r_2, e_2 = r_3r_4 \in E_W$ are crossing. We denote the path $T_{e_1} \cap T_{e_2}$ by P. We denote the shortest path from P to r_i by P_i.

Since e_1 and e_2 are crossing edges, one of $T_{r_1r_3}$ or $T_{r_1r_4}$ contains exactly one node of P. The same is true for $T_{r_2r_3}$ or $T_{r_2r_4}$. Without loss of generality, we assume that the paths $T_{r_1r_3}$ and $T_{r_2r_4}$ contain exactly one node of P. We consider by cases which component of $W\backslash\{e_1, e_2\}$ contains two of r_1, r_2, r_3 and r_4. Note, that this is very similar to the cases laid out in Theorem 3.14 and demonstrated in Figure 4.2.

- Case: r_1 and r_3 (or similarly, r_2 and r_4) are in the same component $W\backslash\{e_1, e_2\}$. If $\sum_{e\in E[P_1]} c(e) + \sum_{e\in E[P_3]} c(e) = 0$, then as we have shown above, e_1 and e_2 are assumed to share an endpoint, and are thus not crossing. So we have that $\sum_{e\in E[P_1]} c(e) + \sum_{e\in E[P_3]} c(e) > 0$. Consider $W' := W \cup \{r_2r_3\}\backslash\{e_1\}$ and $W'' := W \cup \{r_1r_4\}\backslash\{e_2\}$. If $\bar{\nu}_T(W) - \bar{\nu}_T(W') > 0$, this contradicts the mini-

mality of $\bar{v}_T(W)$. Therefore, we can see

$$0 \le c(E)(\bar{v}_T(W') - \bar{v}_T(W)) = \sum_{e \in E[P_3]} \frac{c(e)}{w(e)+1} - \sum_{e \in E[P_1]} \frac{c(e)}{w(e)}$$
$$< \sum_{e \in E[P_3]} \frac{c(e)}{w(e)} - \sum_{e \in E[P_1]} \frac{c(e)}{w(e)+1} = c(E)(\bar{v}_T(W) - \bar{v}_T(W''))$$

Clearly, we have $\bar{v}_T(W'') < \bar{v}_T(W)$, contradicting the minimality of $\bar{v}_T(W)$.

- Case: r_2 and r_3 (or similarly, r_1 and r_4) are in the same component of $W \setminus \{e_1, e_2\}$. In this case, consider $W' := \{r_1 r_3, r_2 r_4\} \setminus \{e_1, e_2\}$. If $\sum_{e \in E[P]} c(e) = 0$, then clearly $\bar{v}_T(W') = \bar{v}_T(W)$, but W' has one fewer crossing pair, contradicting the assumption that W minimizes the number of such pairs, thus $\sum_{e \in E[P]} c(e) > 0$. Without loss of generality, we can assume that $|V(P)| > 1$, because if $|V(P)| = 1$ then we can reduce to the previous case by relabelling the nodes r_1, r_2, r_3, and r_4. Therefore, we can see the following

$$c(E)\left(\bar{v}_T(W') - \bar{v}_T(W)\right) \le -\sum_{e \in E[P]} \frac{c(e)}{w(e)} < 0$$

Clearly, we have $\bar{v}_T(W') < \bar{v}_T(W)$, contradicting the minimality of $\bar{v}_T(W)$. □

Incidentally, this has the following side implication. The authors of [48] gave a dynamic program (that is also a bottom-up approach) to compute the best possible witness tree obtainable with a marking-and-contraction scheme. Our structural results imply that their dynamic program computes an optimal solution for the EWT problem (though for the purpose of the approximation analysis, being able to compute the best witness tree is not that relevant: being able to bound γ is what matters).

4.2.2 Decomposing Witness Trees

We let $L \subseteq E$ denote the terminal incident edges and $O = E \setminus L$ denote the edges between Steiner nodes. For $e \in O$, let $c(e) = \alpha := \frac{32}{90}$, and for $e \in L$, let $c(e) = 1$. Clearly, the optimal Steiner of such an instance is $T = G$.

For terminal $r_i \in R$, and $x_L^i, x_R^i \in \mathbb{Z}_{\ge 0}$, we call $W(r_i, x_L^i, x_R^i) \subseteq \binom{R}{2}$ a *section* if $\mathscr{W}(r_i, x_L^i, x_R^i)$ is a star over nodes the $r_{i-x_L^i}, \ldots, r_{i+x_r^i}$ with centre r_i. The following lemma shows that we can decompose any laminar witness tree into a family of sections and a path between the centres of the sections.

Lemma 4.4. *Let $W = (R, E_W)$ be a laminar witness tree of the tree $T = (V, R)$. Let $\tilde{\mathscr{I}}(W) \subseteq \{0, 1, \ldots, q+1\}$ denote the indices of terminals whose degree in W is greater than 1. And let $\mathscr{I}(W) \coloneqq \tilde{\mathscr{I}}(W) \cup \{0, q+1\}$.*

Then for every $i \in \mathscr{I}(W)$, there are $x_L^i, x_R^i \in \mathbb{Z}_{\geq 0}$ such that

1. *$W = \bigcup_{i \in \mathscr{I}(W)} \mathscr{W}(r_i, x_L^i, x_R^i) \bigcup \mathscr{P}(\mathscr{I}(W))$, and;*
2. *$\mathscr{W}(r_i, x_L^i, x_R^i) \cap \mathscr{W}(r_j, x_L^j, x_R^j) = \emptyset$ for all $i \neq j \in \in \mathscr{I}(W)$.*

Where $\mathscr{P}(\mathscr{I}(W))$ is the path across the terminals of $\mathscr{I}(W)$ in increasing order of index.

Proof. To prove this claim, we exploit laminarity and the structure of T.

First, we wish to show that if $r_i \in R$ has degree greater than 1 in W, then it is adjacent to at most 2 other terminals that have degree greater than 1. Furthermore, we show that if r_i is adjacent to terminals r_a and r_b with degree greater than 1, then $a < i < b$.

We assume for the sake of contradiction that r_i is adjacent to at terminals r_a, r_b, r_c with degree greater than 1. Without loss of generality we assume that $a < b < i$. Since r_b has degree at least 2 and W contains no parallel edges, r_b is adjacent to some terminal $r_s \neq r, r_a$. However, the path in T from r_a to r_i contains the Steiner node s_b. Therefore the edges $r_b r_s$ and $r_a r_i$ are crossing edges, contradicting the assumption that W is laminar.

We now wish to find the values x_L^i and x_R^i for every $r_i \in \mathscr{I}(W)$. For the terminal $r_0 \in \mathscr{I}(W)$, we set $x_L^0 = 0$. Similarly, for the terminal $r_{q+1} \in \mathscr{I}(W)$ with greatest index, we set $x_R^{q+1} = 0$.

Now, consider terminals $r_i, r_j \in \mathscr{I}(W)$ with $i < j$, that are adjacent in W. By combining the above fact and laminarity we can see that every terminal in $\{r_{i+1}, \ldots, r_{j-1}\}$ is degree 1 and adjacent to either r_i or r_j. Furthermore, by laminarity, there must exist some values $x_R^i, x_L^j \in \mathbb{Z}_{\geq 0}$ with $i + x_R^i = j - x_L^j - 1$, such that $\{r_{i+1}, \ldots, r_{i+x_R^i}\}$ are adjacent to r_i and $\{r_{j-x_L^j}, \ldots, r_{j-1}\}$ are adjacent to r_j.

By repeating this argument for every pair of adjacent terminals in $\mathscr{I}(W)$, we complete our construction. □

Given a witness tree $W = (R, E_W)$, we say the sections found by employing Lemma 4.4 the *canonical* sections of W. See Figure 4.4 for an example of a section where $q = 5$. Note that the canonical sections found by Lemma 4.4 always include the sections $\mathscr{W}(r_0, 0, x_R^0)$ and $\mathscr{W}(r_{q+1}, x_L^{q+1}, 0)$.

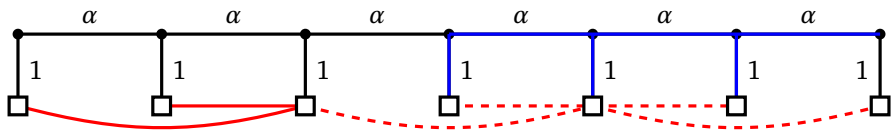


Figure 4.4: Depiction of the lower bound instance with sections for witness tree W marked in red edges. $q = 5$. centres r_0, r_2, r_4 and r_6. There are sections $W(r_0, 0, 0)$, $W(r_2, 1, 0)$, $W(r_4, 1, 1)$, and $W(r_6, 0, 0)$. The section $W(r_4, 1, 1)$ is the red dashed edges. The subtree $T(r_4, 1, 1)$ is shown by blue edges.

We now wish to define a subtree of T that will correspond to a section $\mathscr{W}(r_i, x_L^i, x_R^i)$, allowing us to relate sections to the cost on the edges of T. Given a section $\mathscr{W}(r_i, x_L^i, x_R^i)$, we define $T(r_i, x_L^i, x_R^i) \subseteq T$ as the subtree induced on $\{s_{i-x_L^i}, \ldots, s_{i+x_R^i+1}\} \cup \{r_{i-x_L^i}, \ldots, r_{i+x_R^i}\}$ (obviously, if $T(r_i, x_L^i, x_R^i)$ contains r_{q+1}, then it does not include s_{q+2} as it does not exist). See Figure 4.4 for an example of $T(r_i, 1, 1)$, denoted in blue edges.

The following observation shows that we can decompose T into the subtrees that correspond to the sections of a laminar witness tree W.

Observation 4.1. *Let $W = (R, E_W)$ be a laminar witness tree of the tree $T = (V, R)$. Let the canonical sections of W be $\{\mathscr{W}(r_i, x_L^i, x_R^i)\}_{i \in \mathscr{I}(W)}$. Then the following holds*

1. $T = \cup_{i \in \mathscr{I}(W)} T(r_i, x_L^i, x_R^i)$, *and;*
2. $T(r_i, x_L^i, x_R^i) \cap T(r_j, x_L^j, x_R^j) = \emptyset$ *for all* $i \neq j \in \mathscr{I}(W)$.

The following crucial lemma will be useful in allowing us to replace a section of a laminar witness tree with smaller ones, or merge two sections together into a new section to find new laminar witness trees.

Lemma 4.5. *Let $W = (R, E_W)$ be a laminar witness tree of the tree $T = (V, R)$. Let the canonical sections of W be $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota \in \mathscr{I}(W)}$.*

Then the sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota \in \mathscr{I}(W) \setminus \{i\}} \cup \{\mathscr{W}(r_j, x_L^j, x_R^j), \mathscr{W}(r_k, x_L^k, x_R^k)\}$ are the canonical sections of a laminar witness tree W' iff the following hold:
1) $r_j, r_k \in \{r_{i-x_L^i}, \ldots, r_{i+x_R^i}\}$; 2) $j + x_R^j + 1 = k - x_L^k$; 3) $i - x_L^i = j - x_L^j$, and; 4) $i + x_R^i = k + x_R^k$.

Proof. ($\Rightarrow$) Assume for the sake of contradiction that one of 1)–4) does not hold. If 1) does not hold, then we contradict Observation 4.1.(2).

If 2) does not hold, then first we assume $j + x_R^j + 1 < k - x_L^k$, from which it is clear that $r_{j+x_R^j+1}$ is not covered by $\mathscr{W}(r_j, x_L^j, x_R^j)$, not is it covered by $\mathscr{W}(r_k, x_L^k, x_R^k)$, and since W' is assumed to be laminar, it cannot be adjacent to any other centres, and thus we find a contradiction. If we assume $j + x_R^j + 1 > k - x_L^k$ instead, then r_j is incident to $r_{k-x_L^k}$ which is incident to r_k which is incident to r_j, contradicting the assumption that W' is a tree.

If 3) or 4) do not hold, the proof is similar to the case when 2) does not hold. Thus the claim is proven.

($\Leftarrow$) To see this claim, we only need to show that $\mathscr{W}(r_j, x_L^j, x_R^j) \cup \mathscr{W}(r_k, x_L^k, x_R^k) \cup \{r_j r_k\}$ is a tree spanning the same nodes as $\mathscr{W}(r_i, x_L^i, x_R^i)$. It is not hard to see that r_j and r_k together are adjacent to every terminal $r \in \{r_{j-x_L^j}, \dots, r_{j+x_R^j}\} \cup \{r_{k-x_L^k}, \dots, r_{k+x_R^k}\} = \{r_{i-x_L^i}, \dots, r_{i+x_R^i}\}$, furthermore, it is clear that $\{r_{j-x_L^j}, \dots, r_{j+x_R^j}\} \cap \{r_{k-x_L^k}, \dots, r_{k+x_R^k}\} = \emptyset$. □

For a witness tree $W = (R, E_W)$ with vector w imposed on E, we define the following function that will allow use to compare the cost of witness trees by using their canonical sections $h_W : 2^E \to \mathbb{R}_{\geq 0}$ where $h_W(F) = \sum_{e \in F} H_{w(e)}$. Note, that $h_W(E) = c(E)\bar{v}_T(W)$. The following lemma will allow us to relate the choice of canonical sections of a witness tree to its corresponding subtree of T.

Lemma 4.6. *Let $W = (R, E_W)$ be a laminar witness tree of the tree $T = (V, R)$ with vector w imposed on E. Let $\mathscr{W}(r_i, x_L^i, x_R^i)$ be a canonical section of W for $i = 1, \dots, q$. Then*

$$h_W(T(r_i, x_L^i, x_R^i)) = \alpha \left(\sum_{j=2}^{x_L^i + L_i} H_j + \sum_{j=1}^{x_R^i + R_i} H_j \right) + x_L^i + x_R^i + H_{x_L^i + x_R^i + L_i + R_i}$$

Proof. We first consider edge $e \in L \cap T(r_i, x_L^i, x_R^i)$. Clearly, $w(e) = x_L^i + x_R^i + L_i + R_i$ if e is incident to r_i, and $w(e) = 1$ otherwise. Therefore, $h_W(L \cap T(r_i, x_L^i, x_R^i)) = x_L^i + x_R^i + H_{x_L^i + x_R^i + L_i + R_i}$

Now consider edge $e = s_{i+j} s_{i+j+1} \in O \cap T(r_i, x_L^i, x_R^i)$. For $j = 0, \dots, x_R^i$, since we know that r_i is adjacent to every terminal in $r_{i+1}, \dots, r_{i+x_R^k}$ and $R_i \in \{0, 1\}$ many terminals with index greater than i, is clear that $w(e) = x_R^i - j + R_i$. Similarly, for $j = -1, \dots, x_L^i$, we have $w(e) = x_L^i + 1 + j + L_i$. Therefore,

$$h_W(O \cap T(r_i, x_L^i, x_R^i)) = \sum_{j=-1}^{-x_L^i} H_{w(s_{i+j} s_{i+j+1})} + \sum_{j=0}^{x_R^i} H_{w(s_{i+j} s_{i+j+1})} = \sum_{j=2}^{x_L^i + 1} H_j + \sum_{j=1}^{x_R^i + 1} H_j$$

The claim holds by observing $h_W(T(r_i, x_L^i, x_R^i)) = h_W(L \cap T(r_i, x_L^i, x_R^i)) + h_W(O \cap T(r_i, x_L^i, x_R^i))$. □

The case for $i = q+1$ can very easily be found similarly

Corollary 4.1. *Let $W = (R, E_W)$ be a laminar witness tree of the tree $T = (V, R)$ with vector w imposed on E. Let $\mathscr{W}(r_0, 0, x_R^0)$ and $\mathscr{W}(r_{q+1}, x_L^{q+1}, 0)$ be canonical sections of W. Then*

$$h_W(T(r_{q+1}, x_L^{q+1}, 0)) = \alpha \sum_{j=2}^{x_L^{q+1}+L_{q+1}} H_j + x_L^i + H_{x_L^{q+1}+L_{q+1}}$$

$$h_W(T(r_0, 0, x_R^0)) = \alpha \sum_{j=1}^{x_R^0+R_i} H_j + x_R^0 + H_{x_R^0+R_0}$$

4.2.3 Finding W^*

Let W^* be a witness tree that minimizes $\bar{v}_T(W^*)$. Recall that we can assume W^* to be laminar by Theorem 4.4. To prove Theorem 4.3 we prove structural results about W^* and its canonical sections. Let w^* be the vector imposed on E by W^*. Recall that the terminal incident edges $L \subseteq E$ have cost 1 and edges in $O \subseteq E$ have cost $\alpha := \frac{32}{90}$. Beginning with the following lemma that shows that W^* contains canonical sections with centres r_0 and r_{q+1} that each contain exactly one terminal.

Lemma 4.7. *There is an optimal witness tree W^* that has canonical sections $\mathscr{W}(r_0, 0, 0)$ and $\mathscr{W}(r_{q+1}, 0, 0)$.*

Proof. If W^* does not have a canonical section that contains r_0 as a centre, then let $r_i \in \mathscr{I}(W^*)$ be the centre of canonical section $\mathscr{W}(r_i, x_L^i, x_R^i)$ with least index. Note that since $r_i \neq r_0$ we have $x_L^i > 0$. We apply Corollary 4.5 to find witness tree W' with canonical sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota \in \mathscr{I}(W^*) \setminus \{i\}} \cup \{\mathscr{W}(r_0, 0, 0), \mathscr{W}(r_i, x_L^i - 1, x_R^i)\}$. It is not hard to see that

$$h_{W^*}(T(r_i, x_L^i, x_R^i)) = h_{W'}(T(r_0, 0, 0) \cup T(r_i, x_L^i - 1, x_R^i))$$

The case where W^* does not contain a canonical section that contains r_{q+1} as a centre can be handled similarly.

Now we assume for the sake of contradiction that W^* has canonical section $\mathscr{W}(r_0, 0, x_R^0)$ where $x_R^0 > 0$. The case where W^* has the canonical section $\mathscr{W}(r_{q+1}, x_L^{q+1}, 0)$ where $x_L^{q+1} > 0$, can be handled similarly.

We apply Lemma 4.5 to find witness tree W' that has canonical sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota\in\mathscr{I}(W^*)\setminus\{0\}} \cup \{\mathscr{W}(r_0,0,0), \mathscr{W}(r_1,0,x_R^0-1)\}$. Note that W' exists since $q > \frac{2}{\varepsilon} > 0$, so $q \geq 1$. Let w' be the vector imposed on E by W'. It is not hard to see that for all $e \in E\setminus\{s_0r_0, s_0s_1, s_1r_1\}$, we have $w(e) = w'(e)$, while $w(s_0r_0) = w(s_0s_1) = w'(s_1r_1) = x_R^0 + R_i$, and $w'(s_0r_0) = w'(s_0s_1) = w(s_1r_1) = 1$, and for all other edges they $w(e) = w'(e)$. Thus the difference between $h_{W^*}(T)$ and $h_{W'}(T)$ is:

$$\begin{aligned}&H_{w(s_0r_0)} - H_{w'(s_0r_0)} + H_{w(s_1r_1)} - H_{w'(s_1r_1)} + \alpha(H_{w(s_0s_1)} - H_{w'(s_0s_1)})\\ =&H_{x_R^0+R_i} - 1 + 1 - H_{x_R^0+R_i} + \alpha(H_{x_R^0+R_i} - 1) = \alpha(H_{x_R^0+R_i} - 1) > 0\end{aligned}$$

Thus, $\bar{\nu}_T(W') > \bar{\nu}_T(W^*)$, contradicting the minimality of $\bar{\nu}_T(W^*)$. Demonstrating that $x_L^{q+1} = 0$ in W^* can be shown similarly. □

By applying Lemma 4.7 we assume from now on that W^* has canonical sections $\mathscr{W}(r_0,0,0)$ and $\mathscr{W}(r_{q+1},0,0)$. Applying Corollary 4.1, we see $h_{W^*}(T(r_0,0,0)) = c(s_0r_0)H_{w(s_0r_0)} + c(s_0s_1)H_{w(s_0s_1)} = 1+\alpha$ and we also see $h_{W^*}(T(r_{q+1},0,0)) = c(s_{q+1}r_{q+1})H_{w(s_{q+1}r_{q+1})} = 1$.

The following lemma shows that the centre of every canonical section of W^* is, in some sense, in the "middle" of its terminals.

Lemma 4.8. *Let $\mathscr{W}(r_i, x_L^i, x_R^i)$ be a canonical section of W^*. Then $|x_L^i - x_R^i| \leq 1$.*

Proof. The claim is immediately clear for the cases where $i = 0, q+1$. So we consider the case where $i \in \{1,\dots,q\}$. Note that, by definition of the canonical sections, we have $R_i = L_i = 1$.

We assume for the sake of contradiction that $\mathscr{W}(r_i, x_L^i, x_R^i)$ is a canonical section of W^* with $|x_L^i - x_R^i| > 1$. Furthermore, we assume without loss of generality that $x_R^i > x_L^i + 1$. The case where $x_R^i < x_L^i + 1$ can be handled similarly. We apply Lemma 4.5, and consider witness tree W' with canonical sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota\in\mathscr{I}(W^*)\setminus\{i\}} \cup \{\mathscr{W}(r_{i+1}, x_L^i+1, x_R^i-1)\}$. By applying Lemma 4.6 can see that $h_{W^*}(E) - h_{W'}(E) = h_{W^*}(T(r_i, x_L^i, x_R^i)) - h_{W'}(T(r_{i+1}, x_L^i+1, x_R^i-1))$ is equal to

$$\begin{aligned}\alpha\left(\sum_{j=2}^{x_L^i+1} H_j + \sum_{j=1}^{x_R^i+1} H_j - \sum_{j=2}^{x_L^i+L_1+1} H_j - \sum_{j=1}^{x_R^i} H_j\right) &= \alpha(H_{x_R^i+1} - H_{x_L^i+2})\\ &> \alpha(H_{x_L^i+2} - H_{x_L^i+2}) = 0\end{aligned}$$

Therefore, $\bar{\nu}_T(W') < \bar{\nu}_T(W^*)$, contradicting the minimality of W^*. □

For canonical section $\mathscr{W}(r_i, x_L^i, x_R^i)$ of W^* we can not only apply Lemma 4.8 to assume that $|x_R^i - x_L^i| \le 1$, but we can assume without loss of generality that $x_R^i \ge x_L^i$. To see this, first suppose $x_R^i < x_L^i$, which by Lemma 4.8 implies that $x_R^i + 1 = x_L^i$. We then apply Lemma 4.5 to find witness tree W' with canonical sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota \in \mathscr{I}(W^*)\setminus\{i\}} \cup \{\mathscr{W}(r_{i+1}, x_L^i - 1, x_R^i + 1)\}$. Finally, it is not hard to see by Lemma 4.6 that $\bar{\nu}_T(W^*) = \bar{\nu}_T(W')$.

The following lemma shows that W^* will only have canonical sections that have no more than 6 terminals, which will give us all the ingredients we need to prove Theorem 4.3.

Lemma 4.9. *Let W^* have canonical section $\mathscr{W}(r_i, x_L^i, x_R^i)$. Then $x_L^i + x_R^i + 1 \le 5$.*

Proof. Clearly, for $i \in \{0, q+1\}$ the claim holds. So we assume that $i \in \{1, \dots, q\}$. Note that, by definition of the canonical sections, we have $R_i = L_i = 1$.

We assume for the sake of contradiction that $x_L^i + x_R^i \ge 5$. By Lemma 4.5, we consider witness tree W' with canonical sections $\{\mathscr{W}(r_\iota, x_L^\iota, x_R^\iota)\}_{\iota \in \mathscr{I}(W^*)\setminus\{i\}} \cup \{\mathscr{W}(r_{i-1}, x_L^i - 1, x_R^i - 2), W(r_{i+x_R^i-1}, 1, 1)\}$. By Lemma 4.6, we can see that

$$\begin{aligned}
&h_{W'}(T(r_{i-1}, x_L^i - 1, x_R^i - 2)) + h_{W'}(T(r_{i+x_R^i-1}, 1, 1)) \\
=&\alpha\left(2H_2 + H_1 + \sum_{j=2}^{x_L^i} H_j + \sum_{j=1}^{x_R^i - 1} H_j\right) + x_L^i + x_R^i - 1 + H_{x_L^i + x_R^i - 1} + H_4
\end{aligned}$$

Therefore, the difference between $h_{W^*}(T)$ and $h_{W'}(T)$ is

$$\begin{aligned}
&\alpha\left(\sum_{j=2}^{x_L^i+1} H_j + \sum_{j=1}^{x_R^i+1} H_j - \left(2H_2 + H_1 + \sum_{j=2}^{x_L^i} H_j + \sum_{j=1}^{x_R^i-1} H_j\right)\right) \\
&+ x_L^i + x_R^i + H_{x_L^i + x_R^i + 2} - (x_L^i + x_R^i - 1 + H_{x_L^i + x_R^i - 1} + H_4) \\
=&\alpha(H_{x_L^i+1} + H_{x_R^i} + H_{x_R^i+1} - 1 - 2H_2) + 1 + H_{x_L^i + x_R^i + 2} - H_{x_L^i + x_R^i - 1} - H_4
\end{aligned}$$

We denote this above difference by $P(x_L^i, x_R^i)$. We will show that $P(x_L^i, x_R^i) > 0$ for all $x_L^i + x_R^i > 5$, contradicting the assumption that W^* minimizes $\bar{\nu}_T(W^*)$. We proceed by induction on $x_L^i + x_R^i$.

For our base case, we assume that $x_R^i = 3 \ge x_L^i$. Which is sufficient since $|x_L^i - x_R^i| \le 1$, and thus $x_R^i + x_L^i \ge 5$. We consider the following cases for the value of x_L^i.

1. Case: $x_L^i = 2$. $P(2,3) = \alpha(2H_3 + H_4 - 1 - 2H_2) + 1 + H_7 - 2H_4 = 61/1260 > 0$

2. Case: $x_L^i = 3$. $P(3,3) = \alpha(2H_4 + H_3 - 1 - 2H_2) + 1 + H_8 - H_5 - H_4 = 157/2520 > 0$

And thus our base case holds.

Our inductive hypothesis is to assume the inequality holds for $x_L^i + x_R^i = k \geq 6$. We will show the claim holds when $x_L^i + x_R^i = k+1 \geq 7$ and when $x_R^i \geq 4$. We will show that $P(x_L^i, x_R^i) - P(x_L^i, x_R^i - 1) > 0$, which by the inductive hypothesis, shows that $P(x_L^i, x_R^i) > 0$. The difference between $P(x_L^i, x_R^i)$ and $P(x_L^i, x_R^i - 1)$ is

$$\begin{aligned}
&P(x_L^i, x_R^i) - P(x_L^i, x_R^i - 1) \\
=&\alpha(H_{x_R^i+1} - H_{x_R^i-1}) + H_{x_L^i+x_R^i+2} - H_{x_L^i+x_R^i-1} - H_{x_L^i+x_R^i+1} + H_{x_L^i+x_R^i-2} \\
=&\alpha\left(\frac{1}{x_R^i+1} + \frac{1}{x_R^i}\right) + \frac{1}{x_L^i+x_R^i+2} - \frac{1}{x_L^i+x_R^i-1} \\
\geq&\alpha\left(\frac{1}{x_R^i+1} + \frac{1}{x_R^i}\right) + \frac{1}{2x_R^i+1} - \frac{1}{2x_R^i-2}
\end{aligned}$$

Where the last inequality follows since $x_L^i \geq x_R^i - 1$. Since $x_R^i > 3$, we can see the following inequalities hold

$$\begin{aligned}
&\alpha\left(\frac{1}{x_R^i+1} + \frac{1}{x_R^i}\right) + \frac{1}{2x_R^i+1} - \frac{1}{2x_R^i-2} \geq \frac{2\alpha}{x_R^i+1} - \frac{3}{(2x_R^i+1)(2x_R^i-2)} \\
&\geq \frac{2\alpha}{x_R^i+1} - \frac{3}{4(2x_R^i+1)} \geq \frac{2\alpha}{x_R^i+1} - \frac{3}{7(x_R^i+1)} > 0
\end{aligned}$$

Thus the difference is strictly positive, and the claim holds by induction. □

With Lemma 4.9, we have the ingredients necessary to prove Theorem 4.3.

Proof of Theorem 4.3. Let W^* be such that $\min_W \bar{\nu}_T(W) = \bar{\nu}_T(W^*)$. By applying Lemma 4.4 we can decompose W^* into canonical sections $\{\mathscr{W}(r_\iota, x_R^\iota, x_L^\iota)\}_{\iota \in \mathscr{I}(W^*)}$. And by applying Lemma 4.7 we know that $\mathscr{W}(r_0, 0, 0)$ and $\mathscr{W}(r_{q+1}, 0, 0)$ are canonical sections of W^*. Finally, by Lemma 4.9 we know that for every canonical section $\mathscr{W}(r_i, x_L^i, x_R^i)$ of W^*, we have $x_L^i + x_R^i + 1 \in \{1,2,3,4,5\}$. Now we consider the value of the sum $\sum_{e \in T(r_i, x_L^i, x_R^i)} \frac{c(e) H_{w(e)}}{c(T(r_i, x_L^i, x_R^i))}$ for each case of $x_L^i + x_R^i + 1$, where $c(T(r_i, x_L^i, x_R^i)) = \sum_{e \in T(r_i, x_L^i, x_R^i)} c(e)$.

1. $\frac{h_{W^*}(T(r_i,0,0))}{\alpha+1} = \frac{H_2+\alpha}{\alpha+1} = \frac{167}{122} \approx 1.3688$
2. $\frac{h_{W^*}(T(r_i,0,1))}{2(\alpha+1)} = \frac{\alpha(H_2+1)+H_3+1}{2(\alpha+1)} = \frac{335}{244} \approx 1.373$

3. $\frac{h_{W^*}(T(r_i,1,1))}{3(\alpha+1)} = \frac{\alpha(2H_2+1)+H_4+2}{3(\alpha+1)} = \frac{991}{732} \approx 1.3538$
4. $\frac{h_{W^*}(T(r_i,2,1))}{4(\alpha+1)} = \frac{\alpha(H_3+2H_2+1)+H_4+3}{4(\alpha+1)} = \frac{3793}{2928} \approx 1.3568$
5. $\frac{h_{W^*}(T(r_i,2,2))}{5(\alpha+1)} = \frac{\alpha(2H_3+2H_2+1)+H_6+4}{5(\alpha+1)} = \frac{991}{732} \approx 1.3538$

Let $E_1 = E \setminus (T(r_0,0,0) \cup T(r_{q+1},0,0))$, it is clear that $\frac{\sum_{e\in E_1} c(e)H_{w(e)}}{\sum_{e\in E_1} c(e)} \geq \frac{991}{732}$. Thus, since $\frac{\sum_{e\in E_1} c(e)}{\sum_{e\in E} c(e)} = \frac{q(1+\alpha)}{(q+2)(1+\alpha)-\alpha} > \frac{q}{q+2}$, we can see, for $q > \frac{2}{\varepsilon}$, that $\bar{\nu}_T(W^*) \geq \frac{991}{732}(1-\varepsilon)$. □

Chapter 5

Node Connectivity Augmentation of Highly Connected Graphs

In this chapter, we return to the problem of k-Node-CAP. As we have seen, the problem of k-Node-CAP far less well understand when compared to the problem of k-Edge-CAP, both from a computational complexity and an approximation point of view. From a complexity perspective, there is no known reduction that allows us to focus on a particular value of k, where the k-edge-connectivity case can be reduced down to the case where $k = 2$. The problem is known to be APX-hard for $k \geq 1, k \leq n - n^c$ (with n being the number of nodes of the graph and c being any fixed constant) [63][1] even if all the link weights are 1 (known as the *unweighted setting*), but for high values of k the complexity is not completely settled. Bérczi et al. [17] observe that the problem is polynomial-time solvable for $k = n-2$ and $k = n-3$, and it is NP-hard for $k = n-4$ when links allow general weights (known as the *weighted* setting). Still, the complexity status is open for $k = n - 4$ in the unweighted setting, as well as for any value of k somewhat between $n - 4$ and the previously mentioned bound of $n - n^c$.

Also from an approximation perspective, the results are fewer compared to the edge-connectivity case, and mostly concentrate on small values of k. As

[1] Although not stated explicitly in the reference, the reduction requires that $k \leq n - n^c$ in order to have polynomial running time. A more detailed discussion can be found in Section 5.4

already mentioned in Chapter 3, for $k = 1$, Frederickson and JáJá [37] give a 2-approximation. Recently, this bound has been improved in the unweighted case [4, 51, 71]. A 2-approximation is also known for $k = 2$ [5], which can be improved in the unweighted setting when the input graph is a cycle [42]. For any fixed k, the weighted version of the problem admits a $(4+\varepsilon)$-approximation if n is large enough [72] (see also [28]). For arbitrary values of k, it is still open whether a constant approximation is achievable or not, and the best known factor is $O\left(\log\left(\min\left\{\frac{n}{n-k}, n-k\right\}\right)\right)$ [69]. Note that this implies a $O(1)$-approximation for most values of k, except for "intermediate" ones like, e.g., $n - n^c$ for $0 < c < 1$.

Our results and techniques As a first result, we completely settle the computational complexity status of the problem. In Section 5.5, we prove that $(n-d)$-Node-CAP is NP-hard (in fact, APX-hard) for any $d \geq 4, d = O(n^c)$, with c being any positive constant, even in the unweighted setting. Our result complements the result of Kortsarz et al. [63], and hence completes the picture regarding the complexity status of the problem.

We prove this result by first showing NP-hardness for the case $d = 4$ (i.e. $(n-4)$-Node-CAP), reducing from a variant of SAT denoted as 3-SAT-4, whose optimization version is known to be NP-hard [11]. We then employ an approximation preserving reduction from $(n-d)$-Node-CAP to $(n-(d+1))$-Node-CAP, that holds whenever $d = O(n^c)$, and find the following theorem.

Theorem 5.1. *For any given $d \geq 4$, $d \in O(n^c)$ for any fixed constant $c > 0$, $(n-d)$-Node-CAP is APX-hard.*

Note that when we stated this Theorem in Chapter 1, the statement only referred to NP-hardness for simplicity, as at that time, we had not yet stated the notion of APX-hardness. We remark here that our reduction is different from the hardness construction of [17], as their result heavily relies on the fact that the links have different weights.

We then focus on the approximability of the k-Node-CAP problem for $k = n-4$ which, as we just discussed, corresponds to the highest value of k for which the problem is NP-hard. The best approximation factor known for this problem is a 2-approximation (which follows e.g. from [69]). In Section 5.2, we improve over this result by giving a $\frac{3}{2}$-approximation for the problem.

Theorem 5.2. *There is a polynomial time $\frac{3}{2}$-approximation algorithm for the $(n-4)$-Node-CAP problem.*

In Section 5.3, we improve further the approximability to $\frac{4}{3}$ in the unweighted setting.

Theorem 5.3. *There is a polynomial time $\frac{4}{3}$-approximation algorithm for the unweighted $(n-4)$-Node-CAP problem.*

Our approximation algorithms are combinatorial, and rely on the following ingredients. First, we reformulate the problem as a particular *covering* problem in the complement of the graph, as done also in [17] (see also [18]). In particular, one easily observes that an $(n-d)$-node-connected graph G is *not* $(n-(d-1))$-node-connected if and only if the complement of G has some complete bipartite graphs (which we call *obstructions*) as edge-induced subgraphs. Therefore, the augmentation problem can be reformulated as the problem of selecting links in the complement of G in order to cover all such obstructions (see Definition 5.1). For $d=4$, this covering problem becomes a generalization of the *edge-cover* problem, in which one wants to select links, but in addition to covering all vertices, one also wants to cover all cycles of length 4 in the graph. For the weighted version of this problem, we first compute a suitable partial (infeasible) solution that has at most half of the cost of the optimal one, and then show that we can solve the remaining instance in polynomial time via a reduction to edge-cover. For the unweighted version, the approach is somewhat opposite: our algorithm starts by computing a particular edge cover of the instance, and then adjusts the solution to cover the additional obstructions represented by the 4-cycles, while carefully bounding the cost increase.

As our results show, better constant approximation factors can be obtained using ad-hoc techniques for very large values of k. We believe that studying the problem for such values can be instrumental to the development of useful tools and techniques, which seems to be needed to understand the approximability of the general node-connectivity augmentation problem.

Organization of the Chapter In Section 5.1, we reformulate the $(n-d)$-Node-CAP problem as a *covering* problem that we refer to as d-Obstruction Covering, and further investigate the properties of d-Obstruction Covering instances for the case $d=4$. Then, in Section 5.2, we provide a $\frac{3}{2}$-approximation algorithm for the $(n-4)$-Node-CAP problem, and then in Section 5.3 we provide an improved $\frac{4}{3}$-approximation for the unweighted version of the problem. In Section 5.4, we show that the existing APX-hardness results for k-Node-CAP are incomplete for when $k=n-d$ for a $O(\log n)$ value of d. In Section 5.5, we complement the result of Section 5.4 and prove that the $(n-d)$-Node-CAP problem is APX-hard for any $d \geq 4$, with $d = O(n^c)$ for any fixed constant c.

5.1 From $(n-d)$-Node-CAP to d-Obstruction Covering

As a first step, we reformulate our problem as a covering problem, called the d-Obstruction Covering problem, which we define next. In the following, we denote by $K_{i,j}$ the complete bipartite graph that has i nodes in one bipartition and j nodes in the other bipartition.

Definition 5.1 (d-Obstruction Covering)**.** *We are given a value $d \leq n$, a graph $G = (V, E)$ of n nodes that has no $K_{i,j}$ edge-induced subgraph for any $i, j : i + j > d$, and a subset of links $L \subseteq E$ with associated costs $c(\ell) \geq 0, \forall \ell \in L$. We want to compute a minimum-cost subset of links $F \subseteq L$ such that every $K_{i,j}$ edge-induced subgraph of G with $i + j = d$ contains a link of F.*

For a given d-Obstruction Covering instance, we call $K_{i,j}$ a *forbidden* subgraph if $i + j > d$, and an *obstruction* if $i + j = d$. An obstruction $K_{i,j}$ is *covered* by link $\ell \in L$ if ℓ is an edge of $K_{i,j}$.

