Augmenting Trees to Achieve 2-Node-Connectivity

by

Logan Grout

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2020

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Logan Grout was the sole author of this thesis, with the exception of a few figures from [1] and [17], which were used with only minor modifications. Professors Adjiashvili and Nutov gave their explicit permission for the figures to be used here. Furthermore, this thesis was written under the supervision of Professor Joseph Cheriyan and is based on many numerous discussions with him.

This thesis is mostly expository of the results in [18], [17], and [1]. As such, much of the content is not original. While Logan Grout wrote the entirety of the thesis, the results, ideas, and arguments are similar to work that has appeared before. All results which are not original are cited.

Abstract

This thesis focuses on the *Node-Connectivity Tree Augmentation Problem (NC-TAP)*, formally defined as follows. The first input of the problem is a graph G which has vertex set V and edge set E. We require $|V| \geq 3$ to avoid degenerate cases. The edge set E is a disjoint union of two sets T and L where the subgraph (V, T) is connected and acyclic. We call the edges in T the *tree edges* and the edges in L are called *links*. The second input is a vector $c \in \mathbb{R}^L_{\geq 0}$ (a vector of nonnegative real numbers indexed by the links), which is called the *cost* of the links. We often refer to this graph G and cost vector c as an *instance* of NC-TAP. Given an instance $G = (V, T \cup L)$ and $c \in \mathbb{R}^L_{\geq 0}$ to NC-TAP, a *feasible solution* to that instance is a set of links $F \subseteq L$ such that the graph $(V, T \cup F)$ is 2-connected. The cost of a set of links F is denoted $c(F) := \sum_{l \in L} c_l$. The goal of NC-TAP is to find a feasible solution F^* to the given instance such that the the cost of F^* is minimum among all feasible solutions to the instance.

This thesis is mainly expository and it has two goals. First, we present the current best-known algorithms for NC-TAP. The second goal of this thesis is to explore new directions in the study of NC-TAP in the last chapter. This is an exploratory chapter where the goal is to use the state of the art techniques for TAP to develop an algorithm for NC-TAP which has an approximation guarantee better than factor 2.

Acknowledgements

First and foremost, I would like to thank Joseph Cheriyan, who has been my supervisor, collaborator, and mentor even before I was his Master's student. This thesis would not exist without his guidance, feedback, and numerous discussions. Next, I thank my readers, Jochen Könemann and Chaitanya Swamy, for their comments, insights, and time. I would also like to thank Professors David Adjiashvili from ETH Zurich and Zeev Nutov from The Open University of Israel for giving me permission to use figures from their works in this thesis.

Finally, I want to thank all of the people who gave me emotional support and friendship throughout my academic career. In particular, I want to thank Nishad Kothari for encouraging me to do Undergraduate research, and all of friends in the Party Office and the PMC.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

One of the most fundamental and well-studied family of problems in Combinatorial Optimization and Operations Research is the family of *Survivable Network Design Problems (SNDPs)*. These are problems of finding structures within a network, of minimum cost, such that the structure is still connected in the event of some specified number of node or edge failures. This is a very large family of problems and it includes problems with many applications in transportation and telecommunications. However, this class of problems is too large to study in its entirety in this thesis. Instead we restrict ourselves to the study of a subset of SNDPs.

The *Tree Augmentation Problem (TAP)* is an SNDP where one is given a base network that is a spanning tree. We may assume that the edges of the base network have zero costs. Then we want to purchase additional links between nodes such that the network will still be connected even if any one link fails. This problem was introduced by Frederickson and Jájá in their 1981 paper [10], though they called it the Connected Bridge-Connectivity Augmentation Problem. Frederickson and Jájá also prove that the problem is NP-hard, and they give an algorithm that returns a solution that has cost at most 2 times the cost of the cheapest solution. Their work was inspired by an earlier paper of Eswaran and Tarjan. Since then TAP has been the subject of much study.

Several more algorithms have since been developed for TAP. Some of these are general-purpose algorithms that apply to more general classes of SNDPs, such as the iterative rounding algorithm due to Jain [13] and the primal-dual algorithm due to Goemans et

al., [11]. However, none of these algorithms have been able to achieve an approximation guarantee of less than 2. Improvements have only been obtained in certain special cases. In the case where the base network is such that any pair of nodes can reach each other using a constant number of edges, Cohen and Nutov give an algorithm that outputs a solution of cost at most $(1 + \ln 2)$ times the cheapest solution [6]. In the case where all of the purchasable links have the same cost, which we can assume is one, Kortsarz and Nutov presented an algorithm with an approximation guarantee of 3/2 [14]. We call this case where all links have cost 1 the *unweighted* case. The current best algorithm for the unweighted case is due to Grandoni, Kalaitzis, and Zenklusen who obtain an approximation guarantee of 1.458 [12]. Finally, the last special case of interest to us in this thesis is when all of the purchasable links have a cost between 1 and some constant. Note that this is a generalization of the unweighted case. With this assumption, Adjiashvili gives an algorithm that has an approximation guarantee of $(1.965 + \epsilon)$ [1]. This was improved by Fiorini, Groß, Könemann, and Sanità to $(3/2 + \epsilon)$ [9].

Despite all of the attention that has been given to TAP, very little study has been done of the related problem where we want to augment a base network that is connected to survive any single node failure. This problem, which we call the *Node-Connectivity Tree Augmentation Problem (NC-TAP)* is the focus of this thesis. NC-TAP was also introduced by Frederickson and Jájá in [10], though they call it the Connected Biconnectivity Augmentation Problem. Like with TAP, they show that NC-TAP is NP-hard and give an algorithm with approximation guarantee of 2. To the best of our knowledge, no improvements were made, even for special cases of NC-TAP, until Nutov gave an algorithm for the unweighted case of NC-TAP in [17] which has a factor $(1.916 + \epsilon)$ approximation guarantee.

This thesis is mainly expository and it has two goals. First, we present some of the current best-known algorithms for NC-TAP. In Chapter 2 we present a primal-dual algorithm due to Ravi and Williamson [18, 19] which has an approximation guarantee of 2 [1]. Their algorithm is actually for a more general class of SNDPs, and they present it as a subroutine for yet another SNDP. We present their arguments tailored for NC-TAP as this allows for some simplification. In Chapter 3 we present the algorithm for the unweighted case given by Nutov in [17], which applies the techniques developed by Byrka et al. in [2]. The second goal of this thesis is to explore new directions in the study of NC-TAP, which is done in Chapter 4. This is an exploratory chapter where we attempt to use the state of the art techniques for TAP to develop an algorithm for NC-TAP which has an approximation guarantee better than factor 2. In particular, we study the developments

[1] [18] claimed to have an algorithm for two different SNDPs which they call the {0,1,2}-SNDP and the k-vertex-connectivity problem. A fatal flaw was found in this paper, however the authors published an erratum [19] which addresses the flaw in the case of the {0,1,2}-SNDP, which NC-TAP is a special case of.

made in [1] and apply them to NC-TAP. We have not yet been successful in creating such an algorithm but we believe our study has merit. First, we pin down some core unsolved questions that might motivate future research. Secondly, we uncover a simpler problem related to NC-TAP which is of interest in its own right. We show that this new problem, which we call the *Two-Level Tree Augmentation Problem (TLTP)*, is NP-hard and present an (11/6)-approximation algorithm for it.

Since this thesis is largely expository, many results presented are not original. Some results, while new in a formal sense, are fundamentally based on earlier results and are proven using similar arguments. Furthermore, where appropriate, we utilize figures which appeared in previous publications. Naturally, we give citations for any content which is not original.

Throughout this thesis, we will assume the reader is familiar with the content, definitions, and notations that one might see in first year graduate courses in Graph Theory, Mathematical Programming, and Combinatorial Optimization. If the reader encounters any notation, definitions, or material that is unfamiliar, we refer them to the glossary of this thesis, or else the following standard texts: [8, 20, 22].

1.2 Problem Statement

NC-TAP and TAP are optimization problems, formally defined as follows. The two problems have the same inputs. The first input of either problem is a graph G which has vertex set V and edge set E. We require $|V| \geq 3$ to avoid degenerate cases. The edge set E is a disjoint union of two sets T and L where the subgraph (V, T) is connected and acyclic, ie it is a spanning tree. We often abuse notation and use T where it would make sense to reference a graph, not a set of edges. In this event is is understood we are using T as short-hand for the graph (V, T). We call the edges in T the *tree edges* and the edges in L are called *links*. The second input is a vector $c \in \mathbb{R}^L_{\geq 0}$ (a vector of nonnegative real numbers indexed by the links), which is called the *cost* of the links. We often refer to this graph G and cost vector c as an *instance* of NC-TAP. The cost of a set of links is the sum of the costs of each edge in the solution and is denoted $c(F) := \sum_{l \in L} c_l$. Given an instance $G = (V, T \cup L)$ and $c \in \mathbb{R}^L_{\geq 0}$ to NC-TAP (or TAP), a *feasible solution* to that instance is a set of links $F \subseteq L$ such that the graph $(V, T \cup F)$ is 2-connected (or 2-edge-connected). The goal of NC-TAP (or TAP) is to find a feasible solution F^* to the given instance such that the the cost of F^* is minimum among all feasible solutions to the instance.

1.3 Notation, Definitions, and Assumptions

Before we can write a concise mathematical program for NC-TAP (or TAP), it will be helpful to define some terminology and notation that will be used throughout this thesis.

Let $G = (V, E)$ be a graph. Let $F \subseteq E$ and $S, S' \subsetneq V$ such that S and S' are disjoint and nonempty. We define the *cut from S to S'* with respect to F as $\delta_F(S : S') := \{uv \in F : u \in S, v \in S'\}$. We define the *vertex neighbourhood* of S with respect to F as $\Gamma_F(S) := \{v \in V \setminus S : uv \in F \text{ for some } u \in S\}$. We define $\zeta_F(S) := V \setminus (S \cup \Gamma_F(S))$, the set of all vertices which are neither in S nor in the neighbourhood of S with respect to F. Further, for any subgraph $H \subseteq G$ we let $\text{Comp}(H)$ denote the set of connected components of H. When talking about the components of a subgraph, we will often only be interested in the vertices of that component. So we will abuse notation and often say that the elements of $\text{Comp}(H)$ are the vertex sets of the components of H. We will also use the common notation that for $x \in \mathbb{R}^I$ and $S \subseteq I$,

$$x(S) = \sum_{i \in S} x_i$$

Finally, for a spanning tree T, it is known that there is a unique path between any two vertices u, v. For the spanning tree that is the base network of a TAP or NC-TAP instance, we denote this unique u, v-path as P_{uv}. For any link $l \in V \times V$ we use P_l to mean P_{uv} where u and v are the endpoints of l.

There is a key assumption that can be made about TAP and NC-TAP instances, that relies on the following definition.

Definition 1. Given two link $l_1, l_2 \in V \times V$ we say l_1 is a *shadow* of l_2 if $P_{l_1} \subseteq P_{l_2}$.

The assumption is that if $l \in L$, then L contains all shadows of l. We show that this is a valid assumption for NC-TAP, but the reader should note that it is also valid for TAP [1]. Suppose F is an NC-TAP solution which contains a link l_1 which is a shadow of l_2. We will show that $(F \setminus \{l_1\}) \cup \{l_2\}$ is also an NC-TAP solution. Suppose not, then $G' = (V.(F \setminus \{l_1\}) \cup \{l_2\})$ has a cut node w. Since F is an NC-TAP solution, $F \cup \{l_2\}$ is as well. Thus l_1 connects $G' - w$. This implies that $G' - w$ has exactly two components and w is a vertex on P_{l_1}. However, $P_{l_1} \subseteq P_{l_2}$ since l_1 is a shadow of l_2. Thus the endpoints of l_2 are in different components of $G' - w$, a contradiction.

1.4 Linear Programs for Tree Augmentation

We will explain the Linear Programs (LPs) which we use to design algorithms for TAP and NC-TAP. Throughout our explanation, we will refer to the following example instance. Let the overall all graph be K_4. Let the spanning tree be a star and label the centre node v and the leaves u_1, u_2, u_3. Let the links all have unit cost.

Linear Programs (LPs) for Survival Network Design Problems such as TAP and NC-TAP usually are formed in a similar way. We start with variables $x_e \in \mathbb{R}$ and costs $c_e \in \mathbb{R}_{\geq 0}$ for each edge $e \in E$. Recall, that we only have costs for links. We can extend our cost vector to all edges by saying $c_e = 0$ for all $e \in T$. In this framework, TAP is the problem of finding a minimum cost 2-edge-connected subgraph in G, and NC-TAP is the problem of finding minimum cost 2-connected spanning subgraph. of G.

We begin by considering TAP. Consider a set of vertices, S. We want the solution to be 2-edge-connected, so we know we need two edges from S to the rest of the graph. That is $x(\delta_E(S : V \setminus S)) \geq 2$. Thus a valid LP for TAP is $(TAP1)$, shown below.

$$
\begin{aligned}
(TAP1) \quad \min \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \\
& x(\delta_E(S : V \setminus S)) \geq 2 \quad \forall \emptyset \neq S \subsetneq V \\
& 0 \leq x_e \leq 1 \quad \forall e \in E
\end{aligned}
$$

This LP has an exponential number of constraints, some of which are redundant, and as mentioned above, is really the LP for finding a minimum cost 2-edge-connected spanning subgraph of G. The $(TAP1)$ LP for our K_4 example described above is as follows.

$$
\begin{aligned}
\min \; & 0x_{vu_1} + 0x_{vu_1} + 0x_{vu_1} + x_{u_1u_2} + x_{u_1u_3} + x_{u_2u_3} & & \\
\text{s.t.} \; & & & \\
& x_{vu_1} + x_{vu_1} + x_{vu_1} + 0x_{u_1u_2} + 0x_{u_1u_3} + 0x_{u_2u_3} \geq 2 & & \\
& x_{vu_1} + 0x_{vu_1} + 0x_{vu_1} + x_{u_1u_2} + x_{u_1u_3} + 0x_{u_2u_3} \geq 2 & & \\
& 0x_{vu_1} + x_{vu_1} + 0x_{vu_1} + x_{u_1u_2} + 0x_{u_1u_3} + x_{u_2u_3} \geq 2 & & \\
& 0x_{vu_1} + 0x_{vu_1} + x_{vu_1} + 0x_{u_1u_2} + x_{u_1u_3} + x_{u_2u_3} \geq 2 & & \\
& 0x_{vu_1} + x_{vu_1} + x_{vu_1} + x_{u_1u_2} + x_{u_1u_3} + 0x_{u_2u_3} \geq 2 & & \\
& x_{vu_1} + 0x_{vu_1} + x_{vu_1} + x_{u_1u_2} + 0x_{u_1u_3} + x_{u_2u_3} \geq 2 & & \\
& x_{vu_1} + x_{vu_1} + 0x_{vu_1} + 0x_{u_1u_2} + x_{u_1u_3} + x_{u_2u_3} \geq 2 & & \\
& 0 \leq x_e \leq 1 & & \forall e \in E
\end{aligned}
$$

Note that many of the constraints are satisfied when we set $x_{vu_1} = x_{vu_2} = x_{vu_3} = 1$. This is because, some cuts are covered by enough tree edges that they don't require any links. It is natural to ask if there is a simpler LP which takes advantage of the additional structure of TAP. For any tree edge $e \in T$, we know $T - e$ has exactly two components. Let S be the vertex set of one of these components. We let $\text{cov}(e) = \delta_L(S : V \setminus S)$. Consider the following LP, labelled $(TAP2)$ and referred to as the *Cut LP*.

$$
\begin{aligned}
(TAP2) \quad \min \quad & \sum_{l \in L} c_l x_l \\
\text{subject to} \quad & \\
& x(\text{cov}(e)) \geq 1 \quad \forall e \in T \\
& x_l \geq 0 \quad \forall l \in L
\end{aligned}
$$

Proposition 1. $(TAP1)$ *and* $(TAP2)$ *are equivalent.*

Now we consider the same process for NC-TAP. However, it becomes significantly more complicated. We cannot just consider any set of vertices S and make sure there are two internally disjoint paths between S and $V \setminus S$ because any such path will have no internal vertices. Instead we must consider each pair of disjoint subsets of vertices $S \subseteq V$ and $S' \subseteq V \setminus S$, and make sure that these have two internally-disjoint paths between them. We rely on the fact that if you remove some number, t, of vertices, from a k-connected graph, the resulting graph is still $(k - t)$-connected, and thus $(k - t)$-edge-connected. For every vertex v not in S or S', we assume that v is removed and thus the number of edges we require between S and S' decreases by one. That is, we must satisfy $x(\delta_E(S : S')) \geq 2 - |V \setminus (S \cup S')|$. Consider again our instance on K_4. Now let $S = \{u_1\}$, and $S' = \{v, u_2, u_3\}$. We want a 2-connected graph, which must also be 2-edge-connected, so we know a solution must have two edges between these sets. Now consider consider leaving $S = \{u_1\}$ but instead $S' = \{u_2, u_3\}$. We want the solution to be 2-connected, but there is already a path between the two sets which uses the centre vertex. Thus we only need one more edge between these two sets. This gives rise to the following LP for NC-TAP.

$$
\begin{aligned}
(NC - TAP1) \quad \min \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \\
& x(\delta_E(S : S')) \geq 2 - |V \setminus (S \cup S')| \quad \forall \emptyset \neq S, S' \subsetneq V, S \cap S' = \emptyset \\
& 0 \leq x_e \leq 1 \quad \forall e \in E
\end{aligned}
$$

Again, this LP has an exponential number of constraints and captures the more general problem of finding a minimum cost 2-connected subgraph. However, just like with TAP, it clear to see that some are redundant. Consider again our example instance of K_4 where the initial tree is a star. We know that the only vertex that is a cut node in the star is the centre. So a solution is just a set of links that connects the remaining three vertices. So if we partition the vertices other then v into two sets, no matter how we do it, a solution should have a link between the two sets. In general, the vertices that are cut nodes in an arbitrary tree are precisely the vertices which are not leaves. After removing a non-leaf vertex v, we need the components of $T-v$ to be connected. We use $\text{Comp}(T-v)$ to denote the set of components of $T-v$. If we look at any subset $\mathcal{S}$ of components, a solution must have a link from a component in $\mathcal{S}$ to a component not in $\mathcal{S}$. This gives us the following LP, labelled $(NC-TAP2)$, which we call the *Set-Pair LP*

$$
\begin{aligned}
(NC-TAP2) \quad \min \quad & \sum_{l \in L} c_l x_l \\
\text{subject to} \quad & \\
& x(\delta_L(S, \zeta_T(S)) \geq 1 \quad \forall \text{ non-leaf } v, \text{ and } \forall \emptyset \neq \mathcal{S} \subseteq \text{Comp}(G-v), \\
& \qquad\qquad S := \cup_{\mathcal{S}} S' \\
& x_l \geq 0 \quad \forall l \in L
\end{aligned}
$$

In our K_4 example, the only non-leaf is v and $\text{Comp}(T-v) = \{\{u_1\}, \{u_2\}, \{u_3\}\}$. Thus, the possible choices for S are $\{u_1\}, \{u_2\}, \{u_3\}, \{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_3\}$.

Proposition 2. *LP $(NC-TAP1)$ and LP $(NC-TAP2)$ are equivalent.*

Proof. First we note that since $c_e = 0$ for any $e \in T$, if we take a feasible solution x and replace x_e with 1 for all $e \in T$, then the resulting vector is feasible and has the same objective value. Furthermore, since we don't actually want a minimum cost 2-connected spanning subgraph, but rather a minimum cost set of links that augment the tree T, it makes sense that we always include all tree edges in our solution. Thus we may replace $0 \leq x_e \leq 1$ with $x_e = 1$ for all tree edges $e \in T$.

Next, note that if $|V \setminus (S \cup S')| \geq 2$ then the corresponding constraint is redundant as $x \geq 0$. So we turn our attention to when $|V \setminus (S \cup S')| = 1$, say $V \setminus (S \cup S') = \{v\}$. Consider $G-v$. Clearly, $S' = V \setminus (S \cup v) = \zeta_T(S)$ so it remains to show that S is a union of components of $T-v$. The non-empty, non-exhaustive properties we get for free since $S, S' \neq \emptyset$. Suppose that S is not the union of components of $T-v$, then $T-v$ contains a

component, say C such that $S \cap C$ and $S' \cap C$ are nonempty. Let $u \in S \cap C$ and $w \in S' \cap C$. Since C is connected, there exists a u, w-path that does not use v. There also must exist some tree edge ab in this path such that $a \in S$ and $b \in S'$. Thus $ab \in \delta_T(S : S')$ and thus the constraint $x(\delta_E(S : S')) \geq 2 - |V \setminus (S \cup S')| = 1$ is made redundant by the constraint $x_{ab} = 1$. Thus, for any S, S' such that $|V \setminus (S \cup S')| = 1$, we need only keep the constraints when S is the nonempty, non-exhaustive union of components of $T - v$ for some vertex v, and $S' = \zeta_T(S)$. Note that if v is a leaf of T, then $T - v$ only has a single component and thus no S satisfies this requirement, so we need only consider, non-leaf vertices.

Finally, consider when $|V \setminus (S \cup S')| = 0$. If $|\delta_T(S : S')| \geq 2$ then the constraint is implied by $x_e = 1$ for all tree edges $e \in T$. Further, it is impossible that $|\delta_T(S : S')| = 0$ since T is a spanning tree. So we must have that $|\delta_T(S : S')| = 1$. Let uv be the unique edge in $\delta_T(S : S')$ with $u \in S, v \in S'$. At least one of u or v must be a non-leaf. Without loss of generality, we will assume it is v. We note that S is a component of $T - uv$ and $v \notin S$ and thus S is a component of $T - v$. Specifically it is a nonempty, non-exhaustive union of (one) components of $T - v$. And $S' = \zeta_T(S) \cup v$. Thus, $\delta_E(S : S') \supseteq \delta_E(S : \zeta_T(S)) \cup uv$. But from above, we know that $x(\delta_E(S : \zeta_T(S))) \geq 1$. Thus we have

$$\begin{aligned} x(\delta_E(S : S')) &\geq x_{uv} + x(\delta_E(S : \zeta_T(S)) \\ &\geq x_{uv} + 1 \\ &= 2 \end{aligned}$$

And thus this constraint is redundant. □

This approach is designing LPs is very robust, and works well for problems concerning edge-connectivity, such as TAP. However, it is not useful for designing algorithms for NC-TAP which achieve an approximation guarantee of less than factor 2. Figure 1.1 shows the problem. Consider any (unweighted) TAP/NC-TAP instance where the tree is a star with $k \geq 3$ leaves, and the links form a cycle on the leaves. Both the Cut LP and the Set-Pair LP are satisfied by assigning a weight of $1/2$ to all of the links. This means the optimal value for both the Cut LP and the Set-Pair LP is at most $k/2$. The optimal, integral, TAP solution takes $\lceil k/2 \rceil$ links of the cycle, alternating between taking a link and not taking a link. The worst case scenario is when $k = 3$ and thus the optimal solution uses 2 links. Thus the integrality gap of the Cut LP on this instance is $4/3$. However, Any integral NC-TAP solution must use $k - 1$ links. Thus, the integrality gap of the Set-Pair LP is arbitrarily close to 2.

We introduce a new LP for NC-TAP. The motivation and correctness is fairly clear. After removing any vertex, if one imagines contracting all of the connected components

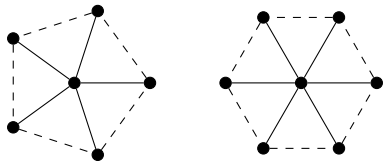


Figure 1.1: Two Stars. One has an odd number of leaves and the other even. $x_l = 1/2$ satisfies both the Cut LP and Set-Pair LP, but not the Partition LP.

into single vertices, then any NC-TAP solution for the original graph must contain a spanning tree on this new graph. One common LP for the Minimum Spanning Tree problem, which is integral [20, Corollary 50.8], has a constraint for every partition of the vertices, and ensures that the number of edges going between any two parts of the partition is at least the total number of parts in the partition, minus one. We will use the notation that $\mathcal{P}(S)$ is the set of partitions of the set S, $|\mathcal{P}|$ is the number of parts in the partition $\mathcal{P} \in \mathcal{P}(S)$, and for a partition of sets of vertices, $\mathcal{P}$, we let $\delta_F(\mathcal{P})$ denote the set of edges in F which have each endpoint in different parts of the partition $\mathcal{P}$. Again, we will consider the example instance on K_4 where the initial tree is a star with centre v and leaves u_1, u_2, u_3. Recall that $\text{Comp}(T - v) = \{\{u_1\}, \{u_2\}, \{u_3\}\}$, so $\mathcal{P}(\text{Comp}(T - v)) = \{[u_1|u_2, u_3], [u_2|u_1, u_3], [u_3|u_1, u_2], [u_1|u_2|u_3]\}$. We also have that $|[u_1|u_2, u_3]| = 2$ and $|[u_1|u_2|u_3]| = 3$. Finally, $\delta_F([u_1|u_2, u_3]) = \{u_1u_2, u_1u_3\}$ and $\delta_F([u_1|u_2|u_3]) = \{u_1u_2, u_1u_3, u_2u_3\}$. With this, we get the following LP, which we call the *Partition LP*.

$$\begin{aligned} \min \quad & \sum_{l \in L} c_l x_l \\ \text{subject to} \quad & \\ & x(\delta_L(\mathcal{P})) \geq |\mathcal{P}| - 1 \quad \forall \text{ non-leaf } v, \text{ and } \mathcal{P} \in \mathcal{P}(\text{Comp}(T - v)) \\ & x \geq 0 \end{aligned}$$

Note that the only non-leaf of the star is the centre, so the Partition LP for the NC-TAP instances shown in Figure 1.1 has only one family of constraints. Furthermore, the partition where every leaf is its own part tells us that the weight on all links must be at least $k-1$, where k is the number of leaves. This means that there is no integrality gap for the Partition LP on such instances. However, the Partition LP is not integral for NC-TAP. It cannot be, since we know NC-TAP is NP-hard [10]. We will show in Section 4.6 that NC-TAP is NP-hard even in some specific special cases, and in Section 4.7 we will show that the Partition LP has an integrality gap of at least $\frac{4}{3}$.

