EPFL

Thèse n° 9946

Approximation Algorithms for Allocation and Network Design

Présentée le 9 juin 2023

Faculté informatique et communications
Laboratoire de théorie du calcul 2
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Etienne Michel François BAMAS

Acceptée sur proposition du jury

Prof. E. Telatar, président du jury
Prof. O. N. A. Svensson, directeur de thèse
Prof. A. Gupta, rapporteur
Prof. F. Grandoni, rapporteur
Prof. F. Eisenbrand, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

To my parents,
and Johanna.

ACKNOWLEDGMENTS

I am indebted to many people whose support and encouragement were crucial during my journey towards completing my PhD thesis. The years I spent at EPFL have been a fantastic experience, and I sincerely think that I have been lucky in many ways until this moment. Now is the time to acknowledge it.

I would like to express my deepest gratitude to my advisor, Ola Svensson, for being an exceptional advisor and a role model as a researcher. His never-ending enthusiasm is truly contagious, and it is one of the reasons why working in the Theory Lab has been so enjoyable to me. He often aims at solving problems known to be very difficult, and seeing his success with that strategy has been very inspiring. Even if he is not a co-author on several papers in this thesis, he definitely had a big influence on all of them. In addition to being a fantastic researcher, Ola always showed support in various situations, from Covid times to life after the PhD. For this, I would like to thank him.

I am also grateful to the members of my thesis committee Fritz Eisenbrand, Fabrizio Grandoni, Anupam Gupta, and Emre Telatar, for the careful reading of this thesis and the interesting discussion during the defense.

I have been very lucky to work with amazing co-authors during my time at EPFL and the results of this thesis would not have existed without them. For this, I would like to thank Marina Drygala, Paritosh Garg, Andreas Maggiori, Lars Rohwedder, and my advisor Ola. I would also like to give a special thanks to Lars, with whom I learned a lot about the Santa Claus problem, which is now one of my favorite problems. I should also thank Louis Esperet, who was my advisor during my master thesis, and is one of the people that inspired me to pursue a PhD.

I am very grateful to Adam, Andreas, Chantal, Muriel, and Pauline with whom we organized the workshop ALPS in 2022, inviting almost 50 people from all over the world to come to EPFL for one week. It was a pleasure organizing this with you. Special thanks to Chantal for also being a great help in many other administrative or organizational aspects of my life, and for always being friendly and welcoming.

During my PhD, I also had the opportunity to travel a lot for academic visits. For this, I must thank Lars for inviting me in his lab in Maastricht, the Simons Institute for giving me the opportunity to visit UC Berkeley, and Rico Zenklusen for inviting me during an entire month at ETH Zurich.

I enjoyed a lot being part of the Theory Group and I would like to thank all my friends and colleagues there for the friendly atmosphere that they created, the very nice activities we did together, and the fun discussions on many topics. Moreover, I am especially grateful to Xinrui for proofreading the introduction of this thesis.

Although I will not attempt to name all of them, I want to thank all my friends in France and Switzerland for making these years so enjoyable. Mention spéciale à mes colocs et amis Arnout, Quentin, et Sylvain pour ces belles

années de doctorat ponctuées d'épisodes de Top Chef, de moments difficiles avec notre ami Shaun T, de randos, d'innombrables virées à KFC, et j'en passe.

Je tiens à exprimer ma reconnaissance particulière envers Géraldine et (un autre) Sylvain, qui m'ont accueilli chez eux pendant plusieurs mois durant l'épidémie de Covid. Je me suis toujours senti le bienvenu chez eux.

Mes plus sincères remerciements vont à ma famille, en particulier mes parents Nathalie et Philippe, mon frère Arthur, et ma soeur Eliette. Sans leur soutien, leur amour, et leurs encouragements sans faille depuis 28 ans, je ne serais sans doute pas là où je suis aujourd'hui.

Je voudrais dédier ces dernières lignes à ma compagne Johanna. Johanna, j'aurais sans doute dû écrire ton nom dans chacun des 3 derniers paragraphes tellement tu es importante à mes yeux. Je veux te remercier pour ces belles années à tes côtés, et j'espère en vivre beaucoup d'autres avec toi.

ABSTRACT

In this thesis, we give new approximation algorithms for some NP-hard problems arising in resource allocation and network design. As a resource allocation problem, we study the Santa Claus problem (also known as the MaxMin Fair Allocation problem) in which we are given m agents, n indivisible resources, and we have to allocate resources in order to maximize the happiness of the least happy agent. For the network design part, we study two key problems: the Steiner Forest problem and the Matching Augmentation problem.

In the first part of this thesis, we give improved guarantees for the Santa Claus problem in two prominent settings. First, we consider the Santa Claus problem in the setting where the happiness of each agent i is an arbitrary linear function f_i. Obtaining a constant factor approximation in that setting is a major open problem in the area of approximation algorithms. In 2009, the MaxMin Arborescences problem was identified as a key special case that appears to capture most of the difficulty of the general problem. Even in that special case, a constant factor approximation has remained elusive, and the current best algorithm only guarantees a polylogarithmic approximation in quasi-polynomial time. We give an exponential improvement to this, an $O(\text{poly}(\log\log(n)))$-approximation in quasi-polynomial time for the MaxMin Arborescences problem. Our second result in this part considers the Santa Claus problem in the restricted assignment case: all the agents have the same happiness function f, but each agent i is only interested in a subset Γ_i of the resources. When f is a monotone submodular function, we show that we can obtain an $O(\log\log n)$-approximation in polynomial time. Before our work, comparable results in this setting were only known for the case where f is a linear function.

In the second part of this thesis, we start with the Steiner Forest problem which is defined as follows. Given a graph G and an arbitrary set of k terminal pairs $\{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$, the goal is to return a minimum-weight subgraph that connects all the pairs. In 1996, Awerbuch, Azar, and Bartal showed that an intuitive greedy algorithm guarantees an $O(\log^2 k)$-approximation. Our first result is to show that, in fact, the greedy algorithm guarantees an $O(\log(k) \cdot \log\log(k))$-approximation, which is nearly tight in light of a known lower bound of order $\Omega(\log k)$. Interestingly, our analysis also gives important insights in the online setting of the problem, in which the pairs are revealed one by one in an adversarial order. Our last result is on the Matching Augmentation problem, a key problem to compute cheap 2-edge connected subgraphs. We give a simple polynomial-time algorithm that guarantees a better-than-2 approximation when compared to the standard relaxation of the problem known as the cut LP. In contrast, previous better-than-2 approximations were much more complicated and did not compare to this simple relaxation.

Keywords— approximation algorithms, Santa Claus, maxmin allocation, network design, Steiner forest, matching augmentation.

RÉSUMÉ

Dans cette thèse de doctorat, nous proposons de nouveaux algorithmes d'approximation pour des problèmes NP-difficiles d'allocation de ressources et de conception de réseaux. Dans le domaine d'allocation de ressources, nous étudions le problème du Père Noël (aussi connu sous le nom de problème d'allocation MaxMin) : étant donné m individus et n ressources indivisibles, le but est d'allouer les ressources afin de maximiser le bonheur de l'individu le moins heureux. Dans le domaine de la conception de réseaux, nous étudions deux problèmes importants : le problème de la forêt de Steiner et le problème d'augmentation de couplage.

Dans la première partie de cette thèse, nous donnons de meilleures garanties pour le problème du Père Noël dans deux cas importants. En premier lieu, nous considérons le problème du Père Noël dans le cas où le bonheur de chaque individu i est défini par une fonction linéaire quelconque f_i. Obtenir une approximation de facteur constant dans ce cas est un problème ouvert majeur dans le domaine des algorithmes d'approximation. En 2009, le problème d'arborescences MaxMin a été identifié comme un cas particulier qui semble capturer l'essence du problème général. Même dans ce cas particulier, une approximation de facteur constant échappe encore largement aux techniques actuelles, et l'état de l'art ne peut garantir qu'une approximation de facteur polylogarithmique en temps quasi-polynomial. Nous proposons une approximation de facteur $O(\text{poly}(\log\log(n)))$ en temps quasi-polynomial pour le problème d'arborescences MaxMin, ce qui constitue une amélioration exponentielle de l'état de l'art. Notre deuxième résultat traite du problème du Père Noël avec des restrictions d'allocation : le bonheur de tous les individus est décrit par la même fonction f, mais chaque individu i n'accepte que les ressources appartenant à un sous-ensemble Γ_i. Dans le cas où f est une fonction sous-modulaire et monotone, nous obtenons une approximation de facteur $O(\log\log n)$ en temps polynomial. Avant nos travaux, des résultats comparables n'étaient connus que dans le cas où f est une fonction linéaire.

Dans la deuxième partie de cette thèse, nous commençons par étudier le problème de la forêt de Steiner : étant donné un graphe G et un ensemble de k paires de terminaux $\{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$, nous devons trouver le sous-graphe de poids minimum qui connecte toutes les paires. En 1996, Awerbuch, Azar, et Bartal ont montré qu'un simple algorithme glouton garantit une approximation de facteur $O(\log^2 k)$. Notre premier résultat est de montrer que l'algorithme glouton retourne en fait une approximation de facteur $O(\log(k) \cdot \log\log(k))$, ce qui est presque optimal en raison d'une borne inférieure connue d'ordre $\Omega(\log k)$. De plus, nos résultats apportent une compréhension nouvelle de la version en-ligne du problème, lorsque les paires sont révélées les unes après les autres par un adversaire. Notre dernier résultat concerne le problème d'augmentation de couplage, un problème important pour trouver des sous-graphes 2-connectés de poids minimum. Nous proposons un algorithme simple (et de complexité polynomiale) qui garantit une approximation de facteur strictement inférieur à 2 par rapport à

la valeur de la relaxation linéaire du problème. En comparaison, les autres algorithmes connus pour garantir une approximation de facteur strictement inférieur à 2 sont bien plus complexes et ne peuvent pas être comparés à cette relaxation linéaire.

Mots-clefs— algorithmes d'approximation, Père Noël, allocation maxmin, conception de réseaux, forêt de Steiner, augmentation de couplage.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Many problems faced every day can be phrased as a discrete optimization problem, in which there is a finite set of possible solutions. Some examples (among many) include finding the closest supermarket, finding the best assignment of students to universities, or even how to design the energy network of a country. It is quite clear that some of these questions are of significant and global importance for a big part of the population. Many of these problems are now solved automatically by a computer that runs an algorithm, and it is therefore natural that these algorithms have a high importance in our lives. Even seemingly minor decisions can have big influence on society at large when millions of individuals rely on them, and this naturally creates the need for a thorough and rigorous understanding of these algorithms.

Regrettably, many of these problems can have a complex structure which often makes it very difficult (if not impossible) to find the best solution in a reasonable amount of time. This obstacle motivated the study of *approximation algorithms*, which are algorithms designed to find approximate solutions to optimization problems. Approximation algorithms are often used when it is impractical or impossible to find the best solution, because the time required to find that solution is prohibitive. The output of these algorithms may not be perfect, but can still provide valuable insights and help us make better decisions. This is the main topic of this thesis, which focuses on the design and analysis of approximation algorithms with provable, worst-case guarantees.

As a concrete illustration, imagine the following situation. You are managing a factory in which there is a set M of m identical machines that can perform jobs for you. Each day you are given a new set J of n jobs to be distributed among the m machines, where job i takes time p_i to be completed by a machine (the machines can perform several types of jobs, which may not need the same time). Because you want to save electricity and go home earlier, you are interested in how to assign these tasks to the machines so as to minimize the overall completion time (also called the makespan). In future references, we will call this problem the MAKESPAN problem. A simple solution would be to try all possible assignments of jobs to machine and keep the best one; this is what we call an *exhaustive search*. But how long would your computer need to perform this exhaustive search? Let us perform a quick calculation, imagine you have 40 tasks and 10 machines (this is a very small factory): then there are 10^{40} possible assignments. Unfortunately, this number is already astronomically big. Assuming we had *all* the computational power of the world (often estimated to be around 10^{20} basic operations per second, see [57] and references therein), then we would need at least 317 years to find the best assignment. With 44 jobs we would need more than a million years.

This exponential growth of the number of solutions is sometimes referred to as *combinatorial explosion*. Very often, this makes it prohibitive to try an

exhaustive search and necessitates the design of a faster algorithm. Traditionally, an algorithm is considered to be *efficient* if it runs in time that is a polynomial function of the input size, and the class of problems for which there is such an algorithm is denoted by P. Unfortunately, the MakeSpan problem belongs to a wide and significant class of problems called NP-*hard problems*, for which it is widely believed that there is no such efficient algorithm[1].

Fortunately, this is not the end of the story. Here, we were looking for an *exact* solution to the Makespan problem but an *approximate* solution might be satisfying as well. For instance, for the Makespan problem, there exists an algorithm that runs in polynomial time and returns a solution that is *always* within 1% of the optimal solution (see [54]). This result shows that it is possible to sacrifice a tiny bit of accuracy to save one of the most valuable resources, time. Clearly, it is not worth waiting for millions of years in order to improve the solution by 1%.

The above example falls into the realm of approximation algorithms, which are widely used when facing NP-hard problems. We say that an algorithm has *approximation ratio* α if it always returns a solution that is not worse than α times the cost of the best solution (for instance, the above algorithm has approximation ratio 1.01). Now, a fundamental question that comes to mind is how much accuracy do we need to lose in order to be able to find an efficient algorithm? To answer this question, researchers have designed increasingly elaborate algorithms to get better and better approximation ratios. These results often came with very interesting insights and new algorithmic techniques that turned out to be useful for other problems. Unfortunately, during this quest of the best approximation guarantee, another crucial property often got lost on the way: simplicity. This is also very important, because extremely complicated algorithms are difficult to use in practice and often remain in the world of theory. For many problems, it is therefore tempting to design a simple and intuitive algorithm. In the Makespan problem, we can consider the following very simple algorithm: schedule the jobs one by one (in any order), so that each job gets assigned to the machine that has been assigned the least amount of work so far. This is very natural as we intuitively want to balance the loads on the machines. It turns out that this algorithm has an approximation ratio of 2 (see [85]). It is extremely simple and runs very fast in practice, which might be an advantage over a more complicated algorithm with a better approximation ratio.

In this thesis, we focus on problems that are known to be NP-hard, which means that there is probably no efficient algorithm to solve the problem (like the Makespan problem defined above). We start by focusing on the Santa Claus problem, a fundamental problem in resource allocation. Second, we study two key network design problems: the Steiner Forest problem and the Matching Augmentation problem. For the Santa Claus problem, we aim at finding the best approximation ratio that an efficient algorithm can guarantee (without worrying about the simplicity of the algorithm). For the network design problems, we focus on analyzing the approximation ratio guaranteed

1 In fact, NP-hard problems can have polynomial time algorithms only if $\mathrm{P} = \mathrm{NP}$, an assertion widely believed to be false. The $\mathrm{P} \neq \mathrm{NP}$ conjecture is one of the Millenium Prize Problems, a list of seven of the most important problems in mathematics.

by a simple and intuitive heuristic. Interestingly, all our results leverage some linear programming relaxation of the problem at hand. We will elaborate further on this concept at the end of this introduction (see Section 1.3).