As already observed in [17, 18], $(n-d)$-Node-CAP is equivalent to solving d-Obstruction Covering in the complement of the original graph, hence being an equivalent reformulation.

Theorem 5.4. *There is an approximation-preserving reduction from $(n-d)$-Node-CAP to the d-Obstruction Covering problem.*

We remark that [17] shows this for $d = 4$. Here we generalize this for other values of d (see [18] for a similar discussion). To prove the Theorem we first prove the following lemma.

Lemma 5.1. *Let $G = (V, E)$ be a connected graph. G has a vertex cut V' of size r if and only if $\bar{G}$ contains a subgraph $K_{i,|V|-r-i}$ for some $1 \leq i < |V| - |V|'$.*

Proof. Assume V' is a cut of size r in G, then $G[V \setminus V']$ has at least two connected components. Let C be a connected component of $G[V \setminus V']$. Then in $\bar{G}$, the induced subgraph on $V \setminus V'$ contains a $K_{|C|,V-|V'|-|C|}$. Now assume $\bar{G}$ contains a subgraph $K_{i,|V|-r-i}$ for some $1 \leq i < |V| - |V|'$. Let (X, Y) be the bipartition of this subgraph. We argue that $V' := V \setminus (X \cup Y)$ is a cut of size r. Observe that by deleting V', there will be no edges between X and Y. Furthermore $|V'| = |V| - (|X| + |Y|) = |V| - (|V| - r) = r$. Thus V' is clearly a cut of size r. □

Proof of Theorem 5.4. Given an instance I of $(n-d)$-Node-CAP on graph $G = (V, E)$, with link set L and cost $c : E \to \mathbb{R}$ for the edges we obtain the instance

of $I' := (\bar{G}, d, L, c)$ of d-Obstruction Covering problem in which $\bar{G}$ is the complement of G. Observe that, using Lemma 5.1, $\bar{G}$ contains no forbidden subgraphs. Hence, again by Lemma 5.1, $G \cup L$ is $(n-d+1)$-vertex-connected if and only if L has a link in every obstruction of $\bar{G}$. This implies that for every subset L' of L, L' is a feasible solution for I if and only if L' is a feasible solution for I'. □

Using Theorem 5.4, we can focus on showing a $\frac{3}{2}$-approximation algorithm for the 4-Obstruction Covering problem. We denote by $(G = (V, E), d, L, c)$ a d-Obstruction Covering instance. If $c(\ell) = 1$ for all $\ell \in L$, then we call this instance *unweighted* and refer to it as (G, d, L) instead. Note that an unweighted instance (G, d, L) is equivalent to instance (G, d, E, c) where $c(e) = 1$ for $e \in L$, and $c(e) = \infty$ otherwise. We refer to $K_{2,2}$ edge-induced subgraphs of G as *squares*, and to nodes that do not belong to any square as *lonely nodes*. For any subgraph C of G, we denote by $V(C)$ the set of its vertices, and by $E(C)$ the corresponding set of edges.

Definition 5.2. *A graph is called a ladder if it has maximum degree* 3*, and one can label its nodes as* $v_1, v_2, \ldots, v_r, u_1, u_2, \ldots, u_r$ *(*$r \geq 2$*) such that, for every* $1 \leq i < r$*, the subgraph induced by the nodes* u_i*,* v_i*,* v_{i+1}*, and* u_{i+1} *is a square with edges* $u_i v_i, v_i v_{i+1}, v_{i+1} u_{i+1}, u_{i+1} u_i$*. We refer to the number of squares in this sequence* $(r-1)$ *as the length of the ladder. A graph is called a hexagon if it has a spanning edge-induced subgraph that is isomorphic to the graph in Figure 5.1.(b).*

For a given ladder, a labelling of the nodes according to the above definition is called a *consistent* labelling.

Let C be a hexagon or a ladder that is an edge-induced subgraph of a given graph $G = (V, E)$. We will let $\delta(V(C))$ denote the set of edges between $V(C)$ and $V \setminus V(C)$, and to simplify notation, we will let $\delta(C) := \delta(V(C))$. We say that the degree of C is $|\delta(C)|$ (i.e., the number of edges between $V(C)$ and $V \setminus V(C)$). We

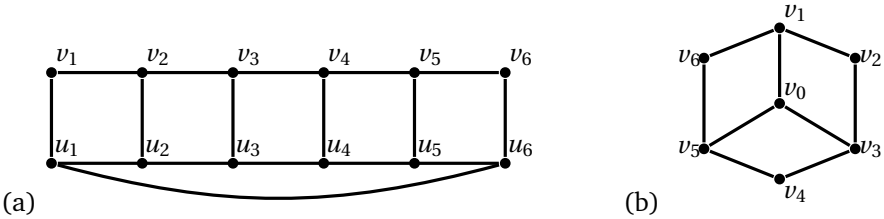


Figure 5.1: (a) An example of a ladder C, with $r = 6$. (b) An example of a hexagon.

say that a node $v \in V(C)$ is a *corner* of C, if v is incident to an edge in $\delta(C)$. Observe that if G is a graph of a 4-Obstruction Covering instance, then a corner v has degree 3 because $K_{1,4}$ is a forbidden subgraph. In this case, note that a hexagon has at most three corners and a ladder has at most four corners. From now on, whenever we use the word subgraph, we refer to edge-induced subgraph (unless specified otherwise).

Let $(G = (V, E), 4, L, c)$ be a 4-Obstruction Covering instance. The following theorem is a useful structural result, which states that we can decompose G into node-disjoint hexagons, lonely nodes, and ladders of maximal length.

Theorem 5.5. *We can find in polynomial time node-disjoint subgraphs $R, G_1, G_2, \ldots, G_k$ of G such that:*

- *Each G_i is a hexagon or a ladder of maximal length. If G_i is a ladder, then we also have a consistent labelling.*
- *$V(R) = V \setminus (V(G_1) \cup V(G_2) \cup \ldots \cup V(G_k))$ is a set of lonely nodes.*
- *For every square subgraph S of G, there exists an index i such that $V(S) \subseteq V(G_i)$.*

In order to prove Theorem 5.5, we need the following technical lemmas.

Lemma 5.2. *Let H and S be hexagon and square subgraphs of G respectively, such that $V(S) \cap V(H) \neq \emptyset$, and $V(S) \not\subseteq V(H)$. Then, in polynomial time, we can find a ladder C in G of length three, and a consistent labelling of C, such that $V(C) = V(H) \cup V(S)$.*

Proof. Denote the nodes of S by $uwzv$, and without loss of generality assume $v \in V(S) \cap V(H)$, and $w \in V(S) \setminus V(H)$. If $S \cap H = \{v\}$, then v is a corner of H that is adjacent to two nodes in H and two nodes in $S \setminus H$, so v has degree 4 which is a contradiction. If $S \cap H = \{u, v\}$, then they cannot be adjacent or else they will again have degree 4. So $S \cap H = \{u, v, z\}$ where u and z are corners of H. Which implies that v is not a corner in H. It is not hard to see that S combined with two of the squares in H gives a ladder C of length three such that $V(C) = V(H) \cup V(S)$. Note that as $|C|$ is constant in size, then by enumeration we can compute a consistent labelling of C. □

Lemma 5.3. *Let C be a ladder in G with a consistent labelling. Let S be a square in G, such that $V(S) \cap V(C) \neq \emptyset$, and $V(S) \not\subseteq V(C)$. Then, in polynomial time, we can find a subgraph $C' \subseteq G$, such that $V(C) \subsetneq V(C')$, where C' is either a ladder with a consistent labelling or a hexagon.*

Proof. If the length of C is 1 then $C \cup S$ is a ladder of length 2, as G does not contain a $K_{2,3}$ forbidden subgraph nor a $K_{1,4}$ forbidden subgraph. In this case, since $|C|$ is constant in size, we can compute a consistent labelling.

So assume the length of C is at least 2, and hence $r \geq 3$. Let $\{u_1, \ldots, u_r, v_1, \ldots, v_r\}$ be the consistent labelling of C. Since $V(S) \not\subseteq V(C)$ and the degree of nodes in G is at most 3, then S contains edges xy and $x'y'$ such that x and x' are distinct corners of S and y and y' are (not necessarily distinct) vertices in $V(S) \setminus V(C)$. Observe that x and x' must have precisely two neighbours from $V(C)$ as they are degree 3 in G, and are thus corners of C.

Without loss of generality, we assume $x = u_r$. We distinguish the following cases: 1) If $x' = v_r$, then $v_r u_r y y'$ must be a square in G. Therefore $C' := S \cup C$ is a ladder of size r. We can label $u_{r+1} := y$ and $v_{r+1} := y'$. 2) If $x' = v_1$, then in the square S there must be a path of length at most two in C from u_r to v_1 in C. However this contradicts the fact that a such path does not exist as $r \geq 3$. 3) If $x' = u_1$, then in the square S there must be a path P of length at most two in C from u_r to v_1 in C. As $r \geq 3$, in order for such a path to exist we must have $r = 3$, and $P = u_1 u_2 u_3$. Therefore $S = u_1 u_2 u_3 y$, and we have $y = y'$. In this case $S \cup C$ is a Hexagon. □

With Lemmas 5.2 and 5.3, we can prove Theorem 5.5.

Proof of Theorem 5.5. We find the decomposition by performing a sequence of steps. At the i-th step, assume we have subgraphs $R_i, G_1, G_2, \ldots, G_{i-1}$ of G such that:

1. $R_i, G_1, G_2, \ldots, G_{i-1}$ are pairwise node-disjoint
2. Each G_j is either a ladder with a consistent labelling or a hexagon.
3. For every square S of G and every $1 \leq j < i$, if $V(S) \cap V(G_j) \neq \emptyset$ then $V(S) \subseteq V(G_j)$.
4. $V(R_i) = V \setminus (V(G_1) \cup V(G_2) \ldots \cup V(G_{i-1}))$.

Initially, we set $i = 1$, and note that $R_1 = G$. At a given step i, if $V(R_i)$ contains only lonely nodes, then $R_i, G_1, G_2, \ldots, G_{i-1}$ are the desired subgraphs. Otherwise we can find a square S such that $V(S)$ and $V(G_1) \cup V(G_2) \ldots \cup V(G_{i-1})$ are disjoint, by property (3). S itself is a ladder of length one, and it is easy to find a consistent labelling for S. Now, we extend S to a maximal ladder or hexagon in R_i by repeatedly applying Lemmas 5.2 and 5.3, as we can enumerate in polynomial time all squares and hexagons in R_i, and check if they intersect with the nodes

of S. In this way, we can find in polynomial time a ladder or a hexagon G_i, such that $V(S) \subseteq V(G_i) \subseteq V(R_i)$ and for every square S' of G, if $V(S') \cap V(G_i) \neq \emptyset$ then $V(S') \subseteq V(G_i)$. Note that if G_i is a ladder, then Lemmas 5.2 and 5.3 also yield a consistent labelling. Furthermore, by construction we keep that $G_1, G_2, ..., G_i$ are node-disjoint. Now we increase i by one and go to the next step. As the size of $V(R_i)$ is decreasing at each step, the process terminates in polynomial time. At termination, we reach the desired subgraphs. □

The decomposition provided by Theorem 5.5 is useful for our purposes, because it is possible to develop a Dynamic Programming based algorithm that solves the 4-Obstruction Covering problem when restricted to a ladder or a hexagon (in fact, something slightly more general than that). The following theorem, provides a precise statement of which instances can be solved.

Theorem 5.6. *Let $C = (V, E)$ be a ladder with a given consistent labelling or a hexagon. Given edge weights $w : E \to \mathbb{R}$ and node covering requirements $h : V \to \{0, 1\}$, we can find in polynomial time a minimum cost set $F \subseteq E$ such that, for every square S in C, $F \cap E[S] \neq \emptyset$, and for every $v \in V$ with $h(v) = 1$, at least one edge of F is incident to v.*

Proof. Assume $u_1, \ldots, u_{r+1}, v_1, \ldots, v_{r+1}$ is the given consistent labelling of C given by Theorem 5.5. If C has at most 10 nodes, then we can enumerate over the possible edge covers to find the minimum cost covering F. Notice that a hexagon has 7 nodes, thus, we can assume that C is a ladder of length at least 6.

We denote by D the possible edges in C with both endpoints in $\{v_1, u_1, v_{r+1}, u_{r+1}\}$. We will compute a different solution for every possible $2^{|D|}$ combination of edges from D in F. If we have $e = xy \in D \cap F$, then we set $h(x) = h(y) = 0$, as these endpoints are covered by $e \in F$. Furthermore, if $v_1u_1 \in D \cap F$, then we set $w(v_1u_1) = 0$ to ensure we have v_1u_1 in the solution and we avoid double counting. The case where $v_{r+1}u_{r+1} \in F \cap D$, is handled similarly. Lastly, if the edges of D form a square B, then we require that $|F \cap D| \geq 1$. Observe that since $r \geq 5$, other than the squares S_i, the only possible square of C is B.

Before we define our dynamic program, in order to simplify notation, for each S_i, $i = 1, \ldots, k$, we define $t_i := v_iv_{i+1}, b_i := u_iu_{i+1}, L_i := v_iu_i$, and, $L_{i+1} := v_{i+1}u_{i+1}$. We are now ready to define our dynamic program.

We compute from S_1 to S_i the minimum number of links needed to cover obstructions $S_1, \ldots, S_i$ and satisfy the requirements $h(v_j)$ and $h(u_j)$ for $j < i$, given whether t_i, b_i, L_{i+1} are in F or not. We let $\mathscr{X}_{L_i}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i} \in \{0, 1\}$ be the indicator variable for whether our solution includes L_i, t_i, and b_i respectively or not. We store our solution in the table $A[i, \mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i}]$.

For $i = 1$, $A[1, \mathscr{X}_{L_2}, \mathscr{X}_{t_1}, \mathscr{X}_{b_1}]$ is equal to

$$\min_{\mathscr{X}_{L_1}, \mathscr{X}_{t_1}, \mathscr{X}_{b_1} \in \{0,1\}} \{w(L_1)\mathscr{X}_{L_1} + w(L_2)\mathscr{X}_{L_2} + w(t_1)\mathscr{X}_{t_1} + w(b_1)\mathscr{X}_{b_1}\}$$

Subject to the requirements that (1) $\mathscr{X}_{t_1} \vee \mathscr{X}_{L_2} = 1$ if $h(v_1) = 1$ to ensure that the coverage requirement of v_1 is met, (2) $\mathscr{X}_{b_1} \vee \mathscr{X}_{L_2} = 1$ if $h(u_1) = 1$ to ensure that the coverage requirement of u_1 is met, and (3) $\mathscr{X}_{t_1} \vee \mathscr{X}_{b_1} \vee \mathscr{X}_{L_2} = 1$ or $u_1 v_1 \in F \cap D$ to ensure that S_1 is covered. Since we are considering all the possible cases by enumeration it should be clear that $A[1, \mathscr{X}_{L_2}, \mathscr{X}_{t_1}, \mathscr{X}_{b_1}]$ is computed correctly.

For $i \in \{1, \ldots, r-1\}$ we set $A[i+1, \mathscr{X}_{L_{i+2}}, \mathscr{X}_{t_{i+1}}, \mathscr{X}_{b_{i+1}}]$ to

$$\min_{\mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i} \in \{0,1\}} \{w(L_{i+1})\mathscr{X}_{L_{i+1}} + w(t_{i+1})\mathscr{X}_{t_{i+1}} + w(b_{i+1})\mathscr{X}_{b_{i+1}} + A[i, \mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i}]\}$$

Subject to the following requirements: (1) $\mathscr{X}_{t_i} \vee \mathscr{X}_{t_{i+1}} \vee \mathscr{X}_{L_{i+1}} = 1$ if $h(v_{i+1}) = 1$ to ensure that the coverage requirement of v_{i+1} is met, (2) similarly, $\mathscr{X}_{b_i} \vee \mathscr{X}_{b_{i+1}} \vee \mathscr{X}_{L_{i+1}} = 1$ if $h(u_{i+1}) = 1$ to ensure that the coverage requirement of u_{i+1} is met, and (3) $\mathscr{X}_{t_{i+1}} \vee \mathscr{X}_{b_{i+1}} \vee \mathscr{X}_{L_{i+1}} \vee \mathscr{X}_{L_{i+2}} = 1$ to ensure that S_{i+1} is covered.

Now by induction we can show that $A[i, \mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i}]$ is computed correctly. We know that this is true for $i = 1$. Assume we have computed all the cells $A[i-1, \mathscr{X}_{L_{i+2}}, \mathscr{X}_{t_{i+1}}, \mathscr{X}_{b_{i+1}}]$ for some i. For each configuration of $\mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i} \in \{0,1\}$, we can apply the inductive hypothesis to $A[i, \mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i}]$, to find the minimum cost number of links needed to cover obstructions $S_1, \ldots, S_i$, and satisfy requirements $h(v_j)$ and $h(u_j)$ for $j < i$. We take the minimum of the configurations of $\mathscr{X}_{L_{i+1}}, \mathscr{X}_{t_i}, \mathscr{X}_{b_i} \in \{0,1\}$ and the claim holds.

To find the minimum solution, we find

$$\min_{\mathscr{X}_{L_{r+1}}, \mathscr{X}_{t_r}, \mathscr{X}_{b_r} \in \{0,1\}} A[r, \mathscr{X}_{L_{r+1}}, \mathscr{X}_{t_r}, \mathscr{X}_{b_r}],$$

subject to the requirements 1) $\mathscr{X}_{t_r} \vee \mathscr{X}_{L_{r+1}} = 1$ if $h(v_{r+1}) = 1$; 2) $\mathscr{X}_{b_r} \vee \mathscr{X}_{L_{r+1}} = 1$ if $h(u_{r+1}) = 1$, and; 3) $x_{L_{r+1}} = 0$ if $u_{r+1} v_{r+1} \notin F \cap D$. □

In particular, we can use Theorem 5.6 to solve each ladder or hexagon of degree zero optimally.

Corollary 5.1. *Consider a 4-Obstruction Covering instance $(G, 4, L, c)$, and a given decomposition of G into lonely nodes R, and disjoint hexagon or ladder subgraphs $G_1, \ldots, G_k$ (as in Theorem 5.5). For any G_i of degree 0, one can find in polynomial time an optimal covering of every obstruction in G_i.*

Thanks to Corollary 5.1, we can assume that G contains no hexagons or ladders of degree 0. Moreover, we assume that G is connected, as we can apply our algorithm to each connected component individually.

5.2 $\frac{3}{2}$-Approximation for Weighted $(n-4)$-Node-CAP

In this section, we will prove the following theorem.

Theorem 5.7. *There is a polynomial time $\frac{3}{2}$-approximation algorithm for $(n-4)$-Node-CAP.*

We first give a high-level description of the algorithm. By Theorem 5.4 we focus on 4-Obstruction Covering instances. For a 4-Obstruction Covering instance $(G,4,L,c)$, we consider a fixed optimal solution opt, with cost $c(opt)$. We apply Theorem 5.5 to G to find node-disjoint hexagons and ladders $G_1,\ldots,G_k$, and lonely nodes R. Our algorithm first finds a partial solution F_1, with cost $c(F_1) \leq \frac{1}{2}c(opt)$, that covers at least the corners of hexagons and ladders $G_1,\ldots,G_k$ whose degree is at least three. We then remove F_1 from E, leaving an instance with only ladders and hexagons of degree 1 or 2. We then compute a special edge cover F_2 (that consists of links), to take care of all the remaining obstructions, with cost $c(F_2) \leq c(opt)$. Putting F_1 and F_2 together then defines a feasible solution of cost at most $\frac{3}{2}c(opt)$. We now give more details about the computation of these partial solutions.

To compute the partial solution F_1, we make use of the following useful Corollary of Petersen's Theorem [75].

Corollary 5.2. *Let G be a 4-regular graph. Then, we can decompose G into two 2-regular spanning subgraphs in polynomial time.*

To prove this, we first recall Petersen's Theorem.

Theorem 5.8 (Petersen's Theorem [75])**.** *Let G be a $2k$-regular graph, for some integer k. Then, G can be decomposed into k 2-regular spanning subgraphs.*

We are now ready to prove Corollary 5.2.

proof of Corollary 5.2. Using Petersen's Theorem G has a 2-regular spanning subgraph. We can compute one such subgraph G_1 in polynomial time by computing a maximum 2-matching, which is well known to be polynomial time. Now G_1 and $G_2 := G \setminus E(G_1)$ are the desired subgraphs. □

We also make use of a slight generalization of the Edge Cover problem, denoted as Minimum N-Edge Cover problem. Given a graph $G = (V, E)$, a subset $N \subseteq V$, and an edge cost function $c : E \to \mathbb{R}$, we want to find a minimum cost subset $E' \subseteq E$, such that the endpoints of E' contain N. The following lemma follows easily from the fact that the edge Cover problem is polynomial-time solvable.

Lemma 5.4. *The Minimum N-edge Cover problem can be solved efficiently.*

Proof. If $N = V(G)$, then we can solve the problem by finding the minimum edge cover of G, which is known to be solvable in polynomial time. Now assume $N \subsetneq V(G)$. In this case we construct a graph G', that contains G and has an additional vertex v. Now for every $u \in V(G) \setminus N$, we add an edge of weight zero from v to u in G'. We compute the minimum weight edge cover E' of G'. Assume E_v is the set of all edges incident to v in G'. We argue that $E'' := E' \setminus E_v$ is the minimum weight N-edge cover of G. For this purpose it is sufficient to observe that if E_1 is an N-edge cover for G then $E_1 \cup E_v$ is an edge cover of G'. Similarly if E_2 is an edge cover for G' then $E_2 \setminus E_v$ is an edge cover of G. Therefore the cost of the minimum N-edge cover of G is indeed equal to the cost of the minimum edge cover of G. Therefore our solution, $E'' := E' \setminus E_v$ is a minimum N-edge cover. □

We now consider node-disjoint hexagons and ladders $G_1, \ldots, G_k$, and lonely nodes R obtained from Theorem 5.5. Denote the corners of hexagons and ladders of degree at least 3 by N, that is, each node of a ladder or hexagon that is adjacent to a node outside the hexagon or ladder containing it is in N. Using Lemma 5.4, we compute a minimum cost N-edge cover on (V, L), that we denote by E_C. As nodes of N (together with their neighbours) induce a subset of the obstructions of G that must be covered (they are nodes of degree 3, hence each with its neighbours induces $K_{1,3}$), it is clear that $c(E_C) \leq c(opt)$. We then construct a 4-regular graph G' using E_C, and apply Corollary 5.2 to find a subset of links of cost at most $\frac{1}{2}c(opt)$.

We begin with $G' = (V' = \emptyset, E' = \emptyset)$, and costs $c' : E' \to \mathbb{R}$. For every G_i, $i = 1, \ldots, k$ of degree at least 3 in G, add corresponding node g_i to V'.

We next add edges to E': consider the links of E_C in an arbitrary but fixed order, say $\ell_1, \ldots, \ell_{|E_C|}$ (recall that $E_C \subseteq L$). For any endpoint u of ℓ_i that is not incident to any ℓ_j for $j < i$, we say that u is *satisfied* by ℓ_i. Note, each corner is satisfied by exactly one link ℓ_i.

Case: link $\ell_i = uv \in E_C$, two endpoints in N in (possibly equal) subgraphs G_a and G_b, $a, b \in \{1, \ldots, k\}$ (see Figure 5.2 for an example of this case). If both u and

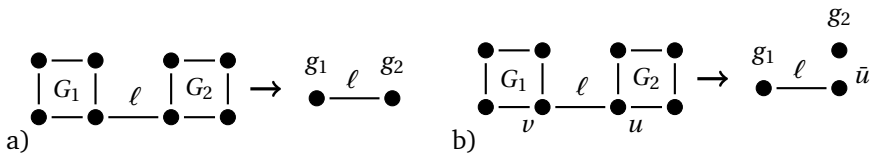


Figure 5.2: a) We have a link ℓ that satisfies corners of G_1 and G_2. When we construct $G' = (V', E')$, we add nodes g_1 and g_2 to V' (if they haven't been added already), and edge $g_1 g_2$ with label ℓ.
b) We have a link $\ell = uv$ that satisfies corner v of G_1 but does not satisfy corner u of G_2. When we construct $G' = (V', E')$, we add nodes g_1 and g_2 to V' (note that g_2 would already have been added since u is satisfied) as well as dummy node $\bar{u}$, and edge $g_1 \bar{u}$ with label ℓ.

v are satisfied by ℓ_i, then we add an edge $g_a g_b$, with label ℓ_i, to E' with cost $c'(g_a g_b) = c(\ell_i)$ (adding a loop in G' with cost $c(\ell_i)$ if $G_a = G_b$). If exactly one endpoint $u \in \ell_i$ is not satisfied by ℓ_i, say $u \in G_a$, then we add a dummy node $\bar{u}$ to V' and add edge $\bar{u} g_b$, with label ℓ_i, to E' with cost $c'(\bar{u} g_b) = c(\ell_i)$. If both endpoints of ℓ_i are satisfied already, then ℓ_i could be removed from the edge cover, and E_C is not minimum, thus, we can ignore this case.

Case: link $\ell_i = uv \in E_C$ has exactly one endpoint in N, say $v \in G_a$, for some $a \in \{1, \ldots, k\}$ (see Figure 5.3 for an example of this case). If v is satisfied by ℓ_i, we add a dummy node $\bar{u}$ to V' and add an edge $\bar{u} g_a$, with label ℓ_i, and cost $c'(\bar{u} g_a) = c(\ell_i)$ to E'. If this v is already satisfied, then we can simply remove ℓ_i from E_C (u may be a degree 3 lonely node, which will be covered in a later step).

Notice that, if G' is not 4-regular, then we can add a polynomial number of dummy nodes to V' and zero cost dummy edges to E' such that G' becomes 4-regular. We show this with the following lemma.

Lemma 5.5. *If G' is not 4-regular, then we can add a polynomial number of dummy nodes to V' and zero cost dummy edges to E' such that G' becomes 4-regular.*

Proof. For every $g_i \in V'$ with degree less than 4 in G', we add a dummy node $\bar{g}_i$ to V' and (possibly parallel) zero cost edges $g_i \bar{g}_i$ to E' so that g_i has degree 4. Observe that the dummy nodes in $V' \setminus V(g_1 \cup \cdots \cup g_k)$ have degree at most 4. For every dummy node $v \in V'$ with degree 1 or 2, we add a self loop of cost zero to E', increasing the degree of v by 2, so that v has degree 3 or 4.

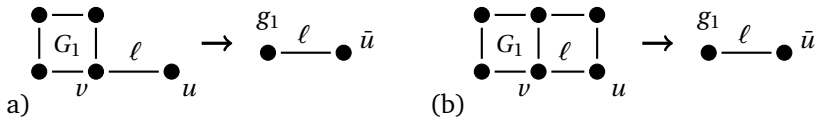


Figure 5.3: a) We have a link $\ell = uv$ that satisfies corner v of G_1 and whose other endpoint is not in a hexagon or ladder. When we construct $G' = (V', E')$, we add node g_1 to V' (if it hasn't been added already) and dummy node $\bar{u}$, and edge $g_1\bar{u}$ with label ℓ.
b) We have a link $\ell = uv$ with both endpoints in G_1, but v is not a corner of G_1. When we construct $G' = (V', E')$, we add node g_1 to V' (if it hasn't been added already) and dummy node $\bar{u}$, and edge $g_1\bar{u}$ with label ℓ.

As the sum of degrees of nodes in G' is even, and all nodes in V' have degree 3 or 4, we can see that there must be an even number of degree 3 dummy nodes. Therefore, we can pair up these dummy nodes in an arbitrary manner and add zero cost edges to E' between these pairs so that every node in G' has degree 4. □

We apply Lemma 5.5 to add a polynomial number of dummy nodes and edges to make G'. By applying Corollary 5.2, we find two edge disjoint 2-regular spanning subgraphs H_1 and H_2 of G', such that $H_1 \cup H_2 = G'$ and $\min\{c(E(H_1)), c(E(H_2))\} \leq \frac{1}{2}c(opt)$. The minimum cost set $H \in \{H_1, H_2\}$ induces our choice of $F_1 \subseteq L$, by selecting F_1 to be equal to the set of links that are labels in H.

In order to compute the second part of the solution, which we denote by F_2, we use F_1 to find a 4-Obstruction Covering instance, $(G'' = (V'', E''), 4, L'', c'')$, with cost $c'' : E'' \to \mathbb{R}$, such that the subgraphs $G'_1, \ldots, G'_k$ that are found by Theorem 5.5 have degree at most 2.

We begin with $G'' = G$. Recall that $G_1, \ldots, G_k$ are computed by Theorem5.5. For every link $f \in F_1$, if $f \in \delta(G_i)$, for some $i \in \{1, \ldots, k\}$, we remove f from E''. If $f \in E[G_i]$ for some $i \in \{1, \ldots, k\}$, then according to our previous discussion, f satisfies at least one corner of $V(G_i)$, let u be the endpoint of f. Since $u \in N \cap V(G_i)$, u has degree 3 and must be incident to an edge $uv \in \delta(G_i)$. Add dummy node $\bar{u}$ to G'', remove uv from G'' and replace it with $\bar{u}v$, giving it label uv, with cost $c''(\bar{u}v) = c'(uv)$. Finally, we set $c''(f) = 0$.

Lemma 5.6. *Let H, F_1, and $(G'', 4, L'', c'')$ be computed as above. $(G'', 4, L'', c'')$ is a 4-Obstruction Covering instance such that:*

1. *The subgraphs $G''_1,\ldots,G''_k$ of G'' found by Theorem 5.5 each have degree at most 2;*
2. *The cost of the optimal solution to $(G'',4,L'',c'')$ is at most $c(opt)$, and;*
3. *Given a solution F to $(G'',4,L'',c'')$ of cost $c''(F)$, we can find in polynomial time a solution to $(G,4,L,c)$ of cost $c''(F)+c(F_1)$.*

Proof. **1)** To see the first claim, we initially assume without loss of generality that the subgraphs G_1 ,…, G_k of G and the subgraphs $G'_1,\ldots,G'_k$ of G'', are enumerated such that $V(G_i)=V(G''_i)$. For each $i=,1\ldots,k$ show that G''_i has degree 1 or 2 by considering the cases of the edges of H incident to g_i.

- *Case:* H has a loop on g_i. Since H is a 2-regular graph, that induces F_1, there is a link $\ell\in E[G'_i]$ that is the unique link in F_1 with an endpoint in $V(G'_i)$. For each endpoint of ℓ that is a corner, by the construction of G'' we create dummy nodes of these corners and change the incident links in $\delta(G'_i)$ with links incident to these dummy nodes instead. Since H contains the loop on g_i, G_i has degree at least 3, therefore, G''_i will have degree 1 or 2 in G''.
- *Case:* H has two (possibly parallel) edges incident to g_i, denoted f_1 and f_2. Note that under our construction of H, each corner is satisfied by exactly one link ℓ_i. Furthermore, for every $G_i\in\{G_1,\ldots,G_k\}$ that has degree 3 or 4, we can see that that in G'', G''_i is incident to $4-|\delta(G'_i)|\in\{0,1\}$ dummy edges using this observation.

  Therefore, if the degree of G''_i in G is 3, then H must contain at least one edge whose label is incident to G''_i. Similarly, if the degree of G''_i in G is 4, then both edges incident to G''_i in H have labels that are incident to G''_i in G. In either case, the degree of G''_i is 2.

Therefore, when we create G'' by either removing edges in $\delta(G'_i)\cap F_1$, or creating dummy nodes so that an edge in $\delta(G'_i)$ is not incident to that dummy node, we reduce the degree of G'_i to 1 or 2.

2) Let opt'' denote the optimal solution to $(G'',4,L'',c'')$. The set of obstructions of G'' is a (not necessarily strict) subset of the obstructions of G, so links in opt (or corresponding links created by replacing nodes with dummy nodes) define a feasible solution to $(G'',4,L'',c'')$. Therefore, $c(opt'')\le c(opt)$.

3) Given F, we can find a partial solution F_2 to $(G,4,L,c)$ by taking the links of F that are in both E'' and E, and the links of $F\cap E''$ that have a label in E. Clearly we have $c(F_2)=c''(F)$. The only obstructions that are uncovered by F_2

are a subset of N, which are covered by F_1. Therefore, we have a solution for (G, L, c) of cost $c(F_2) + c(F_1) = c''(F) + c(F_1)$. □

The last ingredient is to show that the instance $(G'', 4, L'', c'')$ is solvable in polynomial time. To prove this, we prove a slightly more general statement. Given a 4-Obstruction Covering instance $(G, 4, L, c)$ with subgraphs $G_1, \ldots, G_k$ found by Theorem 5.5, we can find in polynomial time a partial solution F that covers every degree 1 or 2 subgraph $G_i \in \{G_1, \ldots, G_k\}$ and every degree 3 node for minimum cost $c(F)$. To see this, we will first show that there is a polynomial time reduction to an instance of the N-edge Cover problem $G' = (V', E')$ with edge costs c' and optimal solution F', such that $c'(F') = c(F)$.

First, we extend the definition of c from just the links L to the edges E by setting $c(e) = \infty$ for all $e \in E \backslash L$. To construct the N-edge Cover instance $G' = (V', E')$ with edge costs c', we will replace each degree 1 or 2 subgraph $G_i \in \{G_1, \ldots, G_k\}$ with a certain subgraph with certain edge costs, which we will call either a 1-gadget or 2-gadget. We will add all degree 3 nodes to N, and a subset of the nodes of each gadgets to N and show that the optimal N-edge cover F' has cost $c'(F') = c(F)$.

Initially, G' is equal to the subgraph induced on G by the nodes that are not in any degree 1 or 2 $G_i \in \{G_1, \ldots, G_k\}$.

1-Gadgets. Consider degree 1 $G_i \in \{G_1, \ldots, G_k\}$ with corner v (see Figure 5.4 (a) and (b) for an example treatment of this case). We apply Theorem 5.6 twice to the subgraph induced on $V(G_i)$, for the following cases.

- Case: For the case where we require that v is covered by a link in $E[G_i]$, we define requirement function $h : V(G_i) \to \{0, 1\}$ by $h(v) = 1$, and for $u \in V(G_i) \backslash \{v\}$, $h(u) = 1$ if u has degree 3, and $h(u) = 0$ otherwise. We then apply Theorem 5.6 to the subgraph induced on $V(G_i)$, with edge weights $w(e) = c(e)$ for $e \in E[G_i]$ that do not have v as an endpoint and $w(e) = \infty$ for $e \in E[G_i]$ that have v as an endpoint, and requirement function h. We denote the cost of this by c_1^i.

- Case: For the case where we require that v is not covered by a link in $E(G_i)$, we define requirement function $h' : V(G_i) \to \{0, 1\}$ by $h'(v) = 0$, and for $u \in V(G_i) \backslash \{v\}$, $h'(u) = 1$ if u has degree 3, and $h'(u) = 0$ otherwise. We then apply Theorem 5.6 to the subgraph induced on $V(G_i)$, with edge weights $w'(e) = c(e)$ for $e \in E[G_i]$, and requirement function h'. We denote the cost of this by c_0^i.

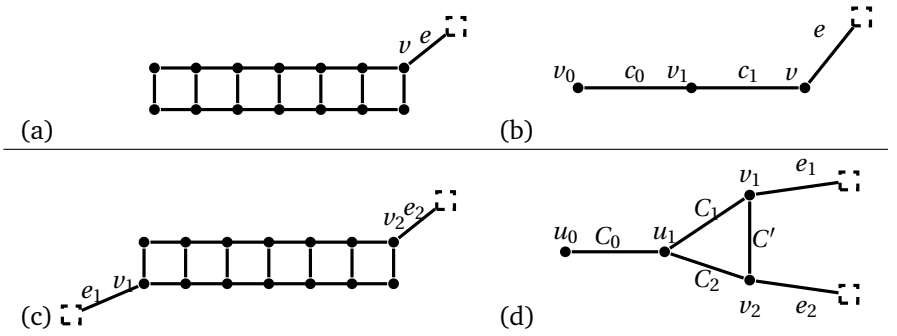


Figure 5.4: An example of the gadget for covering instances of degree 1 and 2 ladders. (a) a ladder C, with corner v. The minimum cost of covering all obstructions but v in C is c_0, and the minimum cost of covering all obstructions in C is c_1. (b) the gadget where we replace C with the path v_0, v_1, v, with edge costs c_0 and c_1. (c) A ladder C with corners v_1 and v_2. The minimum cost of covering: all obstructions is $c_{1,2}$; of covering all obstructions but for v_1 is c_2; of covering all obstructions but for v_2 is c_1; of covering all but obstructions but for v_1 and v_2 is c_0. (d) the gadget where we replace C with the node v_1, v_2, u_1, u_0, with edge costs c_0, c_1, c_2, and $c_{1,2} - \min\{c_0, c_1, c_2\}$.

We then add the nodes v, v_1, and v_0 to V', and add the edges v_0v_1 and v_1v to E', as well as the edge in $\delta(G_i)$ to E' (as this is a single edge incident to v). We give these edges costs $c'(v) = c(v)$, $c'(v_0v_1) = c_0^i$ and $c'(v_1v) = c_1^i$.We add v and v_1 to N. We call this a 1-gadget, with costs c_0^i and c_1^i. We repeat this process of finding 1-gadgets for each degree 1 subgraph $G_i \in \{G_1, \ldots, G_k\}$. See Figure 5.4 (b) for a visual representation of a 1-gadget.