Chapter 2

2-Approximation via Primal-Dual

2.1 Introduction

In this chapter, we present an algorithm due to Ravi and Williamson [18], which they call AUGMENT, that returns a solution F to a given instance of NC-TAP such that $c(F) \leq 2 \cdot \text{OPT}$ where OPT is the minimum cost of all solutions of the given instance. The algorithm Ravi and Williamson created is actually a subroutine of an algorithm that solves a slightly more general problem than NC-TAP, which they call the $\{0, 1, 2\}$-Survivable Network Design Problem. In particular, when they use AUGMENT, the given set of edges that they are augmenting need not form a spanning tree, and rather than a solution making the graph 2-connected, a solution satisfies given connectivity requirements. Specifically, as additional input, we are given integers $r_{ij} \in \{0, 1, 2\}$ for each pair of vertices $i, j \in V$ and a solution must augment the given graph such that there are at least r_{ij} internally disjoint paths between i and j. In our presentation of the algorithm, we assume that the set of edges that we augment does form a spanning tree as this is all that is needed in the setting of NC-TAP, and it simplifies many proofs. However, we keep the generalized connectivity requirements, as this has some potential uses elsewhere in the study of NC-TAP.

We reiterate some notation which is used heavily throughout this chapter. Let $G = (V, E)$ be a graph. Let $F \subseteq E$ and $S \subseteq V$. We define the *vertex neighbourhood* of S with respect to F as $\Gamma_F(S) := \{v \in V \setminus S : uv \in F \text{ for some } u \in S\}$. Then we define $\zeta_F(S) := V \setminus (S \cup \Gamma_F(S))$.

Ravi's and Williamson's algorithm is a Primal-Dual algorithm, meaning that it is based on a particular LP relaxation of NC-TAP and its dual LP. In particular, it is based on the

Set-Pair LP for NC-TAP. This is sufficient for the goal of developing a factor 2 approximation algorithm but it does mean that the analysis is tight.

2.2 Equivalence of NC-TAP and AUGMENT

Ravi and Williamson, as a subroutine, describe a problem that they call AUGMENT. The input to AUGMENT is a graph $G = (V, E)$, a set of edges $T \subseteq E$, a cost c_e for each edge $e \in L := E \setminus T$, and, for each pair of distinct vertices i, j a requirement $r_{ij} \in \{0, 1, 2\}$. They then define an indicator function $h : 2^V \to \{0, 1\}$ as follows. $h(S) = 1$ exactly when there exists $i \in S$ and $j \in \zeta_T(S)$ such that $|\Gamma_T(S)| \leq r_{ij} - 1$. A solution to AUGMENT is a set of edges $F' \subseteq L$ such that for every $S \subseteq V$ such that $h(S) = 1$, then $|\delta_{F'}(S : \zeta_T(S))| \geq 1$. They give an integer program, which they call (AUG) which captures AUGMENT.

$$
\begin{aligned}
(AUG) \quad \min \quad & \sum_{e \in E \setminus T} c_e x_e \\
\text{subject to} \quad & \\
& \sum_{e \in \delta_L(S : \zeta_T(S))} x_e \geq h(S) \quad \emptyset \neq S \subsetneq V \\
& x_e \in \{0, 1\} \quad e \in L
\end{aligned}
$$

We will show that the linear relaxation of the above integer program, when $G = (V, E = T \cup L)$ and $c \in \mathbb{R}^L_{\geq 0}$ are an instance of the NC-TAP, and $r_{ij} = 2$ for every distinct pairs of vertices i, j, is equivalent to the Set-Pair LP of that NC-TAP instance, which from Section 1.4 is

$$
\begin{aligned}
\min \quad & \sum_{l \in L} c_l x_l \\
\text{subject to} \quad & \\
& x(\delta_L(S, \zeta_T(S)) \geq 1 \quad \forall \text{ non-leaf } v, \text{ and } \forall \emptyset \neq \mathcal{S} \subseteq \text{Comp}(G - v), \\
& \qquad\qquad S := \cup_{\mathcal{S}} S' \\
& x_l \geq 0 \quad \forall l \in L
\end{aligned}
$$

Note that $E \setminus T = L$. So we can show the equivalence by showing that $h(S) = 1$ exactly when S is the non-exhaustive union of components of $T - v$ for some non-leaf vertex v. Let $h(S) = 1$. Thus there exists $i \in S$ and $j \in \zeta_T(S)$ such that $|\Gamma_T(S)| \leq 1$. Note that since T is a spanning tree of T, for any $\emptyset \neq S \subsetneq V$ we have $|\Gamma_T(S)| \geq 1$, so $|\Gamma_T(S)| = 1$. Say $\Gamma_T(S) = \{v\}$. Note that S has no neighbours in $T - v$ and thus S must be the non-empty union of some components of $T - v$ and it cannot be the union of all of them since $\zeta_T(S)$ is non-empty.

Now suppose S is the nonempty, non-exhaustive union of components of $T - v$ for some non-leaf v. Since S is the union of components of $T - v$ it must be that $\Gamma_T(S) = \{v\}$. Thus S and $\zeta_T(S)$ are both non-empty. Let $i \in S$ and $j \in \zeta_T(S)$. Since $|\Gamma_T(S)| = 1 \leq 2 - 1 = r_{ij} - 1$, we have that $h(S) = 1$.

When we allow for arbitrary connectivity requirements, all that can no longer be said is that all nonempty, non-exhaustive unions of components of $T - v$ are violated sets. However, we do still have that all violated sets must be such a set.

We will spend the rest of this chapter proving the following.

Theorem 3. *[18] There exists a polynomial time algorithm that, given an instance of NC-TAP, $G = (V, T \cup L)$ with cost vector c, computes an NC-TAP solution $F' \subseteq L$ such that $c(F)$ is at most twice the objective value of an optimal solution to the Set-Pair LP.*

2.3 Uncrossing Violated Sets

Before describing the algorithm, we first give a few vital definitions and a key lemma from [18].

Definition 2. A vertex set $\emptyset \neq S \subsetneq V$ is *violated* with respect to a set of links $F \subseteq L$ if $h(S) = 1$ and $\delta_F(S : \zeta_T(S)) = \emptyset$.

As a consequence, we have that, for any violated set S, $\Gamma_{T \cup F}(S) = \Gamma_T(S)$, and by extension $\zeta_{T \cup F}(S) = \zeta_T(S)$. Also, $|\Gamma_T(S)| = 1$. Also, for a violated set S, $\zeta_T(\zeta_T(S)) = S$. Both of these are true for any set where $|\Gamma_T(S)| = 1$ and $\delta_F(S : \zeta_T(S)) = \emptyset$, but we will only need these facts for violated sets.

Definition 3. A vertex set $\emptyset \neq S \subsetneq V$ is *active* with respect to a set of links $F \subseteq L$ if it is violated with respect to F and no proper subset of S is violated with respect to F. That is an active set is an inclusion-wise minimal violated set.

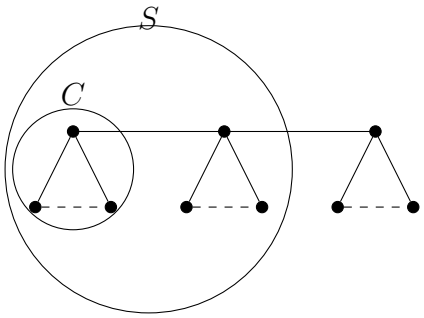


Figure 2.1: S and C are examples of violated sets and C is also active.

An illustration of violated and active sets is given Figure 2.1. As will always be the case in this thesis, solid lines are tree edges and dotted lines are links in F.

Definition 4. We say two sets, A and B, *cross* if $A \setminus B, B \setminus A$, and $A \cap B$ are all nonempty.

Active sets play a vital role in the algorithm and its analysis. The next lemma is useful in its own right, but also has the important corollary that active sets, with respect to a fixed $F \subseteq L$, are disjoint.

Lemma 4. *[18, Lemma 3.6] Let $F \subseteq L$. If A and B are violated sets with respect to F and they cross, then either $A \setminus (B \cup \Gamma_T(B))$ and $B \setminus (A \cup \Gamma_T(A))$ are violated, $A \cap B$ and $A \cup B$ are violated, or $A \setminus B$ and $B \setminus A$ are violated, with respect to F.*

In the event that $A \setminus (B \cup \Gamma_T(B))$ and $B \setminus (A \cup \Gamma_T(A))$ are violated, we will call A and B type I crossing violated sets. Otherwise we call them type II crossing violated sets.

Proof. Let $\Gamma_T(A) = \{\gamma_A\}$ and $\Gamma_T(B) = \{\gamma_B\}$. Because A and B cross, we know there exists $u \in A \cap B$ and $v \in B \setminus A$. Consider the unique u, v-tree path. Specifically consider the first vertex on this path which is not in A, starting from u. This vertex is a neighbour of A and thus must be γ_A. We now consider two cases.

Case 1: $\gamma_A \in B \setminus A$

Figure 2.2 illustrates the structure arising from this case, which turns out to be type I crossing violated sets. Recall from section 2 that we know that A is a union of components of $T - \gamma_A$. In particular, this means that there is a path from $A \setminus B$ to γ_A. This path must be entirely withing A (except for γ_A) and there is a point where it crosses from $A \setminus B$ to B. Thus $\gamma_B \in A \setminus B$. Hence the only neighbour of A is in $B \setminus A$ and the only neighbour of B is in

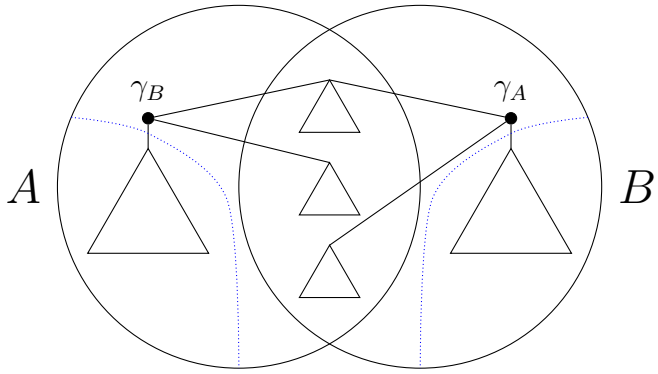


Figure 2.2: Type I Crossing Sets (arising from Case 1 of Lemma 4)

$A \setminus B$. But the graph is connected. Thus we must have that $A \cup B = V$. In particular, since A is violated, there exists $i_A \in A = \zeta_T(B \setminus (A \cup \Gamma_T(A))$ and $j_A \in \zeta_T(A) = B \setminus (A \cup \Gamma_T(A)$ such that $r_{i_A j_A} = 2$. Thus, $B \setminus (A \cup \Gamma_T(A)$ is violated. Similarly, since B is violated, there exists $i_B \in B = \zeta_T(A \setminus (B \cup \Gamma_T(B))$ and $j_B \in \zeta_T(B) = A \setminus (B \cup \Gamma_T(B)$ such that $r_{i_B j_B} = 2$. Thus, $A \setminus (B \cup \Gamma_T(B)$ is violated.

Case 2: $\gamma_A \notin A \cup B$

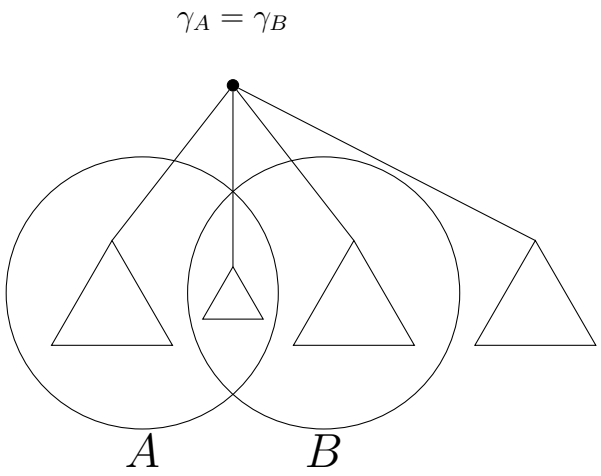


Figure 2.3: Type II Crossing Sets (arising from Case 2 of Lemma 4)

Figure 2.3 illustrates the structure arising from this case, which turns out to be type

II crossing violated sets. Consider a path from $A \setminus B$ to $B \setminus A$. This must use γ_A and thus this path leaves $A \cup B$ before entering B. Specifically, the vertex it uses right before entering $A \cup B$ must be γ_B. However, any vertex in $\Gamma_T(A \cap B)$ would be either in $\Gamma_T(A)$ or $\Gamma_T(B)$ and thus must be either γ_A or γ_B. However, a vertex in $\Gamma_T(A \cap B)$ which is not in $A \cup B$ would be in both $\Gamma_T(A)$ and $\Gamma_T(B)$. Since, $\gamma_A, \gamma_B \notin A \cup B$, we must have that $\Gamma_T(A) = \Gamma_T(B) = \Gamma_T(A \cup B)$.

Furthermore, $A \setminus B$, $B \setminus A$, and $A \cap B$ are all the nonempty, non-exhaustive union of components of $T - \gamma_A = T - \gamma_B$, and thus all are potentially violated sets. Whether or not they are violated depends on if there is a pair of vertices that are required to be 2-connected such that one is inside the set, and the other is outside of the set and its neighbour set. Similarly, $A \cup B$ is also potentially violated, but this requires that there exist vertices outside of $A \cup B$, which there may not.

Subcase 2.1: There exists $i \in A \cap B \subseteq A \cup B$ and $j \in \zeta_T(A \cup B) \subseteq \zeta_T(A \cap B)$ such that $r_{ij} = 2$.

Then $A \cap B$ and $A \cup B$ are violated.

Subcase 2.2: There exists $i \in A \setminus B \subseteq \zeta_T(B \setminus A)$ and $j \in B \setminus A \subseteq \zeta_T(A \setminus B)$ such that $r_{ij} = 2$.

Then $A \setminus B$ and $B \setminus A$ are violated.

Subcase 2.3: The previous cases do not apply.

There must exist $i_A \in A$ and $j_A \in \zeta_T(A)$ such that $r_{i_A j_A} = 2$. Similarly there exists $i_B \in B$ and $j_B \in \zeta_T(B)$ such that $r_{i_B j_B} = 2$. One of the following must happen:

(1) $i_A \in A \setminus B \subseteq A \cup B$ and $j_A \in \zeta_T(A \cup B) \subseteq \zeta_T(A \setminus B)$. So $A \setminus B$ and $A \cup B$ are violated.

(2) $i_A \in A \cap B \subseteq \zeta_T(B \setminus A)$ and $j_A \in B \setminus A \subseteq \zeta_T(A \cap B)$. So $B \setminus A$ and $A \cap B$ are violated.

Similarly, one of the following must happen:

(i) $i_B \in B \setminus A \subseteq A \cup B$ and $j_B \in \zeta_T(A \cup B) \subseteq \zeta_T(B \setminus A)$. So $B \setminus A$ and $A \cup B$ are violated.

(ii) $i_B \in A \cap B \subseteq \zeta_T(A \setminus B)$ and $j_B \in A \setminus B \subseteq \zeta_T(A \cap B)$. So $A \setminus B$ and $A \cap B$ are violated.

If (1) and (i) are true or (2) and (ii) are true, then $A \setminus B$ and $B \setminus A$ are violated. If (1) and (ii) are true, or (2) and (i) are true, then $A \cap B$ and $A \cup B$ are violated.

□

2.4 Computing All Active Sets

We mentioned in the previous section that active sets are vital to the algorithm, which we will present in the next section. In this section, we prove that given an instance of NC-TAP and a set $F \subseteq L$, we can compute the collection of all active sets with respect to F, which we denote as $\mathcal{C}$, in polynomial time. The rest of the algorithm will be straightforward and thus one can easily see it runs in polynomial time, as desired.

We will make use of the following lemma, which is a standard application of Network Flow Theory.

Lemma 5. *[20,] Let $G = (V, E)$ be an undirected graph and $s, t \in V$ a pair of distinct, non-adjacent vertices. One can compute the inclusion-wise minimal set C such that $\Gamma_G(C)$ is a minimum size s, t-separating set.*

Proof. Construct a directed graph $D = (N, A)$ as follows. For each $v \in V$ there are two vertices $v^{in}, v^{out} \in N$ and an arc $(v^{in}, v^{out}) \in A$ of unit capacity. For each $uv \in T \cup F$ there are two arcs $(u^{out}, v^{in}), (v^{out}, u^{in}) \in A$ of infinite capacity. It is well known that the maximum value of an $s^{out} - t^{in}$ flow in D is equal to the number of internally disjoint $s - t$ paths in G. Furthermore, one can recover C from the residual directed graph of D with respect to this maximum flow, which we will call D'. C will be the set of all vertices $v \in V$ such that v^{in} and v^{out} are both reachable from s^{out} in D', and $\Gamma_{T \cup F}(C)$ will be the set of vertices $v \in V$ such that v^{in} is reachable from s^{out} in D' but v^{out} is not. □

Lemma 6. *[18, Theorem 5.1] For any set $F \subseteq L$ one can compute the set of all active sets with respect to F in polynomial time.*

Proof. Suppose C is an active set with respect to F. The it must be the case that there is some $s \in C$ and $t \in \zeta_T(C)$ such that $\Gamma_{T \cup F}(C) = \Gamma_T(C) = \{x\}$. We can determine C using the above lemma with the graph $(V, T \cup F)$.

Thus we can compute $\mathcal{C}$ as follows. Start with $\mathcal{C} = \emptyset$. For each pair $s, t \in V$, compute C and $\Gamma_{T \cup F}(C)$ in polynomial time, adding C to $\mathcal{C}$ if and only if $|\Gamma_{T \cup F}(C)| = 1$. □

2.5 Description of the Algorithm

The AUGMENT algorithm described by Ravi and Williamson is adapted from the Primal-Dual algorithm given by Williamson, Goemans, Mihail, and Vazirani [21]. As such the algorithm has two stages. In the first stage, which is the loop from lines 4 to 8, we start with $F = \emptyset$ and through a sequence of iterations, we will add edges to F until it is a feasible solution to the instance of AUGMENT. In each iteration, the dual variable of all of the active sets is increased until a dual constraint becomes tight for some link l, at which point l is added to F and the active sets are recomputed. The second stage, which is the loop on lines 10 to 14, considers all edges l in F, ordered in the reverse of the order they were added to F in the first stage, and if the edge being considered is redundant for the solution, that is is $F \setminus \{l\}$ is still a feasible solution to the instance, it is removed from F.

Below is the Dual of the LP (AUG), and the full pseudo-code for the AUGMENT algorithm is given in Algorithm 1.

$$\begin{aligned} \max \quad & \sum_{\emptyset \neq S \subsetneq V} h(S) y_S \\ \text{subject to} \quad & \\ & \sum_{S : l \in \delta_L(S : \zeta_T(S))} y_S \leq c_l \quad \forall l \in L \\ & y \geq 0 \end{aligned}$$

Algorithm 1: AUGMENT

```
   // INITIALIZATION
 1 y_S ← 0 for all ∅ ≠ S ⊊ V
 2 F ← ∅
 3 C ← active sets with respect to F
   // STAGE 1
 4 while |C| > 0 do
 5 |   Increase y_C for all C ∈ C until the constraint Σ_{S:l∈δ_L(S:ζ_T(S))} y_S ≤ c_l becomes
   |    tight for some link l ∈ L
 6 |   F ← F ∪ {l}
 7 |   Update C with respect to new F
 8 end
 9 F' ← F
   // STAGE 2
10 for each l ∈ F' in the reverse order they were added to F do
11 |   if F' \ {l} is feasible then
12 |   |   F' ← F' \ {l}
13 |   end
14 end
15 return F'
```

One might notice that initializing $y_S \leftarrow 0$ for all $\emptyset \neq S \subsetneq V$ would take exponential time. However, only the variables corresponding to sets that are at some point active are ever used, and as we saw in the previous section, there is a polynomial number of these.

2.6 Approximation Guarantee of the Algorithm

We wish to show that the algorithm obtains a 2-approximation. Let x^* be an optimal solution to (AUG). In particular, we will show, using LP duality, that

$$\sum_{l \in F'} c_l \leq 2 \sum_{\emptyset \neq S \subsetneq V} h(S) y_S \leq 2 \sum_{l \in L} c_l x_l^*$$

To do this, notice that for any $l \in F'$, we have $c_l = \sum_{S:l \in \delta_{F'}(S:\zeta_T(S))} y_S$ and $y_S > 0$ only if $h(S) = 1$. Thus we can write the inequality we wish to show as

$$\sum_{l \in F'} \sum_{S:l \in \delta_{F'}(S:\zeta_T(S))} y_S \leq 2 \sum_{\emptyset \neq S \subsetneq V} y_S$$

The left-hand side can be re-indexed as

$$\sum_{\emptyset \neq S \subsetneq V} y_S \cdot |\delta_{F'}(S : \zeta_T(S))| \leq 2 \sum_{\emptyset \neq S \subsetneq V} y_S$$

Note that this inequality is true when the algorithm starts, since $y_S = 0$ for all $S \subseteq V$. We will show by induction that this inequality holds after any iteration of the algorithm. Fix some iteration of the algorithm, and assume that at the start of this iteration the inequality holds. Then in the execution of this iteration, the left-hand side of the inequality increases by $\epsilon \cdot \sum_{C \in \mathcal{C}} |\delta_{F'}(C : \zeta_T(C))|$, while the right-hand side increases by $2\epsilon|\mathcal{C}|$. Recall that F is the set of edges chosen by the algorithm up to, but not including the current iteration and $\mathcal{C}$ is the set of active sets with respect to F. It suffices to show

$$\sum_{C \in \mathcal{C}} |\delta_{F'}(C : \zeta_T(C))| \leq 2|\mathcal{C}|$$

To do this, we will prove the following technical lemma.

Lemma 7. *[19] For all $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C : \zeta_{T \cup F}(C))$ there exists a witness set $S_l \subsetneq V$ such that:*

(1) $\delta_{F'}(S_l : \zeta_{T \cup F}(S_l)) = \{l\}$

(2) S_l is violated with respect to F

(3) For each $C \in \mathcal{C}$ either $C \subseteq S_l$ or $C \cap S_l = \emptyset$

(4) The collection of sets $\mathcal{L} := \{S_l : l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C : \zeta_{T \cup F}(C))\} \cup \{V\}$ is laminar

(5) If $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C, \zeta_T(C))$, S_l is the witness set for l and S' is the smallest set in $\mathcal{L}$ properly containing S_l, then the smallest set in $\mathcal{L}$ that contains the active set C is either S_l or S'.

Any collection of sets satisfying (1), (2), and (3) we call a *witness family*. If we have a laminar witness family, meaning that no two sets in the family cross, then that family together with V forms a satisfactory $\mathcal{L}$. We will abuse terminology and also call $\mathcal{L}$, with V, a witness family. Given a laminar witness family $\mathcal{L}$, we can build a tree, denoted H, that captures the structure of $\mathcal{L}$ in the canonical way: $V(H) := \mathcal{L}$, we consider H rooted at the vertex V and for every other vertex $S \in \mathcal{L}$, the parent of S is the smallest set $S' \in \mathcal{L}$ that properly contains S. Not that for any link $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C : \zeta_{T \cup F}(C))$ there is

a unique corresponding set $S_l \in \mathcal{L} \setminus \{V\}$ and thus a unique corresponding edge in H, the edge $\{S_l, S'\}$ where S' is the parent of S_l.

We will now colour the tree H. The default colour for a vertex $S \in \mathcal{L}$ will be blue. For each active set C, we colour the smallest set $S \in \mathcal{L}$ that contains C red. We say that C is *associated* with S. Let H_r be the set of vertices of H coloured red. For each $S \in H_r$ let $\mathcal{C}(S)$ be the set of active sets associated with S.

Now consider any red vertex S. Now consider any active set C associated with S and any $l \in \delta_{F'}(C : \zeta_T(C))$. Let $\{S_l, S'\}$ be the edge in H corresponding to l. Then C is associated with either S_l or S' by property (4) of lemma 7. But C is associated with S, so S is one of the endpoints of this edge. Thus $\sum_{C \in \mathcal{C}(S)} |\delta_{F'}(C : \zeta_T(C))|$ is less than or equal to the degree in H of S, which we denote $\deg(S)$. Further, since each $C \in \mathcal{C}$ is associated with exactly one $S \in H_r$ we can say $\sum_{C \in \mathcal{C}} |\delta_{F'}(C : \zeta_T(C))| = \sum_{S \in H_r} \sum_{C \in \mathcal{C}(S)} |\delta_{F'}(C : \zeta_T(C))| \leq \sum_{S \in H_r} \deg(S)$.

Note that any inclusion-wise minimal witness set S is violated, and thus must contain an active set C, which would necessarily be associated with S. Thus such sets are coloured red. Furthermore, the leaves of H are the inclusion-wise minimal witness sets, with the possible exception of V which is not a witness set, but can be a leaf. Thus V is the only vertex of H that can possibly be a blue leaf.

Finally this allows us to conclude,

$$\begin{aligned}
\sum_{C \in \mathcal{C}} |\delta_{F'}(C : \zeta_T(C))| &\leq \sum_{S \in H_r} \deg(S) \\
&= \sum_{S \in H} \deg(S) - \sum_{S \in H \setminus H_r} \deg(S) \\
&= 2(|V(H)| - 1) - \sum_{S \in H \setminus H_r} \deg(S) \\
&\leq 2(|V(H)| - 1) - 2(|V(H)| - |H_r| - 1) - 1 \\
&= 2|H_r| - 1 \\
&< 2|\mathcal{C}|
\end{aligned}$$

where the third line holds because H is a tree, the fourth line holds because all but possibly one blue vertex is not a leaf and thus has degree at least 2, and finally the last line holds because each red vertex has at least one unique active set associated with it.

Thus, in order to prove Theorem 3, it suffices to prove Lemma 7.

2.7 Existence of a Witness Family

Fix an iteration in the algorithm. Let $\mathcal{C}$ be the set of active sets at the start of this iteration. Let F be the set of links chosen by the algorithm up to, but not including, this iteration. Recall that F' is the set of links returned by the algorithm.

We will prove Lemma 7 by proving a short sequence of lemmas, getting increasingly stronger. Here we simply prove that a witness family exists.

Lemma 8. *[18, Lemma 4.5] For all $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C : \zeta_{T \cup F}(C))$ there exists a witness set $S_l \subsetneq V$ such that:*

(1) $\delta_{F'}(S_l : \zeta_{T \cup F}(S_l)) = \{l\}$

(2) S_l is violated with respect to F

(3) For each $C \in \mathcal{C}$ either $C \subseteq S_l$ or $C \cap S_l = \emptyset$

That is, there exists a witness family.