1.1 OUR CONTRIBUTIONS

We give new approximation algorithms for various settings of the Santa Claus problem, a fundamental problem in resource allocation, and two interesting problems in the area of network design. We detail here these contributions.

1.1.1 *The Santa Claus problem*

First, we study the *Santa Claus problem*, also known as *MaxMin Fair Allocation*. In this problem, we have to assign resources to players in order to make the least happy player as happy as possible. Formally, we have a set P of m players, a set R of n resources, and each player i is equipped with an arbitrary valuation function $f_i : S \subseteq R \mapsto \mathbb{R}_+$. We have to find a partition $(S_1, \ldots, S_i, \ldots, S_m)$ of the resource set such that

$$\min_{i \in P} f_i(S_i)$$

is maximized (note that a resource can be assigned to at most one player).

The most classic version of this problem is when each f_i is an arbitrary linear valuation function. We can rephrase this case as the setting where player i assigns value v_{ij} to resource j and the goal is to find an assignment of resources to players $\sigma : R \mapsto P$ such that

$$\min_{i \in P} \sum_{j : \sigma(j) = i} v_{ij}$$

is maximized. Proving if a constant factor approximation can be obtained efficiently or not is a major open problem in the area of approximation algorithms [13, 78, 85]. In fact, there is a huge gap in our understanding of this fundamental problem. The state-of-the-art gives only an $O(n^{\varepsilon})$-approximation in time $n^{O(1/\varepsilon)}$ for any fixed $\varepsilon > 0$, and an $O(\text{polylog}(n))$-approximation in quasi-polynomial time[2] [21]. On the other hand, it is only known that getting a $(2 - \varepsilon)$-approximation (for any fixed $\varepsilon > 0$) is NP-hard [17, 68]. The mentioned algorithm is obtained by reducing to a problem similar to the *MaxMin Arborescences* problem, which is defined as follows: Given a directed graph $G = (V, E)$ with sources and sinks, our goal is to find vertex disjoint arborescences rooted in the sources such that at each non-sink vertex of an arborescence the out-degree is at least k, where k is to be maximized. This problem can be seen as a special case of the Santa Claus problem (see the beginning of Part i for more details).

This special case is of particular interest, as it seems to capture most of the difficulty of the Santa Claus problem with linear valuation functions. Indeed, an $O(n^{\varepsilon})$-approximation in time $n^{O(1/\varepsilon)}$ for any fixed $\varepsilon > 0$, and an

2 Quasi-polynomial refers here to a running time of $n^{\text{poly}(\log n)}$, which is only slightly worse than polynomial time.

$O(\text{polylog}(n))$-approximation in time $n^{O(\log n)}$ were first obtained only for the MaxMin Arborescence problem before being generalized to the general linear valuations case [15]. Our main result on this problem is to obtain an exponential improvement over previous works, an $O(\text{poly}(\log \log n))$-approximation in time $n^{O(\log n)}$ for the MaxMin Arborescence problem. We also show that getting a better than $(\sqrt{e/(e-1)} - \varepsilon)$-approximation for the MaxMin Arborescence problem is NP-hard for any fixed $\varepsilon > 0$.

Lastly, we turn our attention to the Santa Claus problem with valuation functions $f_1, f_2, \ldots, f_m$ that are submodular and not linear. This is of interest, as linear functions fail to capture properties that are often crucial. For instance, submodular function capture the *diminishing returns* property in economics, which is particularly relevant in the context of the Santa Claus problem. In the linear case, a popular line of research (started by Bansal and Srividenko [14]) has considered the *restricted assignment case*. This is the special case where each player i is given a set of desired resources Γ_i and the individual valuation functions are defined as $f_i(S) = f(S \cap \Gamma_i)$ for a global linear function f. This can also be interpreted as maximizing $\min_i f(S_i)$ with additional assignment restrictions, i.e., resources can only be assigned to certain players. In that case, it has been known for a long time how to obtain an $O(1)$-approximation in polynomial time [7, 39]. Our second result in Part i makes comparable progress for the submodular variant: If f is a monotone submodular function, we can in polynomial time compute an $O(\log \log(n))$-approximate solution.

1.1.2 *Simple Algorithms for Network Design*

In the second part of this thesis, we are interested in the design and analysis of *simple* algorithms for problems in the wide area of *network design*, which contains problems about designing cheap networks that are robust to edge failures. Being able to design such networks is a fundamental question, both in practice and in theory. The first problem we study is the *Steiner Forest problem*: we are given a weighted graph $G = (V, E)$ with a weight function $w : E \mapsto \mathbb{R}_+$ and a set of k pairs of vertices $\mathcal{P} := \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$. The goal is to compute the cheapest set of edges such that any two vertices belonging to the same pair are connected (note that some vertices might not appear in any pair). This is a generalization of two famous optimization problems: Minimum Spanning Tree, which is the special case where $\mathcal{P}$ contains all possible pairs, and Steiner Tree, where there is a special vertex s that belongs to all the pairs in $\mathcal{P}$. The Steiner Forest problem was at the heart of many new algorithmic techniques and several polynomial-time algorithms are known to guarantee a factor 2 approximation for this problem [47, 58]. However, these algorithms are fairly involved and are not necessarily the most intuitive algorithms. The simplest algorithm is arguably the *greedy* algorithm which can be informally defined as follows:

1. Order the pairs in $\mathcal{P}$ in an arbitrary order.
2. For $i = 1$ to k, connect $\{s_i, t_i\}$ with the shortest path in the current metric, then contract the metric along the chosen path.

Although simple an intuitive, greedy proved itself challenging to analyze. Awerbuch, Azar, and Bartal [8] showed with a beautiful argument that this algorithm guarantees a factor $O(\log^2(k))$ approximation for any ordering of $\mathcal{P}$. They also showed that this algorithm is no better than $\Omega(\log(k))$ (even if we select a specific ordering of $\mathcal{P}$), and they conjectured that greedy gives in fact an $O(\log(k))$-approximation. Whether this conjecture is true is particularly interesting, because it would also mean that greedy is the optimal algorithm in the online setting, where the pairs are given in an adversarial order. Unfortunately, this conjecture has not seen any progress since its formulation. Our first result in this part is to show that if we order the pairs in non-increasing order of distance in the graph G (i.e. such that $d_G(s_i, t_i) \geqslant d_G(s_j, t_j)$ for all $j > i$), then greedy guarantees an $O(\log(k) \cdot \log\log(k))$-approximation. While it does not imply an improved bound for the online setting, our proof has important consequences even in that case.

Then, we turn our attention to the *Matching Augmentation problem* (MAP), which is defined as follows. We are given a graph $G = (V, E)$, a weight function w such that $w(e) \in \{0, 1\}$ for all $e \in E$ and the set of edges that are given weight 0 forms a matching M. The goal is to compute the cheapest set $E' \subseteq E$ of edges such that $G' = (V, E')$ is 2-edge connected (note that it is forbidden to take two copies of the same edge $e \in E$). The name "matching augmentation" stems from the fact that the matching M can be included in the solution for free, and the challenge only remains in how to "augment" this matching to make it 2-edge connected. This problem has recently received significant attention as an important step towards better approximation algorithms for finding cheap 2-edge connected subgraphs. A 2-approximation can be obtained for this problem via many standard techniques, and the challenge is to overcome this barrier to obtain better-than-2 approximations. Previous works on this problem culminated in a $\frac{13}{8}$-approximation algorithm [45]. However, these algorithms and their analysis are very involved (the mentioned result is proven in a paper of 60 pages) and do not compare against the problem's well-known LP relaxation called the cut LP. Our second result in this part is a simple algorithm that, guided by an optimal solution to the cut LP, first selects a depth-first search (DFS) tree and then finds a solution to MAP by computing an optimum augmentation of this tree. Using properties of extreme point solutions, we show that our algorithm always returns (in polynomial time) a better-than-2 approximation when compared to the cut LP. We thereby also obtain an improved upper bound on the integrality gap of this natural relaxation. The algorithm is simple enough to be described in 3 basic steps, and the analysis takes roughly 5 pages.

1.2 HOW TO READ THIS THESIS

This thesis is split into two main parts, each part presenting a detailed motivation of the problems studied (including a fairly complete literature review), followed by the proof of our results. Part i contains our results on the Santa Claus problem, while Part ii contains our results on the Steiner Forest problem and the Matching Augmentation problem. In Chapter 8, we conclude the thesis with some interesting open problems arising out of our work. For clarity and better readability, some proofs are deferred to the Appendix in Part iii. Before moving on to the main body of the thesis, we finish this chapter by Section 1.3, which contains an introduction to Linear Programming, a fundamental tool in all our results.

1.3 LINEAR PROGRAMMING

A major challenge on the way to our results is to compare the cost of the solution returned by an algorithm to the cost of the optimum solution, which is unknown. To make matters worse, the optimum solution is often very unstable to small changes in the instance and very difficult to reason about in general. Fortunately, *linear programming relaxations* can be used to cope with these issues. In this section, we introduce this powerful tool which we will use extensively in this thesis.

Linear programs play a crucial role in the design and analysis of approximation algorithms. We illustrate its role on the MAKESPAN problem we defined earlier. In the previous version of MAKESPAN we considered, all the machines were identical. But in general, it might be that the factory has several types of machines, which can perform different sets of jobs. To account for this, we define p_{ij} as the time that machine i needs to finish job j. For instance, if a machine cannot perform a specific job, one can model this by setting $p_{ij} = \infty$ for the corresponding i, j. This more general version is called makespan minimization on unrelated machines, and we will refer to this version as UNRELATED MAKESPAN. It is significantly more challenging than the version on identical machines. In particular, all the results we cited for the MAKESPAN problem do not hold anymore for the UNRELATED MAKESPAN problem. For instance, it is easy to check that the simple factor 2 approximation for MAKESPAN does not guarantee any finite approximation ratio for UNRELATED MAKESPAN.

A natural question arises: what is the best approximation ratio an efficient algorithm can guarantee for the UNRELATED MAKESPAN problem? To tackle this question, we will use a linear program, a concept that we define next.

Definition 1.1 (Linear Program (LP) [85]). *A linear program is formulated in terms of some number of* decision variables *that represent some sort of decision that needs to be made. The variables are constrained by a number of linear inequalities and equalities called* constraints. *Any assignment of real numbers to the variables such that all of the constraints are satisfied is called a* feasible solution. *A linear program is said to be* feasible *if it admits a feasible solution. In addition to the constraints, linear programs are defined by a linear function of the decision variables called the* objective function. *The linear program seeks*

to find a feasible solution that either maximizes or minimizes this objective function. Such a solution is called an optimal solution.

A breakthrough result by Khachiyan [60] shows that there is an efficient (i.e. polynomial time) algorithm that decides, for any given linear program, if there exists an optimal solution or not. Moreover, if an optimal solution exists, then this algorithm returns such a solution in polynomial time.

We are now ready to use this concept in the context of approximation algorithms. To this end, let us consider a slightly easier version of the UNRELATED MAKESPAN problem where instead of building the assignment we only want to decide if there is an assignment of value less than T for $T > 0$ that is given in input. This is the decision version of the problem where the algorithm simply has to answer **YES** or **NO**. While this may seem restrictive, we can combine an algorithm for the decision version with any standard binary search framework to find the smallest T such that there exists an assignment of value at most T (let us call this optimum value T^*). This is often called an *estimation* algorithm, in contrast to a *constructive* algorithm which also returns a schedule that achieves the claimed objective T^*.

However, even the decision version of UNRELATED MAKESPAN is NP-hard, hence we need to relax a little bit the problem. Fix an $\alpha \geqslant 1$, we say that a decision algorithm is an α-approximation to the decision version of the problem if:

1. For any $T \geqslant T^*$, the algorithm returns **YES**.
2. If $T < T^*/\alpha$, the algorithm returns **NO**.
3. If $T^*/\alpha \leqslant T < T^*$, the algorithm can output **YES** or **NO** indifferently.

We recall here that T is given as input to the algorithm. Combining an α-approximation to the decision problem with a binary search gives an α-approximation to the estimation problem, i.e. an algorithm that returns in polynomial time a value $\hat{T}$ such that $T^* \leqslant \hat{T} \leqslant \alpha T^*$. In the rest of this section, we focus on obtaining an α-approximation to the decision problem, with α as close to 1 as possible (note that α is always at least 1). A landmark result by Lenstra, Shmoys, and Tardos [68] shows that this is possible for $\alpha = 2$. The rest of the section is devoted to provide some brief introduction to this result. To this end, we formulate a linear program and our estimation algorithm will simply return **YES** if the linear program is feasible, and **NO** otherwise. To decide if the linear program is feasible we can use Khachiyan's algorithm mentioned above. Now, we are ready to formulate the decision variables and constraints of the linear program.

For any machine i, job j we define the decision variable x_{ij} that takes value 1 if the job j is assigned to machine i and 0 otherwise. A solution to our problem can now be described as the binary vector $x := (x_{ij})_{i \in M, j \in J}$. In order for the solution to be feasible for our problem, it needs to respect a few constraints, that we can enforce using linear inequalities. First, any job j needs to be assigned to one machine to be processed therefore it must be that

$$\sum_{i \in M} x_{ij} = 1,$$

for all $j \in J$. Finally, it must be that the load on any machine is no more than T hence we can write

$$\sum_{j \in J} x_{ij} p_{ij} \leqslant T,$$

for any machine $i \in M$. Here we are simply designing a decision algorithm that outputs **YES** if and only if there is a feasible solution to the linear program, hence the objective function does not matter and we can simply define it to be the constant function equal to 0, which we omit for clarity. In summary, we have rephrased the decision version of the Unrelated Makespan problem into the following set of constraints:

$$\sum_{j \in J} x_{ij} p_{ij} \leqslant T \qquad \forall i \in M \tag{1.1}$$

$$\sum_{i \in M} x_{ij} = 1 \qquad \forall j \in J \tag{1.2}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in M, \forall j \in J\,. \tag{1.3}$$

Unfortunately, this is not a linear program, because of the last constraint $x_{ij} \in \{0, 1\}$ (which is not linear). The above program is often called an *integer linear program*, because the decision variables are forced to be integers. Solving integer linear programs in general is NP-hard, and this is not surprising since the above integer linear program is equivalent to the exact decision version of Unrelated Makespan, which is also NP-hard. Hence a common approach in algorithm design is to relax the integer constraints. The fact of relaxing the constraint naturally gave the name of *linear programming relaxation*. Here, we relax the integer constraints into $x_{ij} \geqslant 0$ to obtain the following linear program.

$$\sum_{j \in J} x_{ij} p_{ij} \leqslant T \qquad \forall i \in M \tag{1.4}$$

$$\sum_{i \in M} x_{ij} = 1 \qquad \forall j \in J \tag{1.5}$$

$$x_{ij} \geqslant 0 \qquad \forall i \in M, j \in J\,. \tag{1.6}$$

This is the linear programming relaxation of our decision problem. Because it is now a *linear* program, we can use Khachiyan's algorithm to decide in polynomial time if there exists a feasible solution. Let us denote by T_{LP} the smallest T such that the above LP is feasible. Then clearly $T_{\mathrm{LP}} \leqslant T^*$, because Constraint (1.6) is less restrictive than Constraint (1.3). Recall that our decision algorithm simply returns **YES** if and only if the LP is feasible. Therefore if $T \geqslant T^*$, then the algorithm always returns **YES**. To guarantee an α-approximation, it only remains to prove that

$$\frac{T^*}{T_{\mathrm{LP}}} \leqslant \alpha$$

always holds. This ratio is often referred to as the *integrality gap* the relaxation, and it measures how strong the relaxation is. The bigger $\frac{T^*}{T_{\mathrm{LP}}}$ is, the less useful the relaxation is.