2**-Gadgets.** Consider degree 2 subgraph $G_i \in \{G_1, \ldots, G_k\}$, with corners v_1 and v_2. We apply Theorem 5.6 four times to the subgraph induced on $V(G_i)$, each time subject to one of the following requirements: 1) both v_1 and v_2 are covered by links in $E[G_i]$; 2) v_1 is covered by links in $E[G_i]$ but v_2 is not; 3) v_2 is covered by links in $E[G_i]$ but v_1 is not, and; 4) neither v_1 nor v_2 are covered by links in $E[G_i]$. (See Figure 5.4.(c) and (d) for an example treatment of this case).

For each $v \in V(G_i)$, we let $h(v) = 1$ if v is degree 3 and 0 otherwise. We apply Theorem 5.6 to the subgraph induced on $V(G_i)$ in the following ways:

1. Case: neither v_1 nor v_2 are covered by links in $E[G_i]$. Define requirement

function $h_0 : V(G_i) \to \{0,1\}$ by $h_0(v_1) = h_0(v_2) = 0$, and for $u \in V(G_i)\backslash\{v_1, v_2\}$, $h_0(u) = 1$ if u is degree 3, and $h_0(u) = 0$ otherwise. We define edge costs $w_0 : E[G_i] \to \mathbb{R}_{\geq 0}$, with $w_0(e) = c(e)$ for all $e \in E[G_i]$ that are not incident to v_1 or v_2, and $w_0(e) = \infty$ for $e \in E[G_i]$ that are incident to v_1 and v_2. We apply Theorem 5.6 to the subgraph induced on $V(G_i)$ with edge costs w_0, and requirement function h_0, and denote the resulting cost by c_0^i.

2. Case: that both v_1 and v_2 are covered by links in $E[G_i]$. Define requirement function $h_{12} : V(G_i) \to \{0,1\}$ by $h_{12}(v_1) = h_{12}(v_2) = 1$, and for $u \in V(G_i)\backslash\{v_1, v_2\}$, $h_{12}(u) = 1$ if u is degree 3, and $h_{12}(u) = 0$ otherwise. We define edge costs $w_{12} : E[G_i] \to \mathbb{R}_{\geq 0}$, with $w_{12}(e) = c(e)$ for all $e \in E[G_i]$. We apply Theorem 5.6 to the subgraph induced on $V(G_i)$ with edge costs w_{12}, and requirement function h_{12}, and denote the resulting cost by c_{12}^i.

3. Case: that v_1 is covered by links in $E[G_i]$ but v_2 is not. Define requirement function $h_1 : V(G_i) \to \{0,1\}$ by $h_1(v_1) = 1$, and $h_1(v_2) = 0$, and for $u \in V(G_i)\backslash\{v_1, v_2\}$, $h_1(u) = 1$ if u is degree 3, and $h_1(u) = 0$ otherwise. We define edge costs $w_1 : E[G_i] \to \mathbb{R}_{\geq 0}$, with $w_1(e) = c(e)$ for all $e \in E[G_i]$ that are not incident to v_2, and $w_1(e) = \infty$ for $e \in E[G_i]$ that are incident to v_2. We apply Theorem 5.6 to the subgraph induced on $V(G_i)$ with edge costs w_1, and requirement function h_1, and denote the resulting cost by c_1^i.

4. Case: that v_2 is covered by links in $E[G_i]$ but v_1 is not. Define requirement function $h_2 : V(G_i) \to \{0,1\}$ by $h_2(v_1) = 0$, and $h_2(v_2) = 1$, and for $u \in V(G_i)\backslash\{v_1, v_2\}$, $h_2(u) = 1$ if u is degree 3, and $h_2(u) = 0$ otherwise. We define edge costs $w_2 : E[G_i] \to \mathbb{R}_{\geq 0}$, with $w_2(e) = c(e)$ for all $e \in E[G_i]$ that are not incident to v_1, and $w_2(e) = \infty$ for $e \in E[G_i]$ that are incident to v_1. We apply Theorem 5.6 to the subgraph induced on $V(G_i)$ with edge costs w_2, and requirement function h_2, and denote the resulting cost by c_2^i.

We now add to V' the nodes v_1 and v_2, and add the edges of $\delta(G_i)$ to E', with costs $c'(v_1) = c(v_1)$ and $c'(v_2) = c(v_2)$. We also add nodes u_1 and u_0 to V', and add edges u_0u_1, u_1v_1, u_1v_2, v_1v_2 to E' with costs $c'(u_0u_1) = c_0^i$, $c'(u_1v_1) = c_1^i$, $c'(u_1v_2) = c_2^i$, and, $c'(v_1v_2) = c_{1,2}^i - \min\{c_0^i, c_1^i, c_2^i\}$.

Note that it is possible that $c'(v_1v_2)$ is negative, but as we will see, this will not cause any problems. We call this subgraph of G' a 2-gadget, with costs c_0^i, c_1^i, c_2^i, and c_{12}^i and add u_1, v_1, and v_2 to N. Lastly, we add every degree 3 node of V' to N.

Lemma 5.7. *For the 4-Obstruction Covering instance $(G, 4, L, c)$, there is a set of links $F \subseteq L$, that covers every degree 1 or 2 $\{G_1, \ldots, G_k\}$ and every degree 3 with*

minimum cost c^* if and only if for the corresponding N-edge Cover instance, (G', c'), described above has an optimal solution $F' \subseteq E'$ with cost c^*.

Proof. **Obstruction cover to N-edge cover.** First, given a 4-Obstruction Covering partial $F \subseteq L$ that covers degree 3 nodes and obstructions of degree 1 or 2 ladders and hexagons of G with minimum cost c^*, we will find an N-edge covering F' of G' with cost c^*. We begin with $F' = \emptyset$.

First, for every $\ell \in F$ that does not have both endpoints in a degree 1 or 2 $G_i \in \{G_1, \ldots, G_k\}$, we add ℓ to our N-edge cover F'. Since F covers every degree 3 node of G, F' covers every degree 3 node in G' that is not part of a gadget (and maybe some nodes that are) for the same cost F takes to cover these obstructions.

Next, consider degree 1 or 2 $G_i \in \{G_1, \ldots, G_K\}$.

Consider the case when G_i is degree 1. If $F \cap E[G_i]$ does not cover the corner v of G_i, then it is clear that $c(F \cup E[G_i]) = c_0^i$. If $F \cap E[G_i]$ does cover v, then it is clear that $c(F \cup E[G_i]) = c_1^i\}$. This follows by the construction of the gadget and the definition of c_0^i and c_1^i. Thus, F' covers the obstructions of the 1-gadget with cost equal to that of F covering the obstructions of G_i.

Now we consider the case when G_i is degree 2. It is not hard to see the following statements hold by construction of the gadget and the definition of c_0^i, c_1^i, c_2^i, and $c_{1,2}^i$.

1. If $F \cap E[G_i]$ covers neither v_1 nor v_2, then $c(F \cup E[G_i]) = c_0^i$.
2. If $F \cap E[G_i]$ covers v_1 but not v_2, then $c(F \cup E[G_i]) = c_1^i$.
3. If $F \cap E[G_i]$ covers v_2 but not v_1, then $c(F \cup E[G_i]) = c_2^i$.
4. If $F \cap E[G_i]$ covers both v_1 and v_2, then $c(F \cup E[G_i]) = c_{1,2}^i$.

In cases $1-3$, we add to F' the edge of the 1-gadget that has the corresponding cost. This covers u_1, and F covers every degree 3 node, so it will use links in $\delta(G_i)$ to cover whichever corner is not covered by $F \cap E[G_i]$. Such an edge is added to F', so v_1 and v_2 will also be covered by F'.

In case 4, we add the edge v_1v_2 to F' and the edge of minimum cost from u_0u_1, u_1v_1, and u_1v_2. By definition of $c'(v_1v_2)$, F' has cost $c_{1,2}^i$ on the gadget. Thus, F' covers the nodes of the gadget with cost equal to that of F.

Therefore, we have $c'(F') = c(F) = c^*$. So it remains to show that given a solution F' with minimum cost c^*, we can find a set of links $F \subseteq L$ with cost $c(F) = c^*$ that satisfies the claim.

***N*-edge cover to Obstruction cover.** Given an N-edge covering F' of G' of minimum cost $c'(F') = c^*$, we will find a set of links $F \subseteq L$ that cover the degree 3 nodes of G and the degree 1 or 2 ladders and hexagons with cost c^*. We begin with $F = \emptyset$ and will add to F step by step.

First, for every edge $e \in F'$ that is not in a gadget we add e to F. So F covers every degree 3 node not in a degree 1 or 2 $G_i \in \{G_1, \dots, G_k\}$. We now show how to use F' to add links to F that cover degree 1 or 2 subgraphs in $\{G_1, \dots, G_k\}$. We consider a fixed degree 1 or 2 subgraph $G_i \in \{G_1, \dots, G_k\}$.

First consider the case where G_i is degree 1. The corresponding 1-gadget in G' has costs c_0^i and c_1^i. Let v denote the corner of G_i and denote the nodes of the corresponding 1-gadget by v, v_0, v_1. We consider in which cases the edges v_0v_1, v_1v are in F'.

- Case: $v_0v_1, v_1v \notin F'$. The node $v_1 \in N$ is not covered by F' and thus F' is not a feasible N-edge cover.
- Case: $v_1v \in F'$. We add to F the links found by applying Theorem 5.6 to $E[G_i]$ with covering requirements h and edge weights w. Thus, F covers every square of G_i and every degree 3 node of G_i with cost $c(G_i \cap F) = c_1^i = c'(v_1v)$.
- Case: $v_0v_1 \in F'$. We add to F the links found by applying Theorem 5.6 to $E[G_i]$ with covering requirements h' and edge weights w'. Thus, $F \cap E[G_i]$ covers every square of G_i and every degree 3 node of G_i except v with cost $c(G_i \cap F) = c_0^i = c'(v_0v_1)$. Note that since F' covers v but does not contain v_1v, it must contain the edge in $\delta(G_i)$, which is incident to v and is in F by construction.
- Case: $v_0v_1, v_1v \in F'$. Since $v_0 \notin N$, we can conclude that at least one of c_0^i or c_1^i is equal to 0. Thus, we can remove v_0v_1 from F' for no increase in cost or loss of feasibility and apply Case 2.

Thus, for any degree 1 $G_i \in \{G_1, \dots, G_k\}$, F covers the squares of G_i and the degree 3 nodes of G_i with the same cost as F' cover the corresponding 1-gadget.

Now, consider the case that G_i is degree 2, and the corresponding 1-gadget in G' has costs c_0^i, c_1^i, c_2^i, and c_{12}^i. Denote by $v_1, v_2 \in V(G_i)$ the corners of G_i. We consider in which cases the edges u_0u_1, u_1v_1, u_1v_2, and v_1v_2 are in F'.

- Case: $\delta(u_1) \cap F' = \{v_1v_2\}$. We add to F the links found by applying Theorem 5.6 to $E[G_i]$ with covering requirements h_{12} and edge weights w_{12}. Thus, F covers every square and degree 3 node of G_i with cost c_{12}^i. Note

that since $u_1 \in N$, F' contains an edge of $\delta(u_1)$ of least cost, the cost of F' on the gadget is $c'(v_1v_2) + \min\{u_1v_1, u_1v_2, u_0u_1\} = c^i_{12}$. For the remaining cases, we assume $v_1v_2 \notin F'$.

- Case: $\delta(u_1) \cap F' = \{u_1v_1\}$. We add to F the links found by applying Theorem 5.6 to $E[G_i]$ with covering requirements h_1 and edge weights w_1. Thus, $F \cap E[G_i]$ covers every square of G_i, every degree 3 node in $V(G_i)$ including v_2, but not v_1, with cost $c(G_i \cap F) = c^i_1$. Note that since F' covers v_1 and v_2, but the gadget does not contain v_1v_2 or u_1v_2, that F' must contain the edge of $\delta(G_i)$ that is incident to v_2, and thus F also contains that edge. Thus, F covers every degree 3 node and every square of G_i.

- Case: $\delta(u_1) \cap F' = \{u_1v_2\}$. This case can be handled similarly to Case 2.

- Case: $\delta(u_1) \cap F' = \{u_0u_1\}$. We add to F the links found by applying Theorem 5.6 to $E[G_i]$ with covering requirements h_0 and edge weights w_0. Thus, $F \cap E[G_i]$ covers every square of G_i and every degree 3 node except v_1 and v_2. Note that since F' covers v_1 and v_2, it must contain both edges of $\delta(G_i)$, and thus by construction F contains those edges as well and thus covers every degree 3 node of G_i.

- Case: $|\delta(u_1) \cap F'| \geq 2$. If $u_0u_1 \in F'$, then we can remove it from F' for no loss of feasibility, and no increase in the cost of F'. If, after removing u_0u_1, $|\delta(u_1) \cap F'| = 1$, then $\delta(u_1) \cap F' \in \{u_1v_1, u_1v_2\}$ and we can consider either Case 2 or 3. So assume that $\delta(u_1) \cap F' = \{u_1v_1, u_1v_2\}$. Without loss of generality assume that $c^i_1 \leq c^i_2$.

  It is clear that $c^i_{12} \leq c^i_1 + c^i_2$, and thus we can replace u_1v_2 with v_1v_2 for a change in the cost of F' on the 1-gadget from $c'(u_1v_1) + c'(u_1v_2) = c^i_1 + c^i_2$ to $c'(u_1v_1) + c'(v_1v_2) = c^i_1 + c^i_{12} - \min\{c^i_0, c^i_1, c^i_2\} \leq c^i_{12} \leq c'(u_1v_1) + c'(u_1v_2)$. Clearly, the feasibility of F' is maintained, and we can apply Case 1 to this gadget.

Thus, for any degree 2 $G_i \in \{G_1, \ldots, G_k\}$, F covers the squares of G_i and the degree 3 nodes of G_i with the same cost as F' cover the corresponding 1-gadget.

Therefore, given minimum N-edge cover F' of G' of cost c^*, we can find links $F \subseteq L$ of G that cover every degree 3 node and every square of degree 1 or 2 $G_i \in \{G_1, \ldots, G_k\}$. □

We now conclude putting everything together to prove Theorem 5.7.

Proof of Theorem 5.7. Given $(n-4)$-Node-CAP instance, we apply Theorem 5.4 to obtain an instance $(G, 4, L, c)$ of 4-Obstruction Covering. By Theorem 5.5, we

obtain node-disjoint ladders and hexagons $G_1, \ldots, G_k$, and lonely nodes R. Using the discussion above, we can compute in polynomial time a partial solution F_1 such that $c(F_1) \leq \frac{1}{2}c(opt)$, to cover a subset of the obstructions (namely, corners of ladders and hexagons of degree at least three in this decomposition). We then construct a new instance $(G'', 4, L'', c'')$, in which the node-disjoint ladders and hexagons found by Theorem 5.5 have degree at most two. By applying Lemma 5.7 to $(G'', 4, L'', c'')$, we can find an optimal solution F_2 for this instance. Then, by applying Lemma 5.6, we can find a solution of cost $c(F_1) + c(F_2) \leq \frac{3}{2}c(opt)$ for $(G, 4, L, c)$. □

5.3 $\frac{4}{3}$-Approximation for Unweighted $(n-4)$-Node-CAP

In this section, we will prove the following theorem.

Theorem 5.9. *There is a polynomial time $\frac{4}{3}$-approximation algorithm for the unweighted $(n-4)$-Node-CAP problem.*

We consider an unweighted 4-Obstruction Covering instance $(G, 4, L)$, and apply Theorem 5.5 to decompose G into lonely nodes R, and node-disjoint hexagons, ladders $G_1, \ldots, G_k$. We fix an optimal solution, denoted opt. We assume that G is connected, by considering each connected component of G in turn.

Our algorithm will apply some preprocessing operations, and then find a minimum size subset of edges $EC_3 \subseteq L$, that covers every degree 3 node in G and the obstructions of every $G_i \in \{G_1, \ldots, G_k\}$ of degree 1 and 2. As the obstructions covered by EC_3 are a subset of all obstructions, we can see $c(EC_3) \leq c(opt)$. We then show that we can modify this solution to get a solution APX that covers the remaining obstructions, and by a careful local charging argument we will show that $c(APX) \leq \frac{4}{3}c(opt)$.

Preprocessing and Computing EC_3. We call a link ℓ *necessary* if there is an obstruction O such that ℓ is the only link that can cover O. Observe that any feasible solution must contain all necessary links. Therefore, we initially take these links as part of our solution and then remove them from E and L and consider the remaining instance. Thus, we assume without loss of generality that no link in L is necessary.

We apply Lemma 5.7 to G with edge costs $c(e) = 1$ for $e \in L$ and $c(e) = \infty$ fpr $e \notin L$ to compute EC_3.

Fixing the solution. Given a subset $H \subseteq \{G_1, \ldots, G_k\}$, we denote by $G[H]$ the subgraph induced by $\cup_{G_a \in H} V(G_a)$, and denote by $E[H]$ the edges of $G[H]$. With EC_3, we prove Lemma 5.8, which states that using EC_3 we can find a partial solution that covers every obstruction covered by EC_3 as well as a every ladder of length at least 2 and hexagon without too much increase in cost. We then prove Lemma 5.9 which covers the remaining obstructions again, without much increase in the cost of the solution. This will lead us to a solution APX with cost at most $\frac{4}{3}|EC_3|$, which is our desired $\frac{4}{3}$-approximation.

Lemma 5.8. *Consider an unweighted 4-Obstruction Covering instance $(G, 4, L)$, that contains no ladder or hexagon of degree 1 or 2, with subgraphs $R, G_1, \ldots, G_k$, found by Theorem 5.5. Consider partial solution $EC_3 \subseteq L$ that covers every degree 3 node of G, but every $G_s \in \{G_1, \ldots, G_k\}$ contains an obstruction that is not covered by EC_3. We can find in polynomial time, the following:*

1. *A (potentially infeasible) solution $APX \subseteq L$;*
2. *A subset $\mathscr{H} \subseteq \{G_1, \ldots, G_k\}$ containing all hexagons and ladders of length at least 2, such that APX covers every obstruction in $G[\mathscr{H}]$, and;*
3. *A subset $Y \subseteq APX \cap \delta(G[\mathscr{H}])$, such that, every $y \in Y$ has an endpoint in R*

such that, $|APX \cap (Y \cup E[\mathscr{H}])| \leq \frac{4}{3}|EC_3 \cap (Y \cup E[\mathscr{H}])|$, and $APX \setminus (Y \cup E[\mathscr{H}]) = EC_3 \setminus (Y \cup E[\mathscr{H}])$.

Proof. We find APX by performing a sequence of steps. We begin with step 1 where we set $APX_1 = EC_3$, $Y_1 = \emptyset$, and $\mathscr{H}_1 = \emptyset$. For each step i, $i \geq 2$, we define

- $APX_i \subseteq L$ to a be a partial solution;
- $\mathscr{H}_i \subseteq \{G_1, \ldots, G_k\}$ contains the set of ladders and hexagons whose obstructions are covered by APX_i, as well as the the set of ladders and hexagons that contain an obstruction covered by APX_i but not covered by APX_{i-1};
- Y_i to be the subset of edges $\subseteq APX_i \cap \delta(G[\mathscr{H}_i])$ such that every $y \in Y_i$ has an endpoint in R.

Our goal is to compute APX_i at the i-th step, $i \geq 2$, such that the following holds:

(1) The obstructions covered by APX_{i-1} are a strict subset of those covered by APX_i;

(2) $|APX_i \cap (Y_i \cup E[\mathscr{H}_i])| \leq \frac{4}{3}|EC_3 \cap (Y_i \cup E[\mathscr{H}_i])|$, and $APX_i \setminus (Y_i \cup E[\mathscr{H}_i]) = EC_3 \setminus (Y_i \cup E[\mathscr{H}_i])$.

Note that since APX_i covers the obstructions covered by APX_{i-1} we have $\mathscr{H}_{i-1} \subseteq \mathscr{H}_i$, and since $EC_3 = \mathscr{H}_1$ already covers every degree 3 node, we must have $\mathscr{H}_{i-1} \subsetneq \mathscr{H}_i$. If in iteration i, every obstruction in a ladder or hexagon $G_s \in \mathscr{H}_i$ is covered, and $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ contains no hexagons or ladders of length at least 2, then we define $\mathscr{H} := \{G_s \in \mathscr{H}_i | \text{every obstruction of } G_s \text{ is covered}\}$, $APX := APX_i$, and $Y := Y_i$ and we are done. Otherwise, we will find a hexagon or ladder of length at least 2, and use it to find $APX_{i+1}, \mathscr{H}_{i+1}$, and Y_{i+1}.

Let $G_s \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ be a hexagon or ladder of length at least 2. We will consider the cases for G_s. First, we will deal with the cases that G_s contains a hexagon. Then, we will assume that $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ contains no hexagons, and will consider the case where G_s is a ladder of length at least 2.

In both cases, for every link we add, we will be able to find at least three links in $\delta(G_s)$ (and possibly $\delta(G_t)$), such that, by adding links to cover (G_s) (and possibly (G_t)), we can charge the addition of these new links to the links in $\delta(G_s)$ (and possibly $\delta(G_t)$) for a $\frac{4}{3}$ fractional increase.

Case: $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ contains hexagon G_s. First, suppose $G_s \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ contains a hexagon. By definition, a hexagon has exactly four degree 3 nodes whose neighbourhoods are in the hexagon. Moreover, the subgraph induced on these nodes forms a claw (see the nodes $\{u_1, u_2, u_2, v_0\}$ in Figure 5.5.(a)). Therefore, in order for EC_3 to cover each of the corresponding nodes in the hexagon of G_s, there must be at least three links $\ell_1, \ell_2, \ell_3 \in EC_3 \cap E[G_s]$. By definition of APX_i, we must have $\ell_1, \ell_2, \ell_3 \in APX_i$. Let $S \subseteq G_s$ denote the square that is not covered by $\mathscr{H}_i$. Since no links are necessary, there is a link $\ell \in E[S] \cap L$. See Figure 5.5.(a) for a visual example of this step.

We set $APX_{i+1} \leftarrow APX_i \cup \{\ell\}$. Therefore, $\mathscr{H}_i \subsetneq \mathscr{H}_{i+1}$, since $\mathscr{H}_{i+1}$ contains G_s, and APX_{i+1} satisfies condition (1). By construction, we have

$$|APX_{i+1} \setminus E[G_s]| = |APX_i \setminus E[G_s]|, \qquad |APX_{i+1} \cap E[G_s]| = |APX_i \cap E[G_s]| + 1$$

Therefore, we have that

$$3 \leq |EC_3 \cap (Y_{i+1} \cup \mathscr{H}_{i+1})| - |EC_3 \cap (Y_i \cup \mathscr{H}_i)| := \Delta$$
$$4 \leq |APX_{i+1} \cap (Y_{i+1} \cup \mathscr{H}_{i+1})| - |APX_i \cap (Y_i \cup \mathscr{H}_i)| = \Delta + 1$$

So the left hand side of our desired inequality increases by at least $\Delta + 1$, while the left hand side increases by at least $\frac{4}{3}\Delta$. We can see that $\Delta \geq 3$ since $\ell, \ell_1, \ell_2 \in EC_3$, so the desired inequality holds.

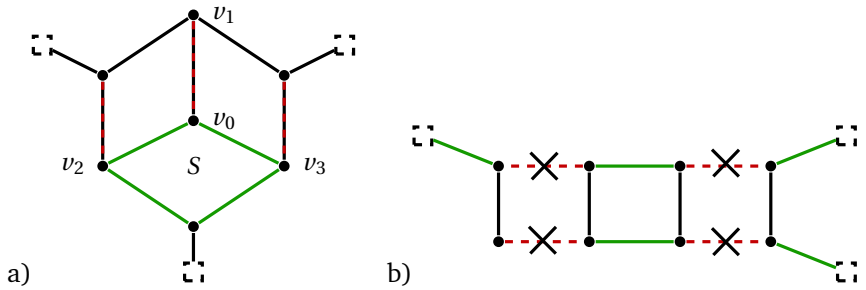


Figure 5.5: (a) A hexagon, with uncovered square S shown in green edges. The nodes v_0, v_1, v_2 and v_3 are degree three and must be covered by links that are in $E(S)$ (example links are shown with dashed red lines). To cover S we can select an edge $\ell \in E(S)$ and add it to the solution, charging its addition to the dashed edges in G_s.
(b) An example of a length 3 ladder. The green edges denote links in the solution APX, and the dashed/crossed out edges denote edges that are not links. Thus, S_1 and S_3 are examples of blockers. Note that we could replace the two links inside the ladder with the other two links in their square to satisfy every obstruction of this ladder.

Case: the set $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ contains no hexagons. So G_s is a ladder of length at least 2 with no hexagon. Applying Theorem 5.5, let G_s have consistent labelling $v_1, \ldots, v_{r+1}, u_1, \ldots, u_{r+1}$. We also enumerate the squares of G_s as $S_j = v_j u_j u_{j+1} v_{j+1}$, for $j = 1, \ldots, r$. Since G_s is degree at least 3, we assume that the labelling of G_s is given such that v_1 is degree 3, if only one of v_1 and u_1 is degree 3. Lastly, we assume that if exactly one of S_1 and S_r are not covered by APX_i, the consistent labelling of G_s is given such that S_r is uncovered. To do this, we first define the set $C := APX_i \cap (E(G_s) \cup \delta(G_s))$, the edges in APX_i with an endpoint in G_s. We will find a set $C' \subseteq L \cap (E(G_s) \cup \delta(G_s))$ that covers every obstruction in G_s such that $|C'| \leq \frac{4}{3}|C|$. We will guarantee this inequality by deleting at least three links from C any time we increase the size of C', with the property that at least one of the links we delete shares an endpoint with the link we add to C'. We will then define $APX_{i+1} := (APX_i \setminus (E(G_s) \cup \delta(G_s)) \cup C'$.

To find C', we will iterate over the squares of G_s, ensuring in each iteration that C' covers the square we consider. We first define some notation. We call the square $S_1 \subseteq E(G_s)$ a *blocker* of APX_i, when a) S_1 is not covered by APX_i, (note that S_r is also uncovered by our enumeration), and; b) when v_1 (resp. u_1) is

degree 3, then $v_1v_2 \notin L$ (resp. $u_1u_2 \notin L$). We call the square $S_r \subseteq E(G_s)$ a *blocker* of APX_i, when a) S_r is not covered by APX_i, and; b) when v_{r+1} (resp. u_{r+1}) is degree 3, then $v_rv_{r+1} \notin L$ (resp. $u_ru_{r+1} \notin L$). See Figure 5.5.(b) for an example where S_1 and S_r are both blockers.

With the definition of blockers, if $r = 2,3$, we can assume without loss of generality that at most one of S_1 and S_r is a blocker.

If $r = 2$, without loss of generality, the nodes v_2 and u_2 are degree 3 and must be covered by APX_i, therefore, at least one of S_1 or S_2 will contain a link and thus cannot be a blocker. In this case, when $r = 2$, and one of the squares is a blocker, we assume that S_2 is the blocker. If $r = 3$, and both S_1 and S_3 are uncovered, since v_2, v_3, u_2, and u_3 are degree 3 nodes, APX_i must contain the links v_2v_3 and u_2u_3. Since S_1 and S_3 are blockers, and no links are necessary, v_2u_2 and v_3u_3 are links. We can replace v_2v_3, and u_2u_3 with v_2u_2 and v_3u_3 in APX_i, C and C' so that G_s now has no blocker (and indeed, no uncovered obstructions).

We now consider the squares of G_s in order $j = 1,\ldots,r$, guaranteeing in each iteration that S_j is covered. Considering by cases if $j = 1$, $j \in \{1,\ldots,r-1\}$, and $j = r$.

Case: $j = 1$. If S_1 is covered by APX_i, we are done with this iteration, so assume that S_1 is not covered by APX_i. We assume v_1 is degree 3, let $v_1x \in \delta(G_s)$ be the incident edge not in $E(G_s)$. If, u_1 is degree, let $u_1y \in \delta(G_s)$ be the incident edge not in $E(G_s)$ to u_1. If v_1x (or potentially u_1y) can be replaced with a link $\ell \in E(S_1)$ without creating an making w (resp. y) part of a newly uncovered obstruction, we replace v_1x (resp. u_1y) with ℓ, and we are one with this iteration. So we assume that v_1x (and potentially u_1y) cannot be replaced. Since APX_i covers every degree 3 node, then v_2 and u_2 are degree 3, thus $v_2v_3, u_2u_3 \in APX_i$. We now consider by cases, whether S_1 is or is not a blocker.

Case: S_1 is not a blocker. Then at least one of v_1v_2 or u_1u_2 is a link adjacent to a corner (v_1 or u_2 respectively). Without loss of generality, assume that $v_1v_2 \in L$. We let then add v_1v_2 to C', and remove three edges from C. That is, $C' \leftarrow C' \cup \{v_1v_2\}$, and $C \leftarrow C \setminus \{v_1x, v_2v_3, u_2u_3\}$.

Case: S_1 is a blocker. By definition, $v_1v_2 \notin L$ (and maybe $u_1u_2 \notin L$). By our arguments on blockers for when $r = 2$ or 3, we can assume $r \geq 4$. By the definition of a blocker and since no links are necessary, v_1u_1, v_2u_2 are links. We consider by cases whether the edges v_3u_3, v_3v_4, and u_3u_4 are links and if so if they are in APX_i.

- Case: $v_3u_3 \in L \cap APX_i$. We set $C' \leftarrow C' \cup \{v_2u_2\} \setminus \{u_2u_3\}$. Clearly, v_2u_2 covers both the obstruction S_1 and the degree 3 node u_2, while v_3u_3 covers degree 3 node u_3. Since the size of C' does not change, we do not need to remove any edges from C.
- Case: $v_3u_3 \in L$ but $v_3u_3 \notin APX_i$. We set $C' \leftarrow C' \cup \{v_2u_2,\ v_3u_3\} \setminus \{v_2v_3, u_2u_3\}$. Clearly, this change covers S_1 in addition to every obstruction covered previously. Since the size of C' does not change, we do not need to remove any edges from C.
- Case: $v_3u_3 \notin L$, and at least one of v_3v_4 or $u_3u_4 \in APX_i$ (observe that, since $v_3u_3 \notin L$ implies that $v_3v_4, u_3u_4 \in L$ since no links are necessary). Without loss of generality, assume that $v_3v_4 \in APX_i$. We set $C' \leftarrow C' \cup \{v_2u_2\} \setminus \{v_2v_3\}$. Since $v_3v_4 \in APX_i$, v_3 is covered, and v_2u_2 covers, v_2, S_1, and S_2. Since the size of C' does not change, we do not need to remove any edges from C.
- Case: $v_3u_3 \notin L$, and $v_3v_4, u_3u_4 \notin APX_i$ (though $v_3v_4, u_3u_4 \in L$ still). We set $C' \leftarrow C' \cup \{v_2u_2, v_3v_4, u_3u_4\} \setminus \{v_2v_3, u_2u_3\}$. Since the size of C' has increased by 1, we need to find 3 links in C to remove. Recall that $r \geq 4$ since S_1 is a blocker, therefore, v_4 and u_4 are degree 3 and must be covered by v_4u_4, or both v_4u_5, u_4u_5. If $v_4u_4 \in APX_i$, then we set $C \leftarrow C \setminus \{v_2v_3, u_2u_3, v_4v_4\}$. If $v_4u_5, u_4u_5 \in APX_i$, then set set $C \leftarrow C \setminus \{v_2v_3, u_2u_3, v_4v_5\}$.

Case: $j \in \{2, \dots, r-1\}$. If S_j is covered, then we are done with this iteration, so we assume that S_j is uncovered. Since $r \neq 1$ or r, the nodes v_j, u_j, v_{j+1}, and u_{j+1} are degree three, and hence covered by APX_i. Therefore, we can see that $v_{j-1}v_j$, $u_{j-1}u_j$, $v_{j+1}v_{j+2}$, $u_{j+1}u_{j+2} \in APX_i$. If removing link $v_{j-1}v_j$ (resp. $v_{j-1}v_j$) from C' and replacing it with a link ℓ in S_j that has v_j (resp. u_j) as an endpoint without leaving v_{j-1} (resp. u_{j-1}) as part of an uncovered obstruction, then we replace it, and we are done with this iteration. Since no link is necessary, we know at least one of v_jv_{j+1}, $v_{j+1}u_{j+1}, u_ju_{j+1}$ is a link. We pick an arbitrary link $\ell \in \{v_jv_{j+1}, v_{j+1}u_{j+1}, u_ju_{j+1}\} \cap L$ from among these. We set $C' \leftarrow C' \cup \{\ell\}$. If ℓ has v_{j+1} as an endpoint, we set $C \leftarrow C \setminus \{v_{j-1}v_j, u_{j-1}u_j, v_jv_{j+1}\}$. If ℓ does not have v_{j+1} as an endpoint, then ℓ has u_{j+1} as an endpoint, and we set $C \leftarrow C \setminus \{v_{j-1}v_j, u_{j-1}u_j, v_jv_{j+1}\}$.

Case: $j = r$. If S_r is covered, then we are done, so assume that S_r is not covered. Since G_s is degree 3, at least one of v_{r+1} or u_{r+1} is degree 3, and if only

one is degree 3, we assume that v_{r+1} is (the other case can be handled similarly). If v_{r+1} (resp. u_{r+1}) is degree 3, then let $v_{r+1}w$ (resp. $u_{r+1}z$) be the link in $\delta(G_s)$ adjacent to v_{r+1} (resp. u_{r+1}). If $v_{r+1}w$ (resp. $u_{r+1}z$) can be replaced with a link in S_r that has endpoint v_{r+1}, without leaving an uncovered obstruction, then we do so, and we are done with this iteration. So we assume that $v_{r+1}w$ (resp. $u_{r+1}z$) cannot be replaced in this way. Since links are not necessary, at least one of $v_r v_{r+1}$ or $v_{r+1}u_{r+1}$ is a link, let $\ell \in L \cap \{v_r v_{r+1}, v_{r+1}u_{r+1}\}$ be an arbitrary such link. We set $C' \leftarrow C' \cup \{\ell\}$, and $C \leftarrow C \setminus \{v_{r-1}v_r, u_{r-1}u_r, v_{r+1}w\}$.

It is not hard to see that this procedure terminates in polynomial time. So it remains to see that whenever we remove edges from C, we reduce the size of $|C|$ by 3. That is, we need to guarantee that when we remove an edge from C it has not been removed in a previous iteration.

When we are covering square S_j, we remove an edge from C when there is no link in APX_i that has one endpoint in S_j that can be replaced with another in S_j that shares that endpoint. When $j = 1$, this happens either when the links in $\delta(G_s)$ cannot be replaced with links in S_1 without creating new uncovered obstructions, or in a special case when S_1 is a blocker. In both cases, since S_1 is the first square, no edges of C will have been removed previously. So assume $j \neq 1, r$, we assume for the sake of contradiction that we try to remove an edge from C in this iteration that has been removed in a previous iteration. That is, one of $v_{j-1}v_j$ or $u_{j-1}u_j$ has already been removed, suppose $v_{j-1}v_j$. By construction, when we remove a link $v_{j-1}v_j$, the node v_{j-1} is the endpoint of a link in C' that is not $v_{j-1}v_j$. Therefore, we could have replaced $v_{j-1}v_j$ with an edge in S_j, and thus would not remove an edge from C, which is a contradiction. For $j = r$, if we remove edges from C, then we remove $v_{r-1}v_r, u_{r-1}u_r$ and one link ℓ in $\delta(G_s)$. The links $v_{r-1}v_r, u_{r-1}u_r$ will not be deleted twice by the argument for $j \neq 1, r$, and the link ℓ is not considered in an earlier iteration.

Using C', we find $APX_{i+1} \coloneqq (APX_i \setminus (E(G_s) \cup \delta(G_s)) \cup C'$, and by construction, we have that $|C'| \leq \frac{4}{3}|C|$. To see that we do not delete a link in $\delta(G_s)$ for any $G_s \in \{G_1, \ldots, G_k\}$ more than once, note that we guarantee that when such a link is deleted the first time, its endpoint is an endpoint of a link in the square that is covered. □

Lemma 5.9. *Consider an unweighted 4-Obstruction Covering instance $(G, 4, L)$, with decomposition into subgraphs $R, G_1, \ldots, G_k$, found by Theorem 5.5, such that every $G_i \in \{G_1, \ldots, G_k\}$ is a ladder of length 1 that has degree 3 or 4. Consider a partial solution $EC_3 \subseteq L$ that covers every degree 3 node, but does not (necessarily) cover the squares of $G_i \in \{G_1, \ldots, G_k\}$.*

We can find in polynomial time, a feasible solution $APX \subseteq L$, such that $|APX| \leq$

$\frac{4}{3}|EC_3|$.

Proof. We begin by defining notation. For a square S, we call a corner $u \in V(S)$ a *good* node if there is a corner $v \in V(S)$ such that $uv \in L$, and we say u is a bad node otherwise (recall that a node of $V(S)$ is a corner if it is incident to an edge in $\delta(S)$). Note that the nodes of $V(S)$ that are not corners are neither good nor bad. Note that since no links are necessary, all nodes of a degree 4 ladder are good nodes.

We find APX by performing a sequence of steps. We begin with step 1 where we set $APX_1 = EC_3$, $Y_1 = \emptyset$, and $\mathscr{H}_1$ equal to the subset of $\{G_1, \ldots, G_k\}$ whose squares are covered by APX_1. For each step i, $i \geq 2$, we define

- $APX_i \subseteq L$ to be a partial solution;
- $\mathscr{H}_i \subseteq \{G_1, \ldots, G_k\}$ to be the set of ladders whose squares are covered by APX_i;
- Y_i to be the subset of edges $\subseteq APX_i \cap \delta(G[\mathscr{H}_i])$ such that every $y \in Y_i$ has an endpoint in R.