Proof. Let $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C : \zeta_{T \cup F}(C))$. Specifically, $l \in F'$ and so the algorithm considered $F' \setminus \{l\}$ and found it to not be a feasible solution. Let S be the violated set with respect to $F' \setminus \{l\}$ the algorithm found. So there is no other element $f \in F' \setminus \{l\}$ such that $f \in \delta_{F'}(S : \zeta_{T \cup F}(S))$. So S satisfies (1). Note that all edges of F were added before l and so when considering l in the edge removal stage, no edge of F has been removed. Thus there is no edge $f \in F$ such that $f \in \delta_F(S : \zeta_T(S))$ so S is violated with respect to F. Since S is violated with respect to F, and each $C \in \mathcal{C}$ is active with respect to F, we get property (3). □

Lemma 9. *[18, Lemma 4.13] There exists a laminar witness family.*

Proof. From Lemma 8, we know that a witness family exists. We will form a laminar witness family by uncrossing any crossing pair of witness sets. Let A and B be crossing sets in the witness family. By Lemma 4 one if the following pairs are violated with respect to F: $A \setminus (B \cup \Gamma_T(B))$ and $B \setminus (A \cup \Gamma_T(A))$, $A \setminus B$ and $B \setminus A$, or $A \cap B$ and $A \cup B$. Replace A and B in the witness family with the appropriate pair of sets. We will show that this is still a witness family, and to show that we can obtain a laminar witness family, we show that performing this substitution decreases the number of pairs of crossing sets in the witness family. Note that A and B crossed, but the sets we replace them with do not.

Note that properties (2) and (3) still hold. Suppose A was the witness set for l_a and B for l_b. We now break into cases depending on what type of crossing sets A and B were.

Case 1: A and B are type I.

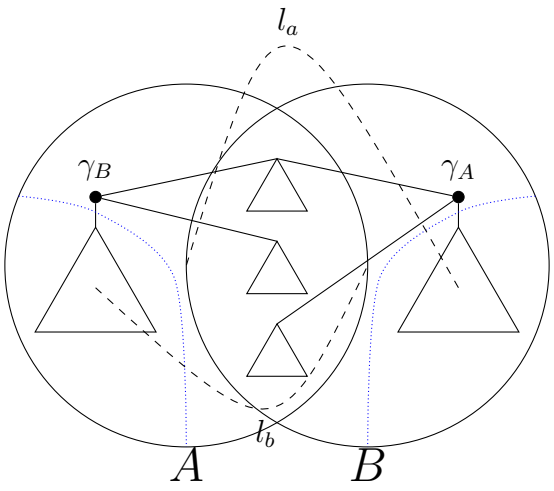


Figure 2.4: Type I Crossing Sets (arising from Case 1 of Lemma 9). Notice that one endpoint of l_a must be within $B \setminus (A \cup \Gamma_T(A))$ and one endpoint of l_b must be within $A \setminus (B \cup \Gamma_T(B))$.

Figure 2.4 illustrates the structure arising from this case. We know that $V = A \cup B$ and thus $\zeta_T(A) = B \setminus (A \cup \Gamma_T(A))$. And since A is violated $\zeta_T(\zeta_T(A)) = A$. Thus $\delta_{F'}(B \setminus (A \cup \Gamma_T(A)) : \zeta_T(B \setminus (A \cup \Gamma_T(A)))) = \delta_{F'}(\zeta_T(A) : \zeta_T(\zeta_T(A))) = \delta_{F'}(\zeta_T(A) : A) = \{l_a\}$. Similarly $\delta_{F'}(A \setminus (B \cup \Gamma_T(B)) : \zeta_T(A \setminus (B \cup \Gamma_T(B)))) = \{l_b\}$. Hence, property (1) still holds and thus we still have a witness family.

We now show that the number of pairs of crossing sets in our family has decreased.

First, suppose X is a set in the witness family which crosses both $A \setminus (B \cup \Gamma(B))$ and $B \setminus (A \cup \Gamma(A))$. Then X must have an element in $A \setminus B$ and another one in $B \setminus A$. So X crosses both A and B and thus uncrossing A and B has decreased the number of crossing pairs.

Now suppose X is a set in our witness family which crosses $A \setminus (B \cup \Gamma(B))$, but does not cross A. Thus there must be an element of X in $A \setminus (B \cup \Gamma_T(B)) \subseteq A$ and an element of X not in $A \setminus (B \cup \Gamma_T(B))$. However, this second element must still be within A and thus must be in $A \cap B$ or $\Gamma_T(B)$. If this second element is in $A \cap B$ then X crosses B. Suppose this second element is in $\Gamma_T(B)$. Recall that there is a single element of $\Gamma_T(B)$,

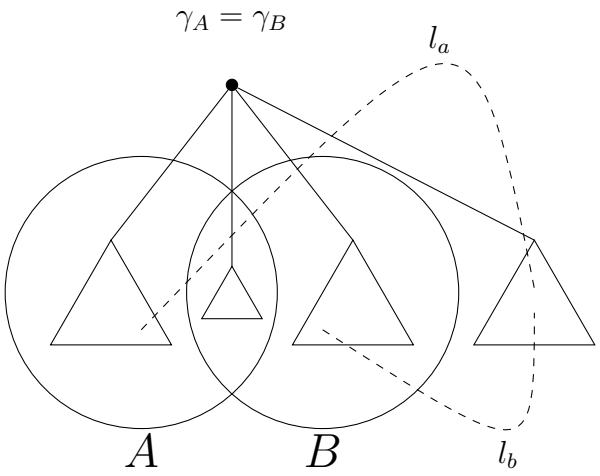


Figure 2.5: Type II Crossing Sets (arising from Case 2 of Lemma 9). Notice that one endpoint of l_a must be within $B \setminus A$ and one endpoint of l_b must be within $A \setminus B$.

call it γ_B. Consider the subtree rooted at γ_B. Note that since X contains b which has a neighbour in B which X cannot contain, in order for X to be violated, it must contain all of the subtree rooted at γ_B. We will show that this subtree is all of $A \setminus B$, and thus that $A \setminus (B \cup \Gamma_T(B)) \subsetneq X$ and thus doesn't cross X. Suppose there is a component of $A \setminus B$ that is not the subtree rooted at γ_B. Call the vertex set of this component X'. Since (V, T) must be connected, $\Gamma_T(X')$ is nonempty. But, as we saw earlier, $A \cup B = V$ and $\Gamma_T(X')$ cannot intersect $A \setminus B$. Thus $\Gamma_T(X')$ must be contained in B. But that means there is at least one vertex of X' in $\Gamma_T(B)$ a contradiction.

A symmetric argument shows that we cannot have a set X in our witness family which crosses $B \setminus (A \cup \Gamma(A))$, but does not cross B.

Case 2: A and B are type II.

Figure 2.5 illustrates the structure arising from this case. Recall that in this case $\Gamma_T(A) = \Gamma_T(B) = \Gamma_T(A \setminus B) = \Gamma_T(B \setminus A) = \Gamma_T(A \cap B) = \Gamma_T(A \cup B) = \{\gamma\}$.

First suppose that A and B are replaced with $A \setminus B$ and $B \setminus A$. Since these are violated, they must be covered by at least one link in F'. Let $uv \in \delta_{F'}(A \setminus B : \zeta_T(A \setminus B))$ with $u \in A \setminus B$. Note that $\zeta_T(A \setminus B) = B \cup (V \setminus (A \cup B \cup \{\gamma\}))$. If $v \in B$ then, since $A \setminus B \subseteq \zeta_T(B)$ we have that $uv \in \delta_{F'}(B : \zeta_T(B))$ and thus $uv = l_b$. If $v \in V \setminus (A \cup B \cup \{\gamma\}) \subseteq \zeta_T(A)$

then $uv \in \delta_{F'}(A : \zeta_T(A))$ then $uv = l_a$. A similar argument shows that if $uv \in \delta_{F'}(B \setminus A : \zeta_T(B \setminus A))$ then $uv = l_a$ or $uv = l_b$.

Thus $\delta_{F'}(A \setminus B : \zeta_T(A \setminus B)) \cup \delta_{F'}(B \setminus A : \zeta_T(B \setminus A)) \subseteq \{e_a, e_b\}$. It suffices to show that we cannot have l_a or l_b in both $\delta_{F'}(A \setminus B : \zeta_T(A \setminus B))$ and $\delta_{F'}(B \setminus A : \zeta_T(B \setminus A))$. This is true because a link in both of these cuts has an endpoint in $A \setminus B$ and the other in $B \setminus A$. But then this link would be in both $\delta_{F'}(B : \zeta_T(B))$ and $\delta_{F'}(A : \zeta_T(A))$, which is a contradiction.

Again, we now show that the number of pairs of crossing sets in our family has decreased. Suppose X crosses $A \setminus B$ but not A. Then we have that $\emptyset \neq (A \setminus B) \cap X \subseteq A \cap X$ and $\emptyset \neq (A \setminus B) \setminus X \subseteq A \setminus X$. So it must be that $X \setminus A = \emptyset$, that is $X \subseteq A$. This tells us that X does not cross $B \setminus A$. Also, $X \setminus (A \setminus B) \neq \emptyset$ so X intersects $A \cap B$. Thus we have that X crosses B. Similarly, if X crosses $B \setminus A$ but not B then it cannot cross $A \setminus B$ and necessarily does cross A.

Now suppose that A and B are replaced with $A \cap B$ and $A \cup B$. Again we see that we cannot have an link in both $\delta_{F'}(A \cap B : \zeta_T(A \cap B))$ and $\delta_{F'}(A \cup B : \zeta_T(A \cup B))$. Such a link has an endpoint in $A \cap B$ and one in $\zeta_T(A \cup B)$. But $\zeta_T(A \cup B) \subseteq \zeta_T(A)$ and $\zeta_T(A \cup B) \subseteq \zeta_T(B)$. Thus this link would be in both $\delta_{F'}(A : \zeta_T(A))$ and $\delta_{F'}(B : \zeta_T(B))$ which is impossible because $\delta_{F'}(A : \zeta_T(A)) \cap \delta_{F'}(B : \zeta_T(B)) = \{l_a\} \cap \{l_b\} = \emptyset$. However, since $A \cap B$ and $A \cup B$ are violated, they must be covered by at least one link in F'.

Let $uv \in \delta_{F'}(A \cap B : \zeta_T(A \cap B))$ with $u \in A \cap B$. Note that $\zeta_T(A \cap B) = (A \setminus B) \cup (B \setminus A) \cup \zeta_T(A \cup B)$. As stated above, It is impossible that $v \in \zeta_T(A \cup B)$. If $v \in A \setminus B \subseteq \zeta_T(B)$ then $uv \in \delta_{F'}(B : \zeta_T(B)) = \{l_b\}$ and thus $uv = l_b$. Similarly, if $v \in B \setminus A$ we get that $uv = l_a$.

Now let $uv \in \delta_{F'}(A \cup B : \zeta_T(A \cup B))$ with $u \in \zeta_T(A \cup B)$. Note that $A \cup B = (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$ but v cannot be in $A \cap B$. If $v \in A \setminus B$ then $uv \in \delta_{F'}(A : \zeta_T(A)) = \{l_a\}$ and if $v \in B \setminus A$ then $uv \in \delta_{F'}(B : \zeta_T(B)) = \{l_b\}$. So $\delta_{F'}(A \cup B : \zeta_T(A \cup B)) \cup \delta_{F'}(A \cap B : \zeta_T(A \cap B)) \subseteq \{l_a, l_b\}$ but neither l_a nor l_b can be in both cuts and thus one cut must contian l_a and the other l_b.

Again, we now show that the number of pairs of crossing sets in our family has decreased. Suppose X crosses $A \cap B$ but not A. Then we have that $\emptyset \neq (A \cap B) \cap X \subseteq A \cap X$ and $\emptyset \neq (A \setminus B) \setminus X \subseteq A \setminus X$. So it must be that $X \setminus A = \emptyset$, that is $X \subseteq A$. This tells us that X does not cross $A \cup B$. But X intersects $A \cap B$ and thus X crosses B. If X crosses $A \cup B$ but not A it must be the case that $A \subseteq X$ or $A \cap X = \emptyset$. If $A \subseteq X$ then $A \cup B \subseteq X$ and so $A \cap B$ and X do not cross. Also, $B \cap X \supseteq B \cap A \neq \emptyset$, $X \setminus B \subseteq X \setminus (A \cup B) \neq \emptyset$ and we must have that $B \setminus X \neq \emptyset$ else $A \cup B \subseteq X$. Thus B crosses X. If $A \cap X = \emptyset$

then $(A \cap B) \cap X = \emptyset$ so $A \cap B$ and X do not cross. Also, we must have that $B \cap X \neq \emptyset$, $B \setminus X \subseteq A \cap B \neq \emptyset$ and $X \setminus B \subseteq A \setminus (A \cup B) \neq \emptyset$ and so B and X cross.

□

We are now ready to prove Lemma 7, restated here, with slightly different terminology.

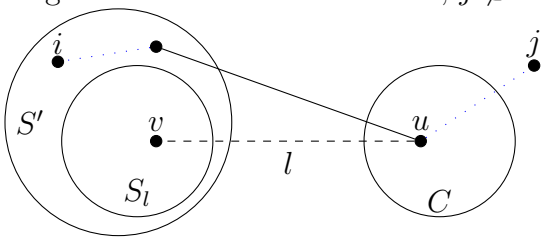
Lemma. *7. There exists a laminar witness family such that, if $l \in \bigcup_{C \in \mathcal{C}} \delta_{F'}(C, \zeta_T(C))$, and (S_l, S') is the edge in H defined by l, then the active set C crossed be l must be associated with either S_l or S'.*

Proof. We know a laminar witness family $\mathcal{L}$ exists. Suppose $l = uv \in \delta_{F'}(C, \zeta_T(C))$ for some $C \in \mathcal{C}$, such that the edge in H corresponding to l is (S_l, S'), but C is not associated with either S_l or S'.

Case 1: $C \cap S' = \emptyset$

In order for S' to not be a witness of l we must have that $u \in \Gamma_T(S')$. Let $v' \in S'$ be the neighbour of u in S'. Let $i \in S', j \in \zeta_T(S')$ be a pair of veretices such that $r_{ij} = 2$. Suppose $j \notin C$. The tree path between i and j must contain a vertex in $\Gamma_T(S')$ and since S' is violated this must be u. However, since the path enters C and we know it must also leave C there must be a vertex $\Gamma_T(C)$ other than u, a contradiction. This is shown in Figure 2.6. If $j \in C$ then $\zeta_T(S')$ crosses C and is also violated, contradicting that C is an active set. This is shown in Figure 2.7 Thus this case cannot happen.

Figure 2.6: Case 1 of Lemma 7, $j \notin C$



Case 2: $C \subseteq X \subsetneq S_e$ for some witness set X

As is illustrated in Figure 2.8, such an X is also a witness set for l so this case cannot happen.

Figure 2.7: Case 1 of Lemma 7, $j \in C$

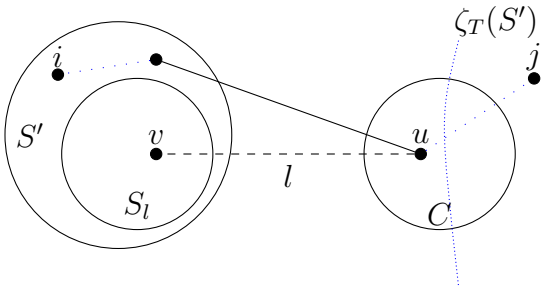
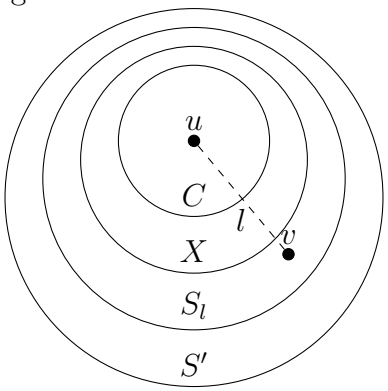


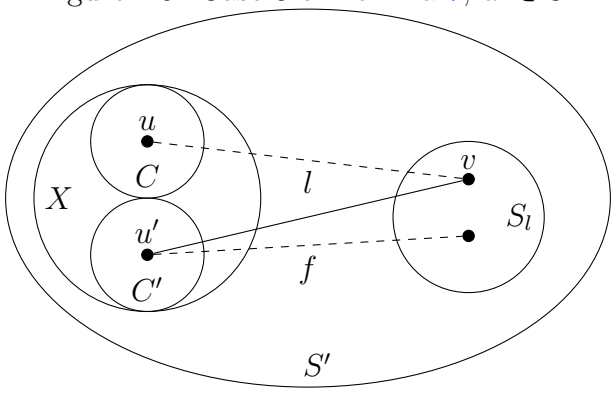
Figure 2.8: Case 2 of Lemma 7



Case 3: $C \subseteq X \subsetneq S'$ for some witness set X

In order for X to not be a witness set for l we must have $v \in \Gamma_T(X)$. Say $u' \in X \setminus C$ such that $u'v \in T$. In order for X and S_e to both be violated, u' must be the unique neighbour of S_e and v the unique neighbour of X. Thus u' and v are cut-nodes. Specifically, deleting either should disconnect some i, j such that $r_{ij} = 2$. without loss of generality, let $i \in X$ and $j \in S_e$. Note that we don't add more edges crossing $\delta(S_l, \zeta_T(S_l)$, except for l. So to stop v from being a cut vertex we must add a link f crossing $\delta(S_l, \Gamma_T(S_l)) = \delta(S_l, \{u'\})$. That is, one endpoint of f is u' and the other is in $S_l \setminus \{v\}$. Thus f is witnessed by X. The edges that are witnessed are, by definition, crossing the boundary of a currently active set. So one endpoint of f is in an active set C'. Suppose that $u' \in C'$, which is illustrated in Figure 2.9. Then we would have $C' \subsetneq X$. As a consequence, $u'v$ would be an edge in $\delta'_F(C', \zeta_T(C'))$ and u' has a path to C so an edge of that path would be a second edge in $\delta'_F(C', \zeta_T(C'))$. This would contradict that C' is active. So it must be that the other endpoint of f is in C' and hence $C' \subseteq S_e$, which is illustrated in Figure 2.10.

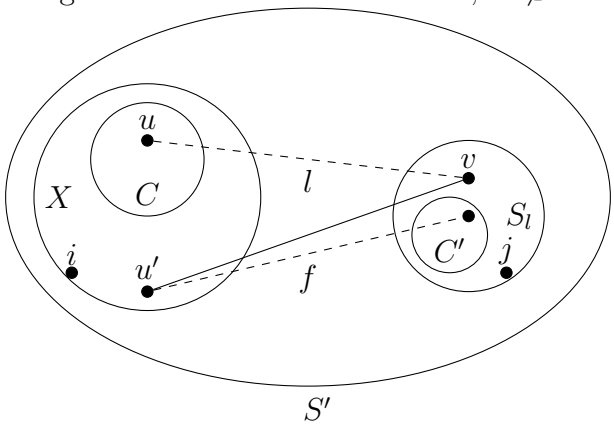
Figure 2.9: Case 3 of Lemma 7, $u' \in C'$



It cannot be the case that $C' \subseteq X' \subsetneq S_e$ for some X' in the laminar witness family. Such a set would be either a witness set for l or a witness set for f. So C' is associated with S_l.

If we replace S_l with $X - u'$ (which is a witness for l) and replace X with $S_l - v$ (a witness for f), C will be associated with $X - u'$ and C' with $S_l - u$.

Figure 2.10: Case 3 of Lemma 7, $u' \notin C'$



□

Now that we have proven Lemma 7 we have also proven Theorem 3.

Chapter 3

Beating Factor 2 in the Unweighted Case

3.1 Introduction

As previously mentioned, NC-TAP has not received much study since its inception by Frederickson and Jájá in their 1981 paper [10]. Algorithms, such as the one due to Ravi and Williamson, presented in the previous chapter, offer more robustness and generality than Frederickson and Jájá's algorithm. However, these were not designed to specifically solve NC-TAP, and in terms of approximation guarantee, do not outperform Frederickson and Jájá's algorithm. To the best of our knowledge, no improvements were made, even for special cases of NC-TAP, until Nutov gave an algorithm for the unweighted case of NC-TAP in [17] which has a factor $(1.916 + \epsilon)$ approximation guarantee.

But even this is really just another case of a more general algorithm being applied to NC-TAP. In 2013, Byrka, Grandoni, Rothvoss, and Sanita [3] developed an iterative, randomized rounding algorithm for finding approximately optimal solutions to the Steiner Tree problem. It is known that many combinatorial optimization problems reduce to the Steiner Tree problem. As such, in 2020, Byrka, Grandoni, and Ameli used this Steiner Tree algorithm to develop an algorithm for the Cactus Augmentation Problem [2]. Soon after, Nutov showed in [17] that the techniques in [2], which leverage the algorithm in [3], could be used to solve several related SNDPs, including NC-TAP.

In this chapter, we present these techniques of [17] and [2], which are the only improvements in NC-TAP for almost 40 years. In particular, we will prove the following.

Theorem 10. *[17, Theorem 1] There is a polytime algorithm for the unweighted NC-TAP that returns a set of links F such that $|F| \leq (4 \ln 2 - \frac{967}{1120} + \epsilon)|F^*|$ where F^* is an optimal solution to the same NC-TAP instance.*

3.2 Defining Incidence Graphs

Let $G = (V, E = T \cup L)$ be an (unweighted) instance of NC-TAP. Remember that T is a spanning tree of G and L is the set of links we can use to augment T to be 2-connected. Also recall that P_{uv} is the unique tree path in T between u and v. We use $|P_{uv}|$ to denote the number of edges in this path. Let $F \subseteq L$. It will be useful to think of F as an NC-TAP solution.

The goal is to transform an instance of NC-TAP to a special instance of a (node weighted) Steiner Tree Problem. To this end we will define a special auxiliary graph which captures the structure of the NC-TAP instance. To keep the definitions clear, it is useful to construct this auxiliary graph in stages. The following definitions are from [17]

Definition 5. The (F, T)-*incidence graph* has node set $F \cup T$ and edge set $\{le : f \in L, e \in T, e \in P_l\}$.

In the context of a Steiner Tree Problem, we will always consider the nodes in T to be the terminals and the nodes in L to be the Steiner nodes.

Definition 6. The *short-cut* (F, T)-*incidence graph* is obtained from the (F, T)-incidence graph by adding edges such that for every terminal node $e \in T$, the neighbours of e form a clique.

Definition 7. The *reduced* (F, T)-*incidence graph* is obtained from the short-cut (F, T)-incidence graph by removing any terminals $e \in T$ that are not leaf edges in (V, T). We let R denote the set of remaining terminals. So the node set of the reduced (F, T)-incidence graph is $F \cup R$, and in the context of the Steiner Tree Problem, R is the set of terminals and F is the set of Steiner nodes.

The figure 3.1 from [17] shows an NC-TAP instance, and the corresponding (F, T)-incidence graph and reduced (F, T)-incidence graph.

Normally, when talking about connectivity, we are concerned with the number of disjoint paths between two vertices. However, no elements of our auxiliary graphs correspond to vertices in the original graph. To remedy this, we make the following definition.

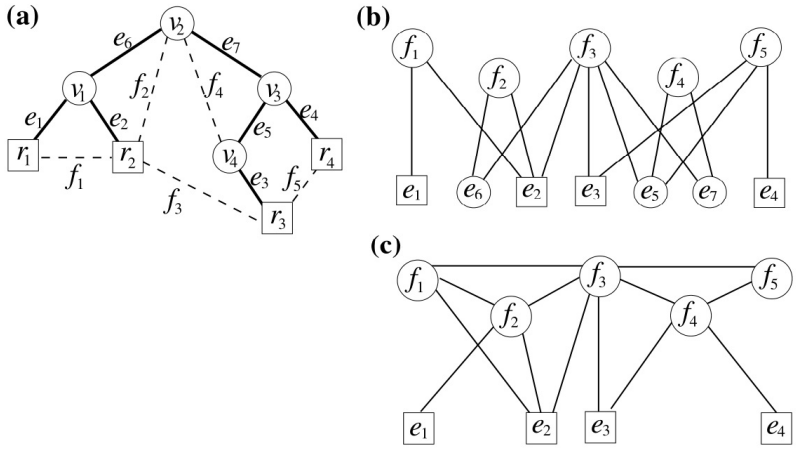


Figure 3.1: [17] (a) Shows an NC-TAP instance. (b) is the (F, T) incidence graph and (c) is the reduced (F, T)-incidence graph.

Definition 8. Let H be the (F, T)-incidence graph of G and let s, t be distinct vertices in V. Let e_s and e_t be the edges in P_{st} incident to s and t, respectively. We say that s and t are *H-reachable* if there is an e_s, e_t-path in H.

3.3 Reduction from NC-TAP to Steiner Tree

Now we present the lemma that is the main result of Nutov's paper, which shows that the Steiner Tree Problem on the auxiliary graphs we have defined is equivalent to the NC-TAP on the original graph. The substance of this reduction is the following lemma.

Lemma 11. *[17, Lemma 3] Let s, t be a pair of distinct vertices in V. There are two internally disjoint s, t-paths in $G' = (V, T \cup F)$ if and only if s and t are H-reachable.*

Before proving this, we show how it can be used to prove what we really want to show.

Lemma 12. *[17, Lemma 3] $G' = (V, T \cup F)$ is 2-connected if and only if the reduced (F, T)-incidence graph H_R has a path between every two terminals.*

Proof. We know G' is 2-connected if and only if there are 2 internally disjoint paths between every pair s, t of leaves of (V, T). This is true, by Lemma 11, if and only if s and t are

H-reachable. Since s and t are leaves, e_s and e_t are leaf edges and thus in R. We claim that an e_s, e_t-path exists in H if and only if an e_s, e_t-path exists in H_R.

Suppose that an e_s, e_t-path exists in H. For any node of this path which corresponds to a non-leaf edge e in T, note that e is not an end node of this path. Thus there is a node before and after it in the path. Call these l_1 and l_2 respectively. By the definition of the short-cut (F, T)-incidence graph, an edge is added between l_1 and l_2. Thus we can see that in the short-cut incidence graph, there is an e_s, e_t-path that is nearly identical to the one in H except that it skips any non-leaf terminals. This will also be an e_s, e_t-path in H_R.

Now suppose that an e_s, e_t-path exists in H_R. The only edges that this may use that do not exist in H are edges that go between nodes corresponding to links. Let $l_1 l_2$ be such an edge. Since $l_1 l_2$ was added when constructing the short-cut (F, T)-incidence graph, it must be the case that there exists a terminal e in H that is adjacent to both. Thus we can replace the edge $l_1 l_2$ with the path $l_1 e, e l_2$. Doing this for all such edges will produce an e_s, e_t-walk in H, which guarantees the existence of an e_s, e_t-path in H. □

The proof of Lemma 11 is a proof by induction. Since each part is non-trivial, for the sake of clarity, we first prove a lemma which captures the inductive step.

Lemma 13. *[17, Lemma 5] If $|P_{st}| \geq 3$ and the last two edges of P_{st} are uv and $vt = e_t$ then*

(i) If there are two internally disjoint paths between s and t in G' then there are two internally disjoint paths between s and v and between u and t in G'.

(ii) If s and v are H-reachable and u and t are H-reachable, then s and t are H-reachable.

(iii) If s and t are H-reachable, then s and v are H-reachable and u and t are H-reachable.

Proof. We prove the three parts in order.