Here, it turns out that $\frac{T^*}{T_{\text{LP}}}$ can be as big as m (the number of machines), hence the approximation ratio α of our algorithm is at least m, far from our goal of a 2-approximation. Consider the following example. There are m machines, and a unique job $\{1\}$ that has load m on all the machines (i.e. $p_{i1} = m$ for any machine i). Then the linear program is feasible for $T = 1$, by setting $x_{i1} = 1/m$ for all machines i. Clearly the job is assigned once in total, while every machine receives only a $1/m$ fraction of the job, for a total load of 1. Hence $T_{LP} \leqslant 1$. However, $T^* \geqslant m$ because no matter where we assign the job, it takes space at least m.

To cope with this, a useful idea is to introduce more restrictive constraints to the linear program. While doing this, we must ensure that an integral solution also satisfies those constraints, as otherwise it would not be a relaxation anymore. In order to find a suitable candidate constraint, we must understand intuitively what went wrong in the previous relaxation. Here, what happened is that the linear programming relaxation is allowed to spread a huge job on many machines, while the integral solution has to assign it *entirely* on a single machine. But clearly, if the objective is T, then any integral solution cannot assign job j on machine j if $p_{ij} > T$. This additional insight can be incorporated into the relaxation in the following sense. For any job j, we denote by $B_T(j)$ the set of machines such that $p_{ij} > T$. We then obtain the following new relaxation.

$$\sum_{j \in J} x_{ij} p_{ij} \leqslant T \qquad \forall i \in M \tag{1.7}$$

$$\sum_{i \in M \setminus B_T(j)} x_{ij} = 1 \qquad \forall j \in J \tag{1.8}$$

$$x_{ij} \geqslant 0 \qquad \forall i \in M, j \in J\,. \tag{1.9}$$

Constraint (1.5) was simply replaced by Constraint (1.8), in which we allow the LP to assign job j to machine i only if $p_{ij} \leqslant T$. Note that the previous example is not feasible for $T = 1$ on this new relaxation. It turns out that this seemingly benign modification of the LP now guarantees an integrality gap of at most 2. This was showed in [68]. We do not give here the proof here as it is beyond the scope of this introduction. To obtain such a result, [68] used a procedure that transforms the feasible solution x into an integral solution $\hat{x}$ (i.e. a solution such that $\hat{x}_{ij} \in \{0, 1\}$). Such a procedure is often referred to as a *rounding scheme*, a crucial concept in many results in approximation algorithms. Indeed, after solving the LP, we obtain a feasible solution x where $x_{ij} \in [0, 1]$ and we would like to show that there exists an integral solution that is not much worse than x. The most intuitive way to proceed is to transform x into an integral vector $\hat{x}$ where $\hat{x}_{ij} \in \{0, 1\}$. Along the way, if we manage to show that

$$\sum_{j \in J} \hat{x}_{ij} p_{ij} \leqslant \alpha \cdot \left(\sum_{j \in J} x_{ij} p_{ij} \right) \tag{1.10}$$

for all $i \in M$, then we will have shown that the integrality gap of the relaxation is at most α, and that our algorithm is an α-approximation. This is precisely what is shown in [68] for $\alpha = 2$, hence obtaining a 2-approximation

for our problem. Finding the good rounding scheme is however non-trivial, as some intuitive rules do not work. For instance, one might be tempted to simply assign each job j to the machine i such that x_{ij} is maximum. This intuitively follows the decisions made by the LP, but unfortunately does not guarantee a factor 2. Another intuitive rounding is simply to assign randomly each job j randomly to machine i with probability x_{ij}, but again this does not guarantee a factor 2. In fact, the rounding scheme of [68] is non-trivial and crucially exploits an additional property of the feasible solution x.

Indeed, [68] assumes that the feasible point x that we need to round is an *extreme point* solution to the LP. A feasible solution x is said to be an extreme point if it cannot be expressed as a convex combination of other feasible points. It turns out that if the LP has a bounded feasible region (which is the case in many relevant settings in approximation algorithms), then there always exists an extreme point that also achieves the optimum value, and an optimum extreme point can be found in polynomial time. These extreme points have some additional properties that are often useful and that we do not develop here. For intuition in our setting, the reader might just think that the extreme point x that we obtain has only $n+m$ coordinates that are non-zero, i.e. the solution is very sparse (note that in total there are $n \cdot m$ coordinates). Using the special structure of extreme points, [68] design a clever rounding scheme that satisfies Equation (1.10) with $\alpha = 2$. Interestingly, this rounding scheme runs in polynomial time, which renders the whole proof constructive and although we were focusing only on estimating the optimum value, we get a constructive algorithm that runs in polynomial time.

While the techniques here are specific to the Unrelated Makespan problem, the general structure of the example illustrates a very successful paradigm in the area approximation algorithms: (1) write a relaxation of the problem, then (2) devise a rounding scheme of the fractional solution. Both steps are crucial and challenging: Step (1) might necessitate to introduce stronger constraints, and carrying out Step (2) successfully involves designing a new algorithm. There is no magic recipe that works for all problems, and overcoming each step is often an interesting mathematical problem on its own.

1.3.1 *The use of LPs in this thesis*

In all the results presented in this thesis, we will use a linear programming relaxation of the problem considered. Our results on the Santa Claus problem and the Matching Augmentation problem essentially fit into the classic framework described above: first write a relaxation of the problem, then round a fractional solution into an integral one. Interestingly, for the Matching Augmentation problem we crucially need that the LP solution has a sparse support, which is satisfied in general by *extreme* points of the linear program. Lastly, for our result on the Steiner Forest problem, the use of linear programming techniques is less obvious, but still important. Indeed, to compare the cost of the greedy solution to the optimum solution, we need to lower bound the cost of the latter. In order to achieve this, we rely on the weak duality theorem for linear programs, which states that the value of any feasible solution

to the dual linear program is a lower bound on the objective of the primal linear program. In our setting, the primal will be a relaxation of the Steiner Forest problem, hence a feasible dual solution gives us a lower bound on the value of any feasible solution to the Steiner Forest problem. This last example is interesting, as it shows that even if the algorithm itself does not use the LP relaxation, this relaxation is still useful in the analysis of the algorithm.

Part I

THE SANTA CLAUS PROBLEM

2 INTRODUCTION

In this part, we focus on the *Santa Claus* problem, a central problem in scheduling and algorithmic fairness. In its full generality, we have a set P of m players, a set R of n resources, and each player i is equipped with an arbitrary valuation function $f_i : S \subseteq R \mapsto \mathbb{R}_+$. We have to find a partition $(S_1, \ldots, S_i, \ldots, S_m)$ of the resource set such that

$$\min_{i \in P} f_i(S_i)$$

is maximized (note that a resource can be assigned to at most one player). This problem is often referred to as *MaxMin Fair Allocation* in the sense that the objective is to find the fairest solution, unlike other objectives like total welfare which maximizes the average happiness. Unfortunately, without any restrictions on the utility functions the Santa Claus problem becomes hopelessly difficult. Consider the following reduction from set packing. There are sets of resources $\{S_1, \ldots, S_k\}$ and all utility functions are equal and defined by $f_i(S) = 1$ if $S_j \subseteq S$ for some j and $f_i(S) = 0$ otherwise. Deciding whether there are m disjoint sets in $S_1, \ldots, S_k$ (a classical NP-hard problem, see [59]) is equivalent to deciding whether the optimum of the Santa Claus problem is non-zero. In particular, obtaining any bounded approximation ratio for Santa Claus in this case is NP-hard. This motivates natural restrictions on the valuation functions. This part gives new results for two special cases of the Santa Claus problem that we call *MaxMin Arborescences*, and the *Restricted Submodular Santa Claus*.

2.1 MAXMIN ARBORESCENCES

The most natural restriction is to assume that all the f_i are linear valuation functions. We can rephrase this as follows: each resource j has unrelated values v_{ij} for each of the player i. The goal is to assign each resource j to a player $\sigma(j)$ such that we maximize the utility of the least happy player, that is,

$$\min_{i \in P} \sum_{j : \sigma(j) = i} v_{ij} \ .$$

The dual of the problem, where one has to minimize the maximum instead of maximizing the minimum is the problem of makespan minimization on unrelated parallel machines (this is exactly the UNRELATED MAKESPAN problem from the introduction of this thesis). Both variants form notoriously difficult open problems in approximation algorithms [13, 78, 85] and there is a common believe that the Santa Claus problem admits a constant approximation if and only if makespan minimization on unrelated machines admits a better-than-2 approximation [13]. Although formally no such reduction

is known, techniques often seem to transfer from one problem to the other, which gives an additional motivation to study this problem.

Bateni, Charikar, and Guruswami [15] identified as a central special case of the Santa Claus problem the restriction that for all values we have $v_{ij} \in \{0, 1, \infty\}$ and for each resource there is at most one player with $v_{ij} = \infty$ and for each player there is at most one resource with $v_{ij} = \infty$. This special case can be rephrased in a graph problem as follows. For each resource j such that there is no player i with $v_{ij} = \infty$, we create a *sink* vertex. For each player i such that there is no resource j with $v_{ij} = \infty$, we create a *source* vertex. For any pair of a player i and resource j such that $v_{ij} = \infty$, we create a vertex in our graph. Note that in this construction, source vertices correspond to a single player in the original instance, sink vertices correspond to a single resource, and other vertices correspond to a single player and a single resource. Next we create directed edges in our graph as follows. For every vertex v that corresponds to a player i in the original instance, we add a directed edge from v to all other vertices in the graph that corresponds to a resource j with $v_{ij} = 1$. The Santa Claus problem in this special case can now be rephrased as follows. We are given a directed graph $G = (V, E)$, a set of sources $S \subseteq V$, and a set of sinks $T \subseteq V$. Our goal is to compute a set of vertex disjoint arborescences rooted in each of the sources S (one for each source). The leaves of these arborescences must be at the sinks (i.e. only vertices in T can be included as a leaf of an arborescence). For all inner vertices we must have an out-degree of k, where k is to be maximized. This problem is called the *MaxMin Arborescences* problem.

Bateni et al. gave a $\max\{\text{poly}(\log n), n^{\varepsilon}\}$-approximation in time $n^{O(1/\varepsilon)}$, which is still the state-of-the-art for this problem. In particular, they obtain a quasi-polynomial time polylogarithmic approximation by setting $\varepsilon = \log\log n / \log n$. In a highly non-trivial way, the same approach was then generalized by Chakrabarty, Chuzhoy, and Khanna [21] to obtain the same result also for the Santa Claus problem. The only hardness known for the Santa Claus problem is that there is no better-than-2 approximation (see [17, 68]). This still leaves a large gap in the understanding of both problems. In particular, the two most pressing questions are whether the polylogarithmic guarantee can also be achieved in polynomial time and whether a sublogarithmic approximation guarantee can be achieved.

While much progress on sublogarithmic approximations has been made on other special cases of the Santa Claus problem, most notable the restricted assignment case (on which we will elaborate later), these results are all based on a popular relaxation of the problem named the *configuration LP*. However, it is known that for the general Santa Claus problem the configuration LP has an polynomial integrality gap and hence these methods seem very unlikely to generalize. Similarly, the MaxMin Arborescences problem is a case where the integrality gap of the configuration LP is already high [15] and as such the algorithmic techniques need to be rethought. Given this and the fact that previous progress on the problem was quickly extended to the general Santa Claus problem, the MaxMin Arborescences problem seems to be an important piece of the puzzle towards the goal of understanding the approximability of the Santa Claus problem.

The rephrasing of this special case of the Santa Claus problem as an arborescence problem also uncovers an intriguing connexion to the *Directed Steiner Tree* problem. In this problem we are given an edge-weighted directed graph with a source and a set of sinks. The goal is to find an arborescence of minimal weight, which is rooted at the source and spans all sinks. It is quite remarkable that the state-of-the-art for this problem is very similar to ours: there is a n^{ε}-approximation algorithm in polynomial time for every fixed $\varepsilon > 0$ and a polylogarithmic approximation algorithm in quasi-polynomial time [22]. Unlike our problem, it was shown that no sublogarithmic (in fact, no $\log^{2-\varepsilon} n$) approximation exists [52].

2.1.1 *Our results for MaxMin Arborescences*

Our main result for MaxMin Arborescences is the following.

Theorem 2.1. *There exists an $O((\log\log n)^{20})$-approximation running in time $n^{O(\log n)}$ for the MaxMin Arborescences problem.*

We remark that we did not try to optimize the exponent of 20 in the approximation guarantee. The main challenge was to get a sublogarithmic guarantee (i.e. an $o(\log n)$-approximation) and approach a constant factor. Indeed, in light of the $\Omega(\log^{2-\varepsilon}(n))$-hardness for directed Steiner Tree [52], it was not even clear if a sublogarithmic guarantee was possible for the MaxMin Arborescences problem. In fact, if one takes a closer look at the literature on directed Steiner Tree problems, we believe our result to be rather surprising. We elaborate now on this. Related to the directed Steiner Tree problem is also the (undirected) group Steiner Tree problem, which can be shown to be a special case. In this problem, we are given an undirected weighted graph and a list of groups that are subsets of vertices. The goal is to compute the cheapest set of edges that is connected and contains at least one vertex from each group. Here, there are more subtile connections to our problem: taking a closer look at the literature one can notice that the challenging instances in the group Steiner Tree problem have a similar structure to the challenging ones for the MaxMin Arborescence problem. More precisely they are layered graphs with $O(\log n)$ layers. Halperin et al. [51] show that the integrality gap of a natural LP relaxation of group Steiner Tree could be amplified from $\Theta(\log(n))$ on $O(1)$-layered instances to $\tilde{\Theta}(\log^2(n))$ on $\Omega(\log(n))$-layered graphs. This construction was later transformed to the hardness result by Halperin and Krauthgamer [52].