Our goal is to compute APX_i at i-th step, $i \geq 2$, such that the following holds:

(1) The obstructions covered by APX_{i-1} are a strict subset of those covered by APX_i;

(2) $|APX_i \cap (Y_i \cup E[\mathscr{H}_i])| \leq \frac{4}{3}|EC_3 \cap (Y_i \cup E[\mathscr{H}_i])|$, and $APX_i \setminus (Y_i \cup E[\mathscr{H}_i]) = EC_3 \setminus (Y_i \cup E[\mathscr{H}_i])$.

Note that since APX_i covers the obstructions covered by APX_{i-1}, we must have $\mathscr{H}_{i-1} \subseteq \mathscr{H}_i$. If in iteration i, we have $\mathscr{H}_i = \{G_1, \ldots, G_k\}$, then we are done as APX_i covers every obstruction. If $\mathscr{H}_i \subsetneq \{G_1, \ldots, G_k\}$, then we will find some $G_s \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$, and using G_s, we will find APX_{i+1}, $\mathscr{H}_{i+1}$, and Y_{i+1}.

We will consider three cases for G_s, for which we now provide a brief description. First, if G_s has at least two nodes that are adjacent to nodes in R or in $\mathscr{H}_i$. Second, if a good node of G_s is adjacent to a good node of some $G_t \notin \mathscr{H}_i$. And finally, if neither of the previous cases hold, then we show G_s is degree 3 and that a good node of G_s must be adjacent to a bad node of some degree 3 ladder $G_t \notin \mathscr{H}_i$. In all cases, we will be able to find at least three links in $\delta(G_s)$ (and possibly $\delta(G_t)$), such that, by adding links to cover G_s (and possibly G_t), we can charge the addition of these new links to the links in $\delta(G_s)$ (and possibly $\delta(G_t)$) for a $\frac{4}{3}$ fractional increase.

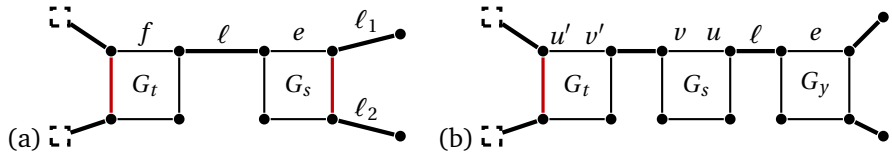


Figure 5.6: In both figures APX_i is shown as bold edges. The red edges are not links. (a) Ladder G_s adjacent to two lonely nodes and ladder G_t. We can see $\{\ell, \ell_1, \ell_2\} = \delta(G_s)$. We can add f and replace ℓ with e, charge the addition of f to $\ell, \ell_1, \ell_2\}$ for an increase from 3 links to 4, but covering both G_s and G_t. (b) Ladders G_s and G_t with adjacent good nodes v and v'. We add uv and $u'v'$ to APX_{i+1} and remove vv', and we replace ℓ with e. Charging the addition of vv' and e to ℓ, vv', and the edge incident to u'.

Case 1: there exists $G_s \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$, such that G_s has two nodes that are adjacent to either a node in R or in $\mathscr{H}_i$. Since G_s is degree at least 3, there are links $\{\ell, \ell_1, \ell_2\} \subseteq \delta(G_s) \cap APX_i$. Without loss of generality, let the links ℓ_1 and ℓ_2 be the ones incident to nodes in $\mathscr{H}_i$ or in R.

Since no links are necessary, there is a link $e \in E(G_s)$ that shares an endpoint with ℓ, and we start with $APX_{i+1} \leftarrow APX_i \cup \{e\}$. Note that this ensures that G_s is in $\mathscr{H}_{i+1}$, and so $\mathscr{H}_i \subsetneq \mathscr{H}_{i+1}$.

If any of ℓ_1, ℓ_2 or ℓ are incident to a node in R, then that link is in Y_{i+1}. Assume ℓ is incident to some ladder G_t. If $G_t \in \mathscr{H}_i$, we are done this iteration. If not, we are going to change APX_{i+1} by swapping two edges. Since ℓ is not necessary, there is link $f \in E(G_t)$ sharing an endpoint with ℓ. We replace ℓ with f in APX_{i+1}, and so also G_t will be in H_{i+1}. Since the endpoints of ℓ are covered in APX_{i+1} by either ℓ or $\{e, f\}$, the obstructions in G_s are covered, and all other obstructions of APX_i are covered by APX_{i+1}, (1) holds (See Figure 5.6). By constructions of APX_i and APX_{i+1}, we have

$$APX_{i+1} \setminus (Y_{i+1} \cup E[\mathscr{H}_{i+1}]) = APX_i \setminus (Y_{i+1} \cup E[\mathscr{H}_{i+1}]) = EC_3 \setminus (Y_{i+1} \cup E[\mathscr{H}_{i+1}])$$

To see that $|APX_{i+1} \cap (Y_{i+1} \cup E[\mathscr{H}_{i+1}])| \leq \frac{4}{3}|EC_3 \cap (Y_{i+1} \cup E[\mathscr{H}_{i+1}])|$, observe that

$$|EC_3 \cap (Y_{i+1} \cup E[\mathscr{H}_{i+1}])| - |EC_3 \cap (Y_i \cup E[\mathscr{H}_i])| := \Delta \geq 3$$

where the inequality follows since $\ell, \ell_1, \ell_2 \notin EC_3 \cap (Y_i \cup E[\mathscr{H}_i])$, but are in $EC_3 \cap (Y_{i+1} \cup E[\mathscr{H}_{i+1}])$. Similarly, the following inequality follows since $APX_{i+1} \cap (Y_{i+1} \cup$

$E[\mathcal{H}_{i+1}]$) contains ℓ_1, ℓ_2, e and exactly one of f or ℓ.

$$|APX_{i+1} \cap (Y_{i+1} \cup E[\mathcal{H}_{i+1}])| - |APX_i \cap (Y_i \cup E[\mathcal{H}_i])| = \Delta + 1$$

So as i increases by one, the increase on the right hand side of the inequality of (2) is at most $\frac{4}{3}$ the increase on the left hand side. Thus, (2) holds.

Case 2: there exist $G_s, G_t \in \{G_1, \ldots, G_k\} \setminus \mathcal{H}_i$ such that a good node of G_s is adjacent to a good node of G_t. Let $u, v \in G_s$ and $u', v' \in G_t$ be good nodes such that $vv' \in L$. Since v and v' are degree 3 nodes that are covered by APX_i, we see $vv' \in APX_i$. We set $APX_{i+1} \leftarrow APX_i \cup \{uv, u'v'\} \setminus \{vv'\}$. In this case, we see that G_s and G_t are in $\mathcal{H}_{i+1}$, and so $\mathcal{H}_i \subsetneq \mathcal{H}_{i+1}$.

Since u and u' are good nodes, they are incident to links $\ell \in \delta(G_s)$ and $\ell' \in \delta(G_t)$ respectively. If ℓ (resp. ℓ') is incident to a node in R, then ℓ (resp. ℓ') is in Y_{i+1}. Assume ℓ is incident to some G_x (the case for ℓ' can be handled similarly if ℓ' is incident to some G_y). If $G_x = G_t$ (similarly if $G_y = G_s$), then the endpoint of ℓ in G_t cannot be adjacent to v' or else $G[G_t \cup G_s]$ forms a ladder of length 3, contradicting our assumption on G.

If $G_x \in \mathcal{H}_i$, then G_x is covered by APX_i and we are done. If $G_x \notin \mathcal{H}_i$, then G_x is not covered by APX_i, and since no links are necessary, there is a link $e \in E(G_x)$ that is adjacent to ℓ. We remove ℓ from APX_{i+1} and add e, and so G_x is in $\mathcal{H}_{i+1}$. Since both endpoints of ℓ are covered by uv and e, APX_{i+1} covers the obstructions covered by APX_i as well as the obstructions in $G[\mathcal{H}_{i+1} \cup Y_{i+1}]$. (See Figure 5.6.)

By a similar argument to the first case, we can see that (1) and (2) hold.

Case 3: Neither of the above cases hold. In this case we wish to show that every $G_s \in \{G_1, \ldots, G_k\} \setminus \mathcal{H}_i$ is degree 3, and that there exists a $G_t \in \{G_1, \ldots, G_k\} \setminus \mathcal{H}_i$ such that G_t has a good node that is adjacent to a bad node of G_s. First, recall that every node of a degree 4 ladder is a good node.

First, we show that the ladders G_s and G_t described always exist when cases 1 and 2 do not hold. We denote by X_3 the set of degree 3 ladders in $\{G_1, \ldots, G_k\} \setminus \mathcal{H}_i$. Denote by N the set of links between a good node and a bad node of different ladders in $\{G_1, \ldots, G_k\} \setminus \mathcal{H}_i$. The following claim will show that the degree 3 ladders that are not covered by APX_i has the same size as N.

Claim 5.1. $|X_3| = |N|$

Proof. In order to show that $|X_3| \geq |N|$, we show that every ladder of degree at least three has at most one bad node. This follows from the fact that as there exists no necessary links, then at least two of the three edges incident to

every corner of a square must be links. Therefore, in a ladder of degree four all corners are good, and in a ladder of degree three at most one corner is bad.

To see $|X_3| \leq |N|$, we first note since case 1 does not hold, a ladder in $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ is adjacent to at most one of: a lonely node, or, a ladder in $\mathscr{H}_i$. Secondly, since case 2 does not hold, the good nodes of ladders in X_3 are not adjacent to the good nodes of any ladder in $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$. Therefore, every degree 3 ladder in $\{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ has at least one good node adjacent to a bad node of another degree 3 ladder, and thus $|X_3| \leq |N|$. And the claim is proven. □

Therefore, since $|X_3| = |N|$, one of the good nodes of each degree 3 ladder in X_3 is adjacent to a bad node of a ladder in X_3, and furthermore, the other good node must be adjacent to a node in R or a ladder in $\mathscr{H}_i$. Since each bad node is adjacent to a degree 3 ladder, the good nodes of degree 4 ladders can only be adjacent to either good nodes, nodes in R, or elements of $\mathscr{H}_i$, and thus must be covered by APX_i since neither case (1) nor (2) hold. Therefore, $X_3 = \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$.

We now consider degree 3 ladder $G_s \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ with bad node s_1, and good nodes s_2, s_3, where s_3 is adjacent to a node in R or a node in $\mathscr{H}_i$. And let s_4 be the node of G_s that is not a corner. Note that $s_1 s_4$ must be a link since there are no necessary links and $s_1 s_2$ cannot be a link by the definition of bad nodes.

By the argumentation above, there is a degree 3 ladder $G_t \in \{G_1, \ldots, G_k\} \setminus \mathscr{H}_i$ with bad node t_1, and good nodes t_2, t_3, where t_3 is adjacent to a node in R or a node in $\mathscr{H}_i$, such that $s_1 t_2 \in APX_i$. Let $e \in \delta(G_s) \cap APX_i$ be incident to s_3, and $f \in \delta(G_t) \cap APX_i$ be incident to t_3. We let $APX_{i+1} \leftarrow APX_i \cup \{s_1 s_4, t_2 t_3\} \setminus \{s_1 t_2\}$, and so G_s and G_t are in $\mathscr{H}_{i+1}$, and $\mathscr{H}_i \subsetneq \mathscr{H}_{i+1}$. The link e (resp. f) is in Y_{i+1} if e (resp. f) is incident to a node in R. By a similar argument to the first case we can see that APX_{i+1}, Y_{i+1}, and $\mathscr{H}_{i+1}$ satisfy (1) and (2).

Note that for every $i \geq 1$, by (2) we have

$$\begin{aligned} |APX_i| &= |APX_i \cap (Y_i \cup E[\mathscr{H}_i])| + |APX_i \setminus (Y_i \cup E[\mathscr{H}_i])| \\ &\leq \frac{4}{3}|EC_3 \cap (Y_i \cup E[\mathscr{H}_i])| + |EC_3 \setminus (Y_i \cup E[\mathscr{H}_i])| \leq \frac{4}{3}|EC_3| \end{aligned}$$

Furthermore, in each iteration we cover a new element of $\{G_1, \ldots, G_k\}$, so after a polynomial number of iterations we will have $\mathscr{H}_i = \{G_1, \ldots, G_k\}$ and we have covered every obstruction. □

With Lemma 5.8 and Lemma 5.9, we have the ingredients necessary to prove Theorem 5.9.

proof of Theorem 5.9. Given an unweighted $(n-4)$-Node-CAP instance, we apply Theorem 5.4 to obtain an instance $(G,4,L)$ of unweighted d-Obstruction Covering. We will show how to find a solution APX with $|APX| \leq \frac{4}{3}|opt|$. Using Theorem 5.5, we decompose G into node-disjoint hexagons and ladders $G_1,\ldots,G_k$, and lonely nodes R. We will compute the solution APX by adding partial solutions.

By applying Lemma 5.7, we compute a minimum cardinality subset of links $EC_3 \subseteq L$ that covers the obstructions contained in every subgraph G_i of degree 1 and 2 and every degree 3 node. Let $\mathscr{H}_0 \subseteq \{G_1,\ldots,G_k\}$ denote the ladders and hexagons covered by EC_3. We set $APX_0 := EC_3 \cap E[\mathscr{H}_0]$, and remove these edges from G to get new 4-Obstruction Covering instance $(G' = (V,E'),4,L')$, where $E' = E \setminus APX_0$, and $L' = L \setminus APX_0$. We apply Theorem 5.5 to find decomposition $R', G'_1,\ldots,G'_{k_1}$. Observe that by construction of G', $G'_1,\ldots,G'_{k_1}$ contain only ladders of degree 3 and 4.

We apply Lemma 5.8 to $(G',4,L')$ with partial solution $EC_3 \setminus APX_0$, to find a new partial solution F_1. We also find a subset of ladders $\mathscr{H}_1 \subseteq \{G'_1,\ldots,G'_{k_1}\}$ whose obstructions are covered by F_1, and a set of links $Y_1 \subseteq F_1 \cap \delta(G'[\mathscr{H}_1])$ where every $y \in Y_1$ has an endpoint in R'. We set $APX_1 := F_1 \cap (Y_1 \cup E'[\mathscr{H}_1])$, and remove APX_1 from G' to get new 4-Obstruction Covering instance $(G'',4,L'')$ where $G'' = (V', E' \setminus APX_1)$ and $L'' = L' \setminus APX_1$. We apply Theorem 5.5 to find decomposition $R'', G''_1,\ldots,G''_{k_2}$. Observe that by Lemma 5.8, $G''_1,\ldots,G''_{k_2}$ contains only ladders of length 1 and degree 3 or 4.

Lastly, we apply Lemma 5.9 with $F_1 \setminus APX_1$ as our partial solution, to find a feasible solution APX_2. We denote the ladders covered by APX_2 by $\mathscr{H}_2 := \{G''_1,\ldots,G''_{k_3}\}$.

We define $APX := APX_0 \cup APX_1 \cup APX_2$. By construction APX_0, APX_1, and APX_2 are pairwise disjoint. To see that APX is a feasible solution to $(G,4,L)$, first observe that $\mathscr{H}_0, \mathscr{H}_1$, and $\mathscr{H}_2$ are also pairwise disjoint and $\mathscr{H}_0 \cup \mathscr{H}_1 \cup \mathscr{H}_2 = \{G_1,\ldots,G_k\}$, so APX covers every obstruction of $\{G_1,\ldots,G_k\}$, in addition, EC_3 covers every obstruction in R, and APX covers every obstruction covered by EC_3 so APX is feasible.

$$\begin{aligned} APX &= APX_0 \cup APX_1 \cup APX_2 = APX_0 \cup (F_1 \cap (Y_1 \cup E'[\mathscr{H}_1])) \cup APX_2 \\ &= (EC_3 \cap E[H_0]) \cup (F_1 \cap (Y_1 \cup E[\mathscr{H}_1])) \cup APX_2 \end{aligned}$$

Therefore, $|APX|$ is equal to

$$
\begin{aligned}
&= |APX_0| + |F_1 \cap (Y_1 \cup E[\mathscr{H}_1])| + |APX_2| \\
&\leq |APX_0| + \frac{4}{3}|(EC_3 \setminus APX_0) \cap (APX_1)| + \frac{4}{3}|F_1 \setminus (APX_1)| \\
&= |APX_0| + \frac{4}{3}|(EC_3 \setminus APX_0) \cap (APX_1)| + \frac{4}{3}|(EC_3 \setminus APX_0) \setminus (APX_1)| \\
&< \frac{4}{3}|APX_0| + \frac{4}{3}|EC_3 \setminus APX_0| = \frac{4}{3}|EC_3| \leq \frac{4}{3}|opt|
\end{aligned}
$$

Where the inequality above follows by application of the inequalities found in Lemmas 5.8, and 5.9. The second equality follows from the equality in Lemmas 5.8. The final inequality above follows since opt is a feasible solution to the N-edge cover problem that EC_3 solves. □

5.4 Missing Hardness Results for k-Node-CAP

Kortsarz et al. [63] prove the following hardness result for 1-Node-CAP.

Theorem 5.10 ([63]). *There exists $\varepsilon_0 > 0$ fixed, such that it is NP-hard to approximate 1-Node-CAP within a factor better than $1 + \varepsilon_0$.*

This result is proved via a reduction from a bounded degree version of 3D Matching. In order to obtain analogous hardness results for k-Node-CAP, it is argued that we can iteratively add $k-1$ nodes to the construction which are connected to all the other nodes in the graph (this way, the graph becomes k connected and the cuts that must be covered are exactly the same). Thus, in general, if we add t dummy nodes to the instance, which has n nodes, we get a graph with $n' = n + t$ nodes which is $(t+1)$-node-connected.

In order for this procedure to be a polynomial-time reduction, it is required that the value of t is polynomially bounded with respect to n, let us say at most n^c for some constant $c > 0$, which implies that the new instance we obtain is a $(n' - n'^{1/c})$-connected graph and we wish to augment its connectivity by one.

In particular, it is not possible for this procedure to reach a $(n'-d)$-connected instance where d is constant or logarithmic, and to have polynomial running time at the same time. This is also consistent with the fact that $(n-3)$-Node-CAP is polynomial-time solvable [17].

5.5 Hardness of Unweighted Node-CAP

In this section, we will prove that the d-Obstruction Covering problem is APX-hard for any $d \geq 4$ with $d = O(n^c)$ for any fixed constant c. To prove this, we introduce a variant of the 3-SAT problem, denoted 3-*SAT*-$(2,2)$.

Definition 5.3 (3-SAT-$(2,2)$ problem)**.** *The* 3-*SAT*-$(2,2)$ *problem is the special case of 3-SAT where each variable occurs exactly twice as a positive literal and twice as a negative literal in the boolean formula $\mathscr{I}$. The corresponding optimization version of the problem is denoted as MAX-*3-*SAT*-$(2,2)$.

Along this section, we will also refer to the following well known special case of the 3-SAT problem, 3-SAT-4.

Definition 5.4 (3-SAT-4 problem)**.** *The 3-SAT-4 problem is the special case of 3-SAT where each variable appears exactly four times in the boolean formula $\mathscr{I}$. The corresponding optimization version of the problem is denoted as MAX-3-SAT-4.*

The 3-SAT-4 problem is known to be NP-hard, and furthermore its optimization version is known to be APX-hard [11]. The following lemma shows that this special case of 3-SAT (and its optimization version) is as hard as the original problem.

Lemma 5.10. *The* 3-*SAT*-$(2,2)$ *problem is NP-hard.*

To prove our hardness results for 4-Obstruction Covering, we will provide a polynomial-time reduction from 3-SAT-$(2,2)$. Let $\mathscr{I}$ be an instance of 3-SAT-$(2,2)$, we will construct an instance $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ of the 4-Obstruction Covering problem as follows:

- For each clause C in $\mathscr{I}$, we add node labelled C to $G_{\mathscr{I}}$. We call these *clause nodes*.
- For each variable $x \in I$, we add a subgraph H_x to $G_{\mathscr{I}}$, where H_x is defined by six *variable nodes* $x_1, \ldots, x_6$, and edges $\{x_1x_2, x_2x_3, x_3x_4, x_4x_5, x_5x_6, x_1x_6, x_2x_5\}$. The edges $\{x_1x_2,\ x_2x_3,\ x_4x_5,\ x_5x_6\}$ are included into $L_{\mathscr{I}}$.
- For (possibly equal) clauses C_1, C_2 containing the positive literals of x, we add edges $\{x_1C_1, x_4C_2\}$ to both $G_{\mathscr{I}}$ and $L_{\mathscr{I}}$. For (possibly equal) clauses C_3, C_4 containing the negative literals of x, we add edges $\{x_3C_3, x_6C_4\}$ to both $G_{\mathscr{I}}$ and $L_{\mathscr{I}}$.

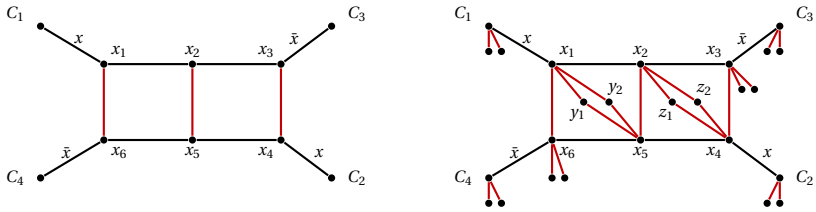


Figure 5.7: Left: Subgraph H_x and clause nodes C_1, C_2, C_3, C_4 containing x, from the reduction in Theorem 5.11, for variable x in $\mathscr{I}$. Here the links of H_x are denoted by the black edges. Right: The extension of $\mathscr{I}$ to larger values of d (in this case, $d = 6$). It can be seen that the set of links is the same, but the obstructions have the requisite number of nodes.

See Figure 5.7 (Left) for a depiction of the gadget. By construction, the graph is 3-regular and no three nodes have two common neighbors, so it is not difficult to show that the defined instance is valid.

Proposition 5.1. *The graph $G_{\mathscr{I}}$ defined above contains neither $K_{1,4}$ nor $K_{2,3}$ as subgraph. Thus, $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ is a valid instance of the 4-Obstruction Covering problem.*

Proof. First of all, since $G_{\mathscr{I}}$ is a 3-regular graph, it does not contain $K_{1,4}$ as subgraph. Therefore, the only forbidden subgraph we need to show does not exist is $K_{2,3}$. Assume, for the sake of contradiction, that F is a $K_{2,3}$ subgraph of $G_{\mathscr{I}}$. Let $v \in F$ be degree 3 in F, we will distinguish two cases:

- If v is a clause node in $G_{\mathscr{I}}$, say corresponding to clause C, then its neighbors are three variable nodes x_i, y_j, z_k for $i, j, k \in \{1, 3, 4, 6\}$, for some (possibly equal) variables x, y, z. If any two such variables are different, then their only common neighbor is v, and hence it cannot be part of a $K_{2,3}$. On the other hand, if the three neighbors of v belong to the same subgraph H_x, by construction there are always two such variable nodes without common neighbors other than v (either x_1 and x_4, or x_3 and x_6), hence again v cannot be part of a $K_{2,3}$
- If v is a variable node in some subgraph H_x, we consider two cases: if $v = x_1$ (the cases x_3, x_4 or x_6 are symmetric), its neighbors in $G_{\mathscr{I}}$ are x_2, x_6 and some clause node, but the only node connected to x_2 and x_6 is x_5, which is not connected to any clause node; hence, x_1 cannot be part of a $K_{2,3}$. If $v = x_2$ (the case of x_5 is symmetric), its neighbors in $G_{\mathscr{I}}$ are

x_1, x_5 and x_3 but they do not have two neighbors in common. Hence, no variable node can be part of a $K_{2,3}$.

□

With Proposition 5.1, we now show using the following two lemmas how to translate a feasible assignment of values for instance $\mathcal{I}$ into an optimal solution of $(G_{\mathcal{I}}, 4, L_{\mathcal{I}})$ and back.

Lemma 5.11. *Let k be the number of variables in the 3-SAT-(2,2) instance $\mathcal{I}$. If $\mathcal{I}$ is satisfiable, then there is a feasible solution to the 4-Obstruction Covering instance $(G_{\mathcal{I}}, 4, L_{\mathcal{I}})$ of size $4k$.*

Proof. Consider a satisfying assignment for instance $\mathcal{I}$. For each variable x in $\mathcal{I}$, if x is set to true (resp. false), then for the (possibly equal) clauses C_1 and C_2 containing the positive (resp. negative) literals of x, we take the links x_1C_1 and x_4C_2 (resp. x_3C_3, x_6C_4), as well as the links $\{x_2x_3, x_5x_6\}$ (resp. $\{x_1x_2, x_4x_5\}$). This way, since the assignment satisfies $\mathcal{I}$, all the obstructions $K_{1,3}$ defined by clause nodes are covered. The $K_{1,3}$ obstructions defined by variable nodes and the $K_{2,2}$ obstructions of each H_x are also covered by the links $\{x_2x_3, x_5x_6\}$ or $\{x_1x_2, x_4x_5\}$ depending on the case. Therefore, we have a feasible solution for the 4-Obstruction Covering instance $(G_{\mathcal{I}}, 4, L_{\mathcal{I}})$ of size $4k$. □

Lemma 5.12. *Let k be the number of variables in the 3-SAT-(2,2) instance $\mathcal{I}$. If there is a feasible solution to the 4-Obstruction Covering instance $(G_{\mathcal{I}}, 4, L_{\mathcal{I}})$ of size $4k$, then $\mathcal{I}$ is satisfiable.*

Proof. Note, for a given variable x in $\mathcal{I}$, any links taken to cover obstructions in subgraph H_x do not cover obstructions from subgraph H_y for variable $y \neq x$. Hence, the size of any feasible solution $F \subseteq L_{\mathcal{I}}$ to $(G_{\mathcal{I}}, L_{\mathcal{I}})$ is equal to $\sum_{x \in \mathcal{I}} |\{\ell \in F | \ell \text{ incident to } v \in V(H_x)\}|$. Observe that every feasible solution must contain at least four links incident to a node in $V(H_x)$ as we must have at least one link incident to every vertex in $\{x_1, x_2, \ldots, x_6\}$. Therefore in a feasible solution of size $4k$, for each variable $x \in \mathcal{I}$ we have exactly four links with an endpoint in $V(H_x)$. Furthermore for each variable x, such a solution has exactly one link incident on each x_i for every $i \in \{1,3,4,6\}$.

We will prove we can assume the feasible solution of size $4k$ to $(G_{\mathcal{I}}, L_{\mathcal{I}})$ has the following structure: for each subgraph H_x, either we have the links between x_1 and x_4 and their corresponding adjacent clause nodes plus x_2x_3 and x_5x_6, or the links between x_3 and x_6 and their corresponding adjacent clause nodes plus x_1x_2 and x_4x_5.

Suppose that for some variable $x \in \mathscr{I}$, we cover the obstructions of H_x with a different set of links.

Case 1, we choose at least three links from H_x: Let us assume without loss of generality that these links are $\{x_1x_2, x_2x_3, x_4x_5\}$. Since the solution is feasible, it must contain either the link between x_6 and its adjacent clause node or the link x_5x_6. In the first case, we can replace x_2x_3 with the link joining x_3 and its adjacent clause node. In the second case, we can replace both x_2x_3 and x_5x_6 with the corresponding links joining them to their adjacent clause nodes, and in both cases every obstruction in H_x is still covered.

Case 2, we choose at most 2 links: We need to cover both of the $K_{2,2}$ obstructions plus x_2 and x_5. In order to do it with two links only, one of them must be incident to x_2 and the other to x_5. Without loss of generality we assume our solution contains x_1x_2 and x_4x_5 and it does not contain x_2x_3 and x_5x_6. Therefore as we need to cover x_3 and x_6 we must have the links from x_3 and x_6 to the clause nodes in our solution. Thus our solution forms the desired configuration.

Notice that, since the solution for $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ is feasible, every clause node is covered. It is possible to find a satisfying assignment to the variables of $\mathscr{I}$ in the following way: For each variable x, if the links incident to x_1 and x_4 and their adjacent clause nodes are taken, then we set x to be true. If the links x_3 and x_6 and their adjacent clause nodes are taken instead, then we set x to be false.

To see that this assignment satisfies $\mathscr{I}$, suppose for the sake of contradiction that some clause $C = (a \vee b \vee c)$ is not satisfied. Then, none of the links between H_a, H_b, H_c and C are taken, and thus C is uncovered. This means that the solution is not covering every obstruction in $(G_{\mathscr{I}}, L_{\mathscr{I}})$, which is a contradiction. □

We can now proceed with the initial hardness result of this section.

Theorem 5.11. *The* 4*-Obstruction Covering problem is NP-hard.*

Proof of Theorem 5.11. Consider an instance $\mathscr{I}$ of 3-SAT-(2,2) defined by k variables. We construct the $(n-4)$-Obstruction Covering instance $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ as described before. Thanks to Lemmas 5.11 and 5.12, $\mathscr{I}$ is satisfiable if and only if $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ admits a feasible solution of size $4k$. Since 3-SAT-(2,2) is NP-hard (Lemma 5.10), the 4-Obstruction Covering problem is NP-hard as well. □

The following lemma shows that the optimization version of this problem is in fact *APX*-hard.

Lemma 5.13. *The MAX-*3*-SAT-*(2,2) *problem is APX-hard.*

Proof. Consider an instance $\mathscr{I}$ of MAX-3-SAT-4 having C clauses, and being $OPT_{\mathscr{I}} = C - t$ the number of clauses satisfied by an optimal solution, where $t \in \mathbb{N}$. We apply the same procedure described in Lemma 5.10 to construct an equivalent instance $\mathscr{I}'$ of MAX-3-SAT-(2,2), and let $OPT_{\mathscr{I}'}$ be the number of clauses satisfied by an optimal solution.

First of all, notice that $\mathscr{I}'$ has clauses that come from $\mathscr{I}$ (possibly with some auxiliary variables instead of the original) plus some auxiliary clauses added in the construction. In particular, if we fix the values of the auxiliary variables so that $x = y_x = z_x = w_x$ for each x where the corresponding modification was added, and also set the variables $d_{y,1}$ and $d_{y,2}$ added in the process to be true, then all the auxiliary clauses are satisfied. This means that $OPT_{\mathscr{I}'} \geq C' - t$, where C' is the number of clauses in $\mathscr{I}'$. Furthermore, we have that $C' \leq 12C$: The number of literals in C is $3C$, and for each literal we include up to four extra clauses (if they were removed and had to be included back we include three, and then we may include four extra clauses per variable).

Suppose now that there is a PTAS for MAX-3-SAT-(2,2). We will run this PTAS on instance $\mathscr{I}'$, obtaining an assignment of values that satisfies at least $(1-\varepsilon)OPT_{\mathscr{I}'}$ clauses. Let us derive an assignment of values for $\mathscr{I}$ by assigning the same values to the variables that are in both $\mathscr{I}$ and $\mathscr{I}'$, and assigning value true to the variables that occur as four positive literals in $\mathscr{I}$ and false to the variables that occur as four negative literals in $\mathscr{I}$. In the worst case, all the unsatisfied clauses in $\mathscr{I}'$ also appear in $\mathscr{I}$, which implies that the derived assignment satisfies at least

$$C - t - \varepsilon(C' - t) \geq C - t - \varepsilon(12C - t)$$

clauses of $\mathscr{I}$.

Notice that $C - t \geq \frac{C}{2}$, because if we consider the assignment that sets every variable to true and the assignment that sets every variable to false, one of them must satisfy at least half of the clauses. Putting everything together, we have that the number of clauses satisfied by our derived assignment for $\mathscr{I}$ is at least

$$C - t - \varepsilon(12C - t) \geq C - t - 24\varepsilon(C - t) + \varepsilon t \geq (1 - 24\varepsilon)OPT_{\mathscr{I}},$$

implying that there exists a PTAS for MAX-3-SAT-4, a contradiction. □

With Lemma 5.13, we are ready to prove the following Theorem.

Theorem 5.12. *The 4-Obstruction Covering problem is APX-hard.*

Proof. Consider a MAX-3-SAT-(2,2) instance $\mathscr{I}$ with optimal solution $OPT_{\mathscr{I}} = m - t$, where m is the number of clauses and $t \in \mathbb{N}$. Consider the 4-Obstruction Covering instance $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ as defined in the proof of Theorem 5.11. Observe that there exists a feasible solution to $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ whose size is $4k + t$, where k is the number of variables in $\mathscr{I}$: For every variable x, if it is set to true in the optimal assignment for $\mathscr{I}$, then we take the links corresponding to the positive literals of the subgraph H_x that connect them to their corresponding clauses plus x_2x_3 and x_5x_6, and otherwise we take the links corresponding to the negative literals instead plus x_1x_2 and x_4x_5. This choice of links will cover every obstruction in the subgraphs H_x, and the nodes corresponding to clauses that are satisfied by the optimal assignment in $\mathscr{I}$. For the clause nodes corresponding to clauses that are not satisfied, we simply take an arbitrary link incident to them. As there are k subgraphs H_x, and their obstructions are covered by four links each, this solution has a cost of $4k + t$.

Let us assume that there exists a PTAS for the 4-Obstruction Covering problem. We will run the PTAS on $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$, obtaining a solution of size at most $(1+\varepsilon)OPT_{(G_{\mathscr{I}},4,L_{\mathscr{I}})} \leq (1+\varepsilon)(4k+t)$, where $OPT_{(G_{\mathscr{I}},4,L_{\mathscr{I}})}$ is the size of the optimal solution for $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$. We will construct an assignment of values for the variables in $\mathscr{I}$ as follows: For each variable x, if only the links connecting the variable nodes corresponding to positive literals of x to their clauses are chosen, then we set the value of x to be true; if only the links connecting the variable nodes corresponding to negative literals of x to their clauses are chosen, then we set the value of x to be false; otherwise, we set the value of x to make true the literal with most incident chosen links that connect them to clause nodes (breaking ties arbitrarily). As argued in the proof of Lemma 5.12, if for some variable x we pick links connecting clause nodes to variable nodes which are not in opposite positions (i.e. corresponding to both positive and negative literals of x), then we require at least five links to cover the obstructions appearing in such a subgraph H_x; furthermore, if four links connecting clause nodes to H_x are chosen, then since x_2 and x_5 are yet to be covered and x_2x_5 is not a link, we need at least six links to cover those obstructions. Thus, at most $t + \varepsilon(4k+t)$ clauses of $\mathscr{I}$ are not satisfied.

Similarly to the proof of Lemma 5.13, we have that $m - t \geq \frac{m}{2}$. Putting everything together, we get that the number of satisfied clauses in $\mathscr{I}$ by the constructed assignment is at least

$$m - t - \varepsilon(4k+t) = m - t - \varepsilon(3m - t) \geq m - t - 6\varepsilon(m-t) + \varepsilon t \geq (1 - 6\varepsilon)OPT_{\mathscr{I}},$$

implying that there is a PTAS for MAX-3-SAT-(2,2). □

It is possible to extend the previous hardness results to further values of d, as the following theorem states (see Figure 5.7 (Right)). First, we extend Theorem 5.11.

Theorem 5.13. *For any given $d \geq 4$, $d \in O(n^c)$ for any fixed constant $c > 0$, the d-Obstruction Covering problem is NP-hard.*

Proof. Given a 3-SAT-(2,2) instance $\mathscr{I}$, we first create the $(n-4)$-Obstruction Covering instance $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ as described in the proof of Theorem 5.11. We will modify it to obtain an d-Obstruction Covering instance $(G'_{\mathscr{I}}, d, L'_{\mathscr{I}})$, and show that finding an optimal covering of the obstructions in $G'_{\mathscr{I}}$ implies whether a satisfying assignment exists for $\mathscr{I}$ or not.

Recall that the instance $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$ has a node for every clause C, and a subgraph H_x for every variable $x \in \mathscr{I}$. For every clause node C, we add $d-4$ dummy nodes to $G'_{\mathscr{I}}$ that are adjacent to C, so that C is degree $d-1$. For subgraph H_x, with nodes $x_1, \ldots, x_6$, we add $d-4$ dummy nodes to $G'_{\mathscr{I}}$ that are adjacent to x_3 and $d-4$ dummy nodes adjacent to x_6. We add $d-4$ nodes to G' that are adjacent to both x_1 and x_5, which we denote by $y_1, \ldots, y_{d-4}$, and we add $d-4$ nodes that are adjacent to both x_2 and x_4, which we denote by $z_1, \ldots, z_{d-4}$. None of these new edges are added to $L'_{\mathscr{I}}$, so in fact we have $L'_{\mathscr{I}} = L_{\mathscr{I}}$.

Notice that every clause and variable node from $G_{\mathscr{I}}$ now is degree $d-1$, and thus is part of a $K_{1,d-1}$ obstruction. Moreover, the nodes $x_1, x_2, x_5, x_6, y_1, \ldots, y_{d-4}$ form a $K_{2,d-2}$ obstruction, as do the nodes $x_2, x_3, x_4, x_5, z_1, \ldots, z_{d-4}$. See Figure 5.7 (Right). Since no set of at least three nodes has more than two neighbours in common in $G'_{\mathscr{I}}$, these are the only obstructions in the instance.

We will prove now that $(G'_{\mathscr{I}}, d, L'_{\mathscr{I}})$ is a valid $n-d$ Obstruction Covering instance. Recall that the forbidden subgraphs of a d-Obstruction Covering instance is a $K_{i,j}$ subgraph where $i + j > d$. First, the dummy nodes adjacent to clause nodes are degree 1, and the clause node are degree $d-1$, so these dummy nodes cannot be part of a forbidden subgraph. Similarly, for variables $x \in \mathscr{I}$, the dummy nodes adjacent to x_3 and x_6 are not part of a forbidden subgraph. The nodes $y_1, \ldots, y_{d-4}$ are degree 2 and thus cannot be part of a forbidden subgraph $K_{i,j}$ for $i > 2$, and the same holds for $z_1, \ldots, z_{d-4}$. Since the maximum degree of $G'_{\mathscr{I}}$ is $d-1$, these nodes cannot be part of a $K_{1,j}$ forbidden subgraph. Thus, they can only possibly be part of a $K_{2,j}$ forbidden subgraph. The neighbours of $y_1, \ldots, y_{d-4}$ are x_1 and x_5 which also share a neighbourhood of x_2 and x_6, which together form a $K_{2,d-2}$ subgraph. In conclusion, the are no forbidden subgraphs in $G'_{\mathscr{I}}$, and hence $(G'_{\mathscr{I}}, d, L_{\mathscr{I}})$ is a valid d-Obstruction Covering instance.