Let C be the cycle in G' formed by the two internally disjoint s, t-paths. If $v \in C$ then we are done. Otherwise, consider P_{sv} (the unique s, v-tree path) and let a be the closest node to v such that a is also in C. Note such a node exists since s is in C. Note that vt and the s, t-path in C which does not use a, together make an s, v-path. also the subpath of P_{sv} from a to v and the s, a-path in C which does not use t is another s, v-path, internally disjoint from the first. See Figure 3.2. One can apply the exact same analysis to find two internally disjoint paths between t and u. This proves (i).

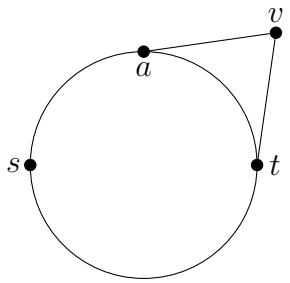


Figure 3.2: Existence of two internally disjoint s, v-paths given two internally disjoint s, t-paths, for Lemma 13 part (i)

Note that s and v being H-reachable means that there is a path between e_s and uv in H. Similarly, there is a path between uv and e_t in H. The union of these paths is an e_s, e_t-path and hence, s and t are H-reachable, proving (ii).

Let Q be an e_s, e_t-path in H. It suffices to show that there exists $f \in F$ that is a node in the path Q which is adjacent to the node uv. This is because taking the subpath of Q from e_s to f and then the edge from f to uv would be an e_s, uv path in H, which would mean that s and v are H-reachable. Similarly, the subpath of Q from e_t to f and then the edge from f to uv would be an e_t, uv path in H, which would mean that t and u are H-reachable. Suppose no such f exists. Let T_s be the connected component of $T \setminus \{uv\}$ containing s and T_t the same for t.Then let F_s be the set of edges in F with both endpoints in T_s. Similarly we define F_t. Note that no links can go from T_s to T_t as such a link would be adjacent to uv in H. But then F_s and F_t partition F and no link in F_s is adjacent to an edge in T_t in H. Similarly, no link in F_t is adjacent to an edge in T_s in H. Thus, there is no path from e_s, which is in T_s, and e_t, which is in T_t, contradicting the existence of Q. □

Proof of Lemma 11. Fix a pair s, t of leaves of (V, T) We prove the lemma by induction on $|P_{st}|$. When $|P_{st}| = 1$ we have that $e_s = e_t$ and thus s and t are trivially H-reachable.

We now show that Lemma 11 is true if $|P_{st}| = 2$. Let s, v, t be the nodes of P_{st}.

First suppose that there are two internally disjoint s, t-paths in G'. Thus there is an s, t-path P' in $G' - v$. Consider the sequence of components of $T - v$ that are visited by P'. Call them $B_1, \ldots, B_k$. Also, let l_i be the link used by P' to go from B_i to B_{i+1}. Now let e_i be the edge with one endpoint in B_i and the other endpoint being v. It is clear that this tree edge is on the tree path between the endpoints of any link which has exactly one endpoint in B_i, in particular, it is on the tree path between the endpoints of l_{i-1} and l_i.

Thus $e_1, l_1, e_2, l_2, \ldots, e_{k-1}, l_{k-1}, e_k$ is a path in H. Finally, since P' starts at s and ends at t we know that $e_1 = e_s$ and $e_k = e_t$ so the above path is actually an e_s, e_t-path and thus s and t are H-reachable.

We now show the induction to finish the proof. Suppose $|P_{st}| \geq 3$ and uv and $vt = e_t$ are the last two edges of P_{st}.

If there are two internally disjoint paths between s and t in G', then by Lemma 13(i) there are two internally disjoint paths between s and v and between u and t. Then, by induction, we know that s and v are H-reachable, and that u and t are H-reachable. Finally by Lemma 13(ii), we have that s and t are H-reachable.

If s and t are H-reachable, then by Lemma 13(ii) we have that s and v are H-reachable, and that u and t are H-reachable. The induction hypothesis guarantees that there are two internally disjoint s, v-paths and two internally disjoint u, t-paths. To see that this guarantees two internally disjoint s, t-paths, we apply Menger's theorem. Consider removing any vertex $x \in V \setminus \{s, t\}$. If x is not on s, t-tree path then clearly s and t are still connected. If x is on the path, but is not v, then there is still a path from s to v. This path, plus the edge vt is an s, t-path. Finally, if $x = v$ then there is still a path from t to u and the s, u-tree path is intact. The union of these two paths is an s, t-path. □

Suppose we are given an unweighted instance of NC-TAP, $G = (V, T \cup L)$. Consider the reduced (L, T)-incidence graph H_R. Let $F \subseteq L$ be a feasible solution to the Node-weighted Steiner Tree Problem on H_R. That is $H'_R := H_R[T \cup F]$ has a path between every two terminals. Note that H'_R is the reduced (F, T)-incidence graph. And thus, by Lemma 12, $G' := (V, T \cup F)$ is 2-connected. Thus, we have successfully transformed an instance of unweighted NC-TAP into the node-weighted Steiner Tree Problem. In total, our algorithm will take a given instance of unweighted NC-TAP $G = (V, T \cup L)$, construct the reduced (L, T)-incidence graph H_R, use some algorithm to obtain a Steiner Tree ST, find F, the set of Steiner nodes with non-zero degree in ST, and return F. So it suffices to develop an algorithm for the node-weighted Steiner Tree Problem with an approximation guarantee of better than 2.

It is important to note that the Steiner-Tree instance that we construct is not completely general. It satisfies a few properties that we will take advantage of.

Proposition 14. *[17] In H_R, each Steiner node is adjacent to at most two terminals.*

Proof. Note that the only terminals in H_R correspond to leaf edges in T and Steiner nodes correspond to links. A leaf edge is covered by a link if and only if the leaf is one of the

endpoints of the link. Thus, in H_R a Steiner node can only be adjacent to at most two terminals, since the corresponding link only has two endpoints. □

Proposition 15. *[17] The neighbours of any terminal in H_R are Steiner nodes and form a clique.*

Proof. We never add edges between terminals, so the neighbours of any terminal are Steiner nodes. And we explicitly add edges so that the neighbours of any terminal form a clique. □

Proposition 16. *[2, Lemma 2] Given a feasible solution F to the Node-Weighted Steiner Tree Problem on H_R, there is a spanning tree in $H_R[T \cup F]$ where all the terminals are leaves.*

Proof. If every terminal e has one neighbour in F, we are done. Suppose e has two or more neighbours in F. Call them l and l'. Since the neighbours of e form a clique, $el, ll', l'e$ is a cycle. Thus $H_R[T \cup F] - l'e$ is still connected and thus contains a tree. Repeating we can remove edges until e is a leaf. □

3.4 Approximation Guarantee

Byrka, Grandoni, Rothvoss, and Sanita [3] developed an iterative, randomized rounding algorithm for finding approximately optimal solutions to the Steiner Tree problem. However, taken as a black box, this algorithm is not good enough to prove an approximation guarantee of less than 2. However, by tweaking their analysis to the special instances of the Steiner Tree problem arising from unweighted NC-TAP instances, we can do better. The analysis of this subroutine for the Steiner Tree problem bounds the cost of the solution returned in terms of any feasible Steiner Tree ST^* rooted at an arbitrary node r. It also relies on what the authors call a *marking scheme* where some child edge of each internal node is marked. Every marking scheme defines a *witness set* $W(e)$, and $w(e) = |W(e)|$, for each edge e. $W(e)$ is the set of all pairs of terminals $\{t', t''\}$ such that the unique t', t''-path in ST contains the edge e and precisely one unmarked edge. The following lemma, which we use, is presented in [2], but follows from arguments of [3].

Lemma 17. *[2, , Lemma 3] Let $ST^* \subseteq E(H_R)$ be a feasible Steiner Tree and $\epsilon > 0$ a real number. Consider any marking scheme of ST^*. Then the Steiner-Tree subroutine runs in $n^{O_\epsilon(1)}$ time and the solution $ST \subseteq E(H_R)$ returned satisfies $|ST| \leq (1+\epsilon)\sum_{e \in ST^*} \mathbb{E}[H_{w(e)}]$ where H_i is the i-th harmonic number, ie $H_i = 1 + \frac{1}{2} + \ldots + \frac{1}{i}$.*

Since we have a special instance of the Steiner Tree problem which possesses the aforementioned properties, we can define a specific marking scheme, with a few properties that will be exploited in the analysis. It will be important for our marking scheme and analysis to define the following notation. For each Steiner node $l \in F^*$ let $d(l)$ be the number of children of l, $s(l)$ the number of children of l which are Steiner nodes, and $t(l)$ the number of children of l which are terminals. Note then that $d(l) = s(l) + t(l)$.

By Proposition 16, we may assume that the optimal Steiner Tree ST^* has every terminal being a leaf. Thus our marking scheme will only be defined for such trees. Root ST^* at some Steiner node r which is adjacent to at least one terminal. For each Steiner node $l \in F^*$, if $t(l) \geq 1$, select one terminal child of l uniformly at random and mark the edge from l to this child. If $t(l) = 0$ select a Steiner child of l uniformly at random and mark the edge from l to this child. Notice that we always favour marking edges that lead to terminals, and that there is precisely one marked edge for each Steiner node, independent of every random choice made by the marking scheme.

Proposition 18. *[2] Conditioning on e being an unmarked edge, $w(e) = 1$ deterministically.*

Proof. Every vertex is incident to at most two marked edges: one to a child and the other, possibly, its parent. Thus, from any vertex v, once you fix a direction, descending or ascending, their is a unique maximal path of marked edges, and this path continues to either descend or ascend until its end. Descending will lead to a terminal, ascending will always result in a Steiner node. Thus, very vertex v has a unique path of marked edges ending in a terminal, and that path descends at every step, when considering v the starting point.

Consider any pair of terminals $\{t', t''\}$ in $W(e)$. The tree path between t' and t'' must use e and since e is unmarked, this can be the only unmarked edge in the path. If $e = uv$ the tree path between t' and t'' must be the union of the tree path between u and t', the tree path between v and t'' and e. But there is a unique path of marked edges from u to a terminal and likewise for v. Thus there is a unique pair of terminals that contain e and no other unmarked edges. □

Lemma 17 bounds the number of edges in the output solution by a function that depends on the edges of another solution. However, we are concerned with the node-weighted Steiner Tree Problem. The following definition from [2] translates the values of $w(e)$ to the appropriate corresponding Steiner node.

Definition 9. Fix a marking scheme. For any Steiner node l the *cost* of l, denoted $c(l)$ is $\mathbb{E}[H_{w(m(l))}]$ where $m(l)$ is the unique marked child edge of l.

Figure 3.3 shows the marking scheme on an example Steiner tree. The red dashed edges are edges that are deterministically marked, because they go from the parent to the unique terminal child. The green dotted edges come in pairs where each have a probability of 1/2 of being marked. The black edges are deterministically unmarked. Three vertices of interest are labelled in Figure 3.3: the root, which is required for defining the marking scheme, and two Steiner nodes, l and l'. These are labelled since they have high cost. $c(l) = 2 + \frac{7}{120}$ and $c(l') = 2 + \frac{1}{12}$. This gives us motivation for the remaining sections of this chapter. l has high cost because it is far from an ancestor with a terminal child, and thus one might have many paths from l which use marked edges. l' has high cost simply because it has a large number of children.

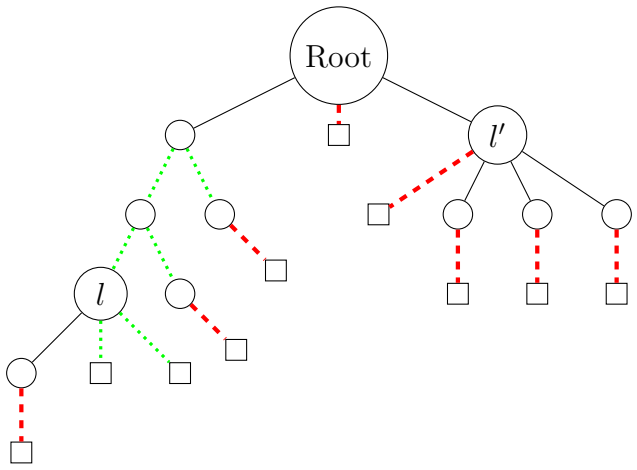


Figure 3.3: A Steiner tree which is marked via our random marking scheme. Red dashed edges are deterministically marked and green dotted edges have a 1/2 probability of being marked. $c(l), c(l') > 2$.

We can now use Lemma 17 to get a bound on the approximation guarantee of our whole algorithm.

Lemma 19. *[2, Lemma 4] Let $F^* \subseteq L$ be the optimal solution to a node-weighted Steiner Tree instance, such that every Steiner node is adjacent to at most two terminals and the neigbours of any terminal form a clique. The algorithm returns a solution F such that $|F| \leq (1+\epsilon)\mathbb{E}[\sum_{l \in F^*} c(l)] + 2\epsilon|F^*|$.*

Proof. By Proposition 16 there is a spanning tree $ST^* \subseteq E(H_R[T \cup F^*])$ where every terminal is a leaf in ST^*. Let $t := |T|$ and $s := |F|$. Thus $|ST^*| = s + t - 1$. Consider the marking scheme on ST^* given above.

By Lemma 17, the solution ST returned by the Steiner-Tree subroutine satisfies $|ST| \leq (1+\epsilon)\sum_{e \in ST^*} \mathbb{E}[H_{w(e)}]$.

Let ST^*_{mar} be the (random) set of marked edges and ST^*_{unm} the (random) set of unmarked edges. Every node in ST^*_{mar} corresponds to a Steiner node in F^*. In particular $\sum_{e \in ST^*_{\text{mar}}} \mathbb{E}[H_{w(e)}] = \sum_{l \in F^*} \mathbb{E}[H_{w(m(l))}] = \sum_{l \in F^*} c(l)$. Thus

$$\sum_{e \in ST^*} \mathbb{E}[H_{w(e)}] = \sum_{e \in ST^*_{\text{mar}}} \mathbb{E}[H_{w(e)}] + |ST^*_{\text{unm}}| = \mathbb{E}[\sum_{e \in ST^*_{\text{mar}}} H_{w(e)}] + t - 1$$

This last equality holds because, deterministically, there are precisely s marked edges in ST^*, as we marked one edge for each Steiner node. Hence, there are $t-1$ unmarked edges. The second equality holds by Proposition 18. So Lemma 17 tells us $|ST| \leq (1+\epsilon)\sum_{l \in F^*} c(l)$. Let F be the set of Steiner nodes with non-zero degree in ST. Now we can see that

$$\begin{aligned}
|F| &= |ST| - t + 1 \\
&\leq (1+\epsilon)\left(\sum_{l \in F^*} c(l) + t - 1\right) - t + 1 \\
&\leq (1+\epsilon)\sum_{l \in F^*} c(l) + \epsilon t \\
&\leq (1+\epsilon)\mathbb{E}[\sum_{l \in F^*} c(l)] + 2\epsilon|F^*|
\end{aligned}$$

□

By dividing both sides of the inequality guaranteed by Lemma 19 by $|F^*|$, we get

$$\frac{|F|}{|F^*|} \leq 2\epsilon + (1+\epsilon)\frac{1}{|F^*|}\sum_{l \in F^*} c(l)$$

We now know that, other then a small ϵ term, our approximation factor is given by the average cost of a Steiner node in our marking scheme.

3.5 Bounding the Cost of a Steiner Node

As previously stated, the first goal is to bound the cost of any Steiner node. To this end, we give the following lemma which gives a messy, technical, and specific bound for each Steiner node. We will refine this to a useable bound afterwards.

Lemma 20. *[2, Lemma 5] Let $l \in F^*$ be a Steiner node which is not the root. Consider the path $l = l_1, l_2, \ldots, l_q$ such that l_q is the lowest proper ancestor of l with a child that is a terminal. Then*

$$c(l) = \sum_{i=1}^{q-2} \frac{(d(l_{i+1}) - 1)H_{d(l_1)+\ldots+d(l_i)-i+1}}{d(l_2) \cdot \ldots \cdot d(l_{i+1})} + \frac{H_{d(l_1)+\ldots+d(l_{q-1})-q+2}}{d(l_2) \cdot \ldots \cdot d(l_{q-1})}$$

Proof. Let $ll' = e = m(l)$ the unique child edge of l which is marked. Recall $c(l) = \mathbb{E}[H_{w(e)}]$. The elements of $W(e)$ correspond one-to-one with paths in the tree between terminals such that the path contains e and precisely one unmarked edge. Recall earlier that we showed that starting at any non-leaf node, there is a unique path from that node to a terminal which uses only marked edges, and furthermore, each edge in these paths is descending. Thus, the paths corresponding to elements of $W(e)$ are the union of an unmarked edge e' and these unique paths of marked edges starting from the two endpoints of e'. Thus one of the endpoints of any unmarked edge we use must be an ancestor of l', since the paths in $W(e)$ must use e. Furthermore, this ancestor must be reachable via only marked edges.

Recall that $c(l) = \mathbb{E}[H_{w(m(l))}]$. We condition the expected value we are trying to compute on the event that the edge $l_i l_{i+1}$ is the first unmarked edge in the path $l_1, l_2, \ldots, l_q$. Note that $l_{q-1}l_q$ is deterministically unmarked. For $1 \leq i \leq q-2$ we know l_{i+1} does not have any terminal children and thus an edge is marked uniformly at random. Thus the probability that $l_i l_{i+1}$ is the first unmarked edge in the path $l_1, l_2, \ldots, l_q$ is $\frac{1}{d(l_2)} \cdot \ldots \cdot \frac{1}{d(l_i)} \frac{d(l_{i+1})-1)}{d(l_{i+1})}$ for all $1 \leq i \leq q-2$. For, $i = q-1$ the probability is $\frac{1}{d(l_2)} \cdot \ldots \cdot \frac{1}{d(l_i)}$.

We now need to count the umber of paths contributing to $w(e)$, given that $l_i l_{i+1}$ is the first unmarked edge in the path $l_1, l_2, \ldots, l_q$. To do this we focus on which unmarked edge we can include in a path that contributes to $w(e)$. For any $1 \leq j \leq i$, we could use any of the $d(l_j) - 1$ unmarked child edges of l_j. We could also the unmarked edge $l_i l_{i+1}$. This gives a total of $d(l_1) - 1 + \ldots + d(l_i) - 1 + 1 = d(l_1) + \ldots + d(l_i) - i + 1$ paths that contribute to $w(e)$. Thus

$$c(l) = \sum_{i=1}^{q-2} \left(\frac{d(l_{i+1}) - 1}{d(l_2) \cdot \ldots \cdot d(l_{i+1})} H_{d(l_1)+\ldots+d(l_i)-i+1} \right) + \frac{H_{d(l_1)+\ldots+d(l_{q-1})-q+2}}{d(l_2) \cdot \ldots \cdot d(l_{q-1})}$$

□

The right hand side of the equation in Lemma 20 motivates the following definition, from [2].

Definition 10. For positive integer i we denote by $\hat{H}_i$ the *i-th super-harmonic number* where

$$\hat{H}_i := \sum_{j \geq 0} \frac{1}{2^{j+1}} H_{i+j} = \frac{1}{2} H_i + \frac{1}{4} H_{i+1} + \frac{1}{8} H_{i+2} + \ldots$$

We will now present a series of properties of these super-harmonic numbers and how they relate to the harmonic numbers. These proofs are mostly just algebraic manipulations, so we defer their proofs to Appendix A.

Proposition 21. *For any positive integer i, $\hat{H}_{i+1} = 2\hat{H}_i - H_i$.*

Proposition 22. *[2] For any positive integer i, $\hat{H}_{i+1} - \hat{H}_i = \hat{H}_i - H_i = \sum_{k \geq 1} \frac{1}{(i+k)2^k}$.*

Proposition 23. *For any positive integer i, $\hat{H}_{i+1} - \hat{H}_i = \hat{H}_i - H_i \leq \frac{1}{i+1}$*

Proposition 24. *$\hat{H}_1 = 2 \ln 2$.*

Now we introduce some extra machinery and propositions to move us from the very local bounds obtained in Lemma 20 to useful ones. For the remainder of this section for a finite sequence of positive integers, $S = (d_1, \ldots, d_k)$ we define

$$f(S) = \sum_{i=1}^{k-1} \frac{(d_{i+1} - 1) H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}} + \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k}$$

Notice that this is the right hand side of the equation of Lemma 20 when S is the sequence of degrees of the path from the given Steiner node to its lowest proper ancestor with a terminal child. However, we can now extend the definition. For an infinite sequence of positive integers $S' = (d_1, d_2, \ldots)$ we define

$$f(S') = \sum_{i \geq 1} \frac{(d_{i+1} - 1) H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}}$$

Lemma 20 gives us a finite sum, we would like to compare this to the super-harmonic numbers, an infinite sum. So for any finite sequence of positive integers $S = (d_1, \ldots, d_k)$ we let $\bar{S} = (d_1, \ldots, d_k, 2, 2, \ldots)$ be the infinite extension of S obtained by appending an infinite sequence of 2 to S.

Proposition 25. *[2] $f(S) \leq f(\bar{S})$ for any finite sequence of positive integers, $S = (d_1, \ldots, d_k)$*

Proposition 26. *[2] Let $\bar{S} = (d_1, \ldots, d_k, 2, 2, \ldots)$ and assume that for some $i \geq 2$ we have $d_i = 1$. Let $\bar{S}_i = (d_1, \ldots, d_{i-1} - 1, d_{i+1} + 1, d_k, 2, 2, \ldots)$. Then $f(\bar{S}) = f(\bar{S}_i)$.*

The proofs of Proposition 25 and Proposition 26 are also deferred to Appendix A. We now have what we need to prove a useful bound on the cost of a Steiner node.

Lemma 27. *[2, Lemma 5] For any Steiner node $l \in F^*$, $c(l) \leq \hat{H}_{d(l)}$.*

Proof. By Propositions 25 and 26, together with Lemma 20, it is sufficient to show that $f(S) \leq \hat{H}_{d_1}$ for any infinite sequence $S = (d_1, \ldots, d_k, 2, 2, \ldots)$ with $k \geq 2$ and $d_i \geq 2$ for all $i \geq 2$. The proof proceeds by induction on k.

When $k = 2$,

$$f(S') = \frac{(d_2 - 1)H_{d_1}}{d_2} + \frac{\hat{H}_{d_1+d_2-1}}{d_2} \leq \frac{H_{d_1}}{2} + \frac{\hat{H}_{d_1+1}}{2} = \hat{H}_{d_1}$$

For $k > 2$ we define $S' = (d_1 + d_2 - 1, d_3, \ldots, d_k, 2, 2, \ldots)$. Thus,

$$f(S) = \frac{(d_2 - 1)H_{d_1}}{d_2} + \frac{f(S')}{d_2} \leq \frac{(d_2 - 1)H_{d_1}}{d_2} + \frac{\hat{H}_{d_1+d_2-1}}{d_2} \leq \frac{H_{d_1}}{2} + \frac{\hat{H}_{d_1+1}}{2} = \hat{H}_{d_1}$$

Where the first inequality holds by the induction hypothesis. □

Consider applying this bound to the example we saw above, which is reproduced here in Figure 3.4. Most nodes have at most two children and thus we can bound there cost by $\hat{H}_2 = 2\ln 2 - 1 < 1.8$. But the root and l have 3 children, and thus we can only bound their cost by $\hat{H}_3 = 8\ln 2 - \frac{7}{2} > 2.045$. Worse, l' has 4 children and thus the bound we would get on its cost is $\hat{H}_4 = 16\ln 2 - \frac{53}{6} > 2.257$. This further suggests that we need more techniques.

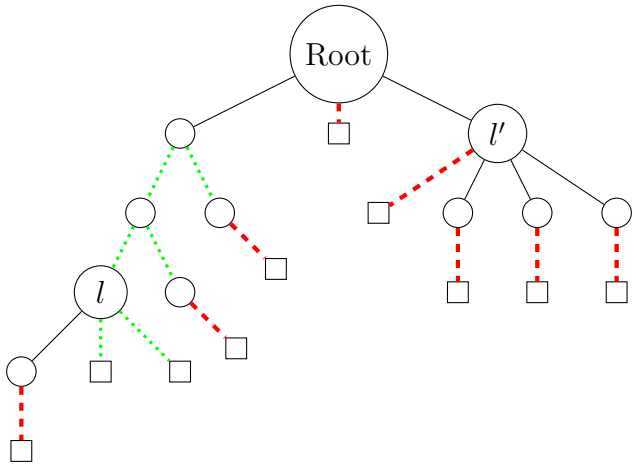


Figure 3.4: Reproduction of Figure 3.3.

3.6 Grouping

We have just seen that we can bound the cost of any Steiner node $l \in F^*$ by $\hat{H}_{d(l)}$. However, obtaining a constant approximation guarantee for the node-weighted Steiner Tree problem using only this bound is hopeless. In order to do that, we rely on two key insights. The first is that what we really need to bound is the average cost, that is $\frac{1}{|F^*|}\sum_{l \in F^*} c(l)$. The second is that for any Steiner node l with no Steiner children, $c(l) \leq \hat{H}_{d(l)} = \hat{H}_{t(l)} \leq \hat{H}_2$. That is, these Steiner nodes with no Steiner children have relatively low cost, and we can use that to offset the cost of more expensive Steiner nodes. Again the following definition is from [2].

Definition 11. A Steiner node l is a *leaf-Steiner node* if it has no Steiner children. Otherwise we call l an *internal-Steiner node*.

We will let F^*_{lf} and F^*_{in} be the sets of leaf-Steiner and internal-Steiner nodes, respectively. The plan is to partition the Steiner nodes into groups such that each group contains one internal-Steiner node l, and $s(l) - 1$ leaf-Steiner nodes. We do this via the following algorithmic process.

Algorithm 2: Grouping Steiner Nodes

1 Initialize all Steiner nodes as unprocessed
2 **while** *there exists an unprocessed node* $l \in F^*_{in}$ **do**
3 Choose such an l whose children are either processed or leaf-Steiner nodes
4 Mark l as processed
5 Initialize $g(l) \leftarrow \{l\}$
6 **for** *each Steiner child* l' *of* l **do**
7 Consider the subtree rooted at l'
8 Let l'' be the unique unprocessed leaf-Steiner node in this subtree
9 Mark l'' as processed
10 $g(l) \leftarrow g(l) \cup \{l''\}$
11 **end**
12 Arbitrarily choose some $l'' \in g(l) \setminus \{l\}$
13 Mark l'' as unprocessed
14 $g(l) \leftarrow g(l) \setminus \{l''\}$
15 **end**
16 Let l^* be the remaining unprocessed leaf-Steiner node
17 Mark l^* as processed
18 **return** $(g(l))_{l \in F^*_{in}}$ *and* l^*

Clearly, this process relies on the truth of the claim made in line 8: that there is a unique unprocessed leaf-Steiner node l'' in the subtree rooted at l'. We also implicitly make the claim that when we are done processing the internal-Steiner nodes, there is exactly one unprocessed leaf-Steiner node. Both of these follow from the following claim.