Before our result, it was quite plausible that such an amplification technique could also apply in the context of the MaxMin Arborescences problem. This would have shown how to amplify a $\Omega(1)$ gap on $O(1)$-layered instances to a $\tilde{\Omega}(\log(n))$ gap on $\Theta(\log(n))$-layered instances. In Appendix A.1, we adapt the construction of [51] to our setting. At first sight, the gap indeed seems to amplify and our construction shows that the previous rounding algorithms of [15, 21] cannot hope to get better than a $\tilde{\Omega}(\log(n))$-approximation. Fortunately, we notice that a simple (but crucial) pruning trick seems to resolve the issue for this specific construction. We note that the group Steiner Tree problem has a rich history and it would be very interesting to see if more techniques could be transferred to the MaxMin Arborescences problem.

Finally, we also show as a side result a hardness of approximation for the MaxMin Arborescences problem.

Theorem 2.2. *For any $\varepsilon > 0$, there is no $(\sqrt{e/(e-1)} - \varepsilon)$-approximation algorithm for the MaxMin Arborescences problem, unless* $\mathrm{P} = \mathrm{NP}$.

It is worthwhile to mention that our hardness result is obtained by a reduction from the max-k-cover problem. Interestingly, the hardness of directed Steiner Tree problems in [52] is obtained by a reduction from set cover combined with the amplification technique mentioned above.

2.2 THE RESTRICTED SUBMODULAR SANTA CLAUS

We introduce here our second result on the Santa Claus problem. As explained in the beginning in this chapter, assuming no restrictions on the valuation functions f_i makes the problem hopelessly difficult: no finite approximation is possible. However, we notice that most of the work on the Santa Claus problem has focused on the case where all valuation functions f_i are linear, and there are some interesting special cases of linear functions where a constant factor is already known. A prominent such example is the so-called *restricted assignment case*.

In the restricted assignment case, the utility functions are defined by one linear function f and a set of resources Γ_i for each player i. Intuitively, player i is interested in the resources Γ_i, whereas the other resources are worthless for him. The individual utility functions are then implicitly defined by $f_i(S) = f(S \cap \Gamma_i)$.

In a seminal work, Bansal and Srividenko [14] provide an $\tilde{O}(\log\log(m))$[1]-approximation algorithm for this case using randomized rounding of the configuration LP. This was improved by Feige [39] to an $O(1)$-approximation. Further progress on the constant or the running time was made since then, see e.g. [5, 6, 24, 25, 35, 53, 76]. These works culminate in a $(4+\varepsilon)$-approximation in polynomial time and an 3.534-approximation that only gives an estimation of the optimum value in polynomial time (but does not return the corresponding assignment in polynomial time).

It is very natural to ask what is possible beyond the case of linear valuation functions. Two naturally arising properties of utility functions are monotonicity and submodularity, see for example the related submodular welfare problem [67, 83] where the goal is to maximize $\sum_i f_i(S_i)$. A function f is monotone, if $f(S) \leqslant f(T)$ for all $S \subseteq T$. It is submodular, if $f(S \cup \{a\}) - f(S) \geqslant f(T \cup \{a\}) - f(T)$ for all $S \subseteq T$ and $a \notin T$. The latter is also known as the *diminishing returns* property in economics. A standard assumption on monotone submodular functions (used throughout this thesis) is that the value on the empty set is zero, i.e., $f(\emptyset) = 0$. Goemans, Harvey, Iwata, and Mirrokni [46] first considered the Santa Claus problem with arbitrary monotone submodular utility functions as an application of their fundamental result on submodular functions. Together with the algorithm of [21] it implies an $O(n^{1/2+\varepsilon})$-approximation in time $n^{O(1/\varepsilon)}$. In the case

1 To be precise, the exact approximation ratio is $O(\log\log(m)/\log\log\log(m))$.

that the valuation functions are all equal, that is, $f_i(S) = f(S)$ for a monotone submodular function f, Krause, Rajagopal, Gupta, and Guestrin gave a constant approximation [65]. We also refer to their work for an application of this problem in sensor placement.

Our second result investigates the approximability of the Santa Claus with monotone submodular functions in the restricted assignment case. That is, all utility functions are defined by $f_i(S) = f(S \cap \Gamma_i)$, where f is a monotone submodular function and Γ_i is a subset of resources for each players i. Before our work, the state-of-the-art for this problem was the $O(n^{1/2+\varepsilon})$-approximation algorithm mentioned above, since none of the previous results for the restricted assignment case with a linear utility function apply when the utility function becomes monotone submodular. In the rest of the thesis, we refer to this problem as the *Restricted Submodular Santa Claus* problem.

2.2.1 *Our results for the Restricted Submodular Santa Claus*

In Chapter 4, we show the following theorem, which essentially generalizes the result of Bansal and Srividenko [14] to the case of submodular functions.

Theorem 2.3. *There is a randomized polynomial time $O(\log\log(n))$-approximation algorithm for the Restricted Submodular Santa Claus problem.*

2.3 PROBABILISTIC LEMMAS

We finish the introduction of this part by stating here two crucial probabilistic lemmas, that we will use extensively for both results.

Lemma 2.4 (Chernoff's bound, see [31])**.** *Let $X_1, \ldots, X_n$ be independent random variables that take value in $[0, a]$ for some fixed a. Let $S_n = \sum_{i=1}^{n} X_i$. Then we have, for any $\delta \geqslant 0$,*

$$\mathbb{P}[S_n \geqslant (1+\delta)\mathbb{E}[S_n]] \leqslant \exp\left(-\frac{\delta^2 \mathbb{E}[S_n]}{(2+\delta)a}\right).$$

Lemma 2.5 (Constructive Lovász Local Lemma, see [70])**.** *Let $\mathcal{X}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. For any $A \in \mathcal{A}$, let $\Gamma_{\mathcal{A}}(A)$ be the set of events $B \in \mathcal{A}$ such that A and B depend on at least one common variable. If there exists an assignment of reals $x : \mathcal{A} \mapsto (0,1)$ such that for all $A \in \mathcal{A}$,*

$$\mathbb{P}[A] \leqslant x(A) \cdot \prod_{B \in \Gamma_{\mathcal{A}}(A)} (1 - x(B)),$$

then there exists an assignment of values to the variables $\mathcal{X}$ not triggering any of the events in $\mathcal{A}$. Moreover, there exists a randomized algorithm that finds such an assignment in expected time

$$|\mathcal{X}| \cdot \sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)}.$$

3 MAXMIN ARBORESCENCES

This chapter is devoted to the proof of Theorem 2.1 and Theorem 2.2. For Theorem 2.1, we start by a brief overview of the main ideas and intuition behind our result before giving the formal proof structure in Section 3.2, and then we provide the three crucial steps of the proof in Section 3.3, Section 3.4, and Section 3.5. We close this chapter by a short proof of Theorem 2.2 in Section 3.6. All the results in this chapter are based on a joint work with Lars Rohwedder which has been accepted for publication at the *Annual ACM Symposium on Theory of Computing* (STOC '23). It is currently available on ArXiv [12].

3.1 OUR TECHNIQUES AND INTUITION

A crucial idea that goes back to previous work [15, 21] is to allow congestion in the solution. Generally, a vertex can only have one incoming edge in the solution, but we relax this constraint. We call the maximum number of times a vertex is used the congestion. The algorithms in [15, 21] employ randomized rounding to obtain a solution with polylogarithmic congestion. The congestion can then be translated into an approximation rate by relatively straight-forward arguments. This polylogarithmic congestion comes from a Chernoff bound that yields an inversely polynomial probability, which is then applied to all vertices with a union bound.

A new ingredient of our algorithm is the notion of local congestion. Roughly speaking, we first compute a solution, which still has polylogarithmic congestion, but when considering only a local part of the solution (say, vertices within a distance of $\ell = O(\log \log n)$ in the arborescences), then this local part needs to have much smaller congestion, i.e., $\mathrm{poly}(\log \log n)$. In other words, if a vertex is used multiple times in the solution, then the occurrences should be far apart in the arborescences.

First, let us describe why it is plausible to be able to obtain such a guarantee. It is already known that by randomized rounding a polylogarithmic congestion can be achieved. The local congestion on the other hand is by definition a very local constraint and hence Lovász Local Lemma (LLL) is natural to employ. This is indeed our approach, although the details are challenging.

Next, we will explain how to arrive at a sublogarithmic congestion. The approach that we call top-to-bottom pruning is very blunt: slighly oversimplifying, we take a given solution (with $\mathrm{poly}(\log n)$ congestion) and start at the sources of the arborescences. We throw away randomly a constant fraction (say, half) of their children. Then we move to the other children and recurse. Clearly, this decreases the approximation rate only insignificantly. However, in expectation the congestion at each vertex decreases drastically. If for example a vertex is at distance d from the source of an arborescence, then the probability of it not being removed is only $1/2^d$. There is, however, a caveat here: suppose that the same vertex occurs in a (relaxed) arborescence

many times and all occurrences are very close to each other. Then there is a high positive correlation between the vertices not being removed, which forms a serious problem. Indeed, this is where the local congestion comes in. It essentially bounds the dependence of occurrences surviving. Again, this proof makes use of LLL, because it seems infeasible to try and make the probability of a vertex's congestion staying above $\mathrm{poly}(\log\log n)$ small enough to apply a union bound.

Since both parts require LLL, it is crucial to bound the dependencies. However, if k is large (say $\Omega(n)$), then even a local part of the arborescence contains many vertices. It then seems unlikely to be able to guarantee locally low congestion for every vertex. Roughly speaking, we will only guarantee the property for a large fraction of the vertices, so that we can make the probability inversely polynomial in k. All other vertices need to be removed from the solution and this is generally very dangerous: even if we remove only a small fraction of vertices, this can lead to other vertex removals becoming necessary, because they now have a low out-degree. If we are not careful, this can accelerate and corrupt the whole solution. We call this the bottom-to-top pruning and we formalize a condition, under which the damage to the solution can be controlled. This condition is then applied in both parts.

This brings us to a discussion on our two pruning techniques. Intuitively, both approaches have complementary merits to each other. Bottom-to-top pruning makes it easy to maintain a low maximum congestion, but difficult to keep a good number of children for every vertex in the arborescence. On the contrary, top-to-bottom pruning makes it very easy to maintain a good number of children, but difficult to keep the maximum congestion under control. Our proof can be seen as a careful combination of those two techniques using LLL. The use of pruning makes the proof fairly involved and one might wonder if this could not be avoided. In particular, it is not clear if the analysis of the previous randomized rounding algorithm (see [15, 21]) is tight or not. However, we argue that an $\Omega(\log n/\log\log n)$ factor seems unavoidable in previous works, and that it is not easily fixable. We now elaborate on this: In previous approaches as in ours, a crucial part of the algorithm is to solve the max-min degree arborescence problem on layered instances (that is, the sources are located in the first layer, and edges can exist only between vertices of consecutive layers). Previous works [15, 21] then solve these instances by rounding an LP relaxation of the problem (we use the same LP relaxation but with a different rounding). An intuitive randomized rounding that appears in previous works is roughly as follows. Assume that the LP says there exists a solution of value k. Then the source samples k children with probability equal to the LP values. These selected children then select k children each, again equal to the LP values (more precisely, values that correspond to conditioning on the previous selections). In that manner, we make progress layer by layer until reaching the last layer. This guarantees a maximum congestion that is at most polylogarithmic in the number of vertices, hence the polylogarithmic approximation ratio. At this point, one might be tempted to argue that very few vertices in our solution will have congestion $\Omega(\log n)$ and that they are not a serious problem. Unfortunately, we show in Appendix A.1 an instance in which the above rounding results in a solution in which *all* the sinks selected in the arborescence have an expected

congestion of $\Omega(\log n/\log\log n)$. While this may seem counter-intuitive, recall that here we are implicitly conditioning by the fact of being selected in the solution. It is non-trivial to recover from this issue: For instance deleting—in a bottom-to-top fashion—the vertices with high congestion will basically remove almost all the sinks which will corrupt the whole solution.

Lastly, there is one important issue that we have not mentioned so far: in a similar way that each vertex loses some fraction of its children in the rounding (compared to the LP relaxation), we would lose some fraction of the sources. This happens also in previous works [15, 21], who then only compute a solution for a $1/\mathrm{poly}(\log n)$ fraction of the sources and repeat it for $\mathrm{poly}(\log n)$ times to cover all sources. This again introduces a congestion of $\mathrm{poly}(\log n)$, which seems difficult to avoid with the randomized rounding approach. First, we only design the randomized rounding to approximate single source instances so that we do not have to cope with this. Then we present a black-box reduction from many sources to one source, which uses a non-trivial machinery that is very different from the randomized rounding approach. Namely, this is by a local search framework, which has already seen a big application in the restricted assignment case of the Santa Claus problem, see related work. Usually the local search is analyzed against the configuration LP, which is not applicable here, so our way of applying it is quite different to previous works.

3.2 FORMAL PROOF STRUCTURE

We start with some simplifying assumptions. Let k be the optimum of the given instance, which we guess via a binary search framework. If $k \leqslant \mathrm{poly}(\log\log n)$, then obtaining a $(1/k)$-approximation is sufficient. This is easy to achieve: it is enough to find $|S|$ vertex-disjoint paths that connect each source to a sink, which can be done by a standard max-flow algorithm. Throughout the paper we assume without loss of generality that k is at least $\mathrm{poly}(\log\log n)$ with sufficiently large constants. Similarly, we assume that n is larger than a sufficiently large constant. Whenever we divide k by some term, for example, $k/\log\log n$, we would normally have to write $\lfloor k/\log\log n\rfloor$. All the divisors considered can be assumed to be much smaller than k, hence any loss due to rounding is only a small constant factor and therefore insignificant. We assume for simplicity that the divisors are always integral and the divisions have no remainder.

We describe solutions using a set of paths instead of directly as arborescences. This abstraction will become useful in the linear programming relaxation, but also in other parts throughout the paper. With a given arborescence, we associate the set of all paths from a source to some vertex in it. Let p be a path from a source to some vertex v. Formally, p is a tuple of vertices $(v_0, v_1, \ldots, v_\ell)$ where v_0 is a source, v_1 a vertex with an edge from v_0, and more generally v_i is a vertex having an edge from $v_{\ell-1}$. We do not explicitly forbid circles, but the properties of a solution will imply that only simple paths can be used. Finally we say that p is a *closed* path if its last vertex is a sink and an *open* path otherwise.

We will denote by $p \circ v$ the path p, to which we add the extra vertex v at the end. We write $|p|$ for the length of path p. A path q is said to the a *descendant* of p if it contains p as a prefix. In that case we call p an *ancestor* of q. We also say that $p \circ v$ is a *child* of p (who is then a *parent* of $p \circ v$). Within a given set of paths P, we denote the sets of children, parent, ancestors, and descendants of a path p by $C_P(p)$, $P_P(p)$, $A_P(p)$, and $D_P(p)$ respectively. By $D_P(p, \ell)$ we denote the descendants $q \in D_P(p)$ with $|q| = |p| + \ell$. Furthermore, we write $D_P(p, \leqslant \ell) = \bigcup_{\ell' \leqslant \ell} D_P(p, \ell')$. The set of paths that ends at a vertex v is denoted by $I_P(v)$. If the set of paths P is clear from the context, we may choose to omit the subscript for convenience.