Since $L'_{\mathscr{I}} = L_{\mathscr{I}}$ and given the structure of the obstructions, $F \subseteq L'_{\mathscr{I}} = L_{\mathscr{I}}$ is a feasible solution to $(G'_{\mathscr{I}}, d, L'_{\mathscr{I}})$ if only if F is a feasible solution to $(G_{\mathscr{I}}, 4, L_{\mathscr{I}})$,

and thus by Theorem 5.11 the result holds.

As a final remark, notice that, if the 3-SAT-(2,2) instance $\mathscr{I}$ has t clauses (and consequently $\frac{3}{4}t$ variables), the size of $G'_{\mathscr{I}}$ is $O(t \cdot d)$. In order for this procedure to be a polynomial-time reduction, it is required that $d \in O(n^{1-c})$ for some constant $c > 0$, as otherwise t must be subpolynomial with respect to n, leading to superpolynomial time for the reduction algorithm.

□

Now we can extend Theorem 5.12, and prove the d-Obstruction Covering problem is APX-hard for any $d \geq 4$. Consider the same construction as in the proof of Theorem 5.13. Since, by construction, the set of links are the same for both instances and the obstructions are in a 1-to-1 correspondence, the reduction is directly approximation-preserving, hence proving the result.

Theorem 5.14. *For any given $d \geq 4$, $d \in O(n^c)$ for any fixed constant $c > 0$, the d-Obstruction Covering problem is APX-hard.*

Chapter 6

Flexible Graph Connectivity

Two fundamental problems in the area of Survivable Network Design are the 2-*edge-connected spanning subgraph* (2ECSS) and the 2-*vertex-connected spanning subgraph* (2VCSS). These problems aim at building a network resilient to a possible failure of an edge or of a vertex, respectively.

In recent years, an interesting generalization has been introduced by Adjiashvili et al. [2], which soon received a lot of attention in the network design community. This generalization is called *flexible* graph connectivity and is the focus of this Chapter. Specifically, Adjiashvili et al. [2] considered a scenario in which not all edges are subject to potential failures. The set of edges is partitioned into *safe* and *unsafe*, and the goal is to construct a network resilient to the failure of unsafe edges. We provide a formal definition of this problem.

Definition 6.1 (Flexible Graph Connectivity Problem (FGC))**.** *Given a graph $G = (V, E)$ and a partition of E into safe edges E_S and unsafe edges E_U, the goal is to find the smallest subset E' of E such that (1) $H = (V, E')$ is connected, (2) and for every edge $e \in E_U \cap E'$, the graph $(V, E' \setminus e)$ is connected.*

Adjiashvili et al. [2] provided the first approximation factor of 2.523 for this problem, which was improved to a 2-approximation by Boyd et al. [14]. When the costs on the edges are uniform Adjiashvili et al. [2] find a $\frac{5}{3}$-approximation, and Boyd et al. [14] provide a $\frac{16}{11}$-approximation.

Next, we define the vertex-connectivity version of the problem, investigated first in [8]. Recall that a *cut-vertex* of a graph is a vertex u such that if we remove u and all its incident edges the number of connected components of the graph increases.

Definition 6.2 (Flexible Vertex-Connectivity Problem (FVC))**.** *Given a graph $G = (V, E)$ and a partition of V into safe vertices V_S and unsafe vertices V_U, the goal is to find the smallest subset E' of E such that (1) $H = (V, E')$ is connected, (2) and for every vertex $u \in V_U$, u is not a cut-vertex of H.*

Note that, when $E_S = \emptyset$, FGC reduces to 2ECSS. Similarly, when $V_S = \emptyset$, FVC reduces to 2VCSS. Hence, these problems are at least as hard as 2ECSS and 2VCSS, respectively.

For FGC and some further generalizations, several approximation results have been developed in the past few years (see e.g. [2, 3, 7, 8, 10, 14, 22, 74], and application of flexible graph based techniques). This list shows a growing interest in this problem and its variants in the literature.

The current best known approximation factor for FGC is given in [14], which is based on the best 2ECSS approximation ratio. At the moment, using the best approximation result available in the literature for 2ECSS, [44], this translates into a bound of $\frac{472}{385} \approx 1.45$.[1]

Of course, a first natural question is the following:

- *Can the approximation factor for FGC be improved?*

Problems involving vertex-connectivity rather than edge-connectivity often turn out to be more challenging to address, and in fact approximation results on FVC are more scarce. The authors in [8] studied the problem in the *weighted* setting, and give a 2-approximation under the assumption to have at least one safe vertex (they target a more general setting of removing k unsafe vertices). Note that a 2-approximation is known for the weighted setting of FGC as well, however as mentioned before, for the standard FGC a significantly better approximation is known. This raises the question of whether a better-than-2 approximation is possible also for FVC.

- *Is there an algorithm for FVC with approximation factor better than 2?*

Finally, we observe that both 2ECSS and 2VCSS have been studied in a generalized setting, called kECSS and kVCSS respectively, where one requires k-edge-disjoint paths (resp. k-vertex-disjoint paths) between each pair of vertices. It turns out that for higher values of connectivity requirements, much better approximations can be developed. In particular, [27, 39, 40] show an approximation factor of $1 + O(\frac{1}{k})$ for kECSS and kVCSS. It is natural to ask what

[1] If one applies the aforementioned 1.3-approximation found in [62], then this approximation becomes $1.\bar{4}$.

happens if we generalize FGC and FVC in a similar way. Specifically, we could aim at building graphs that are k-edge-connected (resp. k-vertex-connected) after the removal of any unsafe edge. The last question we are interested in is then the following:

- *Can we obtain similar approximation bounds as observed for kECSS or kVCSS, if we consider FGC and VFC with a higher value of k as connectivity requirement?*

Our Results In this chapter, we focus on the above questions and answer them in a positive way.

In Section 6.2, we prove Theorem 6.1, which improves the best known approximation factor for FGC. The theorem is proved by giving a refined analysis of the algorithm developed in [14]. Their algorithm relies on an approximation algorithm for 2ECSS as a subroutine, whose analysis is used mainly when the size of the optimal solution is large enough compared to the number of vertices n. Our improvement stems from realizing that 2ECSS can be approximated better than the current best known factor whenever the optimal solution is large compared to n.

Theorem 6.1. *There is a polynomial time $\frac{10}{7}$-approximation algorithm for FGC.*

In Section 6.1 we prove Theorem 6.2, which yields the first approximation algorithm for FVC with an approximation factor strictly better than 2. Its proof constitutes the most technical part of our chapter. It combines two different main algorithms, which rely on non-trivial combinatorial ingredients, like ear-decompositions and matroid intersection.

Theorem 6.2. *There is a polynomial time $\frac{11}{7}$-approximation algorithm for FVC.*

Finally, in Section 6.3 we answer the last question raised, for k-FGC specifically. Where k-FGC is defined as follows.

Definition 6.3 (k-Flexible Graph Connectivity Problem (k-FGC)**.** *Given a graph $G = (V, E)$ and a partition of E into safe edges E_S and unsafe edges E_U, the goal is to find the smallest subset E' of E such that (1) $H = (V, E')$ is 1-edge-connected, (2) and for every k unsafe edges $\{e_1, \ldots, e_k\} \subseteq E_U \cap E'$, the graph $(V, E' \setminus \{e_1, \ldots, e_k\})$ is 1-edge-connected.*

We prove Theorem 6.3, which shows that also k-FGC becomes somewhat easier to approximate when k grows, as it happens for kECSS.[2]

Theorem 6.3. *There is a polynomial time* $1+O\left(\frac{1}{\sqrt{k}}\right)$*-approximation algorithm for* k*-FGC.*

We were not able to extend our arguments to FVC with higher values of connectivity requirement k in a similar manner to k-FGC. We leave this as an open question.

Preliminaries Before we begin, we provide some preliminaries to define and construct some tools that will be helpful later on in the chapter.

Given a graph $G=(V,E)$ and a subset of vertices $U \subseteq V$. We denote by G/U the (multi-)graph obtained by first replacing the vertices of U by a single vertex $\hat{U}$, and then for every edge $uv \in E$ such that $u \in U$ and $v \in V \setminus U$ we add an edge $\hat{U}v$ to E (Note that there can be multiple copies of the same edge $\hat{U}v$ in G/U, and there will be no loops on $\hat{U}$). We sometimes refer to this as *contracting* G by the vertices U. We can define a similar operation on a subset of edges $F \subseteq E$, by taking the vertex sets of connected components of (V,F), and contracting G by these sets one by one. We denote this operation by G/F.

Given a graph $G=(V,E)$, and a subset of vertices $U \subseteq V$, we define $G[U] = (U, \{u_1u_2 \in E | u_1, u_2 \in U\})$, the *induced* subgraph of U. Similarly, we can define an induced subgraph on a subset of edges $F \subseteq E$, and by abuse of notation we use the same notation $G[F] = (\{v \in V | \exists u \in V, uv \in F\}, F)$. We also say a graph $H=(V',E')$ is a spanning subgraph of G if H is a subgraph of G and $V'=V$.

Definition 6.4 (Ear-Decomposition)**.** *Let* $G=(V,E)$ *be a graph. Define an* ear-decomposition *as a sequence* $P_1,\ldots,P_k$*, where* P_1 *is a cycle of* G*, and for each* $i \in \{2,\ldots,k\}$*,* P_i *is either:*

- *a path sharing exactly its two endpoints with* $V(P_1) \cup \ldots \cup V(P_{i-1})$*, or;*
- *a cycle that shares exactly one vertex with* $V(P_1) \cup \ldots \cup V(P_{i-1})$.

$P_1,\ldots,P_k$ *are called* ears*.* P_i *is an* open *ear if it is a path. An ear-decomposition is* open *if for every* $i \in \{2,\ldots,k\}$*,* P_i *is an open ear. We refer to* $|E(P)|$ *as the* length *of* P*.*

[2] We note that a (stronger) approximability of $1+O\left(\frac{1}{k}\right)$ for k-FGC was previously claimed by [2], but their proof is flawed as we explain in Section 6.4.

Given an open ear decomposition $P_1, \ldots, P_k$, we say that P' is a potential open ear of $P_1, \ldots, P_k$ if $P_1, \ldots, P_k, P'$ is itself an open ear decomposition. $P_1, \ldots, P_k$ is an open ear decomposition of a graph $G = (V, E)$ if $(V(P_0) \cup \ldots \cup V(P_k) = V$ and for each i, $E(P_i) \subseteq E$.

We will often abbreviate the adjective '2-vertex-connected' with '2VC'. The following well known result on open ear decompositions can be found in Chapter 4 of [86].

Lemma 6.1 ([86]). *A graph G is 2VC if and only if it has an open ear decomposition.*

We end this section by recalling some useful properties of blocks that were introduced in Chapter 2, which will let us characterize the vertex connectivity of a graph.

Definition 6.5 (Blocks). *Let $G = (V, E)$ be a graph such that $|V(G)| \geq 2$. A block of G is a maximal connected subgraph of G that has at least one edge and has no cut-vertex. Therefore if G has no self-loops, a block is either an induced connected subgraph on two vertices or it is a maximal 2VC subgraph on at least three vertices.*

For proofs and more details on these, we refer the reader to Chapter 4 of [86].

Lemma 6.2 ([86]). *Let $G = (V, E)$ be a graph, and let $\{B_1, \ldots, B_k\}$ be the set of all blocks of G. The following properties hold*

1. *Two blocks share at most one vertex.*
2. *The blocks $B_1, \ldots, B_k$ of G partition E, that is $E = \cup_{i=1}^{k} E(B_i)$, and $E(B_i) \cap E(B_j) = \emptyset$, for $i \neq j$.*
3. *For two distinct edges e_1 and e_2, e_1 and e_2 belong to the same block B_i if and only if there is a cycle in B_i that contains e_1 and e_2.*
4. *If G is connected, G has at most $|V| - 1$ blocks.*

We will require the following useful Lemma.

Lemma 6.3. *Let $G = (V, E)$ be a connected graph with $B(G)$ many blocks. Let H be a connected, spanning subgraph of G with $B(H) > B(G)$ many blocks. In polynomial time, we can find an edge $e \in E(G) \setminus E(H)$ such that $H' := (V(H), E(H) \cup \{e\})$ has fewer than $B(H)$ blocks.*

Proof. H is a subgraph of G, therefore every block of H is either a block in G, or a subgraph of a block in G. Since $B(H) > B(G)$, there must exist a block B of G that contains at least two blocks of H. Let $B_1, \ldots, B_k$ be blocks of H that are subgraphs of B that all share the same vertex, $c \in V(B)$.

We wish to show that there are blocks B_i and B_j, $i, j \in \{1, \ldots, k\}$, and edge $e \in E(G) \setminus E(H)$ between B_i and B_j, that does not contain c as an endpoint. That is, $B_i \cup B_j \cup \{e\}$ is a block.

Pairwise, any edges with c as an endpoint will be part of a cycle in B by Property 2 of Lemma 6.2. So we pick edges e_1 and e_2 arbitrarily of these. By construction there is block B_1 that contains e_1 and block B_2 that contains e_2. Using Property 2 of Lemma 6.2, there is a cycle C in B that contains both e_1 and e_2. Furthermore, the path $C - \{e_1, e_2\}$ starts in B_1 and ends in B_2. Therefore, there is an edge connecting a pair of blocks in $B_1, ; B_k$. □

6.1 $\frac{11}{7}$-Approximation for FVC

In this section we provide a $\frac{11}{7}$-approximation algorithm for Flexible Vertex Connectivity Problem. We assume that our given graph $G = (V, E)$, is a simple graph, that is, G does not contain any loops or parallel edges (note that even if G were not simple, parallel edges and loops would not help in finding a feasible solution), and has $n := |V|$ vertices. Furthermore, we assume that our G does not contain any unsafe cut vertices, as these can be identified in polynomial time, and show that the instance does not have a feasible solution.

We begin this section by showing that we can assume without loss of generality that we can assume that our instance is $2VC$, by splitting the instance on safe cut vertices.

Lemma 6.4. *Let $G = (V, E)$ be a graph and let x be a safe cut-vertex of G. Let V_1 be the vertex set of one connected component of $G[V \setminus \{x\}]$ and let $V_2 = V \setminus (\{V_1\} \cup x)$. Now consider an arbitrary subset E' of E. For $i \in \{1, 2\}$, we define E'_i as the set of edges in E' that have both endpoints in $V_i \cup \{x\}$. Then E' is a feasible FVC solution for G if and only if E'_1 and E'_2 are feasible FVC solutions for $G_1 := G[V_1 \cup \{x\}]$ and $G_2 := G[V_1 \cup \{x\}]$, respectively.*

Proof. Clearly, we can observe that E'_1 and E'_2 partition E'. Assume E'_i is not a feasible FVC solution for G_i for some $i \in \{1, 2\}$. First, suppose $(V_i \cup \{x\}, E'_i)$ is disconnected. Then, there exists a vertex v in V_i that has no paths to x in $(V_i \cup \{x\}, E'_i)$. Therefore, v has no paths to x in (V, E') and hence E' is not a

feasible FVC solution for G. So suppose that $(V_i \cup \{x\}, E'_i)$ is connected, thus, as E'_i is not a feasible FVC solution for G_i, then $(V_i \cup \{x\}, E'_i)$ must contain an unsafe cut-vertex, say $u \in V_i$. Now u is also a cut-vertex in (V, E'), hence the claim.

Now, we assume for each $i \in \{1,2\}$, E'_i is a feasible FVC solution for G_i. Clearly (V, E') is connected. Furthermore, if we remove any unsafe vertex $v_i \in V_i$ from $(V_i \cup \{x\}, E'_i)$, it still remains connected. Therefore, by removing any unsafe vertex from (V, E'), it will remain connected. Hence E' is a feasible FVC solution for G. □

Therefore, to find a β-approximation for the more general instance, we can find an β-approximation for the 2VC instances. In order to obtain a β-approximation for $\beta \geq 1$ one can assume that the input graph (i.e. G) has some additional properties, such as not containing specific subgraphs that we call *forbidden cycles*.

Definition 6.6 (Forbidden Cycle)**.** *We say that a 4-cycle C in G is a forbidden cycle if C has two vertices w and z such that $wz \notin E(C)$ and $\deg_G(w) = \deg_G(z) = 2$.*

The next two lemmas show that we can assume without loss of generality that any instance does not contain a forbidden cycle, either by taking an edge of the cycle that an optimal solution will always take, or identifying an edge that is not contained in some optimal solution.

Lemma 6.5. *Assume G is 2VC and has at least five vertices. Let $C := uwvz$ be a forbidden cycle of G as described above such that $\deg_G(w) = \deg_G(z) = 2$ and u and v are unsafe. Then any optimal solution for G can be decomposed into an optimal solution for $G \setminus \{w\}$ and the edges $\{uw, wv\}$.*

Proof. First, we observe that since G has at least 5 vertices, and is 2VC, there must be a path from u to v that does not contain any edge of C. Furthermore, since u and v are unsafe, it is clear that any feasible solution must contain every edge of C, otherwise, one of u or v would be an unsafe cut-vertex. We wish to show that for every minimal feasible solution F, $F' := F \setminus \{uw, vw\}$ is a feasible solution for the instance $G \setminus \{w\}$.

Since w is adjacent only to u and v in G, and F already contains edges uz and vz, the only vertices that can become cut-vertices when we remove w from F are vertices in C, namely, $u, v,$ and z. We demonstrate by cases that none of these can become cut-vertices of $G' := (V \setminus \{w\}, F')$. See Figure 6.1.

Case: Assume for contradiction that at least one of u or v are cut-vertices of $(V \setminus \{w\}, F')$ (without loss of generality, say u is the cut-vertex). That is, $G' \setminus \{u\}$

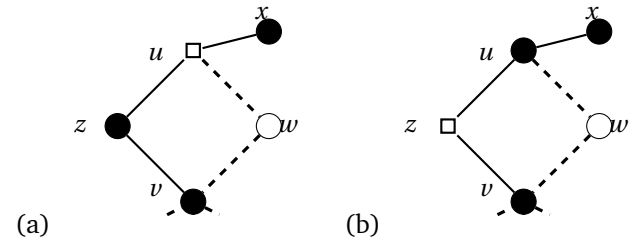


Figure 6.1: As in the statement and proof of Lemma 6.5, we are given a forbidden cycle C, with unsafe vertices u and v, and degree 2 vertices w and z. The edges of F' are shown with solid edges, and F is both the solid lines and the dashed edges incident to w. Since $G \geq 5$, there is an additional vertex $x \notin V(C)$. We wish to show that neither u nor z is a cut-vertex of $(V \setminus \{w\}, F')$, by showing that if so, then that vertex will be cut-vertex separating v from x in (V, F). We consider cases if u or z are cut-vertices of $(V \setminus \{w\}, F')$:
(a) u is a cut-vertex (shown as a square), separating x from z and v, and clearly even with vw, x is still separated from these vertices.
(b) z is a cut-vertex(shown as a square), separating v from u, and in particular, separating v from x. It is clear that in this case u is again a cut-vertex.

has at least two connected components, where one component contains v and another contains some vertex $x \in V \setminus V(C)$. Clearly, this means that in (V, F), every path from v to x contains u. Hence F is not feasible, a contradiction.

Case: So assume for contradiction that z is a cut-vertex in G'. Denote the components of $G' \setminus \{z\}$ as F'_u and F'_v, which are the components containing u and v, respectively. Since G has at least five vertices, without loss of generality we can assume that there is vertex $x \in F'_u$ where $x \notin V(C)$. As z has degree two, we can see that any path in F' from x to v contains u. Therefore, u is a cut-vertex in F', which leads us to the previous case to derive a contradiction. □

Lemma 6.6. *Assume G is 2VC and has at least five vertices. Let $C := uwvz$ be a forbidden cycle of G as described above such that $\deg_G(w) = \deg_G(z) = 2$ and v is safe. W.l.o.g assume that if w is a safe vertex then z is also a safe vertex. Then there exists an optimal solution that does not contain the edge uw.*

Proof. Again, we observe that since G has at least 5 vertices, and is 2VC, there must be a path from u to v that does not contain any edge of C. We wish to show that for every feasible FVC solution F such that $uw \in F$, we can find a

feasible FVC solution F' not containing uw with $|F| \geq |F'|$. In particular, given an optimal solution, we will be able to find an optimal solution that does not contain uw.

Assume $vw \notin F$. Since F is a feasible FVC solution, $uw \in F$. In this case $F' := F \cup \{vw\} \setminus \{uw\}$ is a feasible FVC solution as v is a safe vertex. Thus we can assume that $vw \in F$.

Also F must have an edge incident on z. So far, we realized that F has uw, vw and at least one of the edges in $\{zu, zv\}$. See Figure 6.2.

Now, let $F' := F \cup C \setminus \{uw\}$. Since F contains at least three edges of C, we have $|F| - 1 \leq |F'| \leq |F|$. Furthermore for any vertex $x \in V \setminus \{u, z\}$, x is a cut-vertex in F' if x is a cut-vertex in F. Therefore F' is a feasible FVC solution satisfying our requirements, unless one of z or u is an unsafe cut-vertex of (V, F').

Assume that z is an unsafe cut-vertex of (V, F'). Then there must be exactly one path in (V, F') from u to v, namely, the path u, z, and v. Furthermore, as z is unsafe then w is also unsafe by the assumption of this Lemma on z and w.

As any path from u to v in (V, F') (and hence in F) contains at least one edge of C, F must contain all edges of C; otherwise, since z and w are unsafe then F is not a feasible FVC solution, a contradiction. This implies that $|F| = |F'| + 1$. Let F'_u and F'_v denote the two connected components of $(V \setminus \{z\}, F' \{uz, vz\})$.

Since G has at least 5 vertices, there is a vertex $x \notin V(C)$ in a component of $(V \setminus \{z\}, F' \{uz, vz\})$, without loss of generality say $x \in F'_u$. Since G is 2VC, there is a path P from x to v that does not contain u, as otherwise u would be a cut-vertex of G. Since there are only two components of $(V \setminus \{z\}, F' \{uz, vz\})$, there is an edge $e \in E(P)$ with one endpoint in F'_u and the other in F'_v. See Figure 6.3. Therefore, $F' \cup \{e\}$ is a feasible FVC solution that contain uw.

Lastly, suppose u is an unsafe cut-vertex of (V, F'), we only need to observe that since u is a cut-vertex, there is some vertex x that is in a separate component from v. And the argument is similar to the case where z is an unsafe cut-vertex. □

The following Lemma allows us to assume without loss of generality that our input graph G is 2VC and does not contain a forbidden cycle, by applying the reductions described in Lemmas 6.5 and 6.6 and showing this is a polynomial time approximation preserving reduction.

Lemma 6.7. *For $\beta \geq 1$, if there is a β-approximation for FVC instances that are 2VC and do not contain forbidden cycles, then there is a β-approximation for FVC.*

Proof. We solve the problem recursively. First, observe that we can check the feasibility of the instance in polynomial time. Moreover, if our instance has less

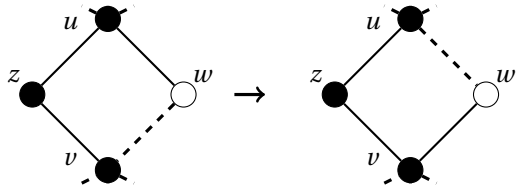


Figure 6.2: As in the statement and proof of Lemma 6.6, we are given a forbidden cycle C, with unsafe vertices u and v, and degree 2 vertices w and z.
in the first figure, the solid edges represent edges in F, and the dashed represent the edges not in F but in C. In the second figure we have the solid edges representing $F' := F \setminus \{uw\} \cup \{vw\}$. Since v is safe, we can replace uw with vw, and not create an unsafe cut-vertex. So F' is a feasible FVC solution.

than five vertices we can solve it in polynomial time by enumeration. If there exists a cut-vertex x in G, then in polynomial time we can find sets V_1, V_2 such that $V_1 \cup V_2 \cup \{x\}$ is a partition of V described by Lemma 6.4. Thus computing a β-approximation on $G_1 := G[V_1 \cup \{x\}]$ and $G_2 := G[V_2 \cup \{x\}]$ and taking the union leads to a β-approximation for G.

Furthermore, if G has a forbidden cycle that satisfies conditions of Lemma 6.5 if we find a β-approximation for $G \setminus \{w\}$ and then add $\{uw, wv\}$ to it we obtain a solution of cost $\beta(opt(G)-2)+2 < \beta opt(G)$. If G has a forbidden cycle that satisfies conditions of Lemma 6.6 then we can find an edge (uw) and remove it from the instance.

Let $T(m)$ be the running time for an instance on m edges. By the above argument we have form $m \geq 3$:

$$T(m) \leq F(m) + \max\{T(m_1) + T(m-m_1), T(m-2), T(m-1)\},$$

where $1 \leq m_1 \leq m-1$ and $F(m)$ is a polynomial function of m, that upper bounds the running time of any of the operations listed above. Note that $T(1) = T(2) = O(1)$ as this small instances can be solved by constant number of operation. Therefore our algorithm has polynomial running time. □

Furthermore, using the next Lemma we assume throughout Section 6.1 that $OPT \geq n$ as otherwise we can solve the problem optimally in polynomial time.

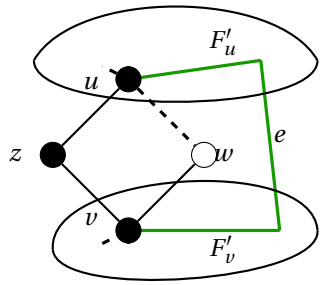


Figure 6.3: As in the statement and proof of Lemma 6.6, we are given a forbidden cycle C, with unsafe vertices u and v, and degree 2 vertices w and z.
Since v is safe, we can replace uw with vw, and not create an unsafe cut-vertex. So F' is a feasible FVC solution.
Here the solid edges represent edges of F'. The green edges show a path from u to v that does not contain an edge of C. Here we depict the case that z is an unsafe cut-vertex of F'. Since G is 2VC, we can pick edge e of this path that connects the two components of $F' \setminus \{z\}$.

Lemma 6.8. *Let OPT be an optimal solution to FVC instance $G = (V, E)$. We have $|OPT| \geq n - 1$. Furthermore, if $|OPT| = n - 1$, then we can find such a solution in polynomial time.*

Proof. As (V, OPT) is connected, $|OPT| \geq n - 1$. If $|OPT| = n - 1$, then by the feasibility of OPT, (V, OPT) is a tree in which the unsafe vertices are leaves. Therefore, the only vertices of OPT with degree at least 2 are safe, and thus form a connected subtree of G.

To find a solution OPT' with $|OPT'| = |OPT|$, we compute a spanning tree on the subgraph induced by the safe vertices. For every unsafe vertex $u \in V$, we buy edge $uv \in E$, where v is a safe vertex v that is adjacent to u. OPT' is a tree, and thus $|OPT'| = n - 1 = |OPT|$. □

6.1.1 Approximation 1

By applying Lemma 6.7, we can assume without loss of generality that G is 2VC and does not contain a forbidden cycle. We also assume that G has at least 5 vertices, as we can handle smaller instances by enumeration. Our algorithm relies on the construction of a certain open ear decomposition which we outline

here. Start with D being a cycle of length at least four. We remark that such a cycle must exist, as $n \geq 4$, and G is 2VC (see [86]). Moreover one can find such a cycle in polynomial time. Now, until there exists a potential open ear of D that has a length of at least 4 for D, we find and add one such potential open ear to D. We observe by Lemma 6.1 that D is 2VC. To show that this procedure terminates in polynomial time we rely on the following simple claim.

Claim 6.1. *Let D be an open ear decomposition. If there exists a potential open ear of D with a length of at least 4, it can be detected in polynomial time.*

Proof. We can check in polynomial time for $x \in V(D)$ and $y_1, y_2, y_3 \notin V(D)$ if there exists such a path that starts with x, y_1, y_2 and y_3 (in this order).

To do this we first check if edges $xy_1, y_1y_2, y_2y_3 \in E$ (if not pick different vertices until all have been scanned). Then we remove y_1, y_2 and x from G and see if there is a path from y_3 to a vertex in $V(D) \setminus x$. As this can be done in polynomial time and there are at most n^4 such tuples, then the claim follows. □

The following Lemma provides an upper bound on the number of edges in D.

Lemma 6.9. *When the algorithm terminates, we have $|E(D)| \leq \frac{4}{3}(|V(D)| - 1)$.*

Proof. We prove that the claim is true by showing that the inequality is invariant at every step of the construction of D by using induction on the number of steps.

In the first iteration, D is a cycle of length at least 4. Thus $4 \leq |E(D)| = |V(D)| \leq \frac{4}{3}(|V(D)| - 1)$.

Now assume the statement holds in the i^{th} iteration. In the $i+1^{st}$ iteration, the number of edges that we add to D is at most $\frac{4}{3}$ the number of vertices added at this step since we add potential open ears of length at least 4. Therefore the above inequality holds also at step $i+1$. □

With this open ear decomposition D, the following key Lemma shows that the edges of $G[V \setminus V(D)]$ have a useful structure, that will be critical to our approximation algorithms.

Lemma 6.10. *The edges of $G[V \setminus V(D)]$ form a (not necessarily perfect) matching.*

Proof. For simplicity, denote $G' := G[V \setminus V(D)]$. If $E(G') = \emptyset$, we are done. Otherwise, consider edge $uv \in E(G')$. We first wish to show that there are vertices $u' \neq v' \in V(D)$, such that $uu', vv' \in E$. That is, uv is part of a potential open ear for D of length three.

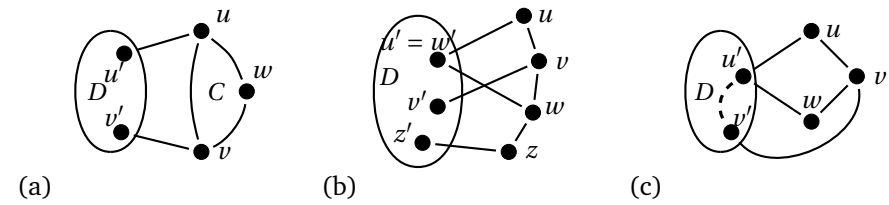


Figure 6.4: (a) Here we have a triangle C, with $V(C) = \{u, v, w\}$, in G'. A potential open ear of length 4 for D is the path uu', uw, wv, vv'. (b) u, v, w, z is a path in G'. Each vertex in this path is adjacent to a vertex $u', v', w', z' \in V(D)$, respectively. Since $w' \neq z'$ by construction, we can find a potential open ear of D of length at least 4 for D. (c) Here the path u, v, w is in neither case a nor b. Thus, u and w are degree 2 in G, with identical neighbourhood. So u', u, v, w is a forbidden square.

Since G is 2VC there are $u' \neq v' \in V(D)$, such that there exist vertex disjoint paths P_u and P_v from u to u' and v to v' respectively. Since $u' \neq v'$, the path $P_u \cup \{uv\} \cup P_v$ is a potential open ear of D. By our construction, there are no potential open ears of length at least 4 possible anymore. Thus we can see the length of the path $P_u \cup \{uv\} \cup P_v$ is at most 3, and hence exactly 3. Therefore, $P_u = \{uu'\}$ and $P_v = \{vv'\}$.

To prove the claim, we assume for contradiction G' has a vertex v of degree at least two. That is, we assume there are edges $uv, vw \in E(G')$ for $u \neq w$. We distinguish the following cases for the path u, v, w:

Case a: uw is an edge of G'. We know there are vertices $u' \neq v' \in V(D)$ with $uu', vv' \in E$. Now $\{u'u, uw, wv, vv'\}$ is the edge set of a potential open ear of D of length four, contradicting our construction of D. See Figure 6.4.(a).

Case b: u, v, w is part of a path P of length three in G'. Without loss of generality let u, v, w, z be a subpath of P in G' ($z \notin \{u, v, w\}$). We know there are vertices $u', v', w', z' \in V(D)$, with $u' \neq v'$, $w' \neq z'$, such that uu', vv', $ww', zz' \in E$. Since $w' \neq z'$, at most one of $u' = w'$ and $u' = z'$ can be true. Assume for contradiction that $u' \neq w'$. Then the path u', u, v, w, w' is a potential open ear for D of length 4. Contradicting our construction of D. If $u' \neq z'$, we can find a similar contradiction. See Figure 6.4.(b).

Case c: u, v, w is neither in case a nor in case b. We know there are vertices $u', w' \in V(D)$, such that uu', $ww' \in E$. If $u' \neq w'$, then the path u', u, v, w, w' is a potential open ear of D of length 4, a contradiction. So we see $u' = w'$. So this

implies that u' is the unique neighbour of u and w in $V(D)$, or again we find a potential open ear of D of length 4. Furthermore, as we assume we are not in case a or b, we can observe that v and u' are the only neighbours of u and w, ie, $deg_G(w) = deg_G(u) = 2$. That is, u, v, w, u' is a forbidden cycle, contradicting our application of Lemma 6.7. □

Since the edges of $G[V \setminus V(D)]$ form a matching, each connected component of $G[V \setminus V(D)]$ is either a singleton or an edge. We will now partition the vertices of $V \setminus V(D)$ into the sets $K_{1,1}, K_{1,2}, K_{2,2}$, and $K_{2,3}$.

We define $K_{1,1} \subseteq V \setminus V(D)$ as the singletons of $G[V \setminus V(D)]$ that have a safe vertex neighbour in $V(D)$. We define $K_{1,2} \subseteq V \setminus V(D)$ as the singletons of $G[V \setminus V(D)]$ that do not have a safe vertex neighbour in $V(D)$. We define $K_{2,2} \subseteq V \setminus V(D)$ as the endpoints of edges uv in $G[V \setminus V(D)]$ that satisfy one of the following: u and v are both adjacent to a (possibly equal) safe vertex in $V(D)$, or one of u or v are safe, and that safe vertex is adjacent to a safe vertex in $V(D)$. Finally, we define $K_{2,3} = V \setminus (V(D) \cup K_{1,1} \cup K_{1,2} \cup K_{2,2})$.

Intuitively, we imagine the set $K_{i,j}$ to represent vertices of the components of $G[V \setminus V(D)]$ with i vertices, where any feasible FVC of solution of G must have at least j edges with endpoints in a component.

We now describe our first algorithm, which will compute a feasible solution APX_1. Starting with $APX_1 := \emptyset$. We first add the edges of D to APX_1. If $V \setminus V(D) = \emptyset$, then APX_1 is feasible since D is 2VC, and we are done. Otherwise, we buy edges in the following way to make APX_1 feasible.

First, for every $v \in K_{1,1}$, buy an edge $uv \in E$ where $u \in V(D)$ is safe (such a u exists, by definition of $K_{1,1}$). Second, for every $v \in K_{1,2}$, we buy arbitrary pair of edges $uv, u'v \in E$, where $u \neq u'$ (these edges exists since G is assumed to be 2VC). Third, for every edge uv of $G[K_{2,2}]$, if u and v are both adjacent to (potentially equal) safe vertices u' and v' in $V(D)$, then we buy the edges uu' and vv'. If at least one of u and v, is not adjacent to a safe vertex in $V(D)$, then by definition of $K_{2,2}$, (without loss of generality) v is safe and is adjacent to a safe vertex in $v' \in V(D)$; In that case, we buy edges uv, vv'. Lastly, for each edge uv in $G[K_{2,3}]$, we buy edge uv and arbitrary pair of edges uu' and vv', where $u' \neq v'$. Observe that such edges must exist again as G is 2VC and $n \geq 4$.

Lemma 6.11. *Our algorithm computes a feasible FVC solution, APX_1, in polynomial time.*

Proof. Recall, D can be computed in polynomial time by Claim 6.1.

Computing and identifying $K_{1,1}, K_{1,2}, K_{2,1}, K_{2,2}$ can also be done in polynomial time. Therefore, the edges we add to APX_1 that share an endpoint with $K_{1,1} \cup$

$K_{1,2} \cup K_{2,2}, \cup K_{2,3}$ can be computed in polynomial time.

We now show that APX_1 is feasible by showing that every time we add a set of edges to APX_1, the subgraph $(V(APX_1), APX_1)$ is a feasible FVC solution for the instance defined by $G[V(APX_1)]$, namely $(V(APX_1), APX_1)$ is connected and has no unsafe cut-vertices ($(V(APX_1), APX_1)$ is not a feasible solution for the original instance). Let us start with $APX_1 \leftarrow D$. The subgraph D is 2VC, and hence $E(D)$ is a feasible FVC solution on $V(D)$. Observe that the edges selected for components of $K_{1,2}$ and $K_{2,3}$ are potential open ears for D of length two and three, respectively. Therefore, as D is 2VC, when we add these edges to APX_1, we have $(V(D) \cup K_{1,2} \cup K_{2,3}, APX_1)$ is 2VC.

For every $v \in K_{1,1}$, we buy edge uv, where $u \in V(D)$ is safe. As by adding these edges to APX_1, v is only adjacent to u then $(V(D) \cup K_{1,2} \cup K_{2,3} \cup K_{1,1}, APX_1)$ is connected and has no unsafe cut-vertex. A similar argument holds also for $K_{2,2}$ and hence this implies APX_1 is feasible. □

By our construction of APX_1, $|APX_1| \leq |E(D)| + |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|$. The following Lemma is clear since $|E(D)| \leq \frac{4}{3}(|V(D| - 1)$, by Lemma 6.9.

Lemma 6.12. $|APX_1| \leq \frac{4}{3}(|V(D)| - 1) + |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|$

We fix an optimal solution OPT to the instance $G = (V, E)$. The following Lemma provides a lower bound on $|OPT|$.