Lemma 28. *The While loop on line 2 of Algorithm 2 maintains the invariant that the subtree rooted at l, where l is either an unprocessed leaf-Steiner node or at a processed internal-Steiner node that does not have a processed parent, contains exactly one unprocessed leaf-Steiner node.*

Proof. Clearly if l is an unprocessed leaf-Steiner node, the invariant is true because l is the only Steiner node in the subtree rooted at l.

Also, when the algorithm begins there are no processed internal-Steiner nodes so the invariant is true when the algorithm begins.

When processing an unprocessed internal-Steiner node l, we know that the children of l are either leaf-Steiner nodes, or processed internal-Steiner whose parent is unprocessed. Note that the leaf-Steiner children of l must be unprocessed because leaf-Steiner nodes

only become processed when processing their ancestors, and internal-Steiner nodes are processed from the bottom of the tree, up towards the root. Thus, the children of l satisfy the hypothesis of the invariant. By induction, the subtree rooted at each child contains exactly one unprocessed leaf-Steiner node. We process all of these, but then unprocess one of them. Thus the subtree rooted at l will have exactly one unprocessed leaf-Steiner node and the invariant still holds. □

We define $a(l) = \frac{1}{|g(l)|} \sum_{l' \in g(l)} c(l')$, for all $l \in F^*_{in} \cup \{l^*\}$. The important thing is that

$$\frac{1}{|F^*|} \sum_{l \in F^*} c(l) \leq \max\{a(l) : l \in F^*_{in} \cup \{l^*\}\}$$

Thus if $a(l) \leq \alpha$ for all $l \in F^*_{in} \cup \{l^*\}$ then, $(1+\epsilon)\alpha + 2\epsilon$ is a bound on the approximation guarantee of the algorithm.

To see how effective this technique is, we again refer to the Steiner tree example seen earlier. It is reproduced in Figure 3.5, now also showing l^* and the groups of all of the Steiner nodes. l' is in a group along with two leaf Steiner nodes. Thus the average cost of a node in this group is $\frac{1}{3}\left(2 + \frac{1}{12} + 1 + 1\right) = \frac{49}{36} < 1.37$. Unfortunately l is in a group all by itself, so we will have to employ another technique, which we will see in the next section.

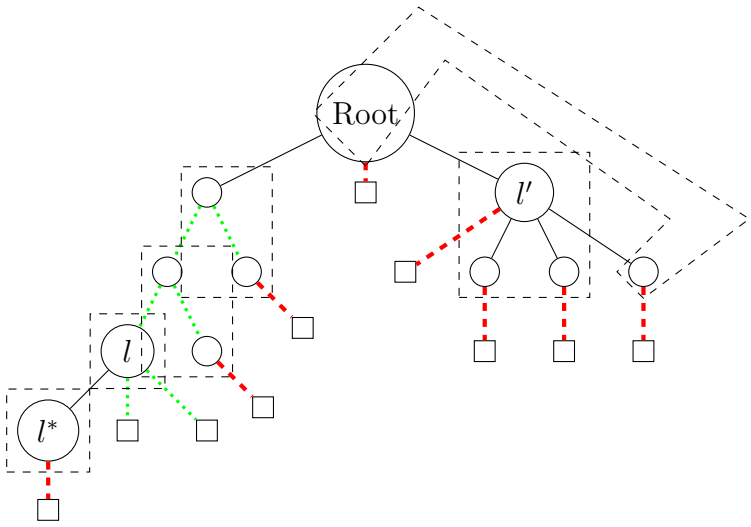


Figure 3.5: A possible grouping for our example Steiner tree. Note one leaf Steiner node has been designated l^*. $c(l')$ can now be shared with two other leaf Steiner nodes. Unfortunately, l is in a group by itself.

We now bound the averaged cost for an arbitrary group. For l^*, we have $a(l^*) = c(l^*) \leq \hat{H}_2$. For any $l \in F^*_{\text{in}}$ we have

$$a(l) = \frac{1}{|g(l)|} \sum_{l' \in g(l)} c(l') \leq \frac{\hat{H}_{s(l)+2} + (s(l)-1)\hat{H}_2}{s(l)} = \frac{\hat{H}_{s(l)+2} - \hat{H}_2}{s(l)} + \hat{H}_2$$

Unfortunately, when $s(l) = 1$ we get the case we saw with the vertex labelled l in Figure 3.5, where $a(l) \leq \hat{H}_3 = 8\ln 2 - \frac{7}{2} \approx 2.045$. The problem is that the terminal children of l increase the cost of l but, unlike the Steiner children of l, they don't contribute to normalizing the average cost of the group.

3.7 Discharging

We have seen that we can bound the cost of any Steiner node $l \in F^*$ by $\hat{H}_{d(l)}$. However, this bound is not good enough to give us an improved approximation ratio for the node-weighted Steiner Tree problem. We mentioned that the problematic nodes are those with a non-zero number of terminal children, which leads to the following definition from [2].

Definition 12. A Steiner node l is a *good father* if it has at least one terminal child and a *bad father* if it has zero terminal children. Steiner nodes which are children of good fathers are called *good* and Steiner nodes which are not good are called *bad*.

We left $F^*_{gf}, F^*_{bf}, F^*_g, F^*_b$ be the sets of good fathers, bad fathers, good nodes, and bad nodes, respectively, in F^*.

We can immediately improve the bound on the cost of good Steiner nodes.

Proposition 29. *For any $l \in F^*_g$ we have $c(l) \leq H_{d(l)}$.*

Proof. If l is the root, we saw that $c(l) \leq H_{d(l)-1} \leq H_{d(l)}$. Otherwise, let l' be the parent of l. Since l is good, we know l' is a good father, and thus has terminal children. Thus, $l'l$ is deterministically unmarked. Let l'' be the child of l such that $l''l$ is marked. Similar to the proof for Lemma 20, we know that paths contributing to $w(e)$ are uniquely determined by their unmarked edge. Further, we know that one endpoint of such an unmarked edge must be an ancestor of l'' which is reachable via marked edges. The only way this is possible is if l is this endpoint. Thus the unmarked edge used must be incident to l. There are $d(l) - 1 + 1 = d(l)$ such edges so $c(l) \leq H_{d(l)}$. □

Unfortunately, the nodes that give rise to groups whose average cost is greater than 2 are, in this new vocabulary, the good fathers, not the good nodes. There could still be a node that is bad, but a good father. Thus we have yet to improve our approximation guarantee. The last idea will be that we will use the cost savings experienced by good nodes to offset the cost of the good fathers. More precisely, we will discharge some of the cost of the good father to its Steiner children, which are by definition, good.

Figure 3.6 shows the same Steiner tree example we have been looking at throughout this chapter. This time, every Steiner node is labelled with a g or a b, depending on if it is good or bad, and with gf or bf depending on if it is a good father or a bad father. Note that l is bad and a good father. l only has a cost above 2, in part at least, because it is a good father, and has terminal children. It's Steiner child, l^* has a very low cost of 1. If l^* could take on some of the cost for l, we could easily show that that the average cost of the Steiner nodes is strictly less than 2. Of course, we want the procedure to be general, so we will discharge cost from every good father to its Steiner children. Say every good father sends p of its cost to each of its Steiner children, which is denoted in Figure 3.6 by blue arrowheads. What value of p would result in the most uniform distribution of costs?

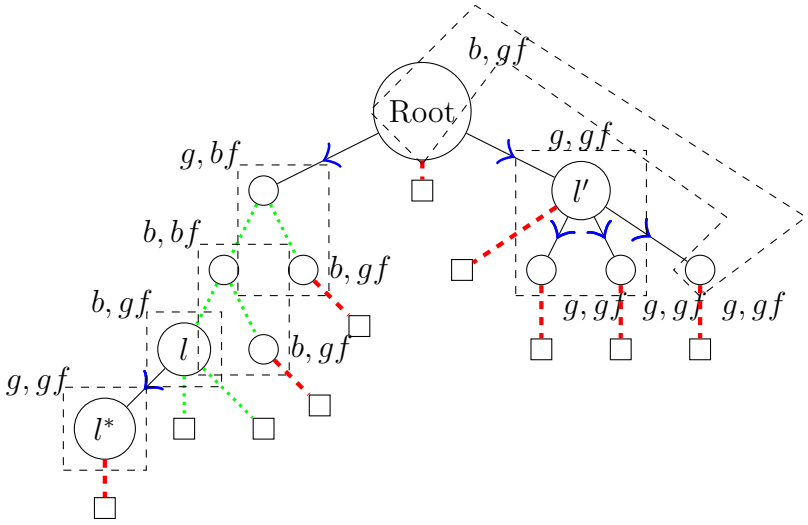


Figure 3.6: Example Steiner Tree with possible grouping. Steiner nodes are labelled as good or bad and as good fathers or bad fathers. Blue arrows denote the discharging of cost from a good father to its Steiner children.

Note that the discharging that happens on the right hand side of the root happens within a group, or happens evenly between the groups $g(\text{Root})$ and $g(l')$ and thus cancels

out. The only good nodes that are increasing in cost are the node directly left of the root, and l^*. The node directly to the left of the root is part of a group with one other leaf Steiner node and this group has an average cost of $\frac{11}{8}$ so after discharging it has a cost of $\frac{11}{8} + \frac{p}{4}$. After discharging, l^* will have a cost of $1 + p$. Thus, in order to ensure that after discharging, all groups have an average cost of strictly less than 2, we have to pick $p \in \left(\frac{7}{120}, 1\right)$. In fact, we can optimize our choice of p to get the best bound possible. If we select p such that $1 + p = 2 + \frac{7}{120} - p$ we get $p = \frac{127}{240}$. After discharging with this p one can check that all groups have an average cost of at most $1 + \frac{127}{240} < 1.53$. Thus we know that the algorithm due to Byrka, Grandoni, Rothvoss, and Sanita will return a Steiner tree whose size is at most 1.53 times the size of our example.

We will now formalize the ideas used in the example above to work in general. Let $p \in [0, \hat{H}_2 - H_2 = 4 \ln 2 - \frac{5}{2}]$. This is a parameter which we will optimize later. The discharging rule is simple: every good Steiner nodes increases its own cost by p by taking this amount away from the cost of its parent, which is a good father. Let c' be the cost function after discharging. Then it is clear that for any Steiner node l,

$$c'(l) \leq \begin{cases} H_{d(l)} + p - s(l)p & \text{if } l \in F_g^* \cap F_{gf}^* \\ H_{d(l)} + p & \text{if } l \in F_g^* \cap F_{bf}^* \\ \hat{H}_{d(l)} - s(l)p & \text{if } l \in F_b^* \cap F_{gf}^* \\ \hat{H}_{d(l)} & \text{if } l \in F_b^* \cap F_{bf}^* \end{cases}$$

Now we can reanalyze the average cost of each group. If l^* is good then $c(l^*) \leq H_2 + p \leq \hat{H}_2$ and if it is bad then $c(l^*) \leq \hat{H}_2$. For any $l \in F_{\text{in}}^*$

$$a(l) \leq \begin{cases} a_{ggf}(s(l)) = \frac{H_{s(l)+2} - \hat{H}_2 + p}{s(l)} + \hat{H}_2 - p & \text{if } l \in F_g^* \cap F_{gf}^* \\ a_{gbf}(s(l)) = \frac{H_{s(l)} - \hat{H}_2 + p}{s(l)} + \hat{H}_2 & \text{if } l \in F_g^* \cap F_{bf}^* \\ a_{bgf}(s(l)) = \frac{\hat{H}_{s(l)+2} - \hat{H}_2}{s(l)} + \hat{H}_2 - p & \text{if } l \in F_b^* \cap F_{gf}^* \\ a_{bbf}(s(l)) = \frac{\hat{H}_{s(l)} - \hat{H}_2}{s(l)} + \hat{H}_2 & \text{if } l \in F_b^* \cap F_{bf}^* \end{cases}$$

We know that for any approximation ratio α, we must have that

$$a_{ggf}(n), a_{gbf}(n), a_{bgf}(n), a_{bbf}(n) \leq \alpha$$

Also, we want to select p such that α is as small as possible. If we can restrict the domain of n to a finite set, then our goal is to minimize a value subject to a finite number of linear constraints, which is easily done via the theory of Linear Programming.

Proposition 30. *[2] For any real $p \in [0, \hat{H}_2 - H_2]$, a_{ggf} obtains its global maximum in the range $\{1, 2, 3\}$, a_{gbf} obtains its global maximum when $n = 1$, a_{bgf} obtains its global maximum in the range $\{6, 7, 8\}$, and a_{bbf} obtains its global maximum when $n = 7$.*

The proof of Proposition 30 is deferred to Appendix B

This gives rise to the following linear program, with variables α and p.

$$\begin{aligned}
&\min \alpha \\
&\text{subject to} \\
&\qquad \alpha \geq H_3 \\
&\qquad \alpha \geq \frac{H_4 + \hat{H}_2 - p}{2} \\
&\qquad \alpha \geq \frac{H_5 + 2\hat{H}_2 - 2p}{3} \\
&\qquad \alpha \geq \frac{H_6 + 5\hat{H}_2 + p}{6} \\
&\qquad \alpha \geq \frac{H_7 + 6\hat{H}_2 + p}{7} \\
&\qquad \alpha \geq \frac{H_8 + 7\hat{H}_2 + p}{8} \\
&\qquad \alpha \geq \hat{H}_3 - p \\
&\qquad \alpha \geq \frac{\hat{H}_7 + 6\hat{H}_2}{7} \\
&\qquad p \leq \hat{H}_2 - H_2 \\
&\qquad p \geq 0
\end{aligned}$$

The optimal solution to this linear program is $p = \frac{7\hat{H}_3 - 6\hat{H}_2 - H_7}{8} = 4\ln 2 - \frac{2953}{1120}$ and $\alpha = \frac{\hat{H}_3 + 6\hat{H}_2 + H_7}{8} = 4\ln 2 - \frac{967}{1120} \approx 1.9092$.

This, together with Lemma 19, proves Theorem 10.

Chapter 4

Towards a General Algorithm Beating Factor 2

4.1 Introduction

Since the barrier of a better than 2-approximation for unweighted NC-TAP has been broken, it is reasonable to attempt to do this for the weighted case. To the best of our knowledge, the techniques in [2] and [17] have little hope of extending to weighted NC-TAP. Instead, we attempt to find direction from another avenue, the bounded-costs case of the (edge-connectivity) Tree Augmentation Problem (TAP). Recall, this problem has the same input as NC-TAP, but a solution is a set of links $F \subseteq L$ such that $(V, T \cup F)$ is 2-edge-connected. For over 30 years, the best algorithms for TAP could only achieve an approximation guarantee of 2, matching the algorithm of Frederickson and Jájá. In 2017, Adjiashvili presented an improvement for the special case that the costs of the links are bounded by a constant.

Lemma 31. *[1, Theorem 1.1] For any fixed $M \in \mathbb{R}_{\geq 1}$ and $\epsilon \in \mathbb{R}_{>0}$ there is an polynomial time $(1.96418 + \epsilon)$-approximation algorithm for TAP, restricted to instances where the cost vector c satisfies $c_l \leq M$ for all $l \in L$.*

Several papers have since improved on Adjiashvili's redsult. First, Fiorini, Groß, Könemann, and Sanità in 2017 [9] were able to give an algorithm which gives a $\frac{3}{2} + \epsilon$-approximation guarantee for TAP under the same conditions: that the costs of the links are bounded by a constant. In parallel, Nutov [16] was able to give an algorithm with

a $\frac{12}{7} + \epsilon$-approximation guarantee. While this is a worse approximation guarantee than that of [9], Nutov's algorithm allows for the costs of the links to be bounded by a function logarithmic in the input size, rather than a constant. Finally, in 2018 Grandoni, Kalaitzis, and Zenklusen [12] improved on Adjiashvili's result in the unweighted case, obtaining a 1.458-approximation algorthm. However, as mentioned in Chapter 1 Section 1.1, we were not able to successfully use Adjiashvili's techniques to develop an algorithm for NC-TAP. As such, we do not explore the extensions of [9, 16, 12] in this thesis.

We recall for the reader, the Cut LP for TAP.

$$\begin{aligned} \min \sum_{e \in L} c_e x_e & \\ x(\text{cov}(e)) &\geq 1 \quad \forall e \in T \\ x &\geq 0 \end{aligned}$$

where $\text{cov}(e) = \delta_F(S : S')$ and S, S' are the vertex sets of the components of $T - v$.

4.2 Summary of Adjiashvili's Results

Adjiashvili's 2017 paper introduced a few major insights into TAP. However, his main ideas are all rooted in his discovery of an algorithm for solving TAP instances where the running time was a function of the number of leaves. This gives us a clear idea for an algorithm. First, decompose the tree into subtrees which have a bounded number of leaves. Second, get an optimal solution for each of these subtrees. Finally, combine these solutions for subtrees into an optimal solution for the original instance. Of course, this is not exactly what Adjiashvili's algorithm does, and it likely isn't possible, but it captures the general idea of the algorithm and motivates the following definition.

Definition 13. Given a tree T and positive number β, a *β-bundle* is a union of at most β, not necessarily disjoint, paths in T, such that this union is connected. The set of all β-bundles in T is denoted $\mathcal{B}_\beta$.

Note that in [1], Adjiashvili's definition of β-bundles do not require that they are connected. However, it was pointed out by Nutov in [16] that we only need to consider β-bundles which are connected. Figure 4.1, which is a modification of a figure from [1], illustrates a 3-bundle in a tree. The idea is that a (connected) β-bundle, B, is a subtree

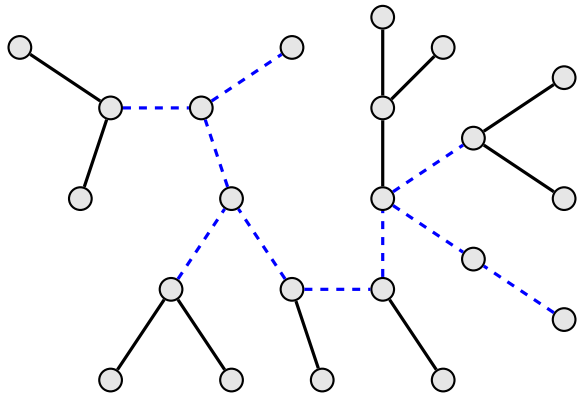


Figure 4.1: [1] The dashed green edges can be obtained as a union of three paths, hence they comprise a 3-bundle.

of the TAP/NC-TAP instance with a bounded number of leaves and thus we can use Adjiashvili's algorithm to determine the cheapest set of links that form a TAP or NC-TAP solution for the subtree given by B. Say that the cheapest set of links that forms a TAP solution for B, which is equivalent to saying the set of links covers the edges in B, has cost $\text{OPT}'(B)$. A solution to the original TAP instance also must cover the edges in B, and thus among all of the links in $\text{cov}(B) := \bigcup_{e \in B} \text{cov}(e)$, we must use a subset of cost at least $\text{OPT}'(B)$. This leads of the following family of valid constraints.

$$\sum_{l \in \text{cov}(B)} c_l x_l \geq \text{OPT}'(B) \quad \forall B \in \mathcal{B}_\beta$$

With the addition of these constraints to the Cut LP, we arrive at what Adjiashvili calls the *Bundle LP*. What these constraints ensure is that, at least for any bundle, that we don't allow a fractional solution which is cheaper than the integer optimal. Thus, the integrality gap of the Bundle LP will be 1 if the instance is a β-bundle. We will also add the similar bundle constraints to the Partition LP. Any link that contributes to preventing a vertex in B from being a cut node must be in $\text{cov}(B)$, so we keep the left-hand side of the inequality the same. Making use of shadows, we can map all of the links in $\text{cov}(B)$ to links whose endpoints are in the subtree defined by B, and these links must make this subtree 2-connected. We use $\text{OPT}(B)$ to mean the minimum cost of a set of links which makes the subtree given by B 2-connected. Then our family of bundle constraints for NC-TAP

are the following.

$$\sum_{l \in \text{cov}(B)} c_l x_l \geq \text{OPT}'(B) \quad \forall B \in \mathcal{B}_\beta$$

Adjiashvili's second key idea is really just an extension of the first. Since Adjiashvili is now basing his approximation off of the Bundle LP, which obtains no integrality gap for instances with a bounded number of leaves, the integrality gap must be caused by global properties of the graph. Thus, Adjiashvili introduces a step in which he decomposes the instance into several "small" instances. Again, these, unfortunately, are not β-bundles themselves, but rather several β-bundles glued together at a vertex. We will give the formal definition of these "small" structures, called β-simple pairs, in Section 4.3. Adjiashvili shows that he can obtain a nearly optimal solution to each of these β-simple pairs, and then combine them into a solution for the original instance by only paying an epsilon factor.

After this decomposition, the main obstacle is rounding the fractional solution to each β-simple pair to an integral solution. Before detailing how this is done, we need the following definitions.

Definition 14. Given an instance of TAP $G = (V, T \cup L)$ with cost vector c, and a root vertex $r \in V$, we call a link $uv \in L$ a *cross-link* if the tree path between u and v contains r. Otherwise we call uv an *in-link*. We let L^{cr} and L^{in} denote the set of cross-links and in-links, respectively. We also use x^{cr} and x^{in} to denote the vectors in $\mathbb{R}^L_{\geq 0}$ such that $x^{\text{cr}}_l = x_l$ for all $l \in L^{\text{cr}}$, $x^{\text{cr}}_l = 0$ for all $l \in L^{\text{in}}$, and $x^{\text{in}}_l = x_l$ for all $l \in L^{\text{in}}$, $x^{\text{in}}_l = 0$ for all $l \in L^{\text{cr}}$

If the fractional solution to a β-simple pair used entirely in-links, then we can split the instance into disjoint β-bundles such that the fractional solution to each of the bundles is also disjoint. Thus we could find an integral solution to each β-bundle separately and this would be directly comparable to the fractional solution. If the fractional solution uses entirely cross-links, then it turns our that the problem is equivalent to a simpler problem, which we can solve directly.

Of course, it is most likely that the fractional solution will use both in-links and cross-links. However, using the known 2-approximation, Adjiashvili can transform any instance into one that uses only in-links, or one that uses only cross-links. This gives two algorithms which give two different approximation guarantees. By taking the minimum cost solution, which will be dependent on whether the cost of the fractional solution comes mostly from in-links, or mostly from cross-links, he achieves an approximation guarantee of better than 2.

To be specific, Adjiashvili proves that, for an instance of TAP which is a β-simple pair with fractional solution x to the Bundle LP, a solution can be found in polynomial time with cost at most $c^\top x^{\text{in}} + 2c^\top x^{\text{cr}}$ or $2\lambda c^\top x^{\text{in}} + \frac{4\lambda}{3(\lambda-1))} c^\top x^{\text{cr}}$ where λ is a parameter that can be optimized.

This gives rise to a reasonable path in applying the techniques of [1] to NC-TAP. There are 3 keys questions that need answering.

- Can we decompose an instance of NC-TAP into small instances?
- Can we solve a small instance so that we only pay a small factor on the in-links?
- Can we solve a small instance so that we only pay a small factor on the cross-links?

It is these questions that we explore in the remainder of this thesis. We would like to reiterate that many of the results in Section 4.3 and Section 4.4 are analogous to results proven by Adjiashvili in [1] and the proofs given are fundamentally the same as in that paper.

4.3 Decomposition

In this section, we present the formal definition of β-simple pairs, as well as the procedure for taking an arbitrary instance and decomposing it into such structures. In doing so, we will uncover some of the difficulties that arise when converting these ideas from TAP to NC-TAP.

Definition 15. Let $G = (V, T \cup L)$ be an instance of TAP or NC-TAP with cost vector c. Let x be a fractional TAP or NC-TAP solution for G. Then the pair (T, x) is *β-simple* if there exists a node $r \in V$, called a *β-centre* for T, such that for every tree $T' \in \text{Comp}(T - r)$ the following hold:

- $\sum_{l \in L \cap (V(T') \times V(T'))} c_l x_l \leq \beta$
- T' has at most β leaves

Figure 4.2, from [1], illustrates a 4-simple pair.

The algorithmic decomposition of an arbitrarily TAP instance, and its fractional solution, into these β-simple pairs is given by the proof to the following theorem.

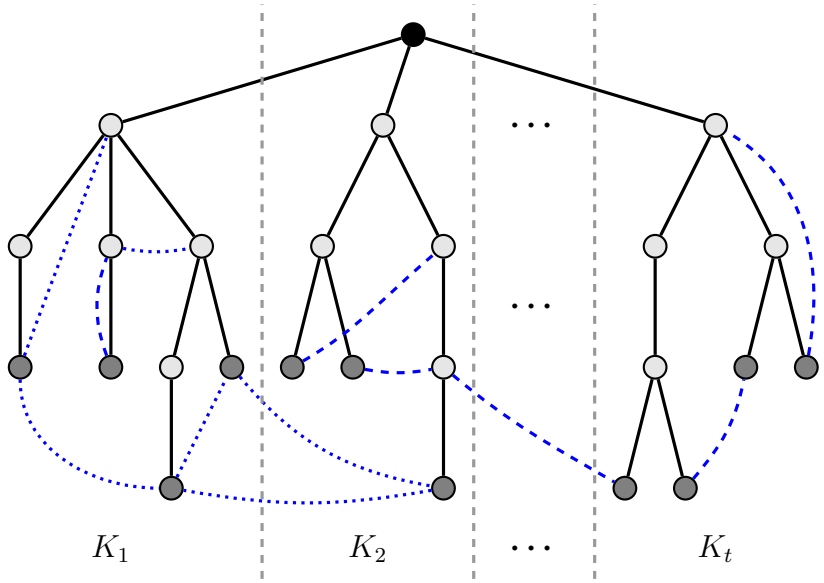


Figure 4.2: [1] A 4-simple pair (T, z) of an (unweighted) TAP instance. The full node is a 4-center. The shown links represent the support of z, with integral (dashed) and half-integral (dotted) links. The number of leaves, as well as the total fractional weight in each subtree is at most 4.