The conditions for a set of paths Q to form a degree-k solution are the following:

1. $(s) \in Q$ for every source s,
2. $|I_Q(v)| \leqslant 1$ for every $v \in V$, and
3. $|C_Q(p)| = k$ for every path $p \in Q$ that is open.

We will also consider a relaxed version of (2), where we allow higher values than 1. Then we call maximum over all $|I_Q(v)|$ the congestion of the solution. The motivation for looking at solutions with (low) congestion is that we can remove any congestion by reducing k by the same factor.

Lemma 3.1. *Let Q be a degree-k solution with congestion K. Then in polynomial time we can compute a degree-k/K solution without congestion.*

Proof. Consider a bipartite multigraph $(U \cup U', F)$ on two copies U, U' of the vertices in V that appear in Q. The graph has an edge $(u, v) \in F$ for every path $p \in Q$ that ends in u and for which $p \circ v \in C(p)$. Then the degree of a vertex $u' \in U'$ is exactly the congestion of this vertex. Similarly, the degree of a vertex $u \in U$ is k times its congestion. We consider a fractional selection of edges $x_e = 1/K$ for each $e \in F$. Here, each vertex $u \in U$ has $\sum_{e \in \delta(u)} x_e \geqslant k/K$ and each vertex $u' \in U'$ has $\sum_{e \in \delta(u')} x_e \leqslant 1$, where $\delta(u)$ are the edges incident to u. As explained at the beginning of the section, we assume that K divides k and therefore K/k is integer. By integrality of the bipartite matching polytope there exists also an integral vector x' that satisfies these bounds. This corresponds to a degree-k/K solution without congestion. □

3.2.1 *Bounded depth solution*

The following proof follows exactly the arguments of a similar statement in [15]. We repeat it here for convenience.

Lemma 3.2. *Let Q be a degree-k solution. Then there exists a degree-$k/2$ solution $Q' \subseteq Q$ where $|p| \leqslant \log_2 n$ for every $p \in Q'$.*

Proof. We iteratively derive Q' from Q. For every $d = 1, 2, \ldots, n$ we consider the paths $p \in Q$ with $|p| = d$. For $d = 1$ we add all these paths to Q'. Then given the paths of length d in Q' we select for each of them the subset of $k/2$ children in Q, which has the least number of descendants (assuming for

simplicity that 2 divides k). For every d we will now bound the total number of descendants in Q of paths of this length, namely

$$n_d = \sum_{p \in Q' : |p| = d} |D_Q(p)| \,.$$

Notice that the descendants are counted in set Q and not Q'. Clearly, we have $n_1 \leqslant n$, since $|Q| \leqslant n$. Then since we remove the children with the largest number of descendants, we get $n_{i+1} \leqslant n_i/2$ for every i. Thus, $n_{\log_2 n} = 0$. □

3.2.2 *Local congestion and layered instances*

A crucial concept in our algorithm are solutions with locally low congestion. It will later be shown that such a solution suffices to derive a solution with (globally) low congestion.

Definition 3.3. *Let Q be a solution and $\ell \in \mathbb{N}$. We say that Q has an* ℓ-local congestion of L, *if for every $p \in Q \cup \{\emptyset\}$ and $v \in V$ we have*

$$|I_{D(p,\leqslant \ell)}(v)| \leqslant L \,.$$

For sake of clarity, let us note the special case of $p = \emptyset$ in the definition above. In this case, $D(p, \leqslant \ell)$ is simply the set of all paths in P with length at most ℓ (potentially starting at different sources). Throughout the paper we use values of the order $\ell = O(\log \log n)$ and $L = \operatorname{poly}(\log \log n)$. The usefulness of local congestion is captured by the following lemma, which we will prove in Section 3.3.

Lemma 3.4. *Let Q be a degree-k solution with ℓ-local congestion of L and global congestion K, where $\ell \geqslant \log_2 K$ and $K \geqslant \log_2 n$. Then we can compute in polynomial time a degree-$k/(8\ell)$ solution $Q' \subseteq Q$, which has global congestion at most*

$$O(\ell^7 L) \,.$$

It remains to show how to compute a solution with low local congestion. The abstraction of local congestion and the lemma on bounded depth allows us to reduce at a low expense to instances in layered graphs.

Definition 3.5. *A layered instance has layers $L_0 \dot\cup L_1 \dot\cup \ldots \dot\cup L_h = V$ such that L_0 consists of all sources and edges go only from one layer L_i to the next layer L_{i+1}.*

Lemma 3.6. *In polynomial time we can construct a layered instance with $h = \log_2 n$ such that if there exists a degree-k solution for the original instance, then there exists a degree-$k/2$ solution for the layered instance. Further, any degree-k' solution with ℓ-local congestion L and global congestion K in the layered instance can in polynomial time be transformed to a degree-k' solution for the original instance with ℓ-local congestion ℓL and global congestion $K \log_2 n$.*

Proof. Let L_0 be the set of sources. Then for each $i = 1, 2, \ldots, \log n$ let L_i be a copy of of all vertices V. We introduce an edge from $u \in L_i$ to $v \in L_{i+1}$ if (u, v) is an edge in the original instance. The new set of sinks is the union of all sink vertices in all copies.

Consider now a degree-k solution Q for the original instance. By Lemma 3.2 there exists a degree-$k/2$ solution Q' where each $p \in Q'$ has $|p| \leqslant \log_2 n$. For each such $p = (v_1, v_2, \ldots, v_t)$ we introduce a path $p' = (v'_1, v'_2, \ldots, v'_t)$ in the layered instance where v'_i is the copy of v_i in L_i.

Now let Q' be a degree-k' solution with ℓ-local congestion L and global congestion C in the layered instance. We transform Q' to a solution Q for the original instance by replacing each path $p' = (v'_1, v'_2, \ldots, v'_t)$ by a path $p = (v_1, v_2, \ldots, v_t)$, where v'_i is a copy of v_i. Since there are only $\log_2 n$ copies of each vertex, the global congestion increases by at most a factor of $\log_2 n$. For the local congestion consider a path $p \in Q$. This path was derived from a path $p' \in Q'$. Notice that any path $q' \in D(p', \leqslant \ell)$ ends in some vertex in $L_{|p|+1}, L_{|p|+2}, \ldots, L_{|p|+\ell}$. Thus, there are only ℓ copies of each vertex that q' can end in. Consequently, the ℓ-local congestion can increase at most by a factor of ℓ. □

Lemma 3.7. *Let $K = 2^{11} \log^3 n$, $\ell = 10 \log\log n$, and $L = 2^{10}\ell^2$. Given a layered instance with a single source and optimum k, we can in time $n^{O(\log n)}$ compute a degree-$k/(64\ell)$ solution with ℓ-local congestion at most L and global congestion at most K.*

This lemma is proven in Section 3.4. The lemmas above would allow us already to obtain our main result for instances with a single source. To generalize to multiple sources we present a black-box reduction on layered instances. This is proved in Section 3.5.

Lemma 3.8. *Suppose we have an α-approximation for the max-min degree bounded arborescence problem on layered instances with a single source running in time $n^{O(\log n)}$. Then there is also a 256α-approximation for layered graphs and an arbitrary number of sources running in time $n^{O(\log n)}$.*

3.2.3 *Connecting the dots*

We complete the proof of the main theorem (Theorem 2.1) as follows.

Theorem 2.1. First we prove the theorem on layered instances with a single source. Let k be the optimum of the given instance. Using Lemma 3.7 we can find a degree-$\Omega(k/\ell)$ solution with ℓ-local congestion L and global congestion K. Here $K = O(\log^3 n)$, $\ell = O(\log\log n)$ with $2^\ell \geqslant K$, and $L = O(\ell^2)$. Next, we apply Lemma 3.4 to turn this into a degree-$\Omega(k/\ell^2)$ solution with global congestion at most $O(\ell^7 L) = O(\ell^9)$. Using Lemma 3.1 we can convert this to a degree-$\Omega(k/\ell^{11})$ solution without congestion. We therefore have an $O(\ell^{11})$-approximation algorithm for a single source on layered graphs and Lemma 3.8 implies that we can extend this to an arbitrary number of sources.

We now turn our attention to instances that are not necessarily layered. Let again k be the optimum. Using Lemma 3.6 we construct a layered instance that is guaranteed to contain a degree-$k/2$ solution. Thus, with our algorithm

for layered instances we can obtain a degree-$\Omega(k/\ell^{11})$ solution for it. This solution has ℓ-local congestion at most 1 and global congestion at most 1. Using Lemma 3.6 we can construct a degree-$\Omega(k/\ell^{11})$ solution for the original (non-layered) instance with ℓ-local congestion at most ℓ and global congestion at most $\log n$. Using again Lemma 3.4 we obtain a degree-$\Omega(k/\ell^{12})$ solution with global congestion at most $O(\ell^8)$. Finally, applying Lemma 3.1 we obtain a degree-$\Omega(k/\ell^{20})$ solution without congestion. In particular, our approximation ratio is

$$O(\ell^{20}) = O((\log\log n)^{20}) .$$

Finally, the overall running time in clearly dominated by the use of Lemmas 3.7 and 3.8, which is $n^{O(\log n)}$. □

3.2.4 *Bottom-to-top pruning*

In this subsection we will describe a method of pruning that is used in the proofs of Lemmas 3.4 and 3.7. Before stating the result we need, we illustrate why this pruning technique can be tricky to use and what are its limitations.

The name "bottom-to-top pruning" refers to the following intuitive strategy. Suppose we are given a degree-k solution with some congestion. A natural approach is to remove paths that have high congestion at their endpoint in the hope to decrease the maximum congestion. However, this can be dangerous. Indeed, removing paths may remove children of some other paths and therefore forcing us to remove them as well. In general, even a very small amount of removals can lead to the whole solution getting corrupted, that is, ultimately sources may need to be removed as well. In fact, a simple calculation shows that if we allow an adversary to delete some fraction β of the paths of length d in our solution and that we try to maintain a solution of degree k/α, then the fraction of children of the sources that we might be forced to delete becomes roughly equal to $\beta e^{d/\alpha} \leqslant \beta e^{\log n/\alpha}$ (recall that at a constant loss we can assume that the depth of the solution is at most $\log n$). In particular if we aim at a sublogarithmic approximation ratio (i.e. $\alpha = o(\log n)$), the bottom-to-top pruning becomes very sensitive to a small number of deletions.

This pruning technique will be useful nonetheless, and the following lemma states a condition under which we can perform bottom-to-top pruning while keeping the damage under control.

Lemma 3.9. *Let Q be a degree-k solution. Let $R \subseteq Q$ be a set of paths that is supposed to be removed. Let $\ell \geqslant 2$ such that for every $p \in Q \setminus R$ we have that at most $k^\ell/(8\ell)^2$ many descendants $q \in D(p, \ell)$ with $q \in R$. Then we can compute in polynomial time a solution $Q' \subseteq Q \setminus R$ such that*

1. *Q' is a degree-$k/2\ell$ solution and*
2. *we have $(s) \in Q'$ for every source s, such that for any distance $\ell' \leqslant \ell$ there are at most $k^{\ell'}/(8\ell)$ many $q \in D((s), \ell')$ with $q \in R$.*

Proof. We assume that no path $p \in R$ has a descendant also in R. This is without loss of generality, since removing the former implies that the latter

will be removed, and omitting the latter from R still keeps the premise of the lemma valid. In particular, this assumption allows us to assert that also paths in R satisfy the bound on the number of descendants in R.

We prune the solution from longest paths to shortest paths: We remove a path if it is in R or if more than $(1 - 1/2\ell)k$ many of its children were removed. Then we prove a stronger variant of (2) inductively, namely, that any path of length $1, 1+\ell, 1+2\ell$, etc. satisfies the implication (or an ancestor of it is removed). Let p be a path with $|p| = 1+t\cdot\ell$ that satisfies the premise of (2), but is not necessarily a singleton. Further, assume that all paths of length $1 + (t + 1)\ell$ satisfy the implication of (2). Let $\ell' \leqslant \ell$. Each of the distance-ℓ' descendants of p has at most $(8\ell)^{-2}k^\ell$ many distance-ℓ descendants in R. Consequently, p has at most

$$k^{\ell'} \cdot (8\ell)^{-2}k^\ell$$

distance $(\ell + \ell')$-descendants in R. Thus, at most $(8\ell)^{-1}k^\ell$ many of p's distance-ℓ descendants have more than $(8\ell)^{-1}k^{\ell'}$ distance-ℓ' descendants belonging to R. Summing over all values of ℓ', we have that at most $1/8 \cdot k^\ell$ many distance-ℓ descendants of p do not satisfy the premise of (2). Next, let us show in a second induction that for every $\ell' \leqslant \ell$, of the distance-ℓ' descendants of p at most

$$\frac{1}{8}\left(1 + \frac{2}{\ell}\right)^{\ell-\ell'+1} k^{\ell'}$$

many are removed. For the base case we sum the distance-ℓ descendants that do not satisfy the premise of (2) and the descendants that are themselves in R, which together are at most

$$\frac{1}{8}k^\ell + \frac{1}{(8\ell)^2}k^\ell \leqslant \frac{1}{8}\left(1 + \frac{2}{\ell}\right)^{\ell-\ell+1} k^\ell\,.$$

Now assume that we removed at most $1/8 \cdot (1 + 2/\ell)^{\ell-\ell'}k^{\ell'+1}$ paths from the distance-$(\ell' + 1)$ descendants. For each distance-ℓ' descendants that we remove because of few remaining children, there are $(1 - 1/(2\ell))k$ many distance-$(\ell' + 1)$ descendants that we removed. The bound from this and the number of distance-ℓ' descendants in R lets us bound the number of distance-ℓ' descendants that we remove by

$$\frac{1}{k}\left(1 - \frac{1}{2\ell}\right)^{-1} \cdot \frac{1}{8}\left(1 + \frac{2}{\ell}\right)^{\ell-\ell'} k^{\ell'+1} + \frac{1}{8\ell}k^{\ell'} \leqslant \frac{1}{8}\left(1 + \frac{2}{\ell}\right)^{\ell-\ell'+1} k^{\ell'}\,.$$

It follows with $\ell' = 1$ that we remove at most $1/8 \cdot (1 + 2/\ell)^\ell k \leqslant e^2/8 \cdot k < (1 - 1/(2\ell))k$ children of p. Hence, p is not removed itself by the procedure. □

3.3 LOCAL TO GLOBAL CONGESTION

This section is to prove Lemma 3.4. Let Q be a degree-k solution with ℓ-local congestion L and global congestion K. We partition Q by length: let Q_i

be the set of paths $p \in Q$ with $|p| = i$. Further, we split the paths into ℓ groups $G_1, G_2, \ldots, G_\ell$, where $G_j = Q_j \cup Q_{j+\ell} \cup Q_{j+2\ell} \cup \cdots$. For some $p \in G_j$ we write $G(p) = G_j$. Roughly speaking, we proceed as follows. We sample from each G_ℓ half of the paths and throw away all others (including their descendants). Then we move to $G_{\ell-1}$ and do the same. We continue until G_1 and then repeat the same a second time, stopping afterwards. The sampling is done in a way that guarantees that each path retains a quarter of its children at the end. We prove with Lovász Local Lemma that in each step we can reduce the congestion significantly. Since it seems unclear how to argue directly about the worst case congestion, we will argue about the congestion aggregated over many paths, which we will formalize next.