Lemma 6.13. $|OPT| \geq \max\{n, |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|\}$

Proof. We have $|OPT| \geq n$ by Lemma 6.8. We now show $|OPT| \geq |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|$. Recall by Lemma 6.10, that $G[V \setminus V(D)]$ is a matching. Consider a connected component F of $G[V \setminus V(D)]$. We consider cases of F. If $V(F) = \{v\} \subseteq K_{1,1}$, OPT has at least one edge incident on v. If $V(F) = \{v\} \subseteq K_{1,2}$, then v is not adjacent to a safe vertex, hence OPT has at least two edges incident on v. If $V(F) = \{u, v\} \subseteq K_{2,2}$, then as (V, OPT) is connected, OPT has at least two edges incident on at least one vertex in $\{u, v\}$. If $V(F) = \{u, v\} \subseteq K_{2,3}$, then as (V, OPT) is connected, OPT has at least two edges incident on at least one vertex in $\{u, v\}$. However as (V, OPT) has no unsafe cut-vertex then by definition of $K_{2,3}$, OPT must have at least three such edges. □

In the next Lemma show that even with the tools we have already developed, we have a $\frac{5}{3}$-approximation, answering the question if there exists an approximation factor less than 2 in the affirmative. We spend the remainder of the section improving on this easier approximation.

Lemma 6.14. *APX_1 is a $\frac{5}{3}$-approximate solution for FVC. Furthermore, if $|K_{1,2}| + |K_{2,3}| \leq 2$, then APX_1 is a $\frac{4}{3}$-approximate solution.*

Proof. With Lemmas 6.11 and 6.13, our approximation factor is upper-bounded by

$$\begin{aligned} S &:= \frac{\frac{4}{3}(|V(D)|-1) + |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|}{\max\{n, |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|\}} \\ &= \frac{\frac{4}{3}(|V(D)|-1) + |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|}{\max\{|V(D)| + |K_{1,1}| + |K_{1,2}| + |K_{2,2}| + |K_{2,3}|, |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|\}} \\ &\leq \frac{\frac{4}{3}|V(D)| - \frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}{\max\{|V(D)| + |K_{1,2}| + |K_{2,3}|, 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|\}}. \end{aligned}$$

Now we consider two case:

- If $|K_{1,2}| + \frac{1}{2}|K_{2,3}| \geq |V(D)|$. In this case we have:

  $$S = \frac{\frac{4}{3}|V(D)| - \frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}{2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}.$$

  Therefore by our condition, we have:

  $$S \leq \frac{\frac{4}{3}(|K_{1,2}| + \frac{1}{2}|K_{2,3}|) - \frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}{2|K_{1,2}| + \frac{3}{2}|K_{2,3}|} < \frac{5}{3}.$$

- Otherwise we have:

$$\begin{aligned} S &= \frac{\frac{4}{3}|V(D)| - \frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}{|V(D)| + |K_{1,2}| + |K_{2,3}|} \\ &= \frac{4}{3} + \frac{-\frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}| - \frac{4}{3}(|K_{1,2}| + |K_{2,3}|)}{|V(D)| + |K_{1,2}| + |K_{2,3}|} \leq \frac{4}{3} + \frac{-\frac{4}{3} + \frac{2}{3}|K_{1,2}| + \frac{1}{6}|K_{2,3}|}{|V(D)| + |K_{1,2}| + |K_{2,3}|} \\ &< \frac{4}{3} + \frac{\frac{2}{3}|K_{1,2}| + \frac{1}{6}|K_{2,3}|}{2|K_{1,2}| + \frac{2}{3}|K_{2,3}|} < \frac{5}{3}. \end{aligned}$$

So the first claim is proven. So now we consider the case when $|K_{2,3}| + |K_{1,2}| \leq 2$.

$$\begin{aligned} S &= \frac{\frac{4}{3}(|V(D)|-1) + |K_{1,1}| + 2|K_{1,2}| + 2|K_{2,2}| + \frac{3}{2}|K_{2,3}|}{\max\{n, |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}|\}} \\ &\leq \frac{\frac{4}{3}(|V(D)|-1) + |K_{1,1}| + 2|K_{1,2}| + 2|K_{2,2}| + \frac{3}{2}|K_{2,3}|}{|V(D)| + |K_{1,1}| + |K_{1,2}| + |K_{2,2}| + |K_{2,3}|} \\ &\leq \frac{\frac{4}{3}|V(D)| - \frac{4}{3} + 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|}{|V(D)| + |K_{1,2}| + |K_{2,3}|} \leq \frac{\frac{4}{3}|V(D)| - \frac{4}{3} + 4}{|V(D)| + 2} = \frac{4}{3}. \end{aligned}$$

□

6.1.2 Approximation 2

In this section, we will provide a second algorithm that relies on the vertex sets defined in Section 6.1.1, that were computed by our first algorithm. Namely, we are interested in $V(D)$, $K_{1,1}$, $K_{1,2}$, $K_{2,2}$, and $K_{2,3}$. This algorithm, when combined with the algorithm of Section 6.1.1 will achieve a $\frac{11}{7}$-approximation. Applying Lemma 6.14, we assume that at least one of $K_{1,2}$ or $K_{2,3}$ is non-empty, as otherwise we immediately have a $\frac{4}{3}$-approximation algorithm.

As in the previous section, we will rely on the fact that D is constructed as a 2VC subgraph of G, and the fact that any feasible solution must take edges incident to vertices in $K_{1,1}$, $K_{1,2}$, $K_{2,2}$, and $K_{2,3}$. However this time we start by buying a minimal set of edges E' incident to $K_{1,1}$, $K_{1,2}$, $K_{2,2}$, and $K_{2,3}$ that are required for feasibility and then we complement these edges with a subset of edges of $G[D]$ to obtain a feasible solution APX_2.

Before we describe our approximation algorithm, we will define the following tools that the algorithm relies on.

Definition 6.7 (Maximum Rainbow Connection Problem)**.** *Given a (multi-)graph G and a coloring $c : E \to \mathbb{N}$ of the edges, find a spanning subgraph of G that minimizes the number of components, while choosing exactly one edge from each colour.*

This problem can be solved to optimality using matroid intersection between the graphic matroid, and the partition matroid on the colour classes.

Lemma 6.15. *Given an instance of the Maximum Rainbow Connection Problem with (multi)-graph $G = (V, E)$, with coloring $c : E \to \mathbb{N}$. We can find in polynomial time, an optimal solution P such that the number of isolated vertices (vertices of degree 0) in (V, P) is minimal with respect to replacing an edge with another edge of the same colour class.*

Proof. For graph $G = (V, E)$, we first define two matroids with ground set E:

- Graph matroid: $M_{graphic}$ over the edges E with independent sets $\mathscr{I}_{graphic} = \{F \subseteq E | F \text{ is acyclic}\}$.
- Partition matroid: partition edges $E = E_1 \cup \cdots \cup E_k$ into k colour classes, $M_{partition}$ over the edges E with independent sets $\mathscr{I}_{partition} = \{F \subseteq E || F \cap E_i| \leq 1, i = 1, \ldots k\}$.

The intersection matroid of $M_{graphic}$ and $M_{partition}$ has independent sets that are acyclic and contain each edge colour at most once. Thus, the intersection matroid $M_{graphic} \cap M_{partition}$ can be solved in polynomial time to find a solution P'. For each colour class that is not in P', we select an arbitrary edge and add it to P' to find P. Note that P' is a maximum forest that uses each colour class at most once, that is, P' has a minimum number of components. Furthermore, by construction P has the same set of components as P' since if it decreased the number of components, P' would not be optimal.

To find a minimal number of singletons with respect to replacing edges of a colour class, we consider the edges of P and if there is an edge that can be swapped to reduce the number of singletons, we make that swap until there are no more improving swaps. Formally, for each edge $e \in P$, we can find all edges in $\tilde{E}$ with colour c, and identify if there is an edge e' with colour c such that $(V, P \cup \{e'\} \setminus \{e\})$ has one fewer singleton. If there is such an edge, we replace P with $P \cup \{e'\} \setminus \{e\}$. Note that by adding e' we reduce the number of connected components in (V, P) by exactly one, and by removing e we increase the number of connected components by at most one. Thus the number of connected components in this process will never increase.

This process terminates in polynomial time, as it reduces the number of singletons by one in each step, and each step takes polynomial time to compute. □

Our goal with the Maximum Rainbow Connection problem is to more cleverly find a minimal set E' than simply taking an *arbitrary* minimal set of edges incident on at least one vertex of $V \setminus V(D)$. Our goal is to select such an E' that also minimizes the number of connected components of (V, E'). To do this we use the edges of E that are incident on $K_{1,1}$, $K_{1,2}$, $K_{2,2}$, and $K_{2,3}$ to create an instance of Maximum Rainbow Connection Problem (and solve it with Lemma 6.15).

We define a set of so-called *pseudo-edges* $\tilde{E}$ with endpoints in $V(D)$ (named pseudo-edges in order to distinguish them from the "real" edges, E), and assign

to each pseudo-edge a unique colour indexed by vertices in $K_{1,2}$ and pairs of adjacent vertices in $K_{2,3}$. For every $u \in K_{1,2}$, and every distinct pair of edges uv_1 and $uv_2 \in E$, we add pseudo-edge v_1v_2 to $\tilde{E}$. Assign v_1v_2 the colour c_u. Intuitively, a pseudo-edge xy with colour c_v (for example) corresponds to a path in E from x to v to y.

For every ordered pair $(u, v) \in K_{2,3} \times K_{2,3}$ such that $uv \in E$, we add pseudo-edges to $\tilde{E}$ in the following way: (1) for every pair of edges uu' and vv' with $u' \neq v'$, we add pseudo-edge $u'v'$ to $\tilde{E}$. Assign $u'v'$ the colour c_{uv}. (2) If u is adjacent to safe vertex $u' \in V(D)$, and for any two neighbours $v_1', v_2' \in V(D)$ of v (if v has at least two neighbours in $V(D)$), we add pseudo-edge $v_1'v_2'$ to $\tilde{E}$. We assign $v_1'v_2'$ the colours c_{uv} (note v is not adjacent to a safe vertex since $u, v \notin K_{2,2}$). (3) If u is safe, then for every distinct pair of edges uv_1 and uv_2, we add pseudo-edge v_1v_2 to $\tilde{E}$. Assign v_1v_2 the colour c_{uv}.

Again, a pseudo-edge xy with colour c_{uv} corresponds to a path from x to y created by a minimum selection of the edges incident on x and y of a feasible FVC solution.

Notice that with $(V(D), \tilde{E})$ (along with the corresponding edge colours we provide) describe an instance of the Maximum Rainbow Connection problem. We use Lemma 6.15 to compute a solution to the Maximum Rainbow Connection problem, which we denote by P. Say that P has α many components, and α_{large} many components with at least two vertices. Then we use the following three algorithms one by one to obtain a feasible solution. We have one last tool we need to provide before we can define for the next step of our algorithm, which is inspired by techniques employed in [44]. This tool will be useful for letting us decide which edges of $\tilde{E}$ to buy.

Definition 6.8 (Good Cycles)**.** *Let $\Pi = (V_1, \ldots, V_k)$, $k \geq 2$, be a partition of the vertex-set of a 2VC simple graph G.*

A good cycle C of Π is a subset of edges with endpoints in distinct subsets of Π such that: (1) C is a cycle of length at least 2 in the graph obtained from G by contracting each V_i into a single vertex one by one (that is, $G/V_1/V_2/\ldots/V_k$); (2) given any two edges of C incident to some V_i, these edges are incident to distinct vertices of V_i unless $|V_i| = 1$; (3) C has an edge incident to at least one V_i with $|V_i| \geq 2$, and; (4) $|C| = 2$ only if both V_i and V_j incident to C have $|V_i|, |V_j| \geq 2$.

The following Lemma allows us to compute good cycles in polynomial time.

Lemma 6.16. *Let $\Pi = \{V_1, \ldots, V_k\}$, $k \geq 2$, be a partition of the vertex-set of a 2VC simple graph G with the following conditions: $G[V_i]$ is connected for all $i = 1, \ldots, k$,*

with at least one set of size at least 2. Furthermore, if there is exactly one $V_i \in \Pi$ with $|V_i| \geq 2$, then there are at least two singletons in Π that are adjacent in G.

Then in polynomial time, one can compute a good cycle C of Π.

Proof. To prove this lemma, we use a similar construction due to [44], which they show can be found in polynomial time.

Definition 6.9 ([44]). *(Nice Cycle and Path). Let $\Pi = \{V_1, \ldots, V_k\}$, $k \geq 2$, be a partition of the verte-set of a graph G. A nice cycle (resp. nice path) N of Π is a subset of edges with endpoints in distinct subsets of Π such that: (1) N induces one cycle of length at least 2 (resp., one path of length at least 1) in the graph obtained from G by by contracting each V_i into a single vertex one by one; (2) given any two edges of N incident to some V_i, these edges are incident to distinct vertices of V_i unless $|V_i| = 1$.*

Lemma 6.17 ([44]). *Let $\Pi = (V_1, \ldots, V_k)$, $k \geq 2$, be a partition of the vertex-set of a 2VC simple graph G. In polynomial time one can compute a nice cycle N of Π.*

Each set $V_i \in \Pi$ induces a connected graph in G, and we call these sets *components* for simplicity. For $V_i \in \Pi$ with $|V_i| \geq 2$, we say that V_i is a large component. For $V_i \in \Pi$ with $|V_i| = 1$, we say that V_i is a singleton of Π. Using the sets of Π, we create a new vertex partitioning Π'' in the following way.

We define $A_1 \subseteq \Pi$ as the singletons of Π that are adjacent to another singleton in Π. Denote by F_1 the set of maximal connected components of A_1 with edges in E. Define $\Pi' \coloneqq \Pi \cup F_1 \setminus A_1$. So basically Π' is a partition obtained from Π by replacing singletons of Π by the vertex sets of the connected components of graph $G[A_1]$.

We define $A_2 \subseteq \Pi'$ as the singletons of Π' that are incident to two distinct edges with endpoint in the same large component of Π. Starting with $F_2 \coloneqq \emptyset$ and $F_2^- \coloneqq \emptyset$. For each $D \in \Pi$, that is adjacent to a singleton in A_2, let $N_{A_2}(D)$ denote the vertices of A_2 that are adjacent to D. Add $D \cup N_{A_2}(D)$ to F_2, add D and every vertex of $N_{A_2}(D)$ to F_2^-, and finally remove $N_{A_2}(D)$ from A_2. When A_2 is empty, define $\Pi'' \coloneqq \Pi' \cup F_2 \setminus F_2^-$. So basically we replace large components and any singletons (that haven't already been processed) that have two edges between them with a single large component.

We define $F_0 \coloneqq \Pi \setminus F_2^-$, the large components that are not subsets of components in F_1. Finally, we define $A_0 \subseteq \Pi''$ the set of singletons of Π''. The following observation is clear from the definition of these sets.

Observation 6.1. *$V(F_0) \cup V(F_1) \cup V(F_2) \cup V(A_0) = V$, and $V(F_0), V(F_1), V(F_2), V(A_0)$ are pairwise disjoint.*

As we are always merging components of Π to create Π' the following observation is clear.

Observation 6.2. *Every component of* Π *is a (not necessarily strict) subset of a component of* Π'.

Thus, Π'' partitions V. To see that $|\Pi''| \geq 2$, we consider cases. If there are at least two large components of Π then $|\Pi''| \geq 2$ since large components of Π are not part of the same component of Π''. If there is exactly one large component of Π then at least one of F_0 or F_2 is non-empty, and by our assumption on Π, there are at least two singletons that are adjacent in G, which shows that F_1 is non-empty. Therefore, Π'' satisfies the conditions of Lemma 6.17.

We compute a nice cycle C using Lemma 6.17 on Π''. It remains to find a good cycle C' on Π. We will find sure C' by using the edges of C, as well as additional edges depending the components that are incident to C. In particular, if C incident to a component D of F_0 or F_2, then C' will be incident to the large component that is contained in D. If C is incident to a component $D \in F_1$, then we guarantee that C' will be incident to at least two singletons of Π that are contained in D. We begin by adding every edge of C to C'. For every component $D \in \Pi''$ that has endpoints of edges of C, we consider by cases which set among F_0, F_1, F_2, and A_0 contain D.

First, if $D \in A_0$ or $D \in F_0$, we do nothing.

Next, if $D \in F_1$. Let $e_1, e_2 \in E(C)$ be the edges of C incident to D. Observe, by definition of a nice cycle, e_1 and e_2 have distinct endpoints $v_1, v_2 \in V(D)$ respectively. By construction, $G[D]$ is connected, so there is a path in $G[D]$ between v_1 and v_2. We add the edges of this path to C'.

Finally, if $D \in F_2$. Let $e_3, e_4 \in E(C)$ be the edges of C incident to D. By definition of a nice cycle e_3 and e_4 have distinct endpoints $v_3, v_4 \in V(D)$ respectively. We consider cases for v_3 and v_4. By construction, there is exactly one large component $L \in \Pi$ such that $L \subsetneq D$.

Case: If $v_3, v_4 \in V(L)$. We are done, as C' now has two edges with distinct endpoints in a large component of Π.

Case: If $v_3 \in V(L)$ but $v_4 \notin V(L)$. (the opposite case where only $v_4 \in V(L)$ is similar). By definition of F_2, v_3 is incident to at least two distinct edges, with endpoints in L, say e_3', e_3''. At most one of these two may have v_4 as an endpoint, without loss of generality say e_3'' has v_4 as an endpoint in this case. We add e_3' to C' and we are done, as C' now has two edges with distinct endpoints in L, a large component of Π.

Case: If $v_3, v_4 \notin V(L)$. By definition of F_1, v_3 and v_4 are incident to at least two distinct edges, with endpoints in L. Denote edges incident to v_3 by e_3', e_3''

and the edges incident to v_4 by e'_4, e''_4. We add e'_3 to C'. At most one of e'_4 and e''_4 can share an endpoint with e'_3, without loss of generality say e'_4. Add e''_4 to C', and we are done as C' now has two edges with distinct endpoints in L, a large component of Π.

By construction, we have the following: (1) C' induces a cycle of length at least 2 in the graph obtained from G by collapsing each $V_i \in \Pi$ into a single vertex; (2) given any two edges of C' incident to some V_i, these edges are incident to distinct vertices of V_i unless $|V_i| = 1$.

It remains to show the remaining conditions of a good cycle: (3) C' has an edge incident to at least one $V_i \in \Pi$ with $|V_i| \geq 2$, and; (4) $|C'| = 2$ only if both V_i and V_j incident to C' have $|V_i|, |V_j| \geq 2$. To see point (3), by construction of the sets F_0, F_1, F_2 and A_0, any component in F_1 and A_0 are adjacent only to components in F_0 and F_2. Thus, since a nice cycle has at least two edges, it will contain a component from F_0 or F_2. And by construction of C, C will be incident to a large component of Π.

To see point (4), observe that since $C \subseteq C'$, so if $|C'| = 2$ then $|C| = 2$. So we consider the cases where $|C| = 2$. If C is incident to a component D in F_1, then C' will have at least three edges by construction, as C' also contains edges of a path in $G[D]$. If C is incident to a component in A_0, then by definition of A_0, C is incident to at least 2 other components in Π'', hence $|C| \geq 3$. Therefore, C can be adjacent only to components in F_0 and F_2. If C is incident to a component in F_2 at a vertex that is a singleton in Π, then $|C'| \geq 3$. Thus, C' is incident to vertices of exactly two large components of Π. And C' satisfies the conditions of a good cycle. □

In our algorithm we will distinguish between connected components that have one vertex (i.e. singletons) and connected components with at least two vertices. Given a graph H, we say that a connected component is *large* if it has at least two vertices. Now we have all the ingredients required to finalize the description of our approximation algorithm. After computation of P, we initialize the set that will eventually be our solution as $APX_2 \leftarrow \emptyset$.

Our plan is to gradually build APX_2. We have three steps, Algorithm 5, 6 and 7, which we apply one after another. These algorithms return edge sets S_1, S_2 and S_3, respectively that will be part of our solution APX_2. We now take time to describe these algorithms in more details. Providing a demonstration of these steps in Figures 6.5 and 6.6.

First we use Algorithm 5, which takes the pseudo-edges P, and find the large components and a subset X_1 of singletons of $(V(D), P)$, buying a subset of edges, S_1, to connect them into a single component A in $(V(D), S_1 \cup P)$ using

Lemma 6.16. Upon termination of this algorithm, we will obtain the additional property that $V(D)\setminus V(A)$ is an independent set in G.

Algorithm 5: Buying Good Cycles

Input: pseudo-edges P
Let Y denote the singletons of $(V(D), P)$
$S_1 \leftarrow \emptyset$
while *There is a good cycle C in $G[V(D)]$ with connected components of $(V(D), P \cup S_1)$ being the vertex partitioning* **do**
  $S_1 \leftarrow S_1 \cup E(C)$
end
$A \leftarrow$ unique large component of $(V(D), S_1 \cup P)$
Let $X_1 \leftarrow Y \cap V(A)$
Return (X_1, Y, S_1, A)

In Algorithm 6, our goal is to buy a minimal set S_2 of edges between $V(A)$ and compute a subset of $V(D)\setminus V(A)$ (which we denote by X_2), such that $(V(A) \cup X_2, P \cup S_1 \cup S_2)$ has one block. We remark that after termination of this algorithm any vertex of $X_3 := V(D) \setminus V(A) \cup X_2)$ is a singleton in $(V(D), P \cup S_1 \cup S_2)$

In Algorithm 7, the goal is to buy a subset S_3 of edges such that $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$ is a feasible FVC solution on $V(D)$ (i.e. $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$ is connected and has no unsafe cut-vertices). For every vertex in $v \in X_3 := V(D)\setminus(V(A) \cup X_2)$ that has a safe neighbour in $V(A) \cup X_2$, we buy one edge from v to one of its safe neighbour in $V(A) \cup X_2$. For any other vertex in X_3 we buy two distinct edges from it to $V(A) \cup X_2$. We define α'_1 as the number of vertices of X_3 that have a safe neighbour, and α'_2 is the number of vertices of X_3 that do not have a safe neighbour. Thus $|X_3| = \alpha'_1 + \alpha'_2$. To maintain a consistent notation for number of components we define $\alpha' := |X_3| = \alpha'_1 + \alpha'_2$. Note that this implies, $\alpha = \alpha_{large} + |X_1| + |X_2| + |X_3| = \alpha_{large} + |X_1| + |X_2| + \alpha'$.

We finalize our solution for the instance by computing S_P, edges with endpoints in $K_{1,1}, K_{1,2}, K_{2,2}$, and $K_{2,3}$, by considering each pseudo-edge $\tilde{e} = v_1v_2 \in P$. If $\tilde{e}$ has colour c_u, then $u \in K_{1,2}$. We add edges uv_1 and uv_2 to S_P.

If $\tilde{e}$ has colour c_{uv}, then $u, v \in K_{2,3}$. We add edges to S_P in exactly one of the following ways (breaking ties in an arbitrary but fixed manner): (1) we add uv_1, vv_2 and uv to S_P if $uv_1, vv_2 \in E$; (2) we add uv_2, vv_1 and uv to S_P if $uv_2, vv_1 \in E$; (3) we add uv_1, uv_2, and uv to S_P if u is safe, and uv_1, uv_2, $uv \in E$; (4) we add vv_1, vv_2, and uv to S_P if v is safe, and vv_1, vv_2, $uv \in E$; (5) we add vv_1, vv_2, and uu_1 to S_P if there exist a safe vertex $u_1 \in V(D)$ such that $uu_1 \in E$,

Algorithm 6: Making Large Component 2VC

Input: Edges $S_1 \subseteq E$, pseudo-edges P, large component A
$X_2, S_2 \leftarrow \emptyset$
while $G[V(A) \cup X_2] \cup P$ *has more than one block* **do**
  Find $v \in V(D) \setminus (V(A) \cup X_2)$ such that $G[V(A) \cup X_2 \cup \{v\}] \cup P$ has fewer blocks than $G[V(A) \cup X_2] \cup P$
  Find edges $e_1 = vu$, $e_2 = vw$, $v \neq w \in V(A)$, such that $(V(A) \cup X_2 \cup \{v\}, P \cup S_1 \cup S_2 \cup \{e_1, e_2\})$ has fewer blocks than $(V(A) \cup X_2, P \cup S_1 \cup S_2)$
  $X_2 \leftarrow X_2 \cup \{v\}$
  $S_2 \leftarrow S_2 \cup \{e_1, e_2\}$
end
while *There exists an edge* $e_3 = uz \in E \setminus S$ *such that* $(V(A) \cup X_2, P \cup S_1 \cup S_2 \cup \{e_3\})$ *has fewer blocks than* $(V(A) \cup X_2, P \cup S_1 \cup S_2)$ **do**
  $S_2 \leftarrow S_2 \cup \{e_3\}$
end
Return (X_2, S_2)

and vv_1, $vv_2 \in E$, and; (6) we add uv_1, uv_2, and vu_1 to S_P if there exist a safe vertex $u_1 \in V(D)$ such that $vu_1 \in E$, and uv_1, $uv_2 \in E$.

For every $v \in K_{1,1}$, buy an edge $uv \in E$ where $u \in V(D)$ is safe. By definition of $K_{1,1}$, u exists. Also, for every $uv \in E(K_{2,2})$, if u and v are both adjacent to safe vertices u' and v' in $V(D)$, then we buy the edges uu' and vv'. If at least one of u and v, (wlog say u) is not adjacent to a safe vertex in $V(D)$, then by definition of $K_{1,2}$, v must be safe, and v must be adjacent to a safe vertex $v' \in V(D)$ in G. In this case we buy edges vv', uv. Observe that by construction, $|S_P| = |K_{1,1}| + 2|K_{1,2}| + 2|K_{2,2}| + \frac{3}{2}|K_{2,3}|$. The output of our algorithm is $APX_2 := S_P \cup S_1 \cup S_2 \cup S_3$. The following Lemma shows that APX_2 is feasible, can be computed in polynomial time, as well as an upper bound on APX_2.

Lemma 6.18. *Our algorithm computes a feasible solution* $APX_2 = S_P \cup S_1 \cup S_2 \cup S_3$, *in polynomial time and* $|APX_2| \leq |V(D)| - 1 + |S_P| + \alpha - 1 - (\frac{\alpha - \alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1)$.

Proof. Let us begin by the following simple claim.

Claim 6.2. $(V(D), P)$ *has at most* $|V(D)| - \alpha$ *blocks.*

Algorithm 7: Making Solution Feasible

Input: Singletons X_2, large component A
$S_3 \leftarrow \emptyset$
$X_3 \leftarrow V(D) \setminus (V(A) \cup X_2)$
$\alpha'_1 \leftarrow 0$, $\alpha'_2 \leftarrow 0$
for *every* $v \in X_3$ **do**
  if *v is adjacent to safe vertex $u \in V(A) \cup X_2$* **then**
    $S_3 \leftarrow S_3 \cup \{uv\}$, $\alpha'_1 \leftarrow \alpha'_1 + 1$
  end
  else
    Find $u, w \in V(A) \cup X_2$ adjacent to v
    $S_3 \leftarrow S_3 \cup \{uv, uw\}$, $\alpha'_2 \leftarrow \alpha'_2 + 1$
  end
end
Return $(X_3, S_3, \alpha'_1, \alpha'_2)$

Proof. Assume that the initial connected components are of size $a_1, ..., a_\alpha$. Then the total number of blocks by Lemma 6.3 is at most $\sum_{i=1}^{\alpha}(a_i - 1) = n - \alpha$. □

Now we will use the next observation for the analysis of Algorithm 5.

Observation 6.3. *Consider good cycle C computed by line 4 in Algorithm 5 and suppose C contains l large connected components. By adding C to S_1 the number of blocks in $(V(D), P \cup S_1)$ is reduced by at least $l - 1$.*

Proof. Let $T_1, \ldots, T_l$ be the set of large connected components of $(V(D), P \cup S_1)$ that are part of cycle C, and let u_i and v_i be the vertices of T_i that are incident on edges of C for $i = 1, \ldots, l$. Now, consider all the edges that belong to paths from u_i to v_i in T_i, which we denote by F_i. Let $F'_i \supseteq F_i$ denote the union of the edge sets of blocks in T_i that contain at least one edge of F_i. Observe that by adding C, the set $E(C) \cup F'_1 \cup F'_2 \cdots \cup F'_l$ merge into a single block B. Thus at least l blocks merge into a single block B and hence the number of blocks decreases by at least $l - 1$. □

In the next two observations we analyze the running time of Algorithm 5 and the cost of the solution obtained by this algorithm.

Observation 6.4. *Algorithm 5 terminates in polynomial time. Furthermore, upon termination of Algorithm 5, $(V(D), P \cup S_1)$ has one large connected component,*

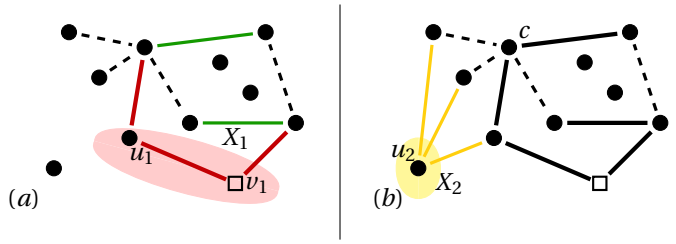


Figure 6.5: A depiction of the edges and vertex sets found by Algorithms 5, 6, and 7 in $V(D)$. Here the unsafe vertices are depicted by black circles. In this example there is only one safe vertex, v_1 in the set $V(D)$ that is shown by a square.
(a) The dashed edges are pseudo-edges P found by Lemma 6.15. Algorithm 5 first computes good cycle on green edges that merges two large components of pseudo-edges, then it finds the red cycle that merges the new large component and 2 singletons. $X_1 = \{u_1, v_1\}$
(b) The yellow edges of the second figure are found by Algorithm 6 which cover the cut-vertex c in the component. The interior vertex is $X_2 = \{u_2\}$.

namely A, and $V(D) \setminus V(A)$ forms an independent set of vertices in graphs $(V(D), P \cup S_1)$ and $G[V(D)]$.

Proof. Note that initially $P \neq \emptyset$ as we are applying Lemma 6.14 and assuming at least one of $K_{1,2}$ and $K_{2,3}$ is not an empty set. Furthermore at each step of Line 4 to 5 in Algorithm 5, we are adding edges of a good cycle to S_1 to merge at least two connected components of $(V(D), P \cup S_1)$ into a single component.

Therefore (1) Line 4 to 5 iterates at most $|V(D)|$ times, and; (2) once the algorithm reaches line 6, we must have at least one large connected component. Furthermore, using Lemma 6.16 we can always find a good cycle as long as we have more than one large connected components in $(V(D), P \cup S_1)$, or at least two singletons are adjacent. Thus, once we reach line 6 of Algorithm 5, we have precisely one large connected component, A in $(V(D), P \cup S_1)$. Additionally, at this stage all the vertices of $V(D) \setminus V(A)$ form an independent set of vertices in $G \cup P$, as otherwise there is another good cycle.

To see that Algorithm 5 terminates in polynomial time, observe that it has at most $|V(D)|$ iterations, and in each iteration we compute a good cycle, which can be done in polynomial time by Lemma 6.16. □

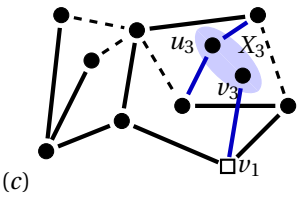


Figure 6.6: Continuing the example of Figure 6.5, the blue edges of the third figure are the edges found by Algorithm 7, which add edges to the solution that bring u_3 and v_3 into $V(D)$ form a feasible FVC solution. The vertex v_1 is a safe vertex so we only add one edge $(x_3 v_1)$ incident on v_3.

Claim 6.3. *At the end of Algorithm 5, (1)* $(V(D), S_1 \cup P)$ *has at most* $|V(D)| - \alpha - \alpha_{large} + 1$ *blocks, and; (2)* $|S_1| \leq 2\alpha_{large} + \frac{3}{2}|X_1| - 2$.

Proof. We started with at most $|V(D)| - \alpha$ blocks by Claim 6.2. Now let $a_1, \ldots, a_t$ be the size of these good cycles, and let l_i be the number of large connected components that appeared in i^{th} good cycle. By Observation 6.3, the total decrease on the number of blocks after these iterations is at least $\sum_{i=1}^{t}(l_i - 1)$. However, $l_i - 1$ is equal to the decrease in the number of large connected components of $(V(D), S_1 \cup P)$ when we add the i^{th} cycle to S_1. Hence, as we start with α_{large} many large connected components then $\sum_{i=1}^{t}(l_i - 1) = \alpha_{large} - 1$. Altogether, these imply that the total number of the blocks at the end is at most $|V(D)| - \alpha - (\alpha_{large} - 1)$, and claim (1) of the observation is proven.

Furthermore, we observe that $\sum_{i=1}^{t}(a_i - 1) = \alpha_{large} + |X_1| - 1$ (This is due to two facts: 1) adding the i^{th} good cycle to S_1 decreases the number of connected components of $(V(D), S_1 \cup P)$ by $a_i - 1$, and; 2) adding all these cycles merge the initial α_{large} many large connected components and $|X_1|$ many singleton components into a single (large) component).

Therefore the total number of edges (i.e. $\sum_{i=1}^{t} a_i$) is equal to $\alpha_{large} + |X_1| - 1 + t$. Assume that among these t good cycles, we have $t' \leq t$ cycles of length 2. Then since each a_i is at least 2, we can see that $t' + 2(t - t') \leq \sum_{i=1}^{t}(a_i - 1) = \alpha_{large} + |X_1| - 1$. Furthermore, since every cycle of length two merges two large connected components into one connected component, we have $t' \leq \alpha_{large} - 1$,.

Thus, $t = \frac{(t' + 2(t-t')) + t'}{2} \leq \frac{1}{2}(\alpha_{large} + |X_l| - 1 + \alpha_{large} - 1)$. Hence $\sum_{i=1}^{t} a_i = \sum_{i=1}^{t}(a_i - 1) + t \leq (\alpha_{large} + |X_1| - 1) + \alpha_{large} - 1 + \frac{|X_l|}{2} = 2\alpha_{large} + \frac{3}{2}|X_1| - 2$, hence the claim. □

Now we analyze the second phase of our algorithm (Algorithm 2).

Observation 6.5. *At the end of Algorithm 6, $(V(D), S_1 \cup S_2 \cup P)$ has α' connected components that are singletons (which is set X_3) and one large connected component that has exactly one block. Furthermore, Algorithm 6 terminates in polynomial time.*

Proof. First recall that since D is defined as an open ear decomposition, $G[V(D)]$ is 2VC by Lemma 6.1. Second, by Observation 6.4, $V(D) \setminus V(A)$ is an independent set.

We show that, as long as $G[V(A) \cup X_2] \cup P$ has more than one block, then there exists a vertex $v \in V(D) \setminus (V(A) \cup X_2)$ such that $G[V(A) \cup X_2] \cup P$ has more blocks than $G[V(A) \cup X_2 \cup \{v\}] \cup P$.

Assume otherwise for sake of contradiction. That is, for every vertex $v \in V(D) \setminus (V(A) \cup X_2)$ there exists a block B_v of $G[V(A) \cup X_2] \cup P$ such that every neighbor of $v \in V(D)$ belongs to $V(B_v)$. Hence any two edges of $G[V(A) \cup X_2] \cup P$ that are not in the same block in $G[V(A) \cup X_2] \cup P$ are not in the same block $G[V(D)] \cup P$. Therefore $G[V(D)] \cup P$ is also not 2VC, a contradiction.

So Line 4 of Algorithm 6 can find such a v. Moreover, v can be found in polynomial time.

Now we show that we can find edges e_1 and e_2 as indicated by line 5 of Algorithm 6. Assume for contradiction this is not the case. Consider any $x \neq y \in V(A) \cup X_2$, such that vx and vy are edges of G. Let E_{xy} denote the edges of $(V(A) \cup X_2 \cup \{v\}, P \cup S_1 \cup S_2)$ that belong to a path from x to y in $(V(A) \cup X_2 \cup \{v\}, P \cup S_1 \cup S_2)$. Then as vx and vy do not satisfy conditions of line 5 of Algorithm 6 then all edges of E_{xy} are in a unique block B_{xy} of $G[V(A) \cup X_2 \cup \{v\}] \cup P$.

There must exist a pair $x', y' \in V(A) \cup X_2$, $x' \neq y'$, such that vx' and vy' are edges of G. Otherwise, every neighbour of v is in a single block, thus $G[V(A) \cup X_2 \cup \{v\}] \cup P$ has the same number of blocks as $G[V(A) \cup X_2] \cup P$, which is a contradiction.

Therefore, there must exist $x, y, x', y' \in V(A) \cup X_2$ such that $B_{xy} \neq B_{x'y'}$. Furthermore, we can assume that $y = y'$, since if $y \neq y'$, then $B_{xy'}$ can be equal to at most one of B_{xy} and $B_{x'y}$, and without loss of generality we can assume that $B_{xy} \neq B_{x'y}$. There is a path in B_{xy} from x to y with at least one edge e, and a path $B_{x'y}$ from x' to y with at least on edge e'. Therefore, there is a cycle in $G[V(A) \cup X_2] \cup P \cup \{vx, vx'\}$ that contains e and e', and by Lemma 6.2, e and e' are in the same block, contradicting our assumption.

Note that as A is the only large connected component of $(V(D), P \cup S_1)$, then the vertices of $A \cup X_2$ is the vertex set of the only connected component of

$(V(D), P \cup S_1 \cup S_2)$.

Finally using Lemma 6.3 while there is an edge $e_3 \in E \setminus S$ such that $(V(A) \cup X_2, P \cup S_1 \cup S_2 \cup \{e_3\})$ that has fewer blocks than $(V(A) \cup X_2, P \cup S_1 \cup S_2)$, then it can be found in polynomial time.

Therefore we have one connected component that is 2VC. Notice that all the vertices not in this component, i.e. the set X_3, are isolated vertices in this graph and hence we have α' small components.