Lemma 32. *[1, Theorem 3.4] Let $\epsilon \in \left(0, \frac{1}{4}\right)$ and $\alpha > 1$ and let x be a solution to the Cut LP on a given TAP instance G with the property that $x(cov(e)) \leq \alpha$ for all $e \in T$. There is a polynomial time algorithm that computes a decomposition $(T^1, z^1), \ldots, (T^k, z^k)$ such that*

(1) T^j is a subtree of T and z^j is a fractional TAP solution for T^j using only links in $V(T^j) \times V(T^j)$ for all $j \in [k]$

(2) The subtrees $T^1, \ldots, T^k$ are obtained from T by removing exactly $k-1$ edges of T

(3) (T^j, z^j) is a β-simple pair for all $j \in [k]$ where $\beta = \frac{6\alpha M}{\epsilon}$

(4) $\sum_{j \in [k]} c^\top z^j \leq \left(1 + \frac{\epsilon}{6}\right) c^\top x$

(5) For any $j \in [k]$ and any set of edges $B \subseteq T^j$ we have $\sum_{l \in cov(B)} c_l z^j_l \geq \sum_{l \in cov(B)} c_l x_l$

Property (1) guarantees that our decomposition does in fact result in several disjoint TAP instances, and property (3) guarantees that they are β-simple. Property (5) tells us

that the fractional solutions to each smaller instance still satisfies the bundle constraints for any bundle contained within the smaller instance. Property (4) tells us that when we add up the cost of the (fractional) solutions to each of these smaller instances, we are within an epsilon factor of the cost of the original LP solution that we are comparing to, while property (2) is essential in proving that we can combine disjoint integral solutions to the smaller instances, by only adding a small number of extra edges. In other words, these properties tell us that this strategy of decomposing an instance into several smaller instances, and then joining the solutions together, will only overpay by a negligible amount.

One should notice the extra assumption in the theorem: that $x(\text{cov}(e)) \leq \alpha$ for all $e \in T$. This assumption is necessary because of how the decomposition algorithm works. As one might guess from the properties guaranteed by the theorem, the decomposition works by removing a tree edge, and then recursing on the resulting two subtrees. When a tree edge uv is removed, we must consider all of the links in the support of x that have each endpoint in a different subtree, that is, the links which cover uv. These links do not appear in either smaller instance, but they contribute to satisfying the constraints present in each smaller instance, and thus must be accounted for. If $u'v'$ is such a link where u' is in the subtree rooted at u and v' is in the subtree rooted at v, then the weight assigned to $u'v'$ is added to the weight assigned to uu' and vv', and then any weight on $u'v'$ is removed. We will now formalize this process.

Definition 16. Let $G = (V, T \cup L)$ and c be an instance of TAP and let $z \in \mathbb{R}^L_{\geq 0}$. Let $uv \in T$ be a tree edge. Let T^u be the tree containing u in $T - uv$, and T^v the tree containing v. The *splitting of z at uv* produces two vectors $z^u, z^v \in \mathbb{R}^L_{\geq 0}$ defined as follows, for any $pq \in L$

$$z^u_{pq} = \begin{cases} z_{pq} & \text{if } p, q \in V(T^u) \setminus \{u\} \\ 0 & \text{if } p, q \in V(T^v) \\ z_{pq} + \sum_{l \in \text{cov}(uv): q \in l} z_l & \text{if } p = u, q \in V(T^u) \end{cases}$$

z^v is defined analogously. Note that $\text{supp}(z^u) \subseteq V(T^u) \times V(T^u)$ and $\text{supp}(z^v) \subseteq V(T^v) \times V(T^v)$.

Figure 4.3, from [1], shows how to split a fractional solution at an edge e.

When we split x at a tree edge e, we end up paying for the links in $\text{cov}(e)$ in both of the smaller instances. This, is why we require that $x(\text{cov}(e)) \leq \alpha$, so that we can bound the cost incurred by "double counting" $\text{cov}(e)$. This requirement that $x(\text{cov}(e))$ be small turns out to only be a technical detail in the case of TAP, but proves to be more of a challenge in NC-TAP.

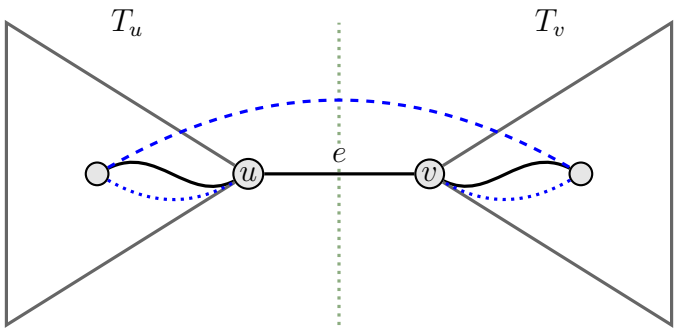


Figure 4.3: [1] The splitting of z at e. The fractional value of the link crossing the cut (dashed) is added to z^u-value and z^v-value of the left and right shadows (dotted) of the link, respectively.

Let $T^h = \{e \in T : x(\text{cov}(e)) \geq \alpha\}$. Adjiashvili preprocesses these heavily covered edges as follows. First, contract all edges in $T \setminus T^h$. Contracting an edge in a tree will still result in a tree, so the resulting graph will still be an instance of TAP. Moreover, $\frac{1}{\alpha}x$ will be a fractional solution to this instance, since $\frac{1}{\alpha}x(\text{cov}(e)) \geq \frac{1}{\alpha}\alpha = 1$ for all $e \in T^h$. Thus, we can use a 2-approximation algorithm for this TAP instance to obtain L_0, which covers all of the edges in T^h and has cost $c(L_0) \leq \frac{2}{\alpha}c^\top x$. It is known that for any graph G, any pair of vertices $u, v \in G$, and any edge $e \neq uv$ that $\kappa'_G(u,v) = \kappa'_{G/e}(u,v)$, where $\kappa'_G(u,v)$ denotes the edge-connectivty betwenn u and v, in G. Thus, all of the edges of T^h are still covered by L_0 in the original graph. By contracting all of the edges that are covered by L_0, we obtain an instance of TAP where $x(\text{cov}(e)) \leq \alpha$ for all tree edges e. Any solution to this new instance, together with L_0, will be a solution to the original TAP instance.

This is where we encounter our first problem when applying these techniques to NC-TAP. While contracting or uncontracting an edge preserves edge-connectivity, both of those operations can effect vertex-connectivity. This is captured by the respective LPs as well. Each tree edge corresponds to a constraint in the Cut LP, and contracting an edge is equivalent to removing the corresponding constraint. However, there is no such relationship between tree edges and constraints of the Set-Pair LP. For example, consider an (unweighted) instance where the overall all graph is K_4 and the spanning tree is a star. Label the centre node v and the leaves u_1, u_2, u_3. Here is the Cut LP for this TAP

instance.

$$\begin{array}{lrcrcrl}
\min & x_{u_1u_2} & + & x_{u_1u_3} & + & x_{u_2u_3} & \\
\text{s.t.} & & & & & & \\
& x_{u_1u_2} & + & x_{u_1u_3} & + & 0x_{u_2u_3} \geq 1 & \\
& x_{u_1u_2} & + & 0x_{u_1u_3} & + & x_{u_2u_3} \geq 1 & \\
& 0x_{u_1u_2} & + & x_{u_1u_3} & + & x_{u_2u_3} \geq 1 & \\
& & & & & x_l \geq 0 & \forall l \in L
\end{array}$$

And here the Cut LP for the instance one gets by contracting the tree edge vu_3 into a node, which we continue to call v. Note that the links u_1u_3 and u_2u_3 become u_1v and u_2v respectively.

$$\begin{array}{lrcrcrl}
\min & x_{u_1u_2} & + & x_{u_1v} & + & x_{u_2v} & \\
\text{s.t.} & & & & & & \\
& x_{u_1u_2} & + & x_{u_1v} & + & 0x_{u_2v} \geq 1 & \\
& x_{u_1u_2} & + & 0x_{u_1v} & + & x_{u_2v} \geq 1 & \\
& & & & & x_l \geq 0 & \forall l \in L
\end{array}$$

Note that the only difference is the removal of the third constraint. In contrast the Set-Pair LP for this instance, before contracting the edge vu_3 is as follows.

$$\begin{array}{lrcrcrl}
\min & x_{u_1u_2} & + & x_{u_1u_3} & + & x_{u_2u_3} & \\
\text{s.t.} & & & & & & \\
& x_{u_1u_2} & + & x_{u_1u_3} & + & 0x_{u_2u_3} \geq 1 & \\
& x_{u_1u_2} & + & 0x_{u_1u_3} & + & x_{u_2u_3} \geq 1 & \\
& 0x_{u_1u_2} & + & x_{u_1u_3} & + & x_{u_2u_3} \geq 1 & \\
& & & & & x_l \geq 0 & \forall l \in L
\end{array}$$

And after contracting the tree edge vu_3 the resulting NC-TAP instance has the following Set-Pair LP.

$$\begin{array}{lrcrcrl}
\min & x_{u_1u_2} & + & x_{u_1u_3} & + & x_{u_2u_3} & \\
\text{s.t.} & & & & & & \\
& x_{u_1u_2} & + & 0x_{u_1u_3} & + & 0x_{u_2u_3} \geq 1 & \\
& & & & & x_l \geq 0 & \forall l \in L
\end{array}$$

Again, this is to be expected because contractions preserve edge-connectivty, but not vertex-connectivity. For now, we will simply assume that no tree edge is heavily covered.

The decomposition for NC-TAP is modified in some technicalities, but is essentially the same procedure. Thus, we will only prove the decomposition result for NC-TAP. But first, we must redefine splitting a tree edge. This is necessary, because if we split our fractional solution at an edge uv and the simply solve T^u and T^v in isolation, then when we put their

solutions together, u and v will be cut nodes. Of course, in TAP there is the analogous problem that uv will be a bridge, but this can be fixed by covering uv with some link, which doesn't necesarrily stop either u or v from being cut nodes.

Definition 17. Let $G = (V, T \cup L)$ and c be an instance of NC-TAP and let $z \in \mathbb{R}^L_{\geq 0}$. Let $uv \in T$ be a tree edge. Let T^u be the tree containing u in $T - uv$, and T^v the tree containing v. Let T'^u be the tree where all of T^v is contracted into a single vertex, v', any self loops at v' are removed, and the edge uv is replaced with uv'. Similarly, T'^v is the tree where all of T^u is contracted into a single vertex, u', any self loops at u' are removed, and the edge uv is replaced with $u'v$. Call the link sets of these trees L^u and L^v. The *splitting of z at uv* produces two vectors $z^u \in \mathbb{R}^{L^u}_{\geq 0}$ and $z^v \in \mathbb{R}^{L^v}_{\geq 0}$ defined as follows, for any $pq \in L^u$

$$z^u_{pq} = \begin{cases} z_{pq} & \text{if } p, q \in V(T^u) \\ 0 & \text{if } p, q \in V(T^v) \\ z_{pq} + \sum_{l \in \text{cov}(uv): q \in l} z_l & \text{if } p = v', q \in V(T^u) \end{cases}$$

z^v is defined analogously.

Note that this is just a formal way of saying that to split z at uv we create two trees, one where we contract T^u into a single vertex, and the other where we contract T^v into a single vertex. In both cases we preserve the weight on the links in the obvious way. Figure 4.4, together with Figure 4.3, shows how to split a fractional solution at an edge e in the NC-TAP context.

Note that the operation of splitting z at e for TAP clearly gave rise to a new instance and feasible fractional solution of TAP where we kept the constraints corresponding to edges in T^u or T^v, depending on which of the two smaller instances we are looking at. Similarly, the splitting operation for NC-TAP produces two NC-TAP instances, one where we have the family of constraints for each non-leaf vertex in T'^u, and the other has the family of constraints for non-leaf vertices in T'^v. Notably, u and v could not have been leaves before splitting z at e, and still are not afterwards. Also, the contracted nodes u' and v' are always leaves and thus don't have any constraints associated with them. Furthermore, the coverage of an edge is preserved. That is $z^u(\text{cov}(e)) = z(\text{cov}(e))$ for all $e \in T'^u$. Recall that we are assuming that $x(\text{cov}(e)) \leq \alpha$ for all tree edges $e \in T$ and thus this preservation is vital. With this updated notion of splitting our NC-TAP instance and solution into two smaller instances and solutions, we wish to prove the following lemma.

Lemma 33. *Let $\epsilon \in \left(0, \frac{1}{4}\right)$ and $\alpha > 1$ and let x be a solution to the Partition LP with Bundle constraints on a given NC-TAP instance G with the property that $x(cov(e)) \leq$*

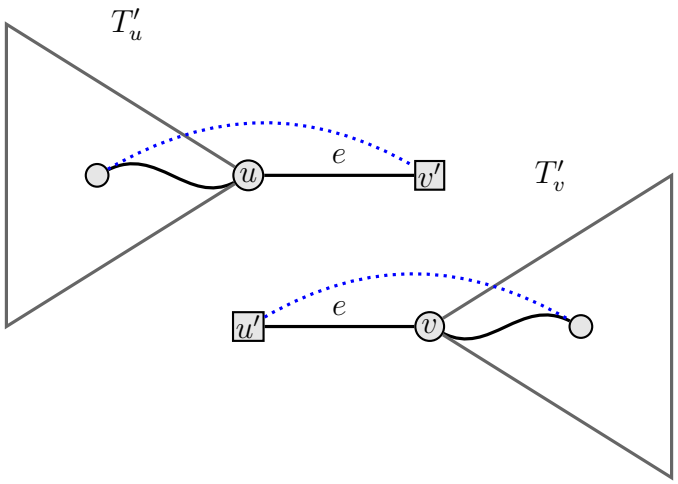


Figure 4.4: The splitting of z at e. The fractional value of the link crossing the cut in Figure 4.3 (dashed) is added to z^u-value and z^v-value of the corresponding links to the contracted node, shown above (dotted).

α for all $e \in T$. There is a polynomial time algorithm that computes a decomposition $(T^1, z^1), \ldots, (T^k, z^k)$ such that

(1) T^j is an instance of NC-TAP (that is also a minor of T) and z^j is a fractional NC-TAP solution for T^j using only links in L^j for all $j \in [k]$,

(2) If F^j is a solution to T^j for each $j \in [k]$ then $\bigcup_{j\in[k]} F^j$ is a solution to T,

(3) (T^j, z^j) is a β-simple pair for all $j \in [k]$ where $\beta = \frac{6M\alpha}{\epsilon}$,

(4) $\sum_{j\in[k]} c^\top z^j \leq \left(1 + \frac{\epsilon}{2}\right) c^\top x$,

(5) For any $j \in [k]$ and any set of edges $B \subseteq T^j$ we have $\sum_{l\in cov(B)} c_l z^j_l \geq \sum_{l\in cov(B)} c_l x_l$. In particular this means that the bundle constraints are still satisfied.

Proof. The decomposition algorithm is going to make use of the splitting z at e operation we just described. Exactly how it is used will be discussed later. First, we will show properties (1), (2), and (5) are true after one splitting, say splitting z at uv. It is easy to see how to apply induction to prove these properties hold after any finite number of splitting operations.

Since T'^u is obtained from T by contractions, it is still a spanning tree, and thus the resulting contracted graph is a valid NC-TAP instance. To see that z^u is a fractional solution, consider any non-leaf vertex w in T'^u, and any partition $\mathcal{P}' \in \mathcal{P}(\text{Comp}(T'^u - w))$. We will show that $z^u(\delta_{L^u}(\mathcal{P}') \geq |\mathcal{P}'| - 1$. As noted above, v' is a leaf so $w \neq v'$. So w is a non-leaf vertex in T. Let $\mathcal{P}$ be obtained from $\mathcal{P}'$ by replacing v', in whatever part it is in, by T^v. $\mathcal{P} \in \mathcal{P}(\text{Comp}(T - w))$. Thus the original Partition LP had a constraint $z(\delta_L(\mathcal{P}) \geq |\mathcal{P}| - 1$ and since z was a feasible solution, $z(\delta_L(\mathcal{P}) \geq |\mathcal{P}| - 1$. Furthermore, $|\mathcal{P}| = |\mathcal{P}'|$ and $\delta_L(\mathcal{P}) = \delta_{L^u}(\mathcal{P}')$. Thus we have that

$$z^u(\delta_{L^u}(\mathcal{P}')) = z(\delta_L(\mathcal{P})) \geq |\mathcal{P}| - 1 = |\mathcal{P}'| - 1$$

Thus T'^u is an NC-TAP instance and z^u satisfies its Partition LP. The same argument shows that T'^v and z^v also satusfy Property (1).

Now suppose $F^u \subseteq L^u$ and $F^v \subseteq L^u$ are solutions to the NC-TAP instances T'^u and T'^v. We will show that $F^u \cup F^v$ is a solution to the original instance. Suppose not. Then there is a non-leaf vertex w such that $T \cup F^u \cup F^v - w$ is disconnected. Note that w must be a non-leaf vertex in either T^u or T^v. Without loss of generality, say it is the former. Then T_v is connected in $T \cup F^u \cup F^v - w$. Thus, there is a connected component, S, of $T \cup F^u \cup F^v - w$ which is entirely contained in T_u and thus in T'_u. However, F^u is a solution for T'_u, so there is a link $l \in \delta_{F^u}(S : \zeta_{T'^u}(S))$ but as we saw above, $\delta_{F^u}(S : \zeta_{T'^u}(S)) = \delta_F(S : \zeta_T(S)$, since $[S|\zeta_T(S)] \in \mathcal{P}(\text{Comp}(T - w)$. Thus we arrive at a contradiction, proving that Property (2) must hold.

To prove Property (5) we will assume $B \subseteq T'^u$. Now consider any link $pq \in \text{cov}(B)$ such that $pq \notin L^u$. It must be that, up to relabeling, $p \in V(T'^u)$ and $q \in V(T'^v)$. The fractional weight of this link, z_{pq}, gets added to the weight of the shadow $z^u_{pv'}$ and so the property holds.

We now look to prove property (4). We can easily see that if $l \in V(T^u) \times V(T^u)$ then $l \in L^u, l \notin L^v$. Similarly, if $l \in V(T^v) \times V(T^v)$ then $l \in L^v, l \notin L^u$. So, $l \in L^u \cap L^v$ if and only if $l \in \text{cov}(e)$. Thus

$$c^\top z^u + c^\top z^v = c^\top z + \sum_{l \in \text{cov}(e)} c_l z_l$$

Thus, if $e_1, \ldots, e_{k-1}$ are the tree edges that we split on in our decomposition algorithm,

then we can say

$$\begin{aligned}
\sum_{j\in[k]} c^\top z^j &= c^\top x + \sum_{j=1}^{k-1} \sum_{l\in \mathrm{cov}(e_j)} c_l x_l \\
&\leq c^\top x + \sum_{j=1}^{k-1} \sum_{l\in \mathrm{cov}(e_j)} M x_l \\
&= c^\top x + M \sum_{j=1}^{k-1} x(\mathrm{cov}(e_j)) \\
&\leq c^\top x + M \sum_{j=1}^{k-1} \alpha \\
&= c^\top x + (k-1)M\alpha \\
&\leq c^\top x + kM\alpha
\end{aligned}$$

Thus, to prove property (4) it suffices to show that $kM\alpha \leq \frac{\epsilon}{2} c^\top x$. This goal motivates our decomposition algorithm.

Initialize $\mathcal{T} = \{(T, x)\}$. When $\mathcal{T} = \emptyset$, terminate. Let $(\hat{T}, z) \in \mathcal{T}$. Let $uv \in \hat{T}$ such that

$$\sum_{l\in L, l\in V[\hat{T}^w]\times V[\hat{T}^w]} c_l z_l \geq \frac{2M\alpha}{\epsilon} \text{ for both } w = u \text{ and } w = v \tag{4.1}$$

If no such $e \in \hat{T}$ exists, report $(\hat{T}, z)$ as part of the final decomposition and remove it from $\mathcal{T}$. Otherwise, obtain $(\hat{T}'^u, z^u)$ and $(\hat{T}'^v, z^v)$ from splitting z at uv, add them to $\mathcal{T}$, and remove $(\hat{T}, z)$ from $\mathcal{T}$. It is clear that a tree edge can only be split at once, so the algorithm must terminate after $|T|$ iterations, and is thus polynomial time.

For each (T^j, z^j) in the final decomposition, let $\bar{T}^j$ be the subtree of T^j where any contracted nodes, which recall are always leavesa, have been removed. By how the decomposition procedure works, we know $\sum_{l\in L\cap(V[\bar{T}^j]\times V[\bar{T}^j])} c_l z_l^j \geq \frac{2M\alpha}{\epsilon}$. Thus,

$$c^\top x \geq \sum_{j=1}^{k} \sum_{l\in L\cap(V[\bar{T}^j]\times V[\bar{T}^j])} c_l z_l^j \geq \frac{2kM\alpha}{\epsilon}$$

Thus proving property (4) holds.

We will now prove that property (3) holds, that is that each (T^j, z^j) is a $\frac{6M\alpha}{\epsilon}$-simple pair. Fix some (T^j, z^j).

Note that when (T^j, z^j) was reported by the decomposition algorithm, it was because there was no edge uv such that

$$\sum_{l \in L, l \in V[(T^j)^w] \times V[(T^j)^w]} c_l z_l \geq \frac{2M\alpha}{\epsilon} \text{ for both } w = u \text{ and } w = v$$

Thus, for every tree edge $e \in T^j$ we must have that for at least one endpoint w of e,

$$\sum_{l \in L, l \in V[(T^j)^w] \times V[(T^j)^w]} c_l z_l < \frac{2M\alpha}{\epsilon}$$

We direct each tree edge so that the head satisfies the above inequality. If both endpoints satisfy the inequality, the edge can be directed arbitrarily. By properties of directed trees, there must be a vertex r with in-degree zero. If there is more than one, pick one arbitrarily to be r. We will show that r is an $\frac{6M\alpha}{\epsilon}$-centre.

By design, for every connected component K of $T^j - r$, $\sum_{l \in L, l \in V[K] \times V[K]} c_l z_l < \frac{2M\alpha}{\epsilon} < \frac{6M\alpha}{\epsilon}$, so the first property of $\frac{6M\alpha}{\epsilon}$-simple pairs hold. It remains to show that K has at most $\frac{6M\alpha}{\epsilon}$ leaves. Assume for a contradiction that it has more. Since each link costs at least 1, all leaves must be incident to some link, and a link can be incident to at most 2 leaves, any fractional solution covering K must have cost strictly greater than $\frac{3M\alpha}{\epsilon}$. Let e' be the tree edge connecting K to r. Then z^j fractionally covers K using only links in $L_K := (L \cap (V[K] \times V[K])) \cup \text{cov}(e')$. However

$$\sum_{l \in L_K} c_l z^j_l < \frac{2M\alpha}{\epsilon} + M\alpha \leq \frac{3M\alpha}{\epsilon}$$

Thus (T^j, z^j) is a $\frac{6M\alpha}{\epsilon}$-simple pair and r is its $\frac{6M\alpha}{\epsilon}$-centre, proving property (3). □

In the remaining sections we will assume our instance and fractional solution are a β-simple pair and that we have a β-centre, r. We will analyze these instances in total isolation.

4.4 Bundle Rounding

In this section we aim to prove the following.

Theorem 34. *Assume that $G = (V, T \cup L)$ is an NC-TAP instance, x is a solution to the β-Bundle LP and (T, x) are a β-simple pair with β-centre r. There is an algorithm which computes a set of links $F \subseteq L$, in time $n^{O(\beta)^{O(1)}}$, such that F is an NC-TAP solution of cost*

$$c(F) \leq c^\top x^{in} + 3c^\top x^{cr}$$

Adjiashvili proved an analogous result in [1], obtaining a solution to TAP of cost at most $c^\top x^{\text{in}} + 2c^\top x^{\text{cr}}$. The algorithm, which is essentially the same for both TAP and NC-TAP, is quite simple. As mentioned earlier in this chapter, there is an algorithm, proven in Lemma 35 later in this section, for solving instances with a bounded number of leaves, to optimality. A β-simple pair is just an instance comprised of many such instances, glued together at the β-centre. The idea will be to modify the fractional solution x such that we can unglue these subtrees, solve them optimally, and put them back together.

Recall that (T, x) is a β-simple pair with β-centre r. For every $T' \in \text{Comp}(T - r)$, let $\bar{T}'$ be the subtree of T induced by $V(T') \cup \{r\}$. These $\bar{T}'$ are the instances with a bounded number of leaves we will solve in isolation using Lemma 35, which appears later.

In order to consider each $\bar{T}'$ separatly, we want there to be no cross-links in $\text{supp}(x)$. We remove the cross-links from $\text{supp}(x)$ with the following simple transformation. Construct $y \in \mathbb{R}^L_{\geq 0}$ as follows. Start with $y = x$. For any cross link uv, which doesn't have r as one of it's endpoints, $y_{uv} = 0$ and $y_{ur} \leftarrow y_{ur} + x_{uv}$ and $y_{vr} \leftarrow y_{vr} + x_{uv}$. The idea is we move the weight from any cross link uv to the two shadows ur and vr, and we know that $c^\top y \leq c^\top x^{\text{in}} + 2c^\top x^{\text{cr}}$ This construction need not, and actually cannot, be a fractional NC-TAP solution to G. However, it is still a fractional solution for each $\bar{T}'$. In fact, the only constraints that will be violated by y are the partition constraints obtained from deleting r.

We know that each $T' \in \text{Comp}(T - r)$ has at most β leaves and thus $\bar{T}'$ has at most $\beta + 1$ leaves, including r. This tells us that $\bar{T}'$ can be solved to optimality using Lemma 35. Also, $\bar{T}'$ is the union of the paths from r to each other leaf. Thus, each $\bar{T}'$ is a β-bundle. Since x was a solution to the β-Bundle Partition LP, it satisfied the constraint that

$$\sum_{l \in \text{cov}(B)} c_l x_l \geq \text{OPT}(B) \quad \forall B \in \mathcal{B}_\beta$$

In particular y also satisfies these constraints. Thus the cost of the solutions for each $\bar{T}'$ returned by the subroutine will be less than or equal to the cost of y restricted to those

subtrees. Since $\text{supp}(y)$ is partitioned by these subtrees, we can say that the union of these solutions, which we will call F', satisfies $c(F') \leq c^\top y \leq c^\top x^{\text{in}} + 2c^\top x^{\text{cr}}$.

However, as mentioned earlier, y, and F', do not satisfy the partition contraints obtained from removing r. We now find a set of links just for satisfying the parition constraints obtained from removing r. However, recall that the family of partition constraints for a single vertex correspond to a Spanning Tree Polytope, which we know is integral [20, Corollary 50.8]. Furthermore, we know that x is in this Spanning Tree Polytope, since it satisfies those constraints. Furthermore, the only links that contribute to satisfying the partition constraints obtained from removing r are by definition, cross-links. Hence x^{cr} is in this Spanning Tree Polytope. Thus, we can compute, in polynomial time, a set of links F'' that satisfy the partition constraints obtained from removing r such that $c(F'') \leq c^\top x^{\text{cr}}$.

Then $F = F' \cup F''$ is a solution to this NC-TAP instance and $c(F) \leq c^\top x^{\text{in}} + 3c^\top x^{\text{cr}}$, proving Theorem 34.

We finally formalize the subroutine that we have alluded to all chapter, which (integrally) solves instances of NC-TAP optimally, and thus finishes the proof of Theorem 34.