Consider a path $p \in Q$ and its close descendants in $D(p, \leqslant \ell)$. Recall, that $D(p, \leqslant \ell)$ contains all descendants of length at most $|p| + \ell$. Intuitively, if many of the direct children of p have high congestion (more precisely, the vertex that they end in), this is bad for p as well: if they have high congestion, we may not be able to keep many of these children for p, which means we might not be able to include p itself in the solution. Let $\mathrm{cong}_G(p)$ be the congestion of the last vertex in p, but restricted to paths in the group $G(p)$. In other words,

$$\mathrm{cong}_G(p) = |\{q \in G(p) \mid q \text{ ends in the same vertex as } p\}| \,.$$

The restriction to other paths in $G(p)$ is only for technical reasons and almost at no cost: if we can achieve that every vertex is used by only few paths in each G_j (i.e., $\mathrm{cong}_G(p)$ is small for all $p \in Q$), the overall congestion can only be worse by a factor ℓ. An important quantity in the following will be the total congestion of descendants $D(p, \ell')$ of p at some distance $\ell' < \ell$, that is,

$$\mathrm{cong}_G(D(p, \ell')) = \sum_{q \in D(p,\ell')} \mathrm{cong}_G(q) \,, \tag{3.1}$$

Since during the procedure the number of children may differ between groups G_j, we will use $k(G_j)$ to describe the current number of children for every open path in G_j. Our intermediate goal will be to bound the totals (3.1) for some $p \in G_j$ in terms of $k(p, \ell') = \prod_{j'=j}^{j+\ell'-1} k(G_j)$ (an upper bound on $|D(p, \ell')|$).

Notice that intially (3.1) can be at most $K \cdot k(p, \ell')$. When sampling down the paths in G_j, we want to show that this reduces (3.1) significantly for paths $p \in G_j$ and all $\ell' < \ell$. This is captured in the following lemma.

Lemma 3.10. *Assume we are given a solution that has an ℓ-local congestion of at most L and global congestion of at most K, where $\ell \geqslant \log K$ and $K \geqslant \log n$. From the paths in G_j form pairs where each pair shares the same parent. Then select i.i.d. one path from each pair and remove it and all its descendants. Let cong_G and cong'_G be the congestion count before and after the removal and similarly k and k' the children count. Then we have with positive probability for every remaining $p \in G_j$ and $\ell' < \ell$ that*

$$\frac{\mathrm{cong}'_G(D(p, \ell'))}{k'(p, \ell')} \leqslant c\ell^4 L + \frac{1}{2}\left(1 + \frac{1}{\ell}\right) \frac{\mathrm{cong}_G(D(p, \ell'))}{k(p, \ell')} \,,$$

where c is a fixed constant. Furthermore, we can obtain such a sampling in expected polynomial time.

Proof. We can rewrite

$$\operatorname{cong}_G(D(p,\ell')) = \sum_{q\in G(p)} \operatorname{cong}_G(D(p,\ell'),D(q,\ell'))\ ,$$

where $\operatorname{cong}_G(D(p,\ell'),D(q,\ell'))$ is the number of pairs $p'\in D(p,\ell'), q'\in D(q,\ell')$ that end in the same vertex. We remove every term in the sum with probability $1/2$, so in expectation the sum will reduce by $1/2$. Furthermore, each term $\operatorname{cong}(D(p,\ell'),D(q,\ell'))$ is bounded by $L\cdot|D(p,\ell')|\leqslant L\cdot k(p,\ell')$, because we have low local congestion. This will give us good concentration. Notice also that $k(p,\ell')=k'(p,\ell')$.

We now group the terms in the sum by their size. For $p\in G_j$ let $G(p,\ell',t)$ be the set of $q\in G(p)=G_j$ with $\operatorname{cong}_G(D(p,\ell'),D(q,\ell'))\in[L\cdot k(p,\ell')\cdot 2^{-(t+1)}, L\cdot k(p,\ell')\cdot 2^{-t})$. Let Q' be the set of paths remaining after sampling down (without taking into account those that are removed recursively). Let c be a large constant to be specified later. Depending on whether t is small or large, we define bad events $\mathcal{B}(p,\ell',t)$ for each $p\in G_j$ as

$$\sum_{q\in G(p,\ell',t)\cap Q'} \operatorname{cong}_G(D(p,\ell'),D(q,\ell')) > 24c\ell^3 L\cdot k(p,\ell') + \frac{1}{2}\left(1+\frac{1}{\ell}\right)\sum_{q\in G(p,\ell',t)} \operatorname{cong}_G(D(p,\ell'),D(q,\ell')) \qquad \text{if } t\leqslant 2\ell,$$

$$\sum_{q\in G(p,\ell',t)\cap Q'} \operatorname{cong}_G(D(p,\ell'),D(q,\ell')) > 24c\ell^3 L\cdot \frac{1}{K}k(p,\ell') + \frac{1}{2}\left(1+\frac{1}{\ell}\right)\sum_{q\in G(p,\ell',t)} \operatorname{cong}_G(D(p,\ell'),D(q,\ell')) \qquad \text{if } t>2\ell.$$

Since from the experiment the total congestion cannot increase, we have a probability of 0 for all bad events where $\operatorname{cong}_{G(p,\ell',t)}(D(p,\ell')) :=$

$$\sum_{q\in G(p,\ell',t)} \operatorname{cong}_G(D(p,\ell'),D(q,\ell')) \leqslant \begin{cases} 12c\ell^3 L\cdot k(p,\ell') & \text{if } t\leqslant 2\ell,\\ 12c\ell^3 L\cdot \frac{1}{K}k(p,\ell') & \text{otherwise.}\end{cases}$$

For the remaining bad events, we will now derive an upper bound on the probabilities. This holds trivially also for the zero probability events. From Chernoff's bound we get

$$\begin{aligned}\mathbb{P}[\mathcal{B}(p,\ell',t)] &\leqslant \exp\left(-\frac{(1/\ell^2)\cdot 1/2\cdot \operatorname{cong}_{G(p,\ell',t)}(D(p,\ell'))}{(2+1/\ell)\cdot 2^{-t}L\cdot k(p,\ell')}\right)\\ &\leqslant \exp\left(-\frac{\operatorname{cong}_{G(p,\ell',t)}(D(p,\ell'))}{6\ell^2\cdot 2^{-t}L\cdot k(p,\ell')}\right)\\ &\leqslant \begin{cases}\exp\left(-2c\ell 2^t\right) & \text{if } t\leqslant 2\ell,\\ \exp\left(-2c\ell 2^t/K\right)\leqslant n^{-10} & \text{otherwise.}\end{cases}\end{aligned}$$

Towards applying LLL, we set the values of the bad events as

$$x(\mathcal{B}(p,\ell',t)) = \begin{cases} \exp\left(-c\ell 2^t\right) & \text{if } t \leqslant 2\ell, \\ n^{-5} & \text{otherwise.} \end{cases}$$

The experiment involves binary variables V. An event $\mathcal{B}(p,\ell',t)$ depends on at most $K \cdot 2^{t+1}$ many variables. A variable V influences at most $2K\ell \cdot 2^{t+1}$ type-t bad events. Notice that $2K\ell \cdot 2^{t+1} \leqslant \exp(c\ell 2^t)/(4K^2)$ for c sufficiently large (since $\ell \geqslant \log K$). Let $\Gamma_t(V)$ be the set of all these events for a specific variable V and a specific value of t. Then

$$\prod_{B \in \Gamma_t(V)} (1 - x(B)) \geqslant \left(1 - \exp\left(-c\ell 2^t\right)\right)^{2K\ell 2^{t+1}} \geqslant 1 - \frac{1}{4K^2} \,.$$

Notice that t can range only from 0 to $\log n$. Furthermore, since every vertex has congestion at most K, the number of paths in Q is at most Kn. Together, we can upper bound the total number of events by $Kn\ell \log n \leqslant n^5$. Thus, for $t \leqslant 2\ell$ and c sufficiently large it holds that

$$\begin{aligned}
\mathbb{P}[\mathcal{B}(p,\ell',t)] &\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \exp(-c\ell 2^t) \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \left(1 - \frac{1}{e}\right)^{4\ell \cdot 2^t} \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \prod_{t'=0}^{\ell} \left(1 - \frac{1}{4K^2}\right)^{K \cdot 2^{t+1}} \cdot \left(1 - \frac{1}{n^5}\right)^{n^5} \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \prod_{B \in \Gamma(\mathcal{B}(p,\ell',t))} (1 - x(B)) \,.
\end{aligned}$$

Similarly, for $t > 2\ell$ we have

$$\begin{aligned}
\mathbb{P}[\mathcal{B}(p,\ell',t)] &\leqslant x(\mathcal{B}(p,\ell',t) \cdot \exp(-c\ell 2^t/K) \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \left(1 - \frac{1}{e}\right)^{4\ell \cdot 2^t/K} \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \prod_{t'=0}^{\ell} \left(1 - \frac{1}{4K^2}\right)^{K \cdot 2^{t+1}} \cdot \left(1 - \frac{1}{n^5}\right)^{n^5} \\
&\leqslant x(\mathcal{B}(p,\ell',t)) \cdot \prod_{B \in \Gamma(\mathcal{B}(p,\ell',t))} (1 - x(B)) \,.
\end{aligned}$$

Hence, by LLL we have with positive probability that none of the bad events occur. If none of them occur, then by summing up bounds fixed p and ℓ' we get

$$\begin{aligned}
\operatorname{cong}'(D(p,\ell')) &\leqslant 2\ell \cdot 24c\ell^3 L \cdot k(p,\ell') \\
&\quad + \log n \cdot 24c\ell^3 L \cdot k(p,\ell')/K \\
&\quad + \frac{1}{2}\left(1 + \frac{1}{\ell}\right) \operatorname{cong}(D(p,\ell')) \\
&\leqslant 100c\ell^4 L \cdot k(p,\ell') + \frac{1}{2}\left(1 + \frac{1}{\ell}\right) \operatorname{cong}(D(p,\ell')) \,.
\end{aligned}$$

□

Lemma 3.11. *Consider a successful run of the random experiment in Lemma 3.10 where we sample down the paths in G_j and satisfy the stated inequalities. For each path p (potentially not in G_j) and every $\ell' \in \{1, 2, \dots, \ell\}$ we have*

$$\frac{\operatorname{cong}'_G(D(p,\ell'))}{k'(p,\ell')} \leqslant c\ell^4 L + \left(1 + \frac{1}{\ell}\right) \frac{\operatorname{cong}_G(D(p,\ell'))}{k(p,\ell')} .$$

Here c is the constant from Lemma 3.10.

Lemma 3.10 only shows that the average congestion reduces for descendants of paths in the group G_j, where we sample down. Conversely, Lemma 3.11 says that for all other groups it does not increase significantly.

Proof. Let $G_{j'} = G(p)$. We can assume without loss of generality that $j' < j \leqslant j' + \ell'$ (modulo ℓ), since the congestion can only decrease and $k(p,\ell')$ in the other case would not change. Let $\ell'' = j - j'$ (modulo ℓ). Further, let $D'(p,\ell')$ be the distance-ℓ' descendants of p after sampling down G_j and $D(p,\ell')$ before it. Then $D'(p,\ell'')$ contains half of the elements $D(p,\ell'')$. Thus,

$$\begin{aligned}
\frac{\operatorname{cong}'_G(p,\ell')}{k'(p,\ell')} &= \frac{1}{k'(p,\ell')} \sum_{q \in D'(p,\ell'')} \operatorname{cong}'_G(q, \ell' - \ell'') \\
&= \frac{2}{k(p,\ell')} \sum_{q \in D'(p,\ell'')} \operatorname{cong}'_G(q, \ell' - \ell'') \\
&\leqslant \frac{2}{k(p,\ell')} \sum_{q \in D'(p,\ell'')} [c\ell^4 L + \frac{1}{2}\left(1 + \frac{1}{\ell}\right) \operatorname{cong}_G(q, \ell' - \ell'')] \\
&\leqslant c\ell^4 L + \left(1 + \frac{1}{\ell}\right) \operatorname{cong}_G(p,\ell') .
\end{aligned}$$

□

Lemma 3.12. *Given a degree-k solution Q with ℓ-local congestion L and global congestion K, we can compute a degree-$k/4$ solution $Q' \subseteq Q$ with*

$$\operatorname{cong}_G(p,\ell) \leqslant 3c\ell^4 L \left(\frac{k}{4}\right)^{\ell} + \left(1 + \frac{1}{e}\right)^2 \frac{K}{2^\ell}$$

for all $p \in Q'$. Here c is the constant from Lemma 3.10.

Proof. We will perform the sampling from Lemma 3.10 for G_ℓ,$G_{\ell-1}$, …,G_1 and then again the same a second time. The reason is that we want that for every group G_j that $G_j, G_{j-1}, \dots, G_{j-\ell}$ (index modulo ℓ) are down-sampled at least once in this order.