Observe that the while loop on line 3 iterates at most $|V(D)|$, since in every iteration of this loop we increase the size of X_2 and since $X_2 \subseteq V(D)$. Furthermore at every iteration of the while loop on line 8, we increase the size of S_2 by adding a new edge of $E(G)$. Thus we will have at most $|E(G)|$ iteration of this while loop. Therefore the algorithm runs in poly-time. □

Claim 6.4. $|S_2| \leq |X_2| + |V(D)| - \alpha - \alpha_{large}$.

Proof. We start with b blocks at the start of the Algorithm 6, where $b \leq |V(D)| - \alpha - \alpha_{large} + 1$ by Claim 6.3. We will consider what happens to the number of blocks of $V(A \cup X_2, P \cup S_1 \cup S_2)$ as we add each edge. Let us partition the edges of S_2 into $F_1, ..., F_{|X_2|}, F'$, where F' is the edges added by line of Algorithm 6 and F_i is the pair of edges added as we add the i-th vertex of X_2 by line 7 of the algorithm. In that case, every time we add an edge of F', we decrease the number of blocks of $V(A \cup X_2, P \cup S_1 \cup S_2)$ by at least one. Furthermore the edges of F_i decrease the number of blocks by at least one. As we have precisely one block in the end, then the total number of edges is:

$$|S_2| = \sum_{i=1}^{|X_2|} |F_i| + |F'| \leq (b-1) + |X_2| \leq |X_2| + |V(D)| - \alpha - \alpha_{large}.$$

□

By construction in Algorithm 7 we add $|S_3| = \alpha'_1 + 2\alpha'_2$ edges. Hence, by Claims 6.3 and 6.4 $|S_1| + |S_2| + |S_3|$ is upper-bounded by:

$$\begin{aligned}
&(2\alpha_{large} + \frac{3}{2}|X_1| - 2) + (|X_2| + |V(D)| - \alpha - \alpha_{large}) + (\alpha'_1 + 2\alpha'_2) \\
=&|V(D)| + \alpha_{large} + |X_2| + \frac{3}{2}|X_1| - 2 - \alpha + 2\alpha' - \alpha'_1 \\
=&|V(D)| + \frac{|X_1|}{2} + (\alpha_{large} + |X_2| + |X_1| + \alpha') - 2 - \alpha + \alpha' - \alpha'_1 \\
=&|V(D)| + \frac{|X_1|}{2} - 2 + \alpha' - \alpha'_1.
\end{aligned}$$

Therefore the total number of edges that we use is $|S_1|+|S_2|+|S_3|+|S_P|$, which is at most:

$$\begin{aligned}
&|S_P| + |V(D)| + \frac{|X_1|}{2} + \alpha' - \alpha'_1 - 2 \\
=&|S_P| + |V(D)| - 2 - \alpha'_1 + \alpha' + \frac{\alpha - \alpha' - \alpha_{large} - |X_2|}{2} \\
\leq&|V(D)| + |S_P| - 2 + \alpha - \frac{\alpha - \alpha'}{2} - \frac{\alpha_{large}}{2} - \alpha'_1.
\end{aligned}$$

Furthermore as $(V(D) \setminus X_3, S_1 \cup S_2 \cup P)$ is 2VC, then $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$ is connected and has no unsafe vertex that is a cut-vertex. Finally we show that $S_1 \cup S_2 \cup S_3 \cup S_P$ is feasible. Let us start by showing that $H := (V, S_1 \cup S_2 \cup S_3 \cup S_P)$ is connected. Note that by our choice of S_P, every vertex v in $K := K_{1,1} \cup K_{1,2} \cup K_{2,1} \cup K_{2,2}$ is either an endpoint of an edge $uv \in S_P$ such that $u \in V(D)$ or is incident on edges $wv', v'v \in S_P$ such that $w \in V(D)$ and $v' \in K$.

Thus, it suffices to show that any two vertices in $V(D)$ are in the same connected component of H. Now, $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$ is connected, and every edge $xy \in P$, S_P contains a path from x to y then all the vertices of $V(D)$ are in the same connected component of H and hence H is connected.

Now we show that H has no unsafe cut-vertex. Assume this is not true and such a cut-vertex u exists. If $u \in K_{1,1}$, it's a leaf of H adjacent to a safe vertex, thus, it cannot be a cut-vertex. If $u \in K_{2,2}$, since u is a cut-vertex, then the degree of u is at least 2 in H, then by our choice of S_P, u is safe.

If $u \in K_{1,2}$, the degree of u in H is two, thus $H \setminus \{u\}$ has two components. Let $w, z \in V$ be the vertices u is adjacent to. Observe by definition of $K_{1,2}$ that $w, z \in V(D)$. Since u is a cut-vertex, there is no path in $H \setminus \{u\}$ from w to z. (Note that this implies that there is exactly one copy of the edge wz in P). Therefore, there is no path from w to z in $(V(D), S_1 \cup S_2 \cup S_3 \cup P \setminus \{wz\})$. So at least one of w and z are cut-vertices of $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$. But since $u \in K_{1,2}$, u is not adjacent to any safe vertices, therefore, w and z are unsafe, contradicting our conclusion that $(V(D), S_1 \cup S_2 \cup S_3 \cup P)$ has no unsafe cut-vertices.

If $u \in K_{2,3}$, the degree of u in H is at least 2. There are two cases to consider by our choice of S_P: (1) u is degree 3, and (2) u is degree 2. If u is degree 3 then by our choice of S_P, u is safe vertex, which is a contradiction. Therefore, we assume u is degree 2.

Let v be the unique vertex in $K_{2,3}$ such that $uv \in E$. If $uv \notin S_P$, then there are vertices $u_1, u_2 \in V(D)$ such that $uu_1, uu_2 \in S_P$, and a safe vertex $v_1 \in V(D)$ such that $vv_1 \in S_P$. This follows a similar argument to the previous case, where $u \in K_{1,2}$.

Lastly, if $uv \in S_P$, there exist vertices $u', v' \in V(D)$ such that $uu', vv' \in S_P$. Since u is a cut-vertex of H, there is no path from u' to v' in $H \setminus \{u\}$. Again this implies that there is exactly one copy of the edge $u'v'$ in P. Therefore there is no path from u' to v' in $(V(D), S_1 \cup S_2 \cup S_3 \cup P \setminus \{u'v'\})$. Recall that by Observation 6.5, $(V(D), S_1 \cup S_2 \cup P)$ has a unique large connected component that has exactly one block B. Furthermore as we assume $|K_{1,2}| + |K_{2,3}| > 2$ by Lemma 6.14, then $|P| = |K_{1,2}| + \frac{1}{2}|K_{2,3}| > 1$. Therefore, as B is a block that has at least two edges (as $|P| \geq 2$), there is a path from u' to v' in $(V(D), S_1 \cup S_2 \cup S_3 \cup P \setminus \{u'v'\})$, a contradiction.

Therefore we can assume that $u \in V(D)$. Now let H' be the graph obtained from H by removing u and let C be a connected components of H'. Observe that by our choice of S_P if $V(C) \cap V(D) = \emptyset$, then u must be a safe vertex, a contradiction. Therefore $V(C)$ has a vertex of D. Now as u is a cut-vertex of H, then by the above argument there must be two vertices v and v' of D such that there is no path in H' from v to v'. Let R be a path from v to v' in $(V(D), S_1 \cup S_2 \cup P)$ that does not contain u. Now we obtain a path R' from v to v' using R, which is a contradiction. We start by setting $R' := R$. Then for any pseudo-edge xy in R' we remove xy from R' and add the unique path of length maximum three from x to y that contains edges of S_P. □

6.1.3 Approximation Factor

We fix an optimal solution, OPT, for the instance $G = (V, E)$. The following Lemma finds a set of lower bounds on $|OPT|$ that depend on terms found by our algorithm, in particular $K_{1,1}, K_{1,2}, K_{2,2}, K_{2,3}, \alpha, \alpha_{large}, \alpha'_1$, and α'_2. We Recall that $|S_P| = |K_{1,1}| + 2|K_{1,2}| + 2|K_{2,2}| + \frac{3}{2}|K_{2,3}|$.

Lemma 6.19. $|OPT| \geq \max\{|S_P| + \alpha - 1, 2K_{1,2} - 2\alpha_{large} + \alpha'_1 + 2\alpha'_2, n\}$

Proof. We are given graph $G = (V, E)$, for every subset of edges $F \subseteq \binom{V}{2}$, we let $\kappa(F)$ denote the number of connected components of graph (V, F).

- We start by showing $|OPT| \geq |K_{1,2}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,2}|$.

  Consider the set of edges $E^* \subseteq OPT$ that have an endpoint in $K_{1,2} \cup K_{2,3}$. We show the following useful claim.

  Claim 6.5. *(V, E^*) has at least $\alpha + 2K_{1,2} + \frac{3}{2}K_{2,3} - |E^*| + K_{2,2} + K_{1,1}$ many connected components.*

Proof. The primary tool for proving this claim is the fact that any subset of pseudo-edges of $\tilde{E}$ that is a feasible solution to the Maximum Rainbow Connection problem, has at least α connected components. To use this fact we first use E^* to create a set $E_1^* \subseteq \binom{V}{2}$, and a set of pseudo-edges. We will guarantee that $|E_1^*| = 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|$ and $\kappa(E_1^*) - \kappa(E^*) \leq |E^*| - |E_1^*|$, and every pseudo-edge of P' has a unique colour. We will use P', and the above fact to show that $\kappa(E_1^*) \leq \alpha + |K_{1,1}| + |K_{2,3}|$, and then show the claim holds.

Note, every time we decrease $|E_1^*|$ by k, $\kappa(E_1^*)$ increases by at most k. This is clear since every time we remove an edge from E_1^*, we increase $\kappa(E_1^*)$ by at most one.

We begin $P' \leftarrow \emptyset$ and $E_1^* \leftarrow E^*$. For every $v \in K_{1,2}$, we remove all but two edges incident to v from E_1^*. For every $uv \in E(G[K_{2,3}])$, if there are at least three edges between u (or v) and $V(D)$, but $uv \notin E_1^*$, then we remove all but two of these edges, namely vv_1, vv_2 from E_1^*. Thus, there exists a pseudo-edge v_1v_2 in $\tilde{E}$ of colour c_v. We add pseudo-edge v_1v_2 to P'.

For every $u, v \in K_{2,3}$ such that $uv \in E$, let E_{uv} be the set of edges of E^* that have at least one endpoint in $\{u, v\}$. We have one of the following cases:

- If $uv \in E^*$; We distinguish two sub-cases:
  Case: there exists three edges of E_{uv} that forms a path $R = u'uvv'$, then we remove all the edges of $E_{uv} \setminus R$. We add the pseudo-edge $u'v' \in \tilde{E}$ with colour c_{uv} to P'.

  Case: Assume there is no such path in E_{uv}. Recall OPT is a feasible solution, and, by definition of $K_{2,3}$ at least one of u or v does not have a safe neighbor in $V(D)$. Then, (w.l.o.g.) u is not incident on any edge of $E_{uv} \setminus \{uv\}$. By feasibility of OPT, v must be safe and E_{uv} must have at least two edges vv_1 and vv_2 such that $v_1, v_2 \in V(D)$. We remove all edges of $E_{uv} \setminus \{uv, vv_1, vv_1\}$ from E_1^*. Finally we observe that by construction $\tilde{E}$ has an edge v_1v_2 with colour c_{uv}. We add this pseudo-edge to P'.
- If $uv \notin E^*$; Again we distinguish following sub-cases:
  Case: If one of u or v is incident on exactly one of the edges of E_{uv}, w.l.o.g. let uu' be that edge. Since OPT is a feasible FVC solution, u' must be safe. Furthermore as $uv \in E$ and $u, v \in K_{2,3}$, then the other endpoint of the edges of E_{uv} that are incident on v must be unsafe. Hence there are at least two edges incident on v. We fix edges $vv_1, vv_2 \in E_{uv}$. Now we remove all the edges of E_{uv} from E_1^*

except for $\{uu', v v_1 v v_2\}$. Finally observe that by construction $\tilde{E}$ has the pseudo-edge $v_1 v_2$ with colour c_{uv}, which we add to P'.

Case: Now assume each of u and v is incident to at least two edges of E_{uv}. If there is a path from u to v that only contains edges of E_1^*, let uu_1 and vv_1 be the first and last edge of this path. We update E_1^* by removing all the edges of E_{uv} from it except for two edge $uu_2, v v_2 \in E_{uv}$ such that $u_2 \neq u_1$ and $v_2 \neq u_2$. Then we add uv to E_1^*. Note that by this operation we decrease the number of edges of E_1^* by $|E_{uv}|-3$ units and the number of connected components of (V, E_1^*) increases by at most $|E_{uv}|-3$. Furthermore we have a pseudo-edge $u_2 v_2$ of colour c_{uv} in $\tilde{E}$ by construction, we add $u_2 v_2$ to P'.

If such a path does not exist then we update E_1^* by removing all the edges of E_{uv} from it except for the two edges $uu_1, vv_1 \in E_{uv}$. Then we add the edge uv to E_1^*. Observe that $|E_1^*|$ decreased by $|E_{uv}|-3$ and the number of connected components of (V, E_1^*) increases by at most $|E_{uv}|-3$. This time we add the pseudo-edge $u_1 v_1$ of colour c_{uv} to P'.

Observe that by construction $|E_1^*| = 2|K_{1,2}| + \frac{3}{2}|K_{2,3}|$. Furthermore by the above argument:

$$\kappa(E_1^*) - \kappa(E^*) \leq |E^*| - |E_1^*|.$$

Furthermore, by construction for every pseudo-edge $p = xy \in P'$ of colour c_v, the edges vx and vy are in E_1^*, and for every pseudo-edge $p = xy \in P'$ of colour c_{uv}, there exists a path consisting only of edges of E_1^* from x to y with inner vertices in $\{u, v\}$. Therefore, any two vertices in D that are in the same connected component of (V, E_1^*) are also in the same connected component of (V, P'). Moreover any vertex $v \in K_{1,2} \cup K_{2,3}$ has a path to at least one vertex of (V, E_1^*), therefore:

$$\kappa(P') = \kappa(E_1^*) - (|K_{1,2}| + |K_{1,2}|).$$

Altogether, we have:

$$\begin{aligned}
&\kappa(E^*) \geq \kappa(E_1^*) - |E^*| + |E_1^*| \geq \kappa(P') - (|K_{1,2}| + |K_{1,2}|) - |E^*| + |E_1^*| \\
=&\kappa(P') + (|K_{1,2}| + \frac{1}{2}|K_{1,2}|) - |E^*| \\
\geq&\alpha + |K_{1,1}| + |K_{1,2}| + |K_{2,2}| + |K_{2,3}| + (|K_{1,2}| + \frac{1}{2}|K_{1,2}|) - |E^*|.
\end{aligned}$$

□

By Claim 6.5, as OPT is connected, then

$$|OPT| - |E^*| \geq \alpha + 2K_{1,2} + \frac{3}{2}K_{2,3} - |E^*| + K_{2,2} + K_{1,1} - 1,$$
$$\Rightarrow |OPT| \geq \alpha + 2K_{1,2} + \frac{3}{2}K_{2,3} + K_{2,2} + K_{1,1} - 1.$$

- $|OPT| \geq n$ follows by our assumption from applying Lemma 6.8.
- It remains to show that $|OPT| \geq 2K_{1,2} - 2\alpha_{large} + \alpha'_1 + 2\alpha'_2$.

  Recall that in Algorithm 7, we compute a subset of singletons in $(V(D), P)$, which we denote by X_3, and that $|X_3| = \alpha'_1 + \alpha'_2$. Let $K'_{1,2} \subseteq K_{1,2}$ be subset of all vertices in $K_{1,2}$ that have at least one neighbor in X_3. Consider $I := K_{1,2} \cup X_3 \setminus K'_{1,2}$. As X_3 and $K_{1,2}$ are independent sets, by definition of $K'_{1,2}$, we see that I is independent.

  In any feasible solution, for any $v \in I$, v either has a safe neighbor, or at least two neighbors. Note, there are precisely α'_1 vertices in I that have a safe neighbor in G, since no vertices of $K_{1,2}$ are adjacent to safe vertices, by definition. Thus, we see $|OPT| \geq 2|I| - \alpha'_1 = 2|K_{1,2}| + \alpha'_1 + 2\alpha'_2 - 2|K'_{1,2}|$.

  Thus it suffices to prove that $|K'_{1,2}| \leq \alpha_{large}$. If $|K'_{1,2}| = 0$, we are done. We assume that $|K'_{1,2}| > 0$. Consider a vertex $v' \in K'_{1,2}$ with neighbour $u \in X_3$. Let P be the set of pseudo-edges computed by Algorithm 5. In $(V(D), P)$, the degree of u is zero (recall that $X_1 \cup X_2 \cup X_3$ were isolated vertices in $(V(D), P)$).

  Note that P has a pseudo-edge xy with colour $c_{v'}$. Observe that as x and y are neighbors of v', then we have the option to replace pseudo-edge xy with either xu or uy (as both will exist since pseudo-edges whose colour is represented by a vertex in $K_{1,2}$ form a clique). Note that this would not change $\kappa(P)$, however if x (or y) has degree two or more in P, then replacing xy with uy (or ux) decreases the number of singletons, a contradiction to the definition of P due to Lemma 6.15. Therefore as such a swap is not possible, then x and y are of degree one in $(V(D), P)$ (i.e. x and y are only incident on the edge xy in P). Therefore x and y form a component of size 2 in P that is not a singleton (and hence large). We associate this component to v', as the unique pseudo-edge of this component has color $c_{v'}$. Therefore every vertex in $K'_{1,2}$ has a unique large connected component of P associated to it and thus $|K'_{1,2}| \leq \alpha_{large}$.

□

Lemma 6.20. *It holds that*

$$\frac{\min\{\frac{4}{3}(|V(D)|-1)+|S_P|, |V(D)|-2+|S_P|+\alpha-(\frac{\alpha-\alpha'}{2}+\frac{\alpha_{large}}{2}+\alpha'_1)\}}{\max\{|S_P|+\alpha-1, 2K_{1,2}-2\alpha_{large}+\alpha'_1+2\alpha'_2, n\}} \leq \frac{11}{7}.$$

Proof. The following inequality will be a useful tool for proving our approximation factor.

$$S := \frac{\min\{\frac{4|V(D)|-4}{3}+|S_P|, |S_P|+|V(D)|-2+\frac{3\alpha}{4}\}}{\max\{|S_P|+\alpha-1, |V(D)|+|K|\}} \leq \frac{11}{7}, \tag{6.1}$$

where $\alpha, |V(D)|, |K|$ and $|S_P|$ are real numbers such that $0 \leq \alpha \leq |V(D)|$, $1 \leq |K| \leq |S_P| \leq 2|K|$.

First suppose that $|S_P| \leq \frac{11}{7}|K|$, then $\frac{\frac{4|V(D)|-3}{3}+|S_P|}{|K|+|V(D)|} \leq \frac{11}{7}$ and the claim holds.

Therefore, we can assume that $|S_P| > \frac{11}{7}|K|$. We consider cases for the values of $|V(D)|, |K|, |S_P|, \alpha$ maximizing S. For simplicity we define the following $UB_1 := \frac{4|V(D)|-4}{3}+|S_P|$, $UB_2 := |S_P|+|V(D)|-3+\frac{3\alpha}{4}$, $LB_1 := |S_P|+\alpha-1$, and $LB_2 := |V(D)|+|K|$. Let $r := |S_P|/|K|$. Observe that $\frac{11}{7} < r \leq 2$.

1. **Case:** First consider the case that $LB_1 = LB_2$ and $UB_1 = UB_2$. Since $UB_1 = UB_2$, we can conclude that $|V(D)| = 2+\frac{9}{4}\alpha$. Since $LB_1 = LB_2$, we can conclude that $|K| = \frac{5\alpha+12}{4(r-1)}$. Therefore,

$$\begin{aligned} S = \frac{UB_1}{LB_2} &= \frac{\frac{4}{3}(\frac{9}{4}\alpha+1)+r(\frac{5\alpha+12}{4(r-1)})}{2+\frac{9}{4}\alpha+\frac{5\alpha+12}{4(r-1)}} \leq \max\left\{\frac{3\alpha+\frac{5\alpha r}{4(r-1)}}{\frac{9}{4}\alpha+\frac{5\alpha}{4(r-1)}}, \frac{\frac{4}{3}+\frac{3}{r-1}}{2+\frac{3}{r-1}}\right\} \\ &\leq \max\left\{\frac{12(r-1)+5r}{9(r-1)+5}, \frac{\frac{4}{3}(r-1)+3r}{2r+1}\right\} = \max\left\{\frac{17r-12}{9r-4}, \frac{\frac{13}{3}r-\frac{4}{3}}{2r+1}\right\} \\ &\leq \max\left\{\frac{34-12}{18-4}, \frac{\frac{26}{3}-\frac{4}{3}}{5}\right\} \leq \frac{11}{7}, \end{aligned}$$

   where the second to last inequality follows as $\frac{7}{11} \leq r \leq 2$, and both $\frac{17r-12}{9r-4}$ and $\frac{\frac{13}{3}r-\frac{4}{3}}{2r+1}$ are increasing functions in terms of r.

2. **Case:** If $\alpha = 0$, then:

$$S = \frac{\min\{\frac{4|V(D)|-4}{3}+|S_P|, |S_P|+|V(D)|-2\}}{\max\{|S_P|-1, |V(D)|+|K|\}} < \frac{|V(D)|+|S_P|-1}{\max\{|S_P|-1, |V(D)|+|K|\}},$$

Thus, by applying the definition of r we can see

$$S < \frac{|V(D)| + r|K| - 1}{\max\{r|K| - 1, |V(D)| + |K|\}} \leq \frac{|V(D)| + r|K| - 1}{\frac{4r}{11}(|K| - 1) + \frac{7}{11}(|V(D)| + |K|)}$$

$$= \frac{|V(D)| + r|K| - 1}{\frac{7}{11}|V(D)| + (\frac{4r}{11} + 1)|K| - \frac{4}{11}} \leq \frac{7}{11}.$$

Again, the last inequality follows as $\frac{7}{11} \leq r \leq 2$.

3. **Case:** If $\alpha = |V(D)|$, then:

$$S = \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \frac{3|V(D)|}{4}\}}{\max\{|S_P| + |V(D)| - 1, |V(D)| + |K|\}}$$
$$\leq \frac{\frac{4|V(D)|-4}{3} + |S_P|}{|S_P| + |V(D)| - 1} \leq \frac{4}{3}.$$

4. **Case:** If $LB_1 = LB_2$ but $UB_1 \neq APX$. Assume for the sake of contradiction that $S > \frac{11}{7}$, and consider cases for $UB_1 > UB_2$ or not.

   Case 4.1) if $UB_1 > UB_2$. We define $\Delta := (UB_1 - UB_2)/(\frac{7}{4} - \frac{4}{3}) = \frac{12(UB_1 - UB_2))}{5}$, $D' := |V(D)| + \Delta$, and $\alpha' := \alpha + \Delta$. Define

$$S' := \frac{\min\{\frac{4D'-4}{3} + |S_P|, |S_P| + D' - 2 + \frac{3\alpha'}{4}\}}{\max\{|S_P| + \alpha' - 1, D' + |K|\}}$$
$$= \frac{\min\{\frac{4D-4}{3} + |S_P| + \frac{4\Delta}{3}, |S_P| + D - 2 + \frac{3\alpha}{4} + \frac{7\Delta}{4}\}}{\max\{LB_1 + \Delta, LB_2 + \Delta\}}$$
$$= \frac{\min\{UB_1 + \frac{4}{3}\Delta, UB_2 + \frac{7}{4}\Delta\}}{\max\{LB_1 + \Delta, LB_2 + \Delta\}}.$$

   By definition of Δ, we have $UB_1 + \frac{4}{3}\Delta = UB_2 + \frac{7}{4}\Delta$, and by the assumption of Case (4) we have $LB_1 + \Delta = LB_2 + \Delta$. Therefore, we can apply Case (1) to see that $S' \leq \frac{11}{7}$.

$$\frac{11}{7} \geq S' = \frac{\min\{UB_1 + \frac{4}{3}\Delta, UB_2 + \frac{7}{4}\Delta\}}{\max\{LB_1 + \Delta, LB_2 + \Delta\}} = \frac{UB_2 + \frac{7}{4}\Delta}{LB_2 + \Delta} \geq \min\left\{\frac{UB_2}{LB_2}, \frac{\frac{7}{4}\Delta}{\Delta}\right\}$$
$$= \min\left\{S, \frac{7}{4}\right\} > \frac{11}{7}.$$

And thus we find a contradiction.

Case 4.2) if $UB_1 < UB_2$. We define $\Delta := \min\{(UB_2 - UB_1)/(\frac{7}{4} - \frac{4}{3}), \alpha\}$, $D' := |V(D)| - \Delta$, and $\alpha' := \alpha - \Delta$. Now define

$$S' := \frac{\min\{\frac{4D'-4}{3} + |S_P|, |S_P| + D' - 2 + \frac{3\alpha'}{4}\}}{\max\{|S_P| + \alpha' - 1, D' + |K|\}}.$$

If $\alpha \leq (UB_2 - UB_1)/(\frac{7}{4} - \frac{4}{3})$, then $\Delta = \alpha$, then S' satisfies the conditions for Case (2), so $S' \leq \frac{11}{7}$. Now suppose that $\alpha > (UB_2 - UB_1)/(\frac{7}{4} - \frac{4}{3})$.

$$= \frac{\min\{\frac{4D-4}{3} + |S_P| - \frac{4\Delta}{3}, |S_P| + D - 2 + \frac{3\alpha}{4} - \frac{7\Delta}{4}\}}{\max\{LB_1 - \Delta, LB_2 - \Delta\}}$$
$$= \frac{\min\{UB_1 - \frac{4}{3}\Delta, UB_2 - \frac{7}{4}\Delta\}}{\max\{LB_1 - \Delta, LB_2 - \Delta\}}.$$

By construction, we have $UB_1 - \frac{4}{3}\Delta = UB_2 - \frac{7}{4}\Delta$. Thus, by Case (1), we have that $S' \leq \frac{11}{7}$. By a similar argument to Case (4), we have $\frac{11}{7} < S \leq S' \leq \frac{11}{7}$, which is a contradiction.

5. **Case:** If $LB_1 < LB_2$. Define $\alpha' = \alpha + \max\{|V(D)| - \alpha, LB_2 - LB_1\}$. Define

$$S' = \frac{\min\{UB_1, |S_P| + |V(D)| - 2 + \frac{3\alpha'}{4}\}}{\max\{|S_P| + \alpha' - 1, LB_2\}}.$$

If $|V(D)| - \alpha \leq LB_2 - LB_1$, then $S' = \frac{\min\{UB_1, |S_P| + |V(D)| - 2 + \frac{3\alpha'}{4}\}}{\max\{LB_1 + LB_2 - LB_1, LB_2\}}$. So S' satisfies the conditions of Case (4), and $S' \leq \frac{11}{7}$. However, by construction we have that $\frac{11}{7} < S \leq S' \leq \frac{11}{7}$. A contradiction.

If $|V(D)| - \alpha > LB_2 - LB_1$, then $S' = \frac{\min\{UB_1, |S_P| + |V(D)| - 2 + \frac{3|V(D)|}{4}\}}{\max\{|S_P| + |V(D)| - 1, LB_2\}}$. In this case, we have $\alpha = |V(D)|$, so S' satisfies the conditions of Case (3), so $S' \leq \frac{11}{7}$. Once again, we have $S \leq S'$, which is a contradiction.

6. **Case:** If $LB_1 > LB_2$. We define $D' = |V(D)| + LB_1 - LB_2$, and we define $S' := \frac{\min\{\frac{4D'-4}{3} + |S_P|, |S_P| + D' - 2 + \frac{3\alpha}{4}\}}{\max\{LB_1, D' + |K|\}} = \frac{\min\{\frac{4D'-4}{3} + |S_P|, |S_P| + D' - 2 + \frac{3\alpha}{4}\}}{\max\{LB_1, LB_2 + LB_1 - LB_2\}}$. S' satisfies the conditions of Case (4), and clearly $S' \geq S$, thus we have $\frac{11}{7} < S \leq S' \leq \frac{11}{7}$.

Using inequality 6.1, we can prove the following inequality.

$$S := \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \alpha - x\}}{\max\{|S_P| + \alpha - 1, |V(D)| + |K|, 2K_{1,2} + 2\alpha - 4x\}} \leq \frac{11}{7}, \tag{6.2}$$

where $\alpha, |V(D)|, |K|$ and $|S_P|$ are real numbers such that $0 \leq x \leq \alpha \leq |V(D)|$, and $1 \leq |K| \leq |S_P| \leq 2|K|$.

For notational simplicity we define the following terms $UB_1 = \frac{4|V(D)|-4}{3} + |S_P|$, $UB_2 = |S_P| + |V(D)| - 2 + \alpha - x$, $LB_1 = |S_P| + \alpha - 1$, $LB_2 = |V(D)| + |K|$, and $LB_3 = 2K_{1,2} + 2\alpha - 4x$.

1. **Case:** if $x \geq \frac{\alpha}{4}$, then

$$S \leq \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \frac{3}{4}\alpha\}}{\max\{|S_P| + \alpha - 1, |V(D)| + |K|\}} \leq \frac{11}{7}.$$

   Where the first inequality follows by applying the Case assumption and the fact that $\max\{LB_1, LB_2, LB_3\} \geq \max\{LB_1, LB_2\}$ and the second inequality holds using inequality 6.1.

2. **Case:** If $UB_2 \leq UB_1$. Define $\Delta := \frac{\alpha - 4x}{5} > 0$, $x' = x + \Delta$, and $\alpha' = \alpha - \Delta$. Thus, $x' = \frac{\alpha'}{4}$. Furthermore, we can define

$$\begin{aligned} S' &:= \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \frac{3}{4}\alpha'\}}{\max\{|S_P| + \alpha' - 1, |V(D)| + |K|, 2K_{1,2} + \alpha'\}} \\ &\leq \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \frac{3}{4}\alpha'\}}{\max\ |S_P| + \alpha' - 1, |V(D)| + |K|\}} \leq \frac{11}{7}. \end{aligned}$$

   Where the second inequality follows by application of inequality 6.1.

   Note that $|S_P| + \alpha' - 1 \leq LB_1$, $|V(D)| + |K| \leq LB_2$, $2K_{1,2} + 2\alpha' - 4x' \leq LB_3$, and $UB_2 = |S_P| + |V(D)| - 2 + \alpha - \Delta - x + \Delta = |S_P| + |V(D)| - 3 + \alpha' - x'$. Thus, we can see that $S \leq S' \leq \frac{11}{7}$.

3. **Case:** If $UB_2 > UB_1$. Define $\Delta := \min\{\alpha - 4x, UB_2 - UB_1\}$, and $\alpha' = \alpha - \Delta$ and

$$S' := \frac{\min\{\frac{4|V(D)|-4}{3} + |S_P|, |S_P| + |V(D)| - 2 + \alpha' - x\}}{\max\{|S_P| + \alpha' - 1, |V(D)| + |K|, 2K_{1,2} + 2\alpha' - 4x\}}.$$

   First observe that as $\max|S_P| + \alpha' - 1, |V(D)| + |K|, 2K_{1,2} + 2\alpha' - 4x\} \leq \max\{|S_P| + \alpha - 1, |V(D)| + |K|, 2K_{1,2} + 2\alpha - 4x\}$ and $\frac{4|V(D)|-4}{3} + |S_P| \leq |S_P| + |V(D)| - 2 + \alpha' - x < UB_2$ then $S' \geq S$. Thus it is sufficient to show that $S' \leq \frac{11}{7}$. We consider two sub-cases. If $\alpha - 4x \leq UB_2 - UB_1$. In which case, $\alpha' = \alpha - \alpha + 4x = 4x$, by applying Case (1) S' one sees that $S' \leq \frac{11}{7}$.

   Else, $\alpha - 4x \geq UB_2 - UB_1$. Then we have that $\frac{4|V(D)|-4}{3} + |S_P| = |S_P| + |V(D)| - 3 + \alpha' - x$, and therefore, Case (2) shows that $S' \leq \frac{11}{7}$.

Lastly, we have the following useful claim.

Claim 6.6. *Define* $x := \frac{\alpha-\alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1$. *Then* $\alpha'_1 + 2\alpha'_2 - 2\alpha_{large} \geq 2\alpha - 4x$

Proof. $2\alpha - 4x = 2\alpha - 4(\frac{\alpha-\alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1) = 2\alpha' - 4\alpha'_1 - \alpha_{large}$
$= 2\alpha'_2 - 2\alpha'_1 - 2\alpha_{large} \leq \alpha'_1 + 2\alpha'_2 - 2\alpha_{large}$ □

We have the ingredients necessary to complete the proof the lemma

To simplify notation, we define $K := K_{1,1} \cup K_{1,2} \cup K_{2,2} \cup K_{2,3}$. Observe that $|K| \leq |S_P| = |K_{1,1}| + 2|K_{1,2}| + 2|K_{2,2}| + \frac{3}{2}|K_{2,3}| \leq 2(|K_{1,1}| + |K_{1,2}| + |K_{2,2}| + |K_{2,3}|) = 2|K|$.

Define $x := \frac{\alpha-\alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1$. Then, by using Claim 6.6 we have $\alpha'_1 + 2\alpha'_2 - 2\alpha_{large} \geq 2\alpha - 4x$. Noting that $n = |V(D)| + |K|$, we have $\frac{\min\{APX_1, APX_2\}}{|OPT|}$ is at most

$$\frac{\min\{\frac{4}{3}(|V(D)|-1) + |S_P|, |V(D)| - 2 + |S_P| + \alpha - x)\}}{\max\{|S_P| + \alpha - 1, 2K_{1,2} + 2\alpha - 4x, |V(D)| + |K|\}}.$$

And by inequality 6.2 this is at most $\frac{11}{7}$. □

With Lemma 6.19, Lemma 6.12, Lemma 6.20, and Lemma 6.18 we have the tools necessary to prove Theorem 6.2.

Proof of Theorem 6.2. Given instance of $(1,1)$-FVC, $G = (V_S \cup V_U, E)$. We apply Lemma 6.7 to assume without loss of generality that G does not contain any forbidden cycles and G is 2VC. We first find solution APX_1, and vertex sets D, $K_{1,1}, K_{1,2}, K_{2,2}$, and $K_{2,3}$. By Lemma 6.11, APX_1 is a feasible solution that we obtain in polynomial time. By Lemma 6.12, we have $|APX_1| \leq \frac{4}{3}(|V(D)| - 1) + |K_{1,1}| + 2|K_{1,2}| + |K_{2,2}| + \frac{3}{2}|K_{2,3}| = \frac{4}{3}(|V(D)| - 1) + |S_P|$.

Using sets $V(D)$, $K_{1,1}, K_{1,2}, K_{2,2}$, and $K_{2,3}$ we apply Lemma 6.15 to compute a set of pseudo-edges P on $V(D)$ with α many components and α_{large} many large components (at least 2 vertices). We then apply Algorithms 5, 6, and 7 as described in Section 6.1.2 to compute edge sets S_1, S_2, and S_3, as well as α'_1 and α'_2, where $\alpha' = \alpha'_1 + \alpha'_2$. We then find edge set S_P by replacing pseudo-edges with corresponding edges, and let $APX_2 = S_P \cup S_1 \cup S_2 \cup S_3$. By Lemma 6.18 computing APX_2 in this way takes polynomial time and $|APX_2| \leq |V(D)| - 1 + |S_P| + \alpha - 1 - (\frac{\alpha-\alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1)$.

By Lemma 6.19, we have $|OPT| \geq \max\{|S_P| + \alpha - 1, 2K_{1,2} - 2\alpha_{large} + \alpha'_1 + 2\alpha'_2, n\}$. Therefore, by Lemma 6.20, we have $\frac{\min\{APX_1, APX_2\}}{|OPT|}$ is at most:

$$\frac{\min\{\frac{4}{3}(|V(D)|-1) + |S_P|, |V(D)| - 2 + |S_P| + \alpha - (\frac{\alpha-\alpha'}{2} + \frac{\alpha_{large}}{2} + \alpha'_1)\}}{\max\{|S_P| + \alpha - 1, 2K_{1,2} - 2\alpha_{large} + \alpha'_1 + 2\alpha'_2, n\}} \leq \frac{11}{7}.$$

□

6.2 FGC Improvement

The goal of this section is to prove Theorem 6.1. We use the algorithm described by [14], and provide a tighter analysis. In particular, their algorithm combines two different algorithms, and takes the best solution output among the two. Our improvement is based on a better analysis of the second one.

The lemma below comes from [14], in particular using their first algorithm. The authors prove the following (see Claim 6.4 of [14]).

Lemma 6.21 ([14]). *Given a FGC instance with optimal solution OPT, one can compute in polynomial time a solution F_1 with $|F_1| \leq |OPT \cap E_S| + \frac{3}{2}|OPT \cap E_U|$.*

The second algorithm used in [14] needs to be described fully, as we modify its analysis.

Algorithm 2. Given a FGC instance defined on a graph G, consider the graph G'' obtained from G by duplicating every safe edge in E. Run an β-approximation algorithm for the 2ECSS problem on G'', and let F_2 be the output. Drop extra copies of safe edges from F_2.

The authors prove the following claim (see Claim 6.5 of [14]).