Lemma 35. *An optimal solution to any NC-TAP instance can be computed in time $n^{k^{O(1)}}$, where k is the number of leaves in the tree.*

Note that this algorithm and proof are exactly the same as given by Adjiashvili in [1].

Proof. Let W be the set of vertices of degree 3 or greater. Since the tree has k leaves, $|W| \leq k$. Let $Q_1, \ldots, Q_p$ be the paths obtained by splitting each node $w \in W$ into $\deg(w)$ copies, each connected to a distinct neighbour of w. Note that these paths $Q_1, \ldots, Q_p$ correspond to paths in T, the endpoints of which are either in W or are leaves. It is clear that $2p = k + \sum_{w \in W} \deg(w)$ and since we started with a tree, $\sum_{w \in W} \deg(w) = 2(n-1) - 2(n-k-|W|) - k = 2|W| - 2 + k$. Thus we get that $p = |W| + k - 2 \leq 2k - 2$.

Next we will show that any optimal solution has at most 2 links between any pair of paths Q_i and Q_j. Suppose not. Then for an optimal solution F^* we have three links between one pair of paths, Q_i and Q_j. Call the three links a_1a_2, b_1b_2, and c_1c_2. Without loss of generality we assume $P_{a_1a_2} \subseteq P_{b_1b_2} \cup P_{c_1c_2}$. Consider $T \cup F^* - a_1a_2$. This has a cut vertex v. Since v is not a cut vertex $T \cup F^*$, we must have that v is a vertex on $P_{a_1a_2}$ and that $T \cup F^* - a_1a_2 - v$ has precisely two components. However, v can only be on one of $P_{a_1c_1}$ or $P_{b_2a_2}$. Without loss of generality, we assume v is not a vertex on $P_{a_1c_1}$. But then $P_{a_1c_1} \cup \{c_1c_2\} \cup P_{c_2a_2}$ is a a_1, a_2-path in $T \cup F^* - a_1a_2 - v$, implying that it is connected,

a contradiction. Thus any optimal solution has at most 2 links between any pair of paths Q_i and Q_j.

This means that any optimal solution will have at most $8k^2$ links between paths. Assume we know what these links were, and call them F'. The remaining links in the optimal solution are fully contained within a path. Call the links that are fully contained in one Q_i, L^{path}. Links in L^{path} only contribute to satisfying partition constraints for vertices which are internal vertices in their path. Such vertices have degree 2 in the original tree and thus there are only two components when you remove such a vertex. For such a vertex v let these two components be S_v and S'_v. The remaining constraints which need to be satisfied are of the form

$$x(\delta_{L^{\text{path}}}(S_v : S'_v)) \geq 1$$

where v is the internal vertex to some path Q_i. We will show that these constraints form a Totally Unimodular (TU) matrix, and thus, the polytope defined by these constraints is integral. Note that not every internal vertex of each Q_i need have a constraint that still needs to be satisfied, but we will assume they do. And if this matrix is TU, the actual constraint matrix we are concerned with will also be TU.

We prove that the constraint matrix is TU by showing its rows satisfy the Ghouila-Houri condition.Note that any subset of constraints corresponds to a subset of vertices. Let X be a subset of vertices. We will show that there exists a subset X^+ such that if one sums all rows of the constraint matrix corrresponding to X^+, and subtracts all rows of the constraint matrix corresponding to $X \setminus X^+$, the resulting vector has entries in $\{0, 1, -1\}$. For each path, Q_i, starting from one of the ends of the path, put every other vertex into X^+, not counting vertices which are not in X. The resulting vector has the following entry for any link $uv \in L^{\text{path}}$.

$$|\{w \in X^+ : w \text{ is an internal vertex of } P_{uv}\}| - |\{w \in X \setminus X^+ : w \text{ is an internal vertex of } P_{uv}\}|$$

However, $P_{uv} \subseteq Q_i$ for some pathj Q_i, and thus the internal vertices on the path P_{uv}, ignoring those not in X alternate between X^+ and $X \setminus X^+$, so the above is one of $1, -1$, or 0.

With this, the algorithm is simple. First, separate the tree at the vertices in W to obtain the paths $Q_1, \ldots, Q_p$. Next, iterate through each possible subset F' of links going between paths such that no pair of paths has more than two links between them. For each of these subsets, compute the minimum cost set of links that 2-connects each path Q_i, call them F_i. Finally, consider $F = F' \cup F_1 \ldots \cup F_p$. If F is a NC-TAP solution, record it and its cost. Once this is done for all possible F', take the solution of minimum cost and output it. One can verify that the running time of this algorithm is indeed $n^{k^{O(1)}}$. □

4.5 Star Rounding

In this section and subsequent sections, we are interested in finding a rounding scheme that does not overpay too much for cross-links. Much like the previous rounding scheme, which worked by simplistically dealing with the cross-links and then having a more sophisticated algorithm for the in-links, this rounding scheme would simplistically deal with the in-links and then round the cross-links in a more careful and sophisticated way.

The result proved by Adjiashvili for TAP is as follows

Lemma 36. *[1, Lemma 3.1] Assume that $G = (V, T \cup L)$ is an TAP instance, x is a solution to the Cut LP and cross-links and in-links are defined with respect to a vertex r. Given any real constant $\lambda > 1$, there is an algorithm which computes a set of links $F \subseteq L$, in polynomial time, such that F is an TAP solution of cost*

$$c(F) \leq 2\lambda c^\top x^{in} + \frac{4\lambda}{3(\lambda - 1)} c^\top x^{cr}$$

The way Adjiashvili deals with in-links is as follows. Let T_λ be the of set of tree edges such that they are $\frac{1}{\lambda}$-covered by in-links. Formally, $T_\lambda = \{e \in T : \lambda x^{\text{in}}(\text{cov}(e)) \geq 1\}$. Adjiashvili then uses a simple 2-approximation algorithm for TAP to find a set of links L_λ that covers T_λ such that $c(F) \leq 2\lambda c^\top x^{\text{in}}$. Then he contracts all covered edges and turns his attention to the resulting TAP instance, which is fractionally solved by $\frac{\lambda}{\lambda - 1} x^{\text{cr}}$.

As mentioned in section 3, when the only links that are available are cross-links, NC-TAP and TAP reduce down to (different) easier problems. In the case of TAP, this simpler problem is the Edge Cover Problem. We say a a vertex is *covered* by a set of edges (or links) if it is incident with at least one edge in the set. The Edge cover problem is the problem of, given a set of vertices V, and edges E with cost vector $c \in \mathbb{R}^E_{\geq 0}$, find a minimum cost $F \subseteq E$ such that F covers all of the vertices in V. The reduction is a corollary of the following lemma.

Lemma 37. *[1, Lemma 5.2] Let $G = (V, T \cup L)$ be a TAP instance where $L = L^{cr}$ with respect to a vertex r. Then $F \subseteq L$ is a feasible solution if and only if F covers all leaves.*

Proof. First, assume F is a feasible solution. Consider any leaf u and its corresponding leaf edge, e_u. Since F is feasible, it must cover e_u. After removing e_u, one of the components is just the isolated vertex u. Thus, in order to cover e_u, F must contain a link incident to u.

Now assume F covers all leaves. Let $e \in T$ be arbitrary. Consider the two components of $T - e$. One of these contains r. Let u be a leaf of the other component which is also a leaf in T. Consider a link $uv \in F$ which is incident to u. Since all links in L, and thus in F are cross-links, P_{uv} contains r. Since e is the only tree edge that goes from the component containing u and the component containing r, v must be in the latter component. Thus uv covers e. Since e was arbitrary, F covers all tree edges and thus is a feasible solution. □

From here, the reduction is simple. Contract all non-leaf vertices into r, then delete the tree edges. Add a special link l_0 which is a self loop at r of cost 0. The Edge Cover Problem on the resulting graph with edge set $L \cup \{l_0\}$ is equivalent to TAP on the original graph. Note that the contraction will delete any link that is not incident to at least one leaf, but Lemma 37 tells us that these are not needed. The special link l_0 simply exists so that the vertex r, which need not be a leaf and thus need not be covered by a TAP solution, can be covered in the Edge Cover Problem at cost 0. Note that we would not want to be able to do this if r was a leaf and thus truly did need to be covered. However, the condition that all links are cross-links implies that if r is a leaf, then every link is incident to r. Thus, in covering the other leaves, r will also be covered, without the need for using l_0. Here, Adjiashvili relies on the following lemma.

Lemma 38. *[1, Lemma 5.3] There is a polynomial time algorithm that, given a graph G, a cost vector $c \in \mathbb{R}^{E[G]}_{\geq 0}$, and a fractional edge cover $x \in \mathbb{R}^{E[G]}_{\geq 0}$, computes an edge cover $F \subseteq E[G]$ with cost $c(F) \leq \frac{4}{3} c^\top x$.*

We would like to follow this same procedure to find a similar reduction for this special class of NC-TAP instances. However, there is currently no known way to deal with the in-links. We can use the 2-approximation algorithm given in Chapter 2 to get a partial solution, but there is no nice combinatorial object in an NC-TAP instance we can associate each constraint with; each non-leaf vertex gives a family of constraints. However, we will continue by studying the case where an NC-TAP instance has only cross-links. Not only could this be useful if the above problem is resolved, but it is also interesting in its own right.

The analogous result to Lemma 37 is the following.

Lemma 39. *Let $G = (V, T \cup L)$ be a NC-TAP instance where $L = L^{cr}$ with respect to a vertex r. Then $F \subseteq L$ is a feasible solution if and only if F covers all leaves and r is not a cut vertex.*

Proof. First, assume F is a feasible NC-TAP solution. That is, $T \cup F$ is 2-connected. Thus $T \cup F$ is 2-edge-connected, so F is also a TAP solution and by Lemma 37 F must cover all leaves. Also, since $T \cup F$ is 2-connected, r is not a cut vertex.

Now assume that F covers all leaves and r is not a cut vertex. Consider any vertex $v \neq r$. We will show that an arbitrary vertex w in $T \cup F - v$ has a path to r. If $w = r$, the empty path suffices. If w is in the same component as r in $T - v$, then the tree path P_{rw} suffices. Otherwise, it must be that v is a vertex on P_{rw}. Consider any leaf u is a descencdent of w, so $P_{rw} \subseteq P_{ru}$. We know u is covered by some link $uz \in F$. We claim that v cannot be on P_{rz}. This is because r must be on P_{uz} since uz must be a cross link, and by the uniqueness of tree paths $P_{uz} = P_{ru} \cup P_{rz}$ and we know v is on P_{ru}. Thus $P_{wu} \cup \{uz\} \cup P_{rz}$ is a w, r-path in $T \cup F - v$. □

At this point we don't have a specific problem we are trying to reduce to, but we would like to do something similar to the reduction above from TAP to the Edge Cover Problem, so that we have a concrete, structured, and simpler problem that we can focus on. We want to do the same contracting of internal vertices, so that we are only considering leaves, which are the important vertices. However, we also need to ensure removing r does not disconnect the graph. If we contract all of the vertices into r then removing the new contracted node r does much more than removing r did in the original graph. Also, removing r removes links that could be used to cover leaves. We contract all edges of the tree which are not incident to any leaves, and are also not incident to r. Then we remove any links which are loops. The resulting graph, with the exception of the links, is a tree with the special property that it is rooted at a vertex r and there is a path of length two or less between r and any other vertex. We say that the tree has *height* two.

The *Two-Level Tree Problem (TLTP)* is the problem of given a graph $G = (V, T \cup L)$, where T is a spanning tree rooted at a vertex $r \in V$ and of height two, and a cost vector $c \in \mathbb{R}^L_{\geq 0}$, find a minimum cost set of links $F \subseteq L$ such that F covers every leaf and $T \cup F - r$ is connected. We will categorize the different types of vertices in a TLTP instance. One category will be just r itself which we continue to call the root. We also care about the leaves of the tree. The remaining vertices will be called *subroots*. Note that while TLTP arised from NC-TAP instances which only used cross-links, for a general TLTP instance it need not be the case that all links are cross-links with respect to the root. For any TLTP instance which only contains cross-links, TLTP is equivalent to NC-TAP.

The Two-Level Tree Problem will be the subject of the next few sections and is the main new contribution of this thesis.

4.6 Hardness of TLTP

In this section we will prove that this new problem, TLTP, is NP-hard. Note that the Edge Cover Problem, which TAP reduces to in these special instances, is known to be in P [20]. This highlights a significant difference between NC-TAP and TAP.

Theorem 40. *TLTP is NP-hard.*

The reduction is from the *3-Dimensional Matching Problem (3DM)*. Here is the formal description of the variant of 3DM we will use. Let W, X, and Y be finite sets such that $|W| = |X| = |Y| = p$. Let $A \subseteq W \times X \times Y$. A 3-dimensional matching M is a subset of A such that if $(w_1, x_1, y_1), (w_2, x_2, y_2) \in M$ then $w_1 \neq w_2, x_1 \neq x_2$, and $y_1 \neq y_2$. The 3-Dimensional Matching Problem is the problem of, given the above, determining whether there exists a 3-dimensional matching M such that $|M| = p$, caled a *perfect 3D matching*. Note that this problem can be viewed as finding a perfect matching in a tripartite hypergraph where edges are incident to exactly 3 vertices, one from each part. Also, we can make the assumption that every element in W, X, and Y is in at least 2 triplets in A. This assumption is a technical one that makes the reduction simpler, but it is a valid assumption. If $w \in W$ were only in one triplet $(w, x, y) \in A$, then we would know that (w, x, y) would have to be in any perfect 3D matching. As such we could remove w, x, and y from the instance, along with any triplet containing any of these elements. Then the resulting instance has a perfect 3D matching M, if and only if $M \cup \{(w, x, y)\}$ is a perfect 3D matching of the original instance. We can do the same with any $x \in X$ or $y \in Y$ such that x or y appear in only one triplet of A.

Assume we are given an instance of 3DM. We will construct an instance of TLTP, of polynomial size compared to the size of the 3DM instance, which we can use to determine whether our instance of 3DM is a "YES" instance, or a "NO" instance, thus proving that 3DM polynomially reduces to TLTP and thus that TLTP is NP-hard.

Here is the reduction. We will assume $W = \{w_1, \ldots, w_p\}$ and the elements of X and Y are similarly defined. First, create a subroot, labelled W whose leaves are precisely the elements of W. For each $j \in \{1, \ldots, p\}$ we create two different subroots: one labelled x_j, the other labelled x'_j. Also, for each $k \in \{1, \ldots, p\}$, create a leaf connected directly to the root, labelled y_k. Now for each $(w_i, x_j, y_k) \in A$ we create a vertex $a_{i,j,k}$ which is a leaf of the subroot x_j and a second vertex $a'_{i,j,k}$ which is a leaf of the subroot x'_j. Then we add three links of unit cost such that $w_i, a_{i,j,k}, a'_{i,j,k}, y_j$ is a path. An example of this construction is shown in Figure 4.5. The idea will be that taking (w_i, x_j, y_k) to be in M will correspond to taking $w_i a_{i,j,k}$ and $a'_{i,j,k} y_k$ to be in our TLTP solution, and not including (w_i, x_j, y_k)

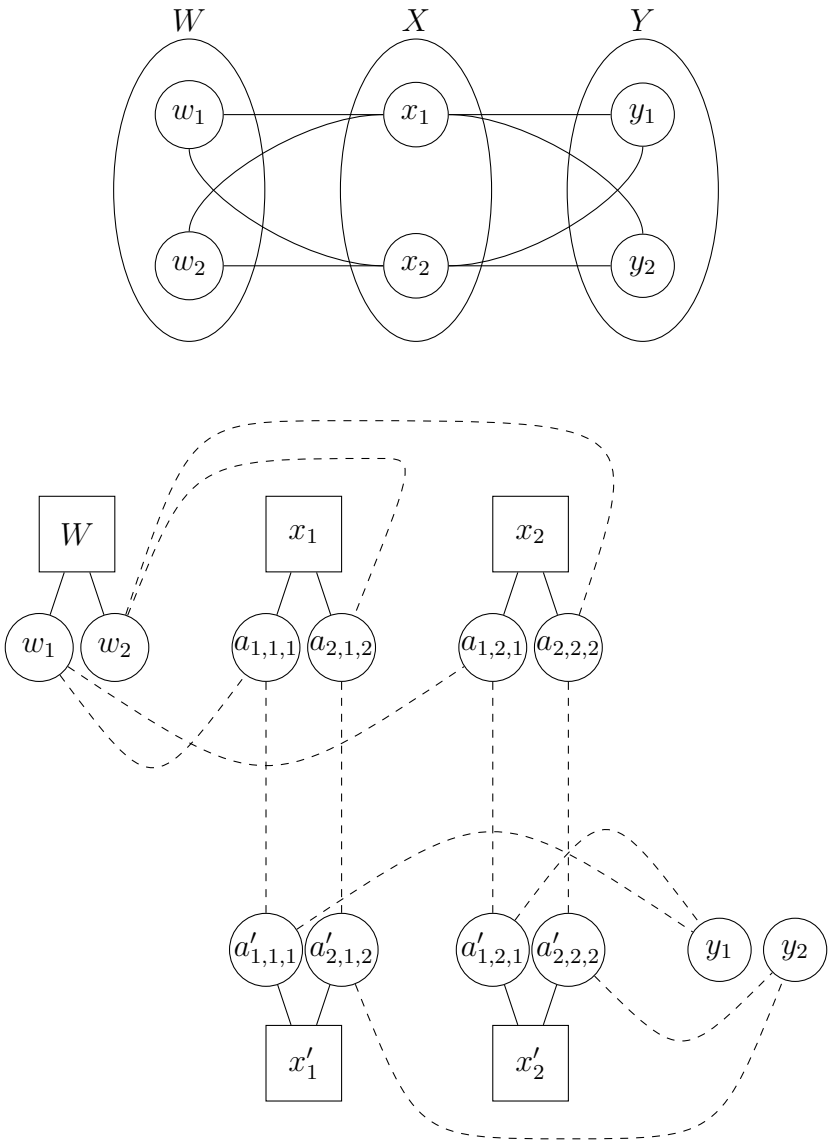


Figure 4.5: The reduction from 3DM to TLTP where $p = 2$ and $|A| = 4$. The upper figure shows a graphical representation of a 3DM instance and the lower figure is the TLTP instance where the root has been omitted. Subroots are shown as square nodes and leaves as round nodes.

will correspond to taking $a_{i,j,k}a'_{i,j,k}$ in our TLTP solution. It is clear that this instance is constructible in polynomial time. Theorem 40 is a corollary of the following lemma.

Lemma 41. *Let $|A| = q$ then there exists a 3-dimensional matching M of size p if and only if the TLTP instance constructed above has a solution of cost $p + q$.*

Proof. First note that the constructed TLTP instance has $2p + 2q$ leaves and a link can cover at most 2 leaves. Thus any solution will have cost at least $p + q$.

Now we will assume that M is a 3-dimensional matching of size p. As mentioned above we will construct F such that for each $(w_i, x_j, y_k) \in M$ we have $w_i a_{i,j,k}, a'_{i,j,k} y_k \in F$ and for each $(w_i, x_j, y_k) \in A \setminus M$ we have $a_{i,j,k}a'_{i,j,k} \in F$. No other links are in F. We have $c(F) = 2|M| + |A \setminus M| = |M| + |A| = p + q$. Now we show F is feasible. Every $w_i \in W$ must be covered by some $(w_i, x_j, y_k) \in M$. Thus every leaf w_i is incident to a link, specifically the link $w_i a_{i,j,k}$. Similarly every leaf y_k must be covered by some link $a'_{i,j,k} y_k$ where $(w_i, x_j, y_k) \in M$. Furthermore, we see that every $a_{i,j,k}$ and $a'_{i,j,k}$ is covered since either $(w_i, x_j, y_k) \in M$ and thus $w_i a_{i,j,k}, a'_{i,j,k} y_k \in F$ or $(w_i, x_j, y_k) \notin M$ in which case $a_{i,j,k}a'_{i,j,k} \in F$. Finally we show that F connects the graph $T - r$ by showing that every component of $T - r$ has a path to the tree rooted at the subroot W. Notice that each y_k leaf must be connected to some $a'_{i,j,k}$ and thus is connected to some x'_j subroot. Each x_j must be in some triplet $(w_i, x_j, y_k) \notin M$, since we are assuming that every element of the 3DM instance appears in at least two triplets of A. Thus the link $a_{i,j,k}a'_{i,j,k} \in F$ which connects each x'_j subtree to it's corresponding x_j subtree. Finally, since each x_j is covered by some $(w_i, x_j, y_k) \in M$, we have that $w_i a_{i,j,k} \in F$ and thus each x_j subtree is connected to the W subtree.

Now we assume that F is a solution to the TLTP instance of cost $p + q$. We will show that it must have the same form as the one we would construct from a perfect 3-dimensional matching, and thus we can use it to construct such an M. Recall that the instance has $2p + 2q$ leaves and thus it must be the case that each leaf is incident to exactly one link in F. In particular every w_i leaf must be matched with a $a_{i,j,k}$ leaf and every y_k leaf must be matched with a $a'_{i,j,k}$ leaf. What we wish to show is that, for a fixed $i, j, k \in \{1, \ldots, p\}$ w_i is matched with $a_{i,j,k}$ if and only if y_k is matched with $a'_{i,j,k}$. Suppose not. Without loss of generality we can assume w_i is matched with $a_{i,j,k}$ but y_k is not matched with $a'_{i,j,k}$. But $a'_{i,j,k}$ is only incident with two links in the TLTP instance: $a'_{i,j,k} y_k$ and $a_{i,j,k}a'_{i,j,k}$. So we must have $a_{i,j,k}a'_{i,j,k} \in F$ but then $a_{i,j,k}$ would be incident to both $a_{i,j,k}a'_{i,j,k}$ and $w_i a_{i,j,k}$, a contradiction. □

Also, note that the TLTP instance that we reduced 3DM to uses only cross-links. As such any solution to this TLTP instance is also an NC-TAP solution on the same graph.

Thus, as a corollary, we know that NC-TAP is NP-hard even in the special case where the tree is of height 2 and all links are cross links.

4.7 An Approximation Algorithm For TLTP

We present an LP based approximation algorithm for the Two-Level Tree Problem. This algorithm is a greedy algorithm, based off of a H_p-approximation algorithm for the Set Cover Problem, which the Edge Cover Problem is a special case of. For more information, see [22].

Let $\mathcal{L}$ be the set of leaves of T. Then the following is an LP relaxation for TLTP.

$$
\begin{aligned}
(P) \quad \min \quad & \sum_{l \in L} c_l x_l \\
\text{subject to} \quad & \\
& x(\delta_L(v) \geq 1 \quad \forall v \in \mathcal{L} \\
& x(\delta_L(\mathcal{P})) \geq |\mathcal{P}| - 1 \quad \forall \mathcal{P} \in \mathcal{P}(\text{Comp}(T - r)) \\
& x \geq 0
\end{aligned}
$$

One should note that this LP is a relaxation of the Partition LP for the same graph viewed as an NC-TAP instance. The leaf covering constraint for a given leaf u is one of the partition constraints that one gets when they consider removing the unique neighbour of u. The family of partition constraints ensuring that the $T - r$ is connected is the same family of partition constraints in the NC-TAP instance obtained by deleting r. This LP gives a very straight forward $\frac{7}{3}$-approximation algorithm. First find a minimum cost spanning tree F' on the components of $T - r$. F' will satisfy the partition constraints and have cost at most $c^\top x$. The remaining leaf-covering constraints form the same Edge-Cover Problem mentioned above, abd we know how to obtain an integral solution F'' of cost at most $\frac{4}{3}c^\top x$. Thus $F = F' \cup F''$ will be a TLTP solution of cost at most $\frac{7}{3}c^\top x$. This can easily be improved to a 2-approximation because there is an LP formulation for the Edge-Cover Problem which is integral and valid for TLTP. Thus, using that LP, the F'' we obtain above would have cost at most $c^\top x$. However, we want an approximation guarantee of better than 2.

We now take the dual of this LP to obtain the following.

$$(D) \quad \max \sum_{v \in \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\text{Comp}(T-r))} (|\mathcal{P}| - 1) z_{\mathcal{P}}$$

subject to

$$\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\text{Comp}(T-r)): l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} \leq c_l \quad \forall l \in L$$

$$y, z \geq 0$$

The algorithm works by only considering one partition at once, rather than the possibly exponential number of them. In particular, the only partition considered at a given iteration, which we refer to as the *active partition*, is the one where every connected component of $T \cup F$ is a part, where F is the set of links selected by the algorithm so far. More formally, that active partition is one where two elements of $\text{Comp}(T - r)$ are in the same part if and only if there is a link in F with an endpoint in each component. By restricting ourselves to just one partition at a time, not only does it become possible for the algorithm to run in polynomial time, but also we obtain an important property: a link can only contribute to at most three constraints that are currently being considered. In particular a link can possibly cover two leaves, and it could also cover the active partition. For a link l, we will let S_l be this set of uncovered objects l covers, restricting ourselves to only allowing the active partition. Again note that the elements of S_l are a mixture of vertices, in particular, leaves, and partitions. Each S_l changes with each iteration, but it

never has a size greater than three. Below is the pseudo-code for the algorithm.

Algorithm 3: Greedy Algorithm for TLTP

$F \leftarrow \emptyset$
$\mathcal{P}$ is the partition where every component of $T - r$ is its own part `// P is the active partition`
while *F is not an TLTP solution* **do**
  for *each link $l \in L$* **do**
    S_l is the set of leaves incident to l which are currently not covered by F
    if $l \in \delta_L(\mathcal{P})$ **then**
      $S_l \leftarrow S_l \cup \{\mathcal{P}\}$
    end
    $\hat{c}_l \leftarrow \frac{c_l}{|S_l|}$
  end
  Let $l^* \in L$ have minimum $\hat{c}_l$
  for $\mu \in S_{l^*}$ **do**
    $w_\mu \leftarrow \hat{c}_{l^*}$
  end
  $F \leftarrow F \cup \{l^*\}$
  if $\mathcal{P} \in S_{l^*}$ **then**
    Let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by combining the two parts in $\mathcal{P}$ that contain an endpoint of l^* into one part
    $\mathcal{P} \leftarrow \mathcal{P}'$
  end
end
return F

Consider a given instance of the Two-Level Tree Augmentation Problem, where $\mathcal{L}$ is the set of leaves of T and $s = |\text{Comp}(T - r)|$. Also consider a run of Algorithm 3 on this instance and let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{s-1} \in \mathcal{P}(\text{Comp}(T - r))$ be the partitions that the algorithm considers active in some iteration and assigns weights to, such that $|\mathcal{P}_i| = s - i + 1$. Let F be the solution that the algorithm outputs and $w \in \mathbb{R}_{\geq 0}^{\mathcal{L} \cup \mathcal{P}(\text{Comp}(T-r))}$ the weights assigned to leaves and partitions. Define $y \in \mathbb{R}_{\geq 0}^{\mathcal{L}}$ and $z \in \mathbb{R}_{\geq 0}^{\mathcal{P}(\text{Comp}(T-r))}$ as follows.

$$\begin{aligned} y_u &= w_u && \forall u \in \mathcal{L} \\ z_{\mathcal{P}_1} &= w_{\mathcal{P}_1} && \\ z_{\mathcal{P}_v^i} &= w_{\mathcal{P}_v^i} - w_{\mathcal{P}_v^{i-1}} && \forall i \in \{2, 3, \ldots, s-1\} \\ z_{\mathcal{P}} &= 0 && \forall \mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) \setminus \{\mathcal{P}_1, \ldots, \mathcal{P}_{s-1}\} \end{aligned}$$

Lemma 42. *$\frac{1}{H_3}y, \frac{1}{H_3}z$ are a feasible solution to (D), the dual of the Two-Level Tree Augmentation LP.*

Proof. Let $l \in L$ be an arbitrary link. We will show that

$$\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} \leq H_3 \cdot c_l$$

which is sufficient to prove the desired result, since the nonnegativity constraints are clearly satisfied.