Let $G_j = G(p)$ and consider the first time that we sample down $G_{j+\ell-1}$ (index taken modulo ℓ). Let $D(p,\ell')$ be the distance-ℓ' descendants of p before this sampling. Then for each $q \in D(p, \ell - 1)$ we have

$$\frac{\operatorname{cong}_G(D(q,1))}{k(q,1)} \leqslant K .$$

This is due to the fact that sampling down cannot increase the congestion on any vertex and initially all vertices have congestion at most K. After sampling according to Lemma 3.10 we have

$$\frac{\mathrm{cong}_G(q,1)}{k(q,1)} \leqslant c\ell^4 L + \left(1+\frac{1}{\ell}\right)\frac{K}{2}\,.$$

In the next step we are sampling down $G_{j+\ell-2}$. We have for each $q \in D(p,\ell-2)$ that

$$\frac{\mathrm{cong}_G(D(q,2))}{k(q,2))} = \frac{1}{k(q,1)}\sum_{q'\in C(q)}\frac{\mathrm{cong}_G(q',1)}{k(q',1)} \leqslant c\ell^4 L + \left(1+\frac{1}{\ell}\right)\frac{K}{2}\,.$$

Thus, after sampling

$$\begin{aligned}\frac{\mathrm{cong}_G(D(q,2))}{k(q,2)} &\leqslant c\ell^4 L + \frac{1}{2}\left(1+\frac{1}{\ell}\right)c\ell^4 L + \left(1+\frac{1}{\ell}\right)^2\frac{K}{4}\\ &\leqslant 2c\ell^4 L + \left(1+\frac{1}{\ell}\right)^2\frac{K}{4}\,.\end{aligned}$$

Continuing this argument, after we sample down G_j we have

$$\frac{\mathrm{cong}_G(p,\ell)}{k(p,\ell)} \leqslant 3c\ell^4 L + \left(1+\frac{1}{\ell}\right)^\ell\frac{K}{2^\ell}\,.$$

After G_j there may be at most ℓ more steps of sampling down, after which we finally have

$$\frac{\mathrm{cong}_G(p,\ell)}{k(p,\ell)} \leqslant 3c\ell^4 L + \left(1+\frac{1}{\ell}\right)^{2\ell}\frac{K}{2^\ell} \leqslant 3c\ell^4 L + \left(1+\frac{1}{e}\right)^2\frac{K}{2^\ell}\,.$$

□

We will now conclude the proof of Lemma 3.4. Using the previous lemma, we obtain a degree-$k/4$ solution Q'. Let $A := 3c\ell^4 L + (1+1/e)^2 K/2^\ell$ be the upper bound on average distance-ℓ congestion. Assuming without loss of generality that c is sufficiently large, we have that $A \leqslant 3c\ell^4 L$. Let R be the set of all paths p with $\mathrm{cong}_G(p) > 16A\ell^2$. We remove these paths using Lemma 3.9. We use the property that each path has at most a $(8\ell)^{-2}(k/4)^\ell$ many of its ancestors at distance ℓ in R. This follows directly from the bounded average congestion. The lemma implies that we can remove those high congestion paths and still keep a solution where each remaining path has $k/(4\cdot 2\ell) = k/8\ell$ children. Since we start with a ℓ-local congestion of at most $L \leqslant 16A\ell^2$ and this cannot be increased by only removing paths, we have that none of the paths of length at most ℓ are removed and thus all sources satisfy the premise of (2) of Lemma 3.9 and remain in the solution. Indeed, the value of $\mathrm{cong}_G(p)$ is now bounded by $16A\ell^2$ for all remaining paths. We recall that the actual congestion is at most a factor ℓ higher than $\max_p \mathrm{cong}_G(p)$, that is,

$$16A\ell^3 \leqslant 64c\ell^7 L\,.$$

3.4 COMPUTING A SOLUTION WITH LOCALLY LOW CONGESTION

The goal of this section is to prove Lemma 3.7. We recall that we are in a layered graph with vertices partitioned into layers $L_0, L_1, \ldots, L_h$ where $h = \log n$ and a single source s. The source s belongs to layer L_0 and edges can only go from vertices in some layer L_i to the next layer L_{i+1}.

In the following we will extensively argue about paths that start in the source. For the remainder of the section every path p that we consider is implicitly assumed to start at the source and then traverses (a prefix of) the layers one by one. Slightly abusing notation, we sometimes use L_i also to denote the set of paths ending in a vertex of L_i, that is,

$$\bigcup_{v \in L_i} I(v) \,,$$

and T to describe the set of closed paths (recall that those are the paths that end at a sink). Let P refer to the set of all possible paths and notice that by virtue of the layers we have that $|P| \leqslant n^h \leqslant n^{\log n + 1}$. We will now describe a linear programming relaxation, which goes back to Bateni et al. [15]. The intuition behind the linear program is to select paths similarly to the way we describe solutions, see Section 3.2. We have a variable $x(p)$ for each path p that in an integral solution takes value 1 if the path p is contained in an arborescence and 0 otherwise.

$$\sum_{q \in C(p)} x(q) = k \cdot x(p) \qquad \forall p \in P \setminus T \tag{3.2}$$

$$\sum_{q \in I(v) \cap D(p)} x(q) \leqslant x(p) \qquad \forall p \in P, v \in V \tag{3.3}$$

$$x((s)) = 1 \tag{3.4}$$

$$x \geqslant 0 \tag{3.5}$$

Here we assume that k is the highest value for which the linear program is feasible, obtained using a standard binary search framework. Moreover, we assume that $k \geqslant 2^{10}(\log\log n)^8$ in the rest of the section. This is at little cost, since a $1/k$-approximation is easy to obtain (see the proof of Theorem 2.1) and is already sufficient for our purposes. The first two types of constraints describe that each open path has many children and each vertex has low congestion (in fact, no congestion). Constraint (3.3) comes from a lift-and-project idea. For integral solutions it would be implied by the other constraints, but without it, there could be situations with continuous variables, where for example we take a path p with only value $1/k$ and then a single child of q with value 1. Such situations easily lead to large integrality gaps, which we can avoid by this constraint.

Since the graph has $h + 1$ many layers, this linear program has $n^{O(h)}$ variables and constraints and therefore can be solved in time $n^{O(h)}$. We refer to this relaxation as the *path LP*. In order to prove Lemma 3.7, we will design a rounding scheme. Before getting to the main part of the proof, we will first preprocess the fractional solution to sparsify its support.

3.4.1 *Preprocessing the LP solution*

Our first step is to sparsify the path LP solution x to get another sparser solution (i.e., with a limited number of non-zero entries). For ease of notation, we might need to take several times a copy of the same path $p \in P$. We emphasize here that two copies of the same path are different objects. To make this clear, we will now have a **multiset** P' of paths but we will slightly change the parent/child relationship between paths. Precisely, for any path $q' \in P'$ is assigned as child to a **unique** copy $p' \in P'$ such that q was a child of p in the set P. With this slight twist, all the ancestors/descendants relationships extend to multisets in the natural way. For instance, we will denote by $D_{P'}(p)$ the set of descendants of p in the multiset P'. Again, the set of closed path in P' will be denoted by T' and s refers to the source. We assume that there is a unique copy of the trivial path $(s) \in P'$.

In this step we will select paths such that each open path has $k \log^2 n$ children instead of the k children one would expect. However, we use a function $y(p)$ that assigns a weight to each path and this weight decreases from layer to layer, modelling that the children are actually picked fractionally each with a $1/4 \log^2 n$ fraction of the weight of the parent. Thus, taking the weights into account we are actually picking only $k/4$ children for each path.

Formally, the preprocessing of the LP will allow us to obtain a multiset of path P' such that

$$\sum_{q \in C_{P'}(p)} y(q) = \frac{k}{4} \cdot y(p) \qquad \forall p \in P' \setminus T' \tag{3.6}$$

$$\sum_{q \in I_{P'}(v) \cap D_{P'}(p)} y(q) \leqslant 2y(p) \qquad \forall p \in P', v \in V \tag{3.7}$$

$$\text{where } y(p) = \frac{1}{(4 \log^2(n))^i} \qquad \forall i \leqslant h, \forall p \in L_i \tag{3.8}$$

We obtain such a solution P' in a similar way as the randomized rounding in [15, 21], which achieves polylogarithmic congestion. The fact that we select more paths, but only fractionally gives us better concentration bounds, which allows us to lose only constant congestion here. For completeness we give the proof in Appendix A.2.

3.4.2 *The main rounding*

We start this part with the sparse multiset of paths P' with the properties as above. The discount value y can be thought of fractionally in the sense that each $p \in P'$ is taken to an extent of $y(p)$. We will proceed to round this fractional solution to an integral solution Q that is *locally nearly good*, a concept we will define formally below. Intuitively, this specifies that the number of paths that have locally high congestion can be removed without losing much with Lemma 3.9. We fix $\ell = 10 \log \log n$ for the rest of this section. Let $\text{cong}(v \mid Q)$ be the global congestion of vertex v induced by Q, that is, the number of paths in Q ending in v. For paths $p \in Q \cup \{\emptyset\}$ and $q \in D_Q(p)$ We denote by $\text{cong}_p(q \mid Q)$ the *local congestion* induced on the endpoint of q by descendants of p. We consider all paths descendants of $\emptyset$.

A locally nearly good solution is a multiset of paths $Q \subseteq P'$ (where again every path has a *unique* parent among the relevant copies of the same path) that has the following properties:

1. one copy of the trivial path (s) belongs to Q;
2. every open path has $k/32$ children;
3. no vertex has global congestion more than $2^{10}\log^3(n)$;
4. for every $p \in Q \cup \{\emptyset\}$ and $\ell' \leqslant \ell$ we have

$$|\{q \in D_Q(p,\ell') \mid \mathrm{cong}_p(q \mid Q) > 2^{10}\ell^2\}| \leqslant \frac{1}{\ell^2}\left(\frac{k}{32}\right)^{\ell'} .$$

From a locally nearly good solution we will then derive a solution of low local congestion by removing all paths with high local congestion using bottom to top pruning (Lemma 3.9). Condition 4 is tailored to ensure that the number of such high local congestion paths is small enough so that the lemma succeeds.

To obtain such a nearly good solution, we will proceed layer by layer, where the top layers are already rounded integrally and the bottom layers are still fractional (as in the preprocessing). However, unlike the case of the preprocessing, we cannot argue with high probability and union bounds, since some properties we want only deviate from expectation by $\mathrm{poly}(\log\log n)$ factors. To obtain a locally nearly good solution, we will use Lovász Local Lemma (LLL) in every iteration, where one iteration rounds one more layer. In the following, we describe the rounding procedure, the bad events and an analysis of their dependencies and finally we apply LLL.

THE RANDOMIZED ROUNDING PROCEDURE. We proceed layer by layer to round the solution P' (with discount y) to an integral solution Q. We start by adding a single copy the trivial path (s) to the partial solution $Q^{(0)}$. Assume we rounded until layer i, that is, we selected the final multiset of paths $Q^{(i)}$ to be used from all paths in $L_{\leqslant i}$.

To round until layer $i+1$ we proceed as follows. Every open path $p \in Q^{(i)} \cap L_i$ selects exactly $k/16$ children where the i-th child equals $q \in C_{P'}(p)$ with probability equal to

$$\frac{1}{k\log^2 n} = \frac{y(q)}{\sum_{q' \in C_{P'}(p)} y(q')} .$$

The selection of each child is independent of the choices made for other children. We then let $Q^{(i+1)}$ be the union of $Q^{(i)}$ and all newly selected paths. The reason that each path selects $k/16$ children instead of the $k/32$ many that were mentioned before is that we will later lose half of the children. We will repeat this procedure until reaching the last layer h and we return $Q = Q^{(h)}$.

DEFINITIONS RELATED TO EXPECTED CONGESTION. In order for this iterated rounding to succeed we need to avoid that vertices get high congestion (in the local and the global sense). It is not enough to keep track only of

the congestion of vertices in the next layer that we are about to round, but we also need to maintain that the expected congestion (over the remaining iterations) remains low on all vertices in later layers. Hence, we will define quantities that help us keep track of them. To avoid confusion, we remark that the quantities we will define do not exactly correspond to the expected congestion of the vertices: notice that we sample less children for each path than P' has. Intuitively, P' has $k/4$ children per open path (see Equation (3.6)), but we only sample $k/16$ many. The quantities we define below would be the expectations if we would sample $k/4$ children instead. Roughly speaking, this gives us an advantage of the form that the expectation always decreases by a factor 4 when we round one iteration. Apart from the intuition on the expectation, the definitions also incorporate a form of conditional congestion, which means that we consider the (expected) congestion based on random choices made so far. This is similar to notions that appear in Appendix A.2 for the preprocessing step.

First, we define the fractional congestion induced by some path $p \in P'$ on a vertex $v \in V$ as follows.

$$\operatorname{cong}(p, v) := \sum_{q \in D_{P'}(p) \cap I_{P'}(v)} y(q).$$

Using this definition, we will define the conditional fractional congestion at step i induced by p on a descendant q as follows (writing v_q for the endpoint of q).

$$\operatorname{cong}_p(q \mid Q^{(i)}) := \begin{cases} |\{q' \in D_{Q^{(i)}}(p) \cap I_{Q^{(i)}}(v_q)\}| & \text{if } q \in L_{\leqslant i} \\ \sum_{q' \in D_{Q^{(i)}}(p) \cap L_i} \frac{1}{y(q')} \operatorname{cong}(q', v_q) & \text{otherwise.} \end{cases}$$

We will use this definition only for $p \in Q^{(i)}$. If v_q belongs to one of the first i layers (i.e., v_q is in the integral part corresponding to the partial solution), this is simply the number of descendants of $p \in Q^{(i)}$ in our partial solution that end at v_q. Otherwise, this is the total fractional congestion induced by all the paths that we actually selected in our partial solution $q \in Q^{(i)} \cap L_i$ and that are descendants of p. The name *conditional* comes from the term $1/y(q')$ which is simply the fact that we condition on having already selected the path q' once integrally. The global congestion at step i on a vertex $v \in V$ is defined similarly with

$$\operatorname{cong}(v \mid Q^{(i)}) := \begin{cases} |\{q \in Q^{(i)} \cap I(v)\}| & \text{if } v \in L_{\leqslant i} \\ \sum_{q \in Q^{(i)} \cap L_i} \frac{1}{y(q)} \operatorname{cong}(q, v) & \text{otherwise.} \end{cases}$$

THE FIRST TYPE OF BAD EVENT: GLOBAL CONGESTION. The naive way to bound the global congestion would be to simply to bound the global congestion on each vertex and make a bad event from exceeding this. However, to manage dependencies between bad events and get better concentration bounds, we partition the set of paths according to how much fractional load their ancestors are expected to put on v and then bound the congestion incurred by each group on v. More precisely, assume we rounded until layer i

and are now trying to round until layer L_{i+1}. Consider any vertex $v \in L_{\geqslant i+1}$. The decisions made in this step of the rounding concern which paths in L_{i+1} will be selected. Fix an integer $t \geqslant 0$ and and let $P_{v,t}^{(i+1)}$ to be the set of all paths $p \in L_{i+1}$ such that

$$\frac{\operatorname{cong}(p,v)}{y(p)} \in (2^{-t}, 2^{-(t-1)}] \,.$$

We define the bad event $\mathcal{B}_1(v,t)$ as the event that

$$\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\operatorname{cong}(p,v)}{y(p)} > 2\mathbb{E}\left[\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\operatorname{cong}(p,v)}{y(p)}\right] + 2^{10} \log n \,.$$

Since $P_{v,t}^{(i+1)}$ partitions the paths in L_{i+1}, the bad events for all t together will bound the increase in the congestion on v: this is because $\operatorname{cong}(v \mid Q^{(i+1)}) = \sum_{p \in Q^{(i+1)} \cap L_{i+1}} \operatorname{cong}(p,v)/y(p)$. We argue this formally in Lemma 3.13 below.