Lemma 6.22 ([14]). *We have $|F_2| \leq 2\beta|OPT \cap E_S| + \beta|OPT \cap E_U|$.*

In order to improve this algorithm we show that one can find a better approximation for 2ECSS if the size of the optimal solution is far enough from n, the number of vertices in G:

Lemma 6.23. *Let G be a 2ECSS instance. One can find in polynomial time a solution APX of size $\frac{4}{3}n + \frac{2}{3}(x-1)$, where $x = |OPT| - n$.*

Proof. If G is 2VC then one can solve the problem using Lemma 6.24. So assume V is not $2VC$. Then in polynomial time we can decompose it into blocks $E_1, ..., E_b$ (see [86] for further details). Let $G_1, ..., G_b$ be the induced subgraph on E_i (i.e. $G_i := G[E_i]$). Furthermore let n_i be the number of vertices of G_i and opt_i be the size of the optimal 2ECSS of G_i. We have:

$$\sum_{i=1}^{b} n_i = n + b - 1.$$

Now let $E' \subseteq E$. One observes that E' is a 2ECSS of G if and only if $E' \cap E_i$ is a 2ECSS of G_i for every $i \in \{1,...,b\}$. Therefore now we can use Lemma 6.24 for each i to get an approximate solution APX_i of size at most $\frac{4}{3}n_i + \frac{2}{3}(opt_i - n_i) - \frac{2}{3}$ for G_i. Thus if we set $APX := APX_1 \cup \cdots \cup APX_b$, we obtain a feasible solution for G of size:

$$|APX| \leq \frac{4}{3}(n+b-1) + \frac{2}{3}(opt-(n+b-1)) - \frac{2}{3}b = \frac{4}{3}n + \frac{2}{3}(opt-n) - \frac{2}{3}.$$

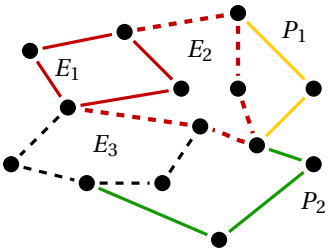
Proving the claim. □

Given Lemma 6.23, we here prove the following lemma, which assumes the instance is 2VC.

Lemma 6.24. *Let G be a 2ECSS instance that is 2VC. One can find in polynomial time a solution APX of size $\frac{4}{3}n + \frac{2}{3}(x-1)$, where $x = |OPT| - n$.*

Proof. We provide a refined analysis of the algorithm provided in [81]. The authors in [81] first find what they call a "nice" ear decomposition. To define it, let's introduce some terminology. The minimum number of ears of 1 one are called trivial, while ears of length 2 and 3 are called short. A vertex is pendant if it is not an endpoint of any non-trivial ear, and an ear is pendant if it is non-trivial and all its internal vertices are pendant. A nice ear decomposition is an ear decomposition with minimum number of even ears, in which there are no trivial ears, all short ears are pendant, and internal vertices of distinct short ears are non-adjacent. See Figure 6.7 for an example of a nice ear decomposition, and a clarification of short ears. Now consider the nice ear decomposition found in [81] and let π denote the number of short ears, π_i denote the number of ears of length i, and $\phi(G)$ denote the number of even length ears.

The authors define two algorithms and take the minimum output of the two. The first algorithm (see Section 5.3 of [81]) outputs a solution ALG_1 which satisfies $|ALG_1| \leq \frac{3}{2}|OPT| - \pi$. The second algorithm simply returns as a solution a nice ear decomposition (which they show exists for a 2VC graph, and can be computed efficiently). Let us call this solution ALG_2. We now provide a bound



(a)

Figure 6.7: Here we have an example of a "nice" ear decomposition, consisting of ears E_1, E_2, E_3, P_1 and P_2, each represented with a different colour or edge shape. Note that the only short ears P_1 and P_2 are open and "pendant". That is, the internal vertices are *only* on their respective ears.

on the size of ALG_2.

$$\begin{aligned}
|ALG_2| &= \sum_{i\geq 2} i\pi_i = 2\pi_2 + 3\pi_3 + 4\pi_4 + \sum_{i\geq 5} i\pi_i \\
&\leq (\frac{5}{4}+\frac{3}{4})\pi_2 + (\frac{5}{4}\cdot 2+\frac{1}{2})\pi_3 + (\frac{5}{4}\cdot 3+\frac{1}{4})\pi_4 + \sum_{i\geq 5}\frac{5}{4}(i-1)\pi_i \\
&\leq \frac{5}{4}(n-1) + \frac{3}{4}\pi_2 + \frac{1}{2}\pi_3 + \frac{1}{4}\pi_4 = \frac{5}{4}(n-1) + \frac{1}{4}(\pi_2+\pi_4) + \frac{1}{2}(\pi_3+\pi_2) \\
&\leq \frac{5}{4}(n-1) + \frac{1}{4}\phi(G) + \frac{1}{2}\pi.
\end{aligned}$$

The second to last inequality follows since an ear of length i has $i-1$ internal vertices, and every vertex of the graph but 1 is an internal vertex of exactly one ear. So the algorithm in [81] returns a solution of size $\leq \min\{|ALG_1|, |ALG_2|\} \leq \frac{1}{3}|ALG_1| + \frac{2}{3}|ALG_2|$ which is

$$\leq \frac{1}{3}\left(\frac{3}{2}OPT - \pi\right) + \frac{2}{3}\left(\frac{5}{4}(n-1) + \frac{1}{4}\phi(G) + \frac{1}{2}\pi\right) = \frac{1}{2}OPT + \frac{5}{6}(n-1) + \frac{1}{6}\phi(G).$$

To prove our claim it is enough to show that the latter term is bounded by $\frac{4}{3}n + \frac{2}{3}(x-1)$. For this, we need to employ Theorem 5 of [81] which states that

$n+\phi(G)-1 \leq |OPT|$, and hence $\phi(G)-1 \leq x$. We then get

$$\begin{aligned}&\frac{4}{3}n+\frac{2}{3}(x-1)-\left(\frac{1}{2}OPT+\frac{5}{6}(n-1)+\frac{1}{6}\phi(G)\right)\\ =&\frac{4}{3}n+\frac{2}{3}(x-1)-\left(\frac{4}{3}n+\frac{1}{2}x+\frac{1}{6}\phi(G)-\frac{5}{6}\right)=\frac{1}{6}(x-\phi(G)+1)\geq 0.\end{aligned}$$

□

We are now ready to prove Theorem 6.1.

Proof of Theorem 6.1. By Lemma 6.21, we have that $ALG_1 \leq |OPT \cap E_S| + \frac{3}{2}|OPT \cap E_U|$.

Let $opt(G'')$ be the size of an optimal solution for the 2ECSS instance G'' considered by Algorithm 2. It is important to observe that $opt(G'') \leq 2|OPT \cap E_S| + |OPT \cap E_U|$. Let us denote $OPT_S := OPT \cap E_S$ and $OPT_U := OPT \cap E_U$. Using Lemma 6.23, we can see that

$$\begin{aligned}|ALG_2| &\leq \frac{4}{3}n+\frac{2}{3}(opt(G'')-n-1) \leq \frac{4}{3}n+\frac{2}{3}(2|OPT_S|+|OPT_U|-n-1)\\ &=\frac{2}{3}(n-1)+\frac{4}{3}|OPT_S|+\frac{2}{3}|OPT_U| \leq (\frac{4}{3}+\frac{2}{3})|OPT_S|+\frac{4}{3}|OPT_U|.\end{aligned}$$

Thus our algorithm provides a solutions of size at most $\min\{|ALG_1|, |ALG_2|\}$ which is

$$\begin{aligned}&\leq \min\left\{|OPT_S|+\frac{3}{2}|OPT_U|, \left(\frac{4}{3}+\frac{2}{3}\right)|OPT_S|+\frac{4}{3}|OPT_U|\right\}\\ &\leq \frac{3}{7}(2|OPT_S|+\frac{4}{3}|OPT_U|)+\frac{4}{7}(|OPT_S|+\frac{3}{2}|OPT_U|)=\frac{10}{7}|OPT_S|+\frac{10}{7}|OPT_U|.\end{aligned}$$

□

6.3 $1+O(1/\sqrt{k})$-approximation for k-FGC

The goal of this section is to prove Theorem 6.3. k-FGC was first tackled in [2], where the authors claimed a $(1+O(\frac{1}{k}))$-approximation. Unfortunately, the analysis of their algorithm has a flaw, rendering the proof incorrect, that seems not salvageable. We discuss this in further detail in Section 6.4. We provide in this section an alternate analysis to a similar algorithm used in [2], finding a slightly larger approximation factor than the one claimed.

The algorithm employed here is simple. We compute a maximum forest F_s on (V, E_S). The connected components of (V, F_s) are contracted (remove loops but keep parallel edges) to find $G/F_s := G' = (V', E')$. Then, we run one of the known $1 + O(\frac{1}{k})$-approximation algorithms for $(k+1)$ECSS due to [27, 39] on G', finding $F \subseteq E'$. As long as F is not a minimal solution for the $(k+1)$ECSS instance defined on G', we find and remove an edge $e \in F$ such that $F \setminus \{e\}$ is still a solution. We then return ALG:$= F_s \cup F$ as our solution.

To prove Theorem 6.3, we need the following lemmas. The first one relies on the fact that a minimal kECSS solution can be partitioned into k forests. As usual, n denotes $|V|$.

Lemma 6.25 ([39])**.** *For a minimal solution F to a given kECSS instance G, $|F| \leq nk$.*

In every k-edge-connected graph the degree of each vertex is at least k, hence:

Lemma 6.26. *For an optimal solution OPT of a kECSS instance G, $\frac{nk}{2} \leq |OPT|$.*

The following lemma is proven in [2].

Lemma 6.27 ([2])**.** *Let $H \subseteq G$ be a feasible solution to a kFGC instance. Then $H/(E_s \cap E(H))$ is $(k+1)$-edge-connected.*

We are now ready to prove Theorem 6.3.

Proof of Theorem 6.3. Fix an optimal solution OPT, which we decompose as $OPT = \cup OPT_U$, where $OPT_S \subseteq E_S$, and $OPT_U \subseteq E_U$. Let β_{k+1} be the best approximation factor for $(k+1)$ECSS. We distinguish two cases.

Case 1: $|F_s| \leq n\frac{\sqrt{k}+2}{\sqrt{k}+3}$. Note that $|ALG| \leq n\frac{\sqrt{k}+2}{\sqrt{k}+3} + \beta_{k+1}|OPT|$. Using Lemma 6.25 we have $|F| \leq (n - |F_s|)(k+1)$. This observation, together with Lemmas 6.26 and 6.27 and the fact that $|OPT_S| \leq |F_s|$ gives

$$\begin{aligned} |OPT| &\geq |OPT_S| + (n - |OPT_S|)\frac{k+1}{2} \\ &\geq |F_s| + (n - |F_s|)\frac{k+1}{2} = n\frac{(k+1)}{2} + \left(1 - \frac{k+1}{2}\right)|F_s|. \end{aligned}$$

By applying this inequality and the assumption of this case, we get

$$|OPT| \geq n\frac{(k+1)}{2} + \left(1 - \frac{k+1}{2}\right)|F_s| \geq n\left(\frac{k+1}{2} + \left(1 - \frac{k+1}{2}\right)\frac{\sqrt{k}+2}{\sqrt{k}+3}\right).$$

Where the second inequality follows since $1-\frac{k+1}{2} \leq 0$ for $k \geq 1$. Therefore, we have

$$|ALG| \leq n\frac{\sqrt{k}+2}{\sqrt{k}+3} + \beta_{k+1}|OPT| \leq \frac{|OPT|(\sqrt{k}+2)}{(\sqrt{k}+3)\left(\frac{k+1}{2}+\left(1-\frac{k+1}{2}\right)\frac{\sqrt{k}+2}{\sqrt{k}+3}\right)} + \beta_{k+1}|OPT|$$
$$= \frac{|OPT|2(\sqrt{k}+2)}{k+2\sqrt{k}+5} + \beta_{k+1}|OPT| < \left(\frac{2}{\sqrt{k}} + \beta_{k+1}\right)|OPT|.$$

Case 2: $|F_s| > n\frac{\sqrt{k}+2}{\sqrt{k}+3}$. We here use a tighter bound on ALG. Namely, $|ALG| \leq n\frac{\sqrt{k}+2}{\sqrt{k}+3} + \beta_{k+1}|OPT_U|$. This holds as OPT_U is a feasible solution to the kECSS instance we obtain contracting F_s: this is because F_s is a maximal forest, hence vertices of each component of (V, OPT_S) are a subset of vertices of some component of (V, F_S). Therefore

$$\frac{ALG}{OPT} \leq \frac{|F_s| + \beta_{k+1}|OPT_U|}{|OPT_S| + |OPT_U|} \leq \max\left\{\frac{|F_s|}{|OPT_S|}, \beta_{k+1}\right\}.$$

We need to bound $\frac{|F_s|}{|OPT_S|}$. By applying Lemmas 6.25 and 6.26 we see that

$$|F_s| + (n-|F_s|)(k+1) \geq |ALG| \geq |OPT| \geq |OPT_S| + (n-|OPT_S|)\frac{k+1}{2}$$
$$= n + (n-|OPT_S|)\frac{k-1}{2} \Rightarrow \frac{2k}{k-1}(n-|F_s|) \geq n - |OPT_S|.$$

Using this inequality with the fact that $\frac{2k}{k-1} \leq 4$ when $k \geq 2$, as well as the case assumption we find

$$n - |OPT_S| \leq \frac{2k}{k-1}(n-|F_s|) < n\frac{2k}{k-1}\left(1-\frac{\sqrt{k}+2}{\sqrt{k}+3}\right) \leq 4n\left(1-\frac{\sqrt{k}+2}{\sqrt{k}+3}\right).$$

Therefore, we have

$$|OPT_S| \geq n\left(1-4+\frac{4(\sqrt{k}+2)}{\sqrt{k}+3}\right) = n\left(1-\frac{4}{\sqrt{k}+3}\right).$$

The last term above is positive for all $k \geq 2$. Using the above inequality and the fact that $|F_s| < n$:

$$\frac{|F_s|}{|OPT_S|} < \frac{n}{n\left(1-\frac{4}{\sqrt{k}+3}\right)} = \frac{1}{1-\frac{4}{\sqrt{k}+3}} = \frac{\sqrt{k}+3}{\sqrt{k}-1} = 1 + \frac{4}{\sqrt{k}-1}.$$

Therefore, in this case we have

$$\frac{|ALG|}{|OPT|} \leq \max\left\{\frac{|F_s|}{|OPT_S|}, \beta_{k+1}\right\} \leq \max\left\{1 + O\left(\frac{1}{\sqrt{k}}\right), \beta_{k+1}\right\}.$$

Combining the inequalities from Case 1 and 2 we find,

$$\frac{|ALG|}{|OPT|} \leq \max\left\{\frac{2}{\sqrt{k}} + \beta_{k+1}, 1 + O\left(\frac{1}{\sqrt{k}}\right), \beta_{k+1}\right\} = 1 + O\left(\frac{1}{\sqrt{k}}\right).$$

Where the last inequality above follows by applying the algorithm in [27, 40], for which we have that $\beta_{k+1} \leq 1 + O(\frac{1}{k})$ and our claim follows. □

6.4 Previous Analysis of $k-$FGC

In [2] the authors address the problem of k-FGC with Theorem 3, which is proven in Section 2 of that paper. We now discuss the relevant elements of the proof of Theorem 3 provided in [2]. The authors provide two approximation algorithms for the problem, and by taking the minimum, they find the desired approximation of $1 + O(\frac{1}{k})$. Here we focus on the second algorithm and its analysis, and we will point out an inequality that is incorrect. We provide here a close depiction of the proof found in [2] in order it identify this issue.

In the construction the authors of [2] provide, they let $I' = (G, F, k)$ be an instance of unweighted k-FGC, where G is the input graph, and F is the set of all safe edges of G. Their approximation algorithm proceeds as follows. First, we compute an edge-set X that forms a maximum forest restricted to the safe edges F of G. Then we compute a $(k+1)$-edge connected spanning subgraph Y' of G/X using the algorithm given in [27].Clearly, the algorithm above runs in polynomial time.

Lemma 6.28. *Let $H \subseteq G$ be a feasible solution to I'. Then $H/(F \cap E(H))$ is $(k+1)$-edge connected.*

Let $Z^* \subseteq G$ be an optimal solution to I' and let $X \cup Y'$ be a solution computed by the algorithm described above. To prove Theorem 3 it remains to bound the size of the solution $X \cup Y'$. Let $\ell := |X|$ and let Y^* be a minimum $(k+1)$-edge connected spanning subgraph of G/X. By Lemma 1, we have that $X \cup Y'$, and $X \cup Y^*$ are feasible solutions to I'.

Recall from Section 1.3 that the optimal solution Z^* consists of 2-edge-connected components that are joined by safe edges $E' \subseteq Z^* \cap F$ in a tree-like

fashion. The maximum forest X also joins these 2-edge-connected components of Z^*, that is, $(Z^* \setminus E') \cup X$ forms a connected graph. Hence, $Z^* \setminus X$ forms a 2-edge connected graph of G/X. Therefore we have that $\text{OPT}(I') = |Z^*| \geq |Y^*|$.

On the other hand, it was shown in [68] that Y' can be partitioned into k spanning forests. Thus Y' contains at most $(k+1)n'$ edges, where $n' := |V(G/X)| = n - \ell$. By Lemma 6.26, we have that any feasible solution to a $(k+1)$ECSS instance on n' vertices needs to contain at least $\frac{k+1}{2}n'$ edges, since each vertex has to have degree at least $k+1$. Therefore, we have that

$$|X| + |Y'| \leq \ell + (k+1)(n-\ell) \leq 2\text{OPT}(I') - k\ell,$$

where $n = |V(G)|$.

Now we show that the second inequality does not necessarily hold. In fact $2\text{OPT}(I') - k\ell$ can potentially be much smaller than $\ell + (k+1)(n-\ell)$. For instance, consider the case that G has a spanning tree containing solely of safe edges and $k \geq 3$. In this case $\text{OPT}(I') = |X| = n - 1$. Therefore $2\text{OPT}(I') - k\ell = (2-k)\ell < 0$. However $\ell + (k+1)(n-\ell) = \ell + (k+1) \geq n$. Thus in this case:

$$\ell + (k+1)(n-\ell) \geq n > 0 > 2\text{OPT}(I') - k\ell.$$

The correct inequality should be:

$$|X| + |Y'| \leq \ell + (k+1)(n-\ell) \leq 2\text{OPT}(I') - \ell,$$

where the second inequality holds by applying Lemma 6.28 to see that $|X| + \frac{(n-|X|)k}{2} \leq \text{OPT}(I')$ and the fact that every $(k+1)$-edge connected subgraph of G/X must have at least $\frac{(n-|X|)}{2}$ edges.

As previously mentioned, the authors of [2] provide a second approximation algorithm. Which simply computes the best $(k+1)$ECSS solution Y instead of Y'.

Lemma 6.29 ([2]). *There is a polynomial time algorithm that has cost at most* $\ell + \beta_{k+1} OPT(I')$.

Taken together, we find an approximation algorithm with approximation factor

$$\begin{aligned}
&\min\{\ell + \beta_{k+1} OPT(I'), 2OPT(I') - 2\ell\} \\
&\leq \frac{1}{2}(\ell + \beta_{k+1} OPT(I')) + \frac{1}{2}(2OPT(I') - \ell) = \frac{1}{2}(\beta_{k+1} + 2) OPT(I') \\
&= \left(\frac{3}{2} + O\left(\frac{1}{k}\right)\right) OPT(I').
\end{aligned}$$

Bibliography

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

[2] David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible graph connectivity: Approximating network design problems between 1- and 2-connectivity. *Mathematical Programming*, 192(1-2):409–441, 2022. (Cited on pages 13, 153, 154, 156, 195, 196, 198, and 199.)

[3] David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. Fault-Tolerant Edge-Disjoint s-t Paths - Beyond Uniform Faults. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on page 154.)

[4] Haris Angelidakis, Dylan Hyatt-Denesik, and Laura Sanità. Node connectivity augmentation via iterative randomized rounding. *Mathematical Programming*, pages 1–37, 2022. (Cited on pages 14, 112, and 216.)

[5] Vincenzo Auletta, Yefim Dinitz, Zeev Nutov, and Domenico Parente. A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph. *Journal of Algorithms*, 32(1):21–30, 1999. (Cited on pages 26 and 112.)

[6] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012. (Cited on pages 15, 18, 20, 21, and 22.)

[7] Ishan Bansal. A constant factor approximation for the (p, 3)-flexible graph connectivity problem. *arXiv preprint arXiv:2308.15714*, 2023. (Cited on page 154.)

[8] Ishan Bansal, Joseph Cheriyan, Logan Grout, and Sharat Ibrahimpur. Extensions of the (p,q)-flexible-graph-connectivity model. *arXiv preprint arXiv:2211.09747*, 2022. (Cited on pages 13, 153, and 154.)

[9] Manu Basavaraju, Fedor V. Fomin, Petr A. Golovach, Pranabendu Misra, M. S. Ramanujan, and Saket Saurabh. Parameterized algorithms to preserve connectivity. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 800–811, 2014. (Cited on pages 26, 30, and 31.)

[10] Matthias Bentert, Jannik Schestag, and Frank Sommer. On the Complexity of Finding a Sparse Connected Spanning Subgraph in a Non-Uniform Failure Model. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on page 154.)

[11] Paul R Berman, AD Scott, and M Karpinski. Approximation hardness and satisfiability of bounded occurrence instances of sat. Technical report, SIS-2003-269, 2003. (Cited on pages 112 and 144.)

[12] Marshall Bern and Paul Plassmann. The steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989. (Cited on page 91.)

[13] Al Borchers and Ding-Zhu Du. The k-steiner ratio in graphs. *SIAM J. Comput.*, 26(3):857–869, 1997. (Cited on pages 32, 45, and 47.)

[14] Sylvia Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. Approximation algorithms for flexible graph connectivity. *Mathematical*

Programming, pages 1–24, 2023. (Cited on pages 13, 153, 154, 155, and 192.)

[15] Jaroslaw Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to Steiner tree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 815–825, 2020. (Cited on pages 26, 27, 28, 30, 31, 33, and 62.)

[16] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013. (Cited on pages 11, 12, 27, 28, 32, 34, 35, 36, 62, 91, 92, 211, and 212.)

[17] Kristóf Bérczi and Yusuke Kobayashi. An algorithm for $(n-3)$-connectivity augmentation problem: Jump system approach. *Journal of Combinatorial Theory, Series B*, 102(3):565–587, 2012. (Cited on pages 111, 112, 113, 114, and 143.)

[18] Kristóf Bérczi and László A. Végh. Restricted b-matchings in degree-bounded graphs. In *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, volume 6080 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2010. (Cited on pages 113 and 114.)

[19] 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 (STOC)*, pages 370–383. ACM, 2021. (Cited on pages 12 and 26.)

[20] Federica Cecchetto, Vera Traub, and Rico Zenklusen. Better-than-4 3-approximations for leaf-to-leaf tree and connectivity augmentation. *Mathematical Programming*, pages 1–35, 2023. (Cited on page 26.)

[21] Deeparnab Chakrabarty, Jochen Könemannnemann, and David Pritchard. Hypergraphic lp relaxations for steiner trees. *SIAM Journal on Discrete Mathematics*, 27(1):507–533, 2013. (Cited on page 92.)

[22] Chandra Chekuri and Rhea Jain. Approximation algorithms for network design in non-uniform fault models. In *50th International Colloquium on*

Automata, Languages, and Programming (ICALP 2023). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023. (Cited on page 154.)

[23] Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part I: stemless TAP. *Algorithmica*, 80(2):530–559, 2018. (Cited on page 26.)

[24] Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part II. *Algorithmica*, 80(2):608–651, 2018. (Cited on page 26.)

[25] Joseph Cheriyan, Tibor Jordán, and R. Ravi. On 2-coverings and 2-packings of laminar families. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 510–520. Springer, 1999. (Cited on pages 10 and 25.)

[26] Joseph Cheriyan, Howard J. Karloff, Rohit Khandekar, and Jochen Könemann. On the integrality ratio for tree augmentation. *Oper. Res. Lett.*, 36(4):399–401, 2008. (Cited on page 26.)

[27] Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30(2):528–560, 2000. (Cited on pages 154, 196, and 198.)

[28] Joseph Cheriyan and László A. Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM J. Comput.*, 43(4):1342–1362, 2014. (Cited on pages 26 and 112.)

[29] Nachshon Cohen and Zeev Nutov. A $(1+\ln 2)$-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theor. Comput. Sci.*, 489-490:67–74, 2013. (Cited on page 26.)

[30] Stephen A. Cook. *The Complexity of Theorem-Proving Procedures*, page 143–152. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023. (Cited on pages 8 and 20.)

[31] Efim Dinic, Alexander Karzanov, and Michael Lomonosov. The system of minimum edge cuts in a graph. *In book: Issledovaniya po Diskretnoĭ Optimizatsii (Engl. title: Studies in Discrete Optimizations), A.A. Fridman, ed., Nauka, Moscow, 290-306, in Russian,*, 01 1976. (Cited on pages 10 and 25.)

[32] Jack Edmonds et al. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1967. (Cited on page 92.)

[33] 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. (Cited on page 26.)

[34] Jon Feldman and Matthias Ruhl. The directed steiner network problem is tractable for a constant number of terminals. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 299–308. IEEE Computer Society, 1999. (Cited on page 33.)

[35] Andreas Emil Feldmann, Jochen Könemann, Neil Olver, and Laura Sanità. On the equivalence of the bidirected and hypergraphic relaxations for steiner tree. *Mathematical programming*, 160(1):379–406, 2016. (Cited on pages 12, 92, 93, and 212.)

[36] Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 817–831. SIAM, 2018. (Cited on page 26.)

[37] Greg N. Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981. (Cited on pages 26, 27, and 112.)

[38] Isaac Fung, Konstantinos Georgiou, Jochen Könemann, and Malcolm Sharpe. Efficient algorithms for solving hypergraphic steiner tree relaxations in quasi-bipartite instances. *arXiv preprint arXiv:1202.5049*, 2012. (Cited on page 92.)

[39] Harold N Gabow. An improved analysis for approximating the smallest k-edge connected spanning subgraph of a multigraph. *SIAM Journal on Discrete Mathematics*, 19(1):1–18, 2005. (Cited on pages 154 and 196.)

[40] Harold N Gabow, Michel X Goemans, Éva Tardos, and David P Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks: An International Journal*, 53(4):345–357, 2009. (Cited on pages 154 and 198.)

[41] Waldo Gálvez, Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Node connectivity augmentation of highly connected graphs. *arXiv preprint arXiv:2311.17010*, 2023. (Cited on pages 14 and 216.)

[42] Waldo Gálvez, Francisco Sanhueza-Matamala, and José A. Soto. Approximation algorithms for vertex-connectivity augmentation on the cycle. In Jochen Koenemann and Britta Peis, editors, *Approximation and Online Algorithms*, pages 1–22, Cham, 2021. Springer International Publishing. (Cited on pages 26 and 112.)

[43] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979. (Cited on pages 11 and 91.)

[44] 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. (Cited on pages 154, 171, and 172.)

[45] Edgar N Gilbert and Henry O Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968. (Cited on page 91.)

[46] Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 223–232, 1994. (Cited on pages 26 and 92.)

[47] Michel X Goemans and Young-Soo Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993. (Cited on page 92.)

[48] Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1161–1176, New York, NY, USA, 2012. Association for Computing Machinery. (Cited on pages 84, 91, 92, and 102.)

[49] 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 (STOC)*, pages 632–645. ACM, 2018. (Cited on page 26.)

[50] Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Improved approximations for flexible network design. *arXiv preprint arXiv:2404.08972*, 2024. (Cited on pages 14 and 217.)

[51] Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanità. Finding Almost Tight Witness Trees. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on pages 14, 112, and 216.)

[52] Dylan Hyatt-Denesik, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Approximations for Throughput Maximization. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on page 216.)

[53] Dylan Hyatt-Denesik, Mirmahdi Rahgoshay, and Mohammad R Salavatipour. Approximations for throughput maximization. *Algorithmica*, pages 1–33, 2024. (Cited on page 216.)

[54] Dylan Vern Phillips Hyatt-Denesik. *Algorithms in Throughput Maximization*. PhD thesis, University of Alberta, 2019. (Cited on page 216.)

[55] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. (Cited on page 26.)

[56] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995. (Cited on page 4.)

[57] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. (Cited on pages 8 and 20.)

[58] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1:47–65, 1997. (Cited on page 91.)

[59] Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *J. Algorithms*, 14(2):214–225, 1993. (Cited on pages 10, 25, 26, and 27.)

[60] Philip Klein and RJJA Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995. (Cited on page 30.)

[61] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006. (Cited on page 4.)

[62] Yusuke Kobayashi and Takashi Noguchi. An approximation algorithm for two-edge-connected subgraph problem via triangle-free two-edge-cover. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 49:1–49:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on page 154.)

[63] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004. (Cited on pages 111, 112, 143, and 212.)

[64] 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. (Cited on page 26.)

[65] Guy Kortsarz and Zeev Nutov. LP-relaxations for tree augmentation. *Discret. Appl. Math.*, 239:94–105, 2018. (Cited on page 26.)

[66] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956. (Cited on page 4.)

[67] 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. (Cited on page 26.)

[68] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of ak-connected graph. *Algorithmica*, 7(1):583–596, 1992. (Cited on page 199.)

[69] Z. Nutov. Approximating minimum-cost edge-covers of crossing bisetfamilies. *Combinatorica*, pages 95–114, 2014. (Cited on pages 12 and 112.)

[70] Zeev Nutov. On the tree augmentation problem. In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA)*, pages 61:1–61:14, 2017. (Cited on page 26.)

[71] Zeev Nutov. 2-node-connectivity network design. In *Proceedings of the 18th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 12806 of *Lecture Notes in Computer Science*, pages 220–235. Springer, 2020. (Cited on pages 11, 26, 27, 30, 31, 38, 40, 41, and 112.)

[72] Zeev Nutov. A $4 + \epsilon$ approximation for k-connected subgraphs. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1000–1009. SIAM, 2020. (Cited on pages 26 and 112.)

[73] Zeev Nutov. Approximation algorithms for connectivity augmentation problems. In *Proceedings of the 16th International Computer Science Symposium in Russia (CSR)*, volume 12730, pages 321–338. Springer, 2021. (Cited on pages 26, 29, and 77.)

[74] Zeev Nutov. Improved approximation algorithms for some capacitated k edge connectivity problems. *arXiv preprint arXiv:2307.01650*, 2023. (Cited on page 154.)

[75] Julius Petersen. Die theorie der regulären graphen. *Acta Math*, 15(3):193–220, 1981. (Cited on page 120.)

[76] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, 49(1):16–34, 2002. (Cited on page 4.)

[77] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. (Cited on page 4.)

[78] Hans Jürgen Prömel and Angelika Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89–101, 2000. (Cited on page 91.)

[79] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *SODA*, pages 770–779, 2000. (Cited on page 91.)

[80] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003. (Cited on pages 15, 19, and 20.)

[81] 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. (Cited on pages 193 and 194.)

[82] Arie Segev. The node-weighted steiner tree problem. *Networks*, 17(1):1–17, 1987. (Cited on pages 33 and 92.)

[83] Vera Traub and Rico Zenklusen. Local search for weighted tree augmentation and steiner tree. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3253–3272. SIAM, 2022. (Cited on page 91.)

[84] Vera Traub and Rico Zenklusen. A (1.5+ ε)-approximation algorithm for weighted connectivity augmentation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1820–1833, 2023. (Cited on page 26.)

[85] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001. (Cited on page 91.)

[86] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001. (Cited on pages 15, 17, 18, 157, 164, and 192.)

[87] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011. (Cited on pages 9, 15, and 21.)

[88] Richard T Wong. A dual ascent approach for steiner tree problems on a directed graph. *Mathematical programming*, 28:271–287, 1984. (Cited on page 92.)

[89] XiaoHua Xu, Yuexuan Wang, Hongwei Du, Peng-Jun Wan, Feng Zou, Xianyue Li, and Weili Wu. Approximations for node-weighted steiner tree in unit disk graphs. *Optim. Lett.*, 4(3):405–416, 2010. (Cited on page 47.)

[90] Alexander Z Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9:463–470, 1993. (Cited on page 91.)

Summary

Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity

In this thesis, we have considered a number of problems in the area of Survivable Network Design from the perspective of designing approximation algorithms. In Chapter 3, we provide a 1.8596-approximation for k-Node-Connectivity Augmentation, for $k = 1$. This is done by utilizing a reduction from connectivity augmentation problems to Node-Steiner Tree instances. We make a key observation about the structure of such instances that allow us to adapt the current best approximation algorithm for Steiner Trees [16] to our instances, as well as to provide a new analysis. The analysis of the algorithm relies on the construction of so-called *witness trees*. The key to the analysis of this chapter is providing a novel construction of these witness trees. In addition, we show a limit to the method of using witness trees to analyze the current approximation algorithm, indicating that in order to find a significantly better approximation, it is likely that one must come up with very different approach.

There are some future directions for research related to results in this chapter that may be worth exploring. There is still a gap between lower bounds on what approximations can be found using a witness tree based analysis, and our approximation factors. It seems likely that one can find an improved approximation by providing a more detailed construction of witness trees, though as we mentioned before, this direction will yield only incremental improvements. Another very interesting direction for future research would be pushing these results to the setting where links have an associated cost and our goal is to minimize the total cost of our augmenting links.

In Chapter 4 we consider the Steiner Tree problem, for Steiner-Claw Free instances, which are instances where the Steiner nodes are adjacent to at most 2

other Steiner nodes, and provide a 1.354-approximation for such instances. Our approach in this chapter again uses the current best approximation algorithm for Steiner Trees [16], where our improvement on relies on finding a construction of witness trees that is tailored to the specific structure of these instances. In addition to this approximation, we provide a matching lower bound, showing that one cannot find a better approximation using these techniques, as the witness trees we found are the best possible for these instances. As a corollary of this approximation we also get an improved bound on the integrality gap of the bidirected cut relaxation for Steiner-Claw Free instances (this follows directly from combining our upper bound with the results in [35]). An open question is whether one can find an upper bound less than 2 for the bidirected cut relaxation in general instances, which is a long-standing open question in the field of approximation algorithms. As an intermediate step, one could consider other restricted instances, much like we consider Steiner-Claw Free instances, to learn about the integrality gap.

In Chapter 5, we continue to look at the problem of Node-Connectivity Augmentation, but for the largest possible values of connectivity instead of the smallest. In particular, we consider $(n-d)$-node-Connectivity Augmentation, for $d = O(n^c)$, with c being a positive constant, where we show that the problem is APX-hard. This result complements the result of Kortsartz et al. [63], who show the problem is APX-hard except for when $d = O(n^c)$, hence, completing the picture regarding the complexity status of the problem. In addition, we provide a pair of approximation algorithms for $(n-4)$-Node-Connectivity Augmentation, one for the weighted case and one for the unweighted case. Both of these algorithms rely on taking the complement of the graph. The goal then becomes to select a minimum cost set of links (which are now a subset of edges of the graph), so that a set of "forbidden" subgraphs each contain at least one link. These forbidden subgraphs correspond to the deficient cuts of the original instance.

One clear direction of research that relates to this chapter is to find an improved approximation algorithm for cases where $d > 4$. The first stumbling block to overcome in this direction is to find a way to nicely characterize the forbidden subgraphs. In the $d = 4$ case, the complement graph takes the form of a set of "chains" (a set of squares that pairwise share at most one edge) that are each adjacent only between the corners. This allowed the design of an algorithm that exploited the structure of this graph. For larger values of d, one would need to identify some sort of structure to exploit.

Finally, in Chapter 6, we consider the recent problem of Flexible Graph Connectivity. We find improved approximations for three particular variants of this

problem; for $(1,1)$-Flexible Edge-Connectivity, $(1,k)$-Flexible Edge-Connectivity, and $(1,1)$-Flexible Vertex-Connectivity. For the this last problem in particular, our approximation is the first one with a better than 2 approximation.

As the area of Flexible Network Design is quite recent, there is a lot of room for improving known approximation results. Furthermore, there is comparatively little known for approximation results in weighted cases, when edges have an arbitrary cost. For the particular problem of $(1,1)$-Flexible Edge-Connectivity, one future direction of research that can be taken is finding a better than $\frac{10}{7}$-approximation for the $(1,1)$-Flexible Edge-Connectivity problem. The algorithm for this case relies in part on computing a maximum forest of safe edges, and it seems likely that an improvement can be found if we consider various sizes of this forest. In particular, when the forest is much larger or smaller, it is not too hard to find an improving approximation, but when the forest is roughly $\frac{4}{7}n$, this seems more challenging.

Course of Life

Born in Edmonton 1994

Bachelor of Science Honors in Applied Mathematics 2012–2017
University of Alberta

Master of Science, Computing Science 2017–2019
Thesis: *Algorithms in Throughput Maximization*
University of Alberta

PhD in Combinatorial Optimization 2020–2024
Thesis: *Approximation Algorithms for Connectivity Augmentation and Flexible Graph Connectivity*
Promotor: *Laura Sanita*
Copromotor: *Frits Spieksma*
Eindhoven University of Technology

List of Publications

D. Hyatt-Denesik has the following publications:

Journals

- Haris Angelidakis, Dylan Hyatt-Denesik, and Laura Sanità. Node connectivity augmentation via iterative randomized rounding. *Mathematical Programming*, pages 1–37, 2022
- Dylan Hyatt-Denesik, Mirmahdi Rahgoshay, and Mohammad R Salavatipour. Approximations for throughput maximization. *Algorithmica*, pages 1–33, 2024

Proceedings and Conference Contributions

- Dylan Hyatt-Denesik, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Approximations for Throughput Maximization. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik
- Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanità. Finding Almost Tight Witness Trees. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik

Masters Thesis

- Dylan Vern Phillips Hyatt-Denesik. *Algorithms in Throughput Maximization*. PhD thesis, University of Alberta, 2019

Pre-prints

- Waldo Gálvez, Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Node connectivity augmentation of highly connected graphs. *arXiv preprint arXiv:2311.17010*, 2023

- Dylan Hyatt-Denesik, Afrouz Jabal Ameli, and Laura Sanita. Improved approximations for flexible network design. *arXiv preprint arXiv:2404.08972*, 2024