Note that since $z_{\mathcal{P}} = 0$ for all $\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) \setminus \{\mathcal{P}_1, \ldots, \mathcal{P}_{s-1}\}$, we have that $\sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} = \sum_{\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_{s-1}\} : l \in \delta_L(\mathcal{P}_i)} z_{\mathcal{P}_i}$.

Furthermore, if $l \in \delta_L(\mathcal{P}_j)$ for some $1 \leq j \leq s-1$, then $l \in \delta_L(\mathcal{P}_i)$ for all $1 \leq i \leq j$. Thus, if we assume $l \in \delta_L(\mathcal{P}_i)$ for some $1 \leq i \leq s-1$, we can let k be the largest index, such that $l \in \delta_L(\mathcal{P}_k)$, and then we have that

$$\sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} = \sum_{i=1}^{k} z_{\mathcal{P}_i} = w_{\mathcal{P}_1} + \sum_{i=2}^{k} \left(w_{\mathcal{P}_i} - w_{\mathcal{P}_{i-1}} \right) = w_{\mathcal{P}_k}$$

Also, $\sum_{v \in l \cap \mathcal{L}} y_v = \sum_{v \in l \cap \mathcal{L}} w_v$, so $\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} = \sum_{v \in l \cap \mathcal{L}} w_v + w_{\mathcal{P}_k}$. If $l \notin \delta_L(\mathcal{P}_i)$ for any $1 \leq i \leq s-1$ then $\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} = \sum_{v \in l \cap \mathcal{L}} w_v$.

Let S_l be the set of elements (leaves and/or partitions), μ that contribute w_μ to the sum, $\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}}$. Note that $|S_l| \in \{1, 2, 3\}$ and that the elements of S_l can be ordered by when Algorithm 3 assigned a weight to that element, say $\mu_1, \ldots, \mu_{|S_l|}$. Consider the iteration where μ_i was assigned a weight. Note that if $\mathcal{P}_k = \mu_j$ for some $i \leq j \leq |S_l|$ then l always crosses the active partition until $\mu_j = \mathcal{P}_k$ is assigned a weight. Similarly, if there is a leaf v such that $v = \mu_j$ for some $i \leq j \leq |S_l|$ then l is incident to v

and v is not incident to any link the algorithm has already purchased. Thus the algorithm considered purchasing l and dividing its cost between $\mu_i, \ldots, \mu_{|S_l|}$. Thus we must have

$$w_{\mu_i} \leq \frac{c_l}{|S_l| - i + 1}$$

And thus we have

$$\begin{aligned}
\sum_{v \in l \cap \mathcal{L}} y_v + \sum_{\mathcal{P} \in \mathcal{P}(\mathrm{Comp}(T-r)) : l \in \delta_L(\mathcal{P})} z_{\mathcal{P}} &= \sum_{i=1}^{|S_l|} w_{e_i} \\
&\leq \sum_{i=1}^{|S_l|} \frac{c_l}{|S_l| - i + 1} \\
&= c_l \sum_{i=1}^{|S_l|} \frac{1}{i} \\
&= H_{|S_l|} \cdot c_l \\
&\leq H_3 \cdot c_l
\end{aligned}$$

□

As a corollary, we obtain the following result.

Theorem 43. *Algorithm 3 outputs a solution F such that $c(F) \leq H_3 \cdot LP_{OPT}$, where LP_{OPT} is the optimal value of (P).*

Proof. Note that the objective value of $\frac{1}{H_3}y, \frac{1}{H_3}z$ as solutions to (D) is

$$\begin{aligned}
&\frac{1}{H_3}\left(\sum_{v\in\mathcal{L}} y_v + \sum_{\mathcal{P}\in\mathcal{P}(\mathrm{Comp}(T-r))} (|\mathcal{P}|-1)z_{\mathcal{P}}\right) \\
&= \frac{1}{H_3}\left(\sum_{v\in\mathcal{L}} y_v + \sum_{i=1}^{s-1}(s-i)z_{\mathcal{P}_i}\right) \\
&= \frac{1}{H_3}\left(\sum_{v\in\mathcal{L}} w_v + (s-1)w_{\mathcal{P}_1} + \sum_{i=2}^{s-1}(s-i)(w_{\mathcal{P}_i} - w_{\mathcal{P}_{i-1}})\right) \\
&= \frac{1}{H_3}\left(\sum_{v\in\mathcal{L}} w_v + \sum_{i=1}^{s-1}(s-i)w_{\mathcal{P}_i} - \sum_{i=1}^{s-2}(s-i-1)w_{\mathcal{P}_i}\right) \\
&= \frac{1}{H_3}\left(\sum_{v\in\mathcal{L}} w_v + \sum_{i=1}^{s-1} w_{\mathcal{P}_i}\right) \\
&= \frac{1}{H_3}\cdot c(F)
\end{aligned}$$

And thus $c(F) \leq H_3 \cdot LP_{\mathrm{OPT}}$ by duality.

□

It turns out that the analysis of this algorithm is tight. Figure 4.6 shows an instance of TLTP where tightness occurs. Notice that, other than the link labelled l^*, each link satisfies one constraint. The link of cost 2 covers the left-most leaf, the link of cost 3 covers the right-most leaf, and the link between the subroots satisfies the only partition constraint. l^* satisfies all three at once. The algorithm will begin by considering the adjusted cost of l^* which is $\frac{c_{l^*}}{3}$, since l^* satisfies three constraints. However, for any $\epsilon > 0$ this is still greater than 2, so instead the algorithm will select the link of cost 2. In the next iteration, the algorithm will give l^* an adjusted cost of $\frac{c_{l^*}}{2}$, since l^* satisfies two constraints that are not yet satisfied. However, this is greater than 3, so the algorithm will select the link of cost 3. In the final iteration, the algorithm will give l^* an adjusted cost of c_{l^*} since the partition constraint is the only constraint left to satisfy. This is greater than 6, so the algorithm will select the link of cost 6. Thus the algorithm selects a solution of cost $2+3+6=11$ however, just l^* is a solution of cost $6+\epsilon$.

In fact the integrality gap of the above LP for TLTP has an integrality gap of at least $\frac{4}{3}$. Figure 4.7 shows an instance of TLTP with an optimal solution of cost 2 and where the

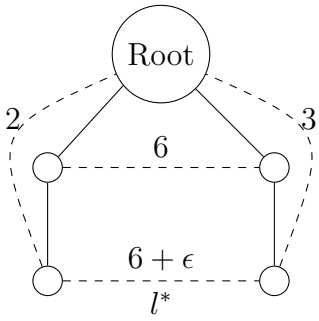


Figure 4.6: An instance of TLTP where Algorithm 3 returns a solution of cost $\frac{11}{6}$ times the optimal cost

above LP has an optimal solution of cost $\frac{3}{2}$. In fact, this example only uses cross-links, and thus the TLTP LP above is exactly the same as the Partition LP for NC-TAP. Thus not only is the integrality gap for the TLTP LP $\frac{4}{3}$ but so is the integrality gap of the Partition LP for NC-TAP, even on instances where the spanning tree has height 2 and which only have cross-links.

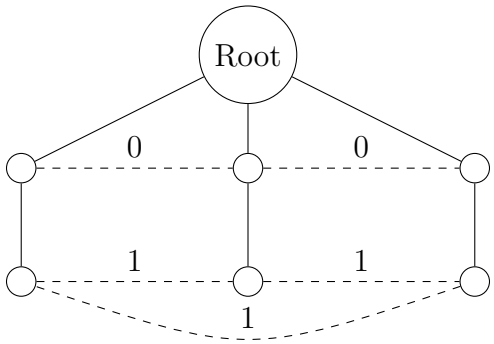


Figure 4.7: An instance of TLTP/NC-TAP where the Partition LP has an integrality gap of $\frac{4}{3}$. An integer solution must take two links of cost 1 to cover the leaves. The LP optimum will assign $x_l = 1$ to the links of cost 0, and $x_l = \frac{1}{2}$ to the links of cost 1. The objective value of this LP solution is $\frac{3}{2}$.

4.8 Next Steps

Unfortunately, there is still a lot of work to be done to use the techniques laid out in [1] for TAP in the setting of NC-TAP. First, we have to do away with the assumption that no tree edge is covered by more than a constant amount in the optimal fractional solution. Second, we have to find a way to transform instances with a large proportion of their cost coming from cross-links into instances where there are only cross-links. However, if we can do this last step by paying a factor of 2 on the in-links, as is done in the case of TAP, we would succeed in extending Adjiashvili's techniques to NC-TAP and obtain $\frac{25}{13}$-approximation for NC-TAP with bounded costs.

References

[1] David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Trans. Algorithms*, 15(2), December 2018.

[2] Jaroslaw Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 815–825. ACM, 2020.

[3] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1), February 2013.

[4] Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part i: Stemless tap. *Algorithmica*, 80(2):530–559, Feb 2018.

[5] Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part ii. *Algorithmica*, 80(2):608–651, Feb 2018.

[6] Nachshon Cohen and Zeev Nutov. A $(1 + \ln 2)$-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. In Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 147–157, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[7] Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors. *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. ACM, 2018.

[8] R. Diestel. *Graph Theory (4th ed.)*. Graduate Texts in Mathematics, Volume 173. Springer-Verlag, Heidelberg, 2010.

[9] Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 817–831. SIAM, 2018. Made available as a preprint at http://arxiv.org/abs/1702.05567, 2017.

[10] Greg N. Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

[11] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, page 223–232, USA, 1994. Society for Industrial and Applied Mathematics.

[12] Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In Diakonikolas et al. [7], pages 632–645.

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

[14] 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.

[15] Lap Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics (No. 46). Cambridge University Press, 2011.

[16] Zeev Nutov. On the tree augmentation problem. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 61:1–61:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[17] Zeev Nutov. 2-node-connectivity network design. *CoRR*, abs/2002.04048, 2020.

[18] R. Ravi and David P. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 18(1):21–43, 1997.

[19] R. Ravi and David P. Williamson. Erratum: An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 34(1):98–107, 2002.

[20] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, Volume 24. Springer-Verlag, Berlin Heidelberg, 2003.

[21] David Williamson, Michel Goemans, Milena Mihail, and Vijay Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15:435–454, 01 1995.

[22] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

APPENDICES

Appendix A

Properties of the Super-Harmonic Numbers and the Related Function

Here we prove propositions 21 through 26, which are about the super-harmonic numbers and the function f which was defined on sequences of integers.

Proposition. *21. For any positive integer i, $\hat{H}_{i+1} = 2\hat{H}_i - H_i$.*

Proof.

$$2\hat{H}_i = 2\sum_{j\geq 0} \frac{1}{2^{j+1}} H_{i+j} = 2\left(\frac{1}{2}H_i + \sum_{j\geq 0} \frac{1}{2^{j+2}} H_{i+j+1}\right) = H_i + \hat{H}_{i+1}$$

□

Proposition. *22. [2] For any positive integer i, $\hat{H}_{i+1} - \hat{H}_i = \hat{H}_i - H_i = \sum_{k\geq 1} \frac{1}{(i+k)2^k}$.*

Proof. From Proposition 21 we have

$$\begin{aligned} \hat{H}_{i+1} - \hat{H}_i &= (2\hat{H}_i - H_i) - \hat{H}_i \\ &= \hat{H}_i - H_i \end{aligned}$$

And

$$\begin{aligned}
\hat{H}_i - H_i &= \left(\sum_{j\geq 0} \frac{1}{2^{j+1}} H_{i+j}\right) - H_i \\
&= \sum_{j\geq 0} \frac{1}{2^{j+1}} \left(H_{i+j} - H_i\right) \\
&= \sum_{j\geq 0} \frac{1}{2^{j+1}} \left(\sum_{k=1}^{j} \frac{1}{i+k}\right) \\
&= \sum_{k\geq 1} \frac{1}{i+k} \left(\sum_{j\geq k} \frac{1}{2^{j+1}}\right) \\
&= \sum_{k\geq 1} \frac{1}{(i+k)2^k}
\end{aligned}$$

□

Proposition. *23. For any positive integer i, $\hat{H}_{i+1} - \hat{H}_i = \hat{H}_i - H_i \leq \frac{1}{i+1}$*

Proof. From Proposition 22 we know

$$\hat{H}_i - H_i = \hat{H}_i - H_i = \sum_{k\geq 1} \frac{1}{(i+k)2^k} \leq \sum_{k\geq 1} \frac{1}{(i+1)2^k} = \frac{1}{i+1}$$

□

Proposition. *24.* $\hat{H}_1 = 2\ln 2$.

Proof.

$$\hat{H}_1 = \sum_{j\geq 1} \left(\frac{1}{2^j} \sum_{i=1}^{j} \frac{1}{i}\right) = \sum_{i\geq 1} \frac{1}{i} \sum_{j\geq i} \frac{1}{2^j} = \sum_{i\geq 1} \frac{2}{i2^i} = -2 \sum_{i\geq 1} \frac{(-1)^{i+1}(\frac{1}{2}-1)^i}{i} = -2\ln\frac{1}{2}$$

Where the last equality follows from the Taylor Series of $\ln x$ at 1. □

Recall that for a finite sequence of positive integers, $S = (d_1, \ldots, d_k)$ we define

$$f(S) = \sum_{i=1}^{k-1} \frac{(d_{i+1}-1)H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}} + \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k}$$

And for an infinite sequence of positive integers $S' = (d_1, d_2, \ldots)$ we define

$$f(S') = \sum_{i \geq 1} \frac{(d_{i+1}-1)H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}}$$

Proposition. *25. [2] $f(S) \leq f(\bar{S})$ for any finite sequence of positive integers, $S = (d_1, \ldots, d_k)$*

Proof.

$$\begin{aligned}
f(\bar{S}) - f(S) &= \sum_{i \geq 1} \frac{(d_{i+1}-1)H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}} - \sum_{i=1}^{k-1} \frac{(d_{i+1}-1)H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}} - \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k} \\
&= \sum_{i \geq k} \frac{(d_{i+1}-1)H_{d_1+\ldots+d_i-i+1}}{d_2 \cdot \ldots \cdot d_{i+1}} - \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k} \\
&\geq \sum_{i \geq k} \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_{i+1}} - \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k} \\
&= \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k} \left(\sum_{i \geq k} \frac{1}{d_{k+1} \cdot \ldots \cdot d_{i+1}} - 1 \right) \\
&= \frac{H_{d_1+\ldots+d_k-k+1}}{d_2 \cdot \ldots \cdot d_k} \left(\sum_{i \geq 1} \frac{1}{2^i} - 1 \right) \\
&= 0
\end{aligned}$$

□

Proposition. *26. [2] Let $\bar{S} = (d_1, \ldots, d_k, 2, 2, \ldots)$ and assume that for some $i \geq 2$ we have $d_i = 1$. Let $\bar{S}_i = (d_1, \ldots, d_{i-1} - 1, d_{i+1} + 1, d_k, 2, 2, \ldots)$. Then $f(\bar{S}) = f(\bar{S}_i)$.*

Proof. We will show these two sums have equal terms. Note that for $j < i - 1$ the j-th terms of $f(\bar{S})$ and $f(\bar{S}i)$ are identical. For $j = i - 1$, since $d_{j+1} - 1 = d_i - 1 = 0$ the j-th

term in $f(S)$ is 0, corresponding to the now non-existent term in $f(\bar{S}i)$. For $j > i-1$, the $j-1$-th item of the sequence $\bar{S}_i$ is the j-th entry of $\bar{S}$, and the $j-1$-th term of $f(\bar{S}i)$ is

$$\frac{(d_{j+1}-1)H_{d_1+\ldots+d_j-d_i-(j-1)+1}}{d_2\cdot\ldots\cdot d_{j+1}\frac{1}{d_i}} = \frac{(d_{j+1}-1)H_{d_1+\ldots+d_j-1-j+1+1}}{d_2\cdot\ldots\cdot d_{j+1}} = \frac{(d_{j+1}-1)H_{d_1+\ldots+d_j-j+1}}{d_2\cdot\ldots\cdot d_{j+1}}$$

which is the j-th term of $f(\bar{S})$. □

Appendix B

Bounding the Domain of Averaged Cost Functions for Steiner Tree Algorithm

Proposition. *30. [2] For any real $p \in [0, \hat{H}_2 - H_2]$, a_{ggf} obtains its global maximum in the range $\{1, 2, 3\}$, a_{gbf} obtains its global maximum when $n = 1$, a_{bgf} obtains its global maximum in the range $\{6, 7, 8\}$, and a_{bbf} obtains its global maximum when $n = 7$.*

Proof. We will look at $a_i(n+1) - a_i(n)$ for all $i \in \{ggf, gbf, bgf, bbf\}$. We will prove this discrete derivative is decreasing, and then find points where it is negative independent of our choice of p. Occassionally we can also find a point where the discrete derivative is positive, giving a lower bound on the domain where the maximizer may exist. Since $a_{bbf}(n)$ is independent of p we start with it.

$$\begin{aligned} a_{bbf}(n+1) - a_{bbf}(n) &= \frac{\hat{H}_{n+1} - \hat{H}_2}{n+1} - \frac{\hat{H}_n - \hat{H}_2}{n} \\ &= \frac{1}{n(n+1)} \left(n\hat{H}_{n+1} - (n+1)\hat{H}_n + \hat{H}_2 \right) \\ &\leq \frac{1}{n(n+1)} \left(1 + \hat{H}_2 - \hat{H}_n \right) \end{aligned}$$

It is clear that this is decreasing. Furthermore,

$$\begin{aligned} a_{bbf}(8) - a_{bbf}(7) &= \frac{1}{56}\left(\frac{7}{8} - \hat{H}_7 + \hat{H}_2\right) \\ &= \frac{1}{56}\left(\frac{7}{8} - 124\ln 2 + \frac{5101}{60}\right) \\ &< \frac{1}{56}\left(\frac{7}{8} - 124\frac{665}{960} + \frac{5101}{60}\right) \\ &= \frac{1}{56}\left(\frac{-4}{960}\right) \\ &< 0 \end{aligned}$$

and

$$\begin{aligned} a_{bbf}(7) - a_{bbf}(6) &= \frac{1}{42}\left(\frac{6}{7} - \hat{H}_6 + \hat{H}_2\right) \\ &= \frac{1}{42}\left(\frac{6}{7} - 60\ln 2 + \frac{2447}{60}\right) \\ &> \frac{1}{42}\left(\frac{6}{7} - 60\frac{437}{630} + \frac{2447}{60}\right) \\ &= \frac{1}{42}\left(\frac{27}{1260}\right) \\ &> 0 \end{aligned}$$

Thus we know that $a_{bbf}(n)$ must obtain its maximum when $n = 7$. Hence, the only value of $a_{bbf}(n)$ we are concerned with is

$$a_{bbf}(7) = \frac{\hat{H}_7 + 6\hat{H}_2}{7} = \frac{152\ln 2}{7} - \frac{5521}{420} < 2$$

So we see that it is possible that the correct choice of p will give us an approximation guarantee better than 2. It turns out that the next easiest function to analyze is a_{bgf}.

$$\begin{aligned} a_{bgf}(n+1) - a_{bgf}(n) &= \frac{\hat{H}_{n+3} - \hat{H}_2}{n+1} - \frac{\hat{H}_{n+2} - \hat{H}_2}{n} \\ &= \frac{1}{n(n+1)} \left(n\hat{H}_{n+3} - (n+1)\hat{H}_{n+2} + \hat{H}_2 \right) \\ &\leq \frac{1}{n(n+1)} \left(1 + \hat{H}_2 - \hat{H}_{n+2} \right) \end{aligned}$$

Again, this is decreasing and

$$\begin{aligned} a_{bgf}(2) - a_{bgf}(1) &= \frac{1}{2} \left(\frac{1}{4} - \hat{H}_3 + \hat{H}_2 \right) \\ &= \frac{1}{2} \left(\frac{1}{4} - 4\ln 2 + \frac{5}{2} \right) \\ &< \frac{1}{2} \left(\frac{1}{4} - 4\frac{69}{100} + \frac{5}{2} \right) \\ &= \frac{1}{2} \left(\frac{-1}{100} \right) \\ &< 0 \end{aligned}$$

Thus $a_{gbf}(n)$ must obtain its maximum when $n = 1$ and thus the only value of $a_{bgf}(n)$ we are concerned with is

$$a_{bgf}(1) = \hat{H}_3 - p$$

Since we want our approximation ration to be strictly less than 2, we will need $\hat{H}_3 - p < 2$, which gives $p > 8\ln 2 - \frac{11}{2}$. With this new lower bound on p, we turn our attention to a_{gbf} a_{ggf}.

$$\begin{aligned} a_{gbf}(n+1) - a_{gbf}(n) &= \frac{H_{n+1} - \hat{H}_2 + p}{n+1} - \frac{H_n - \hat{H}_2 + p}{n} \\ &= \frac{1}{n(n+1)} \Big(nH_{n+1} - (n+1)H_n + \hat{H}_2 - p \Big) \\ &\leq \frac{1}{n(n+1)} \Big(1 + \hat{H}_2 - H_n - p \Big) \end{aligned}$$

Again, this is decreasing and

$$\begin{aligned} a_{gbf}(9) - a_{gbf}(8) &= \frac{1}{72} \left(\frac{8}{9} - H_8 + \hat{H}_2 - p \right) \\ &= \frac{1}{72} \left(\frac{8}{9} - \frac{761}{280} + 4\ln 2 - 1 - p \right) \\ &< \frac{1}{72} \left(\frac{8}{9} - \frac{761}{280} + 4\frac{7}{10} - 1 - p \right) \\ &= \frac{1}{72} \left(\frac{-73}{2520} - p \right) \\ &< 0 \end{aligned}$$

Also,

$$\begin{aligned} a_{gbf}(6) - a_{gbf}(5) &= \frac{1}{30} \left(\frac{5}{6} - H_5 + \hat{H}_2 - p \right) \\ &\geq \frac{1}{30} \left(\frac{5}{6} - H_5 + \hat{H}_2 - \hat{H}_2 + H_2 \right) \\ &= \frac{1}{30} \left(\frac{3}{60} - p \right) \\ &> 0 \end{aligned}$$

Thus $a_{gbf}(n)$ must obtain its maximum when $n \in \{6, 7, 8\}$ and thus the values of $a_{gbf}(n)$

we are concerned with are

$$a_{gbf}(6) = \frac{H_6 + 5\hat{H}_2 + p}{6}$$
$$a_{gbf}(7) = \frac{H_7 + 6\hat{H}_2 + p}{7}$$
$$a_{gbf}(8) = \frac{H_8 + 7\hat{H}_2 + p}{8}$$

Finally,

$$\begin{aligned} a_{ggf}(n+1) - a_{ggf}(n) &= \frac{H_{n+3} - \hat{H}_2 + p}{n+1} - \frac{H_{n+2} - \hat{H}_2 + p}{n} \\ &= \frac{1}{n(n+1)}\left(nH_{n+3} - (n+1)H_{n+2} + \hat{H}_2 - p\right) \\ &\leq \frac{1}{n(n+1)}\left(1 + \hat{H}_2 - H_{n+2} - p\right) \end{aligned}$$

Again the bound is decreasing and

$$\begin{aligned} a_{ggf}(4) - a_{ggf}(3) &= \frac{1}{12}\left(\frac{3}{6} - H_5 + \hat{H}_2 - p\right) \\ &< \frac{1}{12}\left(\frac{1}{2} - \frac{137}{60} + 4\ln 2 - 1 - 8\ln 2 + \frac{11}{2}\right) \\ &< \frac{1}{12}\left(3 - \frac{17}{60} - 4\frac{41}{60}+\right) \\ &= \frac{1}{12}\left(\frac{-1}{60}\right) \\ &< 0 \end{aligned}$$

Thus $a_{ggf}(n)$ must obtain its maximum when $n \in \{1, 2, 3\}$ and thus the values of $a_{ggf}(n)$

we are concerned with are

$$a_{ggf}(1) = H_3$$
$$a_{ggf}(2) = \frac{H_4 + \hat{H}_2 - p}{2}$$
$$a_{ggf}(3) = \frac{H_5 + 2\hat{H}_2 - 2p}{3}$$

□

Glossary

$\Gamma_F(S)$	$:= \{v \in V : \exists u \in S, uv \in F\}$ the *vertex neighbourhood of* S
$\zeta_F(S)$	$:= V \setminus (S \cup \Gamma_F(S))$
$\delta_F(S : S')$	$:= \{uv \in F : u \in S, v \in S'\}$ the *cut from* S *to* S'
$\mathrm{Comp}(H)$	the set of all connected components of subgraph $H \subseteq G$
$\kappa_H(s,t)$	the maximum size of a set of internally-disjoint s,t-paths in $H \subseteq G$
$\kappa'_H(s,t)$	the maximum size of a set of edge-disjoint s,t-paths in $H \subseteq G$
P_{st}	the unique s,t-path in T
Shadow	(of a link l) a link l' such that $P_{l'} \subseteq P_l$
$\mathcal{P}(S)$	the set of partitions of an arbitrary set S
$\lvert\mathcal{P}\rvert$	the number of parts in a partition $\mathcal{P}$
$\delta_F(\mathcal{P})$	$:= \{uv \in F : u$ and v are in different parts of $\mathcal{P}\}$
H_i	$:= \sum_{j=1}^{i} \frac{1}{j}$ the i-th harmonic number
$\hat{H}_i$	$:= \sum_{j \geq 1} \frac{1}{2^{j+1}} H_{i+j}$ the i-th super-harmonic number
β-bundle	a union of at most β paths of T such that the union connected
B_β	the set of all β-bundles
β-simple pair	a union of at most β paths of T such that the union connected