At this point, it is worthwhile to mention that t can only take values in $\{0, 1, \ldots, \log^2 n\}$. Indeed, by Constraint (3.7) we have that

$$\frac{\operatorname{cong}(p,v)}{y(p)} \leqslant 2$$

is ensured for all $p \in P'$. Moreover, we notice that y has a low granularity (i.e. y satisfies Equation (3.8)) which implies that either $\operatorname{cong}(p,v) = 0$ or

$$\frac{\operatorname{cong}(p,v)}{y(p)} \geqslant \frac{1}{(4\log^2 n)^{h-|p|}} \geqslant \frac{1}{(4\log^2 n)^{h}} \,.$$

Hence bad events $\mathcal{B}_1(v,t)$ will be instantiated only for $t = 0, 1, \ldots, \log^2 n$ (assuming here that n is sufficiently large).

Lemma 3.13. *Assuming no bad event $\mathcal{B}_1(v,t)$ has occurred in any iteration up to i, we have for every v that*

$$\operatorname{cong}(v \mid Q^{(i)}) \leqslant 2^{11} \log^3 n \,.$$

Proof. We argue inductively. For $i = 0$ we have

$$\operatorname{cong}(v \mid Q^{(i)}) = \sum_{p \in Q^{(i)} \cap L_i} \frac{\operatorname{cong}(q,v)}{y(q)} = \sum_{s \in S} \operatorname{cong}((s),v) = \sum_{p \in I_{P'}(v)} y(v) \leqslant 2 \,.$$

Now assume that for $i \geqslant 0$ we have

$$\operatorname{cong}(v \mid Q^{(i)}) \leqslant 2^{11} \log^3 n \,.$$

Since the congestion of any $v \in L_{\leqslant i}$ cannot change anymore, we may assume w.l.o.g. that $v \in L_{\geqslant i+1}$. For each t let

$$\mu_t = \mathbb{E}\left[\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\operatorname{cong}(p,v)}{y(p)}\right] .$$

Then $\sum_t \mu_t = 1/4 \cdot \mathrm{cong}(v \mid Q^{(i)}) \leqslant 2^9 \log^3 n$. To see this, denote by $N_q^{(i)}$ the number of copies of a path $q \in P'$ that is selected in $Q^{(i)}$. As a shorthand, we write $P(q)$ for the parent of q in $Q^{(i+1)}$. By linearity of expectation we can write

$$\begin{aligned}
\sum_t \mu_t &= \sum_t \mathbb{E}\Bigg[\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(p)}\Bigg] \\
&= \sum_t \mathbb{E}\Bigg[\sum_{p \in P' \cap P_{v,t}^{(i+1)}} N_p^{(i+1)} \cdot \frac{\mathrm{cong}(p,v)}{y(p)}\Bigg] \\
&= \sum_t \sum_{p \in P' \cap P_{v,t}^{(i+1)}} \mathbb{E}[N_p^{(i+1)}] \cdot \frac{\mathrm{cong}(p,v)}{y(p)} \\
&= \sum_t \sum_{p \in P' \cap P_{v,t}^{(i+1)}} \frac{(k/16) \cdot y(p)}{(k/4) \cdot y(P(p))} \cdot \frac{\mathrm{cong}(p,v)}{y(p)} \\
&= \frac{1}{4} \cdot \Bigg[\sum_t \sum_{p \in P' \cap P_{v,t}^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(P(p))}\Bigg] \\
&= \frac{1}{4} \cdot \Bigg[\sum_t \sum_{p \in P' \cap P_{v,t}^{(i)}} \frac{\mathrm{cong}(p,v)}{y(p)}\Bigg] \\
&= (1/4) \cdot \mathrm{cong}(v \mid Q^{(i)}) \leqslant 2^9 \log^3 n \,.
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathrm{cong}(v \mid Q^{(i+1)}) &= \sum_{p \in Q^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(p)} \\
&= \sum_t \sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(p)} \\
&\leqslant \sum_t [2\mu_t + 2^{10} \log n] \\
&\leqslant 2^{10} \log^3 n + 2^{10} \log^3 n \\
&\leqslant 2^{11} \log^3 n \,.
\end{aligned}$$

□

THE SECOND TYPE OF BAD EVENT: LOCAL CONGESTION. Recall that we want to bound the congestion induced by close descendants of some open path p. Let $\ell' \leqslant \ell$ and $p \in Q^{(i)} \cap L_{i'}$, where $i - \ell' \leqslant i' \leqslant i$. We define

$$\begin{aligned}
N^{(i+1)}(p,\ell') = |\{q \in D_{P'}(p,\ell') \mid \mathrm{cong}_p(q \mid Q^{(i)}) \leqslant 2^{10}\ell^2 \\
\text{and } \mathrm{cong}_p(q \mid Q^{(i+1)}) > 2^{10}\ell^2\}| \,.
\end{aligned}$$

These are the number of vertices with newly high local congestion (counting only the congestion induced by descendants of p). Moreover, note that by Constraint (3.6) and (3.8), we have that $|D_{P'}(p,\ell')| \leqslant (k \log^2 n)^{\ell'}$.

Then, for any $p \in L_{i'} \cap Q^{(i)}$ where $i - \ell - 1 \leqslant i' \leqslant i - 1$, and $t \geqslant 0$, we define the set of *marked* children of p at step $i+1$ as the set $M^{(i+1)}(p)$ of all $q \in C_{Q^{(i)}}(p)$ such that

$$N^{(i+1)}(q, \ell') > \frac{1}{\log^{10} n} \left(\frac{k}{32}\right)^{\ell'}$$

for at least one $\ell' \leqslant \ell$. Notice that the children of p (in $Q^{(i)}$) are already determined because p is in $L_{\leqslant i-1}$. We are now ready to state the second type of bad event. We define the bad event $\mathcal{B}_2(p)$ for any $p \in L_{i'}$ where $i - \ell - 1 \leqslant i' \leqslant i - 1$, as the event that

$$|M^{(i+1)}(p)| \geqslant \frac{1}{\ell^3} \cdot \frac{k}{16} ,$$

which means that a lot of children selected by p become marked at step $i+1$. As we will ensure that this bad event never happens, we can guarantee that a large fraction of children q of each path p always satisfies that $N^{(i+1)}(q, \ell')$ is low for all $\ell' \leqslant \ell$. As we explain later, we will simply remove all other children and this way indirectly prevent the existence of too many locally congested descendants for all the remaining paths. The advantage of this indirect method is that we can apply good concentration bounds on the children and thereby amplify the probability (indeed, note that the event that a child q of p is marked is independent of the same events for other children of p).

THE THIRD TYPE OF BAD EVENT: KEEPING THE SOURCE PATH (s). Because our algorithm will delete in the end all the paths that have been marked in some round, we need to ensure that the source path (s) is never marked. Hence for the first ℓ steps of rounding, we define a single bad event $\mathcal{B}_3(s)$ that is that the source path (s) is marked during this step.

THE PROBABILITIES OF THE BAD EVENTS. In this part, we derive upper bounds for the probability of each bad event. Some bounds are sub-optimal to simplify later formulas.

Lemma 3.14. *For any $v \in V$ and integer t we have that*

$$\mathbb{P}[\mathcal{B}_1(v,t)] \leqslant n^{-10 \cdot 2^{t-1}} .$$

Proof. We need to prove that

$$\mathbb{P}\left[\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(p)} > 2\mathbb{E}\left[\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{\mathrm{cong}(p,v)}{y(p)}\right] + 2^{10} \log n\right] \leqslant n^{-10 \cdot 2^{t-1}} .$$

Notice that $\mathrm{cong}(p,v)/y(p)$ is upper bounded by $2^{-(t-1)}$ by definition of $P_{v,t}^{(i+1)}$ and the sum can be rewritten as a sum over the paths chosen in iteration $i+1$, where a path p contributes $\mathrm{cong}(p,v)/y(p)$ to the sum. These

paths are chosen independently and thus by a standard Chernoff bound we obtain that the probability is at most

$$\exp\left(\frac{-2^{10}\log n}{2\cdot 2^{-(t-1)}}\right) \leqslant n^{-10\cdot 2^{t-1}} .$$

□

Lemma 3.15. *Let $\ell' \leqslant \ell$ and $p \in Q^{(i)} \cap L_{i'}$, where $i - \ell' \leqslant i' \leqslant i$. Then*

$$\mathbb{P}[N^{(i+1)}(p,\ell') > (1/\log^{10} n)\cdot(k/32)^{\ell'}] \leqslant (\log n)^{-40\ell} . \tag{3.9}$$

Moreover,

$$\mathbb{P}[\mathcal{B}_3(s)] \leqslant (\log n)^{-1} . \tag{3.10}$$

Proof. Recall that we are bounding the number of descendants of p that had low local congestion of at most $2^{10}\ell^2$ before, but now get high local congestion. More concretely, consider some $q \in D_{P'}(p,\ell')$ with $\mathrm{cong}_p(q \mid Q^{(i)}) \leqslant 2^{10}\ell^2$. Then

$$\mathbb{E}\left[\mathrm{cong}_p(q \mid Q^{(i+1)})\right] = \frac{1}{4}\cdot \mathrm{cong}_p(q \mid Q^{(i)}) \leqslant 2^8\ell^2 .$$

We do not go into detail for this calculation here but similar calculations have been derived already in Lemma 3.13 or Appendix A.2. This is exactly the same type or argument.

Further, $\mathrm{cong}_p(q \mid Q^{(i+1)})$ can be decomposed into a sum of independent random variables bounded by 2. Indeed, recalling the definition of $\mathrm{cong}_p(q \mid Q^{(i+1)})$ we see that this quantity can be written as a sum over $1/y(q')\cdot \mathrm{cong}_{q'}(q \mid Q^{(i+1)})$, where a paths q' are taken from a certain multiset of chosen paths. Constraint (3.7) of the path LP ensures that this is always at most 2. Those variables are independent because of the randomized rounding that selects children independently of each other. Hence, by a Chernoff bound we have

$$\mathbb{P}\left[\mathrm{cong}_p(q \mid Q^{(i+1)}) > 2^{10}\ell^2\right] \leqslant \exp(-2^7\ell^2) .$$

Therefore, by linearity of expectation we obtain

$$\mathbb{E}[N^{(i+1)}(p,\ell')] \leqslant \frac{(k\log^2 n)^{\ell'}}{\exp(2^7\ell^2)} \leqslant \frac{1}{(\log n)^{2^7\cdot 10\ell}}\left(\frac{k}{32}\right)^{\ell'} .$$

From Markov's inequality it follows that

$$\mathbb{P}[N^{(i+1)}(p,\ell') > (1/\log^{10} n)\cdot(k/32)^{\ell'}] \leqslant (\log n)^{-40\ell} .$$

For the second claim, notice that $\mathcal{B}_3(s)$ is simply the event that the source path (s) is marked, which is equivalent to $N^{(i+1)}((s),\ell') > (1/\log^{10} n)\cdot(k/32)^{\ell'}$ for at least one $\ell' \leqslant \ell$. By a simple union bound, we get the desired result. □

Finally, we upper-bound the probability of $\mathcal{B}_3(p)$.

Lemma 3.16. *Let $p \in L_{i'}$ where $i - \ell - 1 \leqslant i' \leqslant i - 1$. Then we have that*

$$\mathbb{P}[\mathcal{B}_3(p)] \leqslant \exp\left(-\sqrt{k}\right) .$$

Proof. To prove this, note that a child q of p (in the set $Q^{(i)}$) is marked independently of other children of p. This is because q being marked depends only on the random choices made by descendants of q, which are independent of descendants of other children of p. Therefore using Lemma 3.15 and a standard union bound we obtain that each child of p is marked independently with probability at most $\ell \cdot (\log n)^{-40\ell}$. Recall that $k \geqslant 2^{10}(\log\log n)^8 \geqslant (32\ell^3)^2$. We note that p has $k/16$ children in $Q^{(i)}$ and therefore the probability that more than $(1/\ell^3) \cdot (k/16)$ of them are marked is at most

$$\exp\left(-\frac{k}{32\ell^3}\right) \leqslant \exp\left(-\sqrt{k}\right) .$$

□

THE DEPENDENCIES OF THE BAD EVENTS. For any bad event $\mathcal{B}$ we define $\Gamma_1(\mathcal{B})$ to be the set of bad events of the first type $\mathcal{B}_1(v,t)$ that depend on the bad event $\mathcal{B}$. Similarly, let $\Gamma_2(\mathcal{B})$ the set of bad events $\mathcal{B}_2(p)$ that depend on $\mathcal{B}$. We will now upper bound the cardinality of these sets. Note that there is a single bad event of third type $\mathcal{B}_3(s)$ hence it is clear that $\Gamma_3(\mathcal{B}) \leqslant 1$ for any bad event $\mathcal{B}$. For the rest, we remark that the focus here is on simplicity of the terms rather than optimizing the precise bounds.

Lemma 3.17. *For any bad event $\mathcal{B}$, we have that*

$$|\Gamma_1(\mathcal{B})| \leqslant n^2 .$$

Proof. The statement holds trivially, since there are at most $n \cdot (1 + \log^2 n) \leqslant n^2$ bad events of the first type in total (n possibilities of the choice of vertex v and less than n possibilities for the choice of t). □

Before proving the dependencies to events of type 2, we will prove an auxiliary lemma that concerns the events affected by the children picked by one particular path $p \in Q^{(i)} \cap L_i$.

Lemma 3.18. *Let $p \in Q^{(i)} \cap L_i$. Then the choice of children picked by p affects in total at most $\log n$ events of type 2.*

Proof. In order to influence a bad event $B_2(q)$ for some q, it must be that p is a descendant of q in the multiset $Q^{(i)}$. Any path has at most $h \leqslant \log n$ ancestors. □

Lemma 3.19. *Assuming that no bad event has happened until the current iteration of rounding, for any $v \in V$ and integers $t \geqslant 0$, we have*

$$|\Gamma_2(\mathcal{B}_1(v,t))| \leqslant 2^t \cdot (\log n)^{10} . \tag{3.11}$$

Proof. We first notice that the bad event $\mathcal{B}_1(v,t)$ depends only on the random choices made by paths $q \in Q^{(i)} \cap L_i$ that have at least one child $q' \in C_{P'}(q)$ such that

$$\frac{\mathrm{cong}(q',v)}{y(q')} \geqslant 2^{-t}.$$

Let us count how many such paths q can exist in $Q^{(i)}$. For each such path we have that

$$\frac{\mathrm{cong}(q,v)}{y(q)} = \sum_{q' \in C_{P'}(q)} \frac{\mathrm{cong}(q',v)}{y(q)} = \sum_{q' \in C_{P'}(q)} \frac{\mathrm{cong}(q',v)}{(4\log^2 n) y(q')} \geqslant \frac{2^{-t}}{4\log^2 n},$$

where we used here the granularity of y, see Constraint (3.8). Since we assume that no bad event happened so far, we can apply Lemma 3.13 to get

$$\sum_{q \in Q^{(i)} \cap L_i} \frac{\mathrm{cong}(q,v)}{y(q)} = \mathrm{cong}(v \mid Q^{(i)}) \leqslant 2^{11} \log^3 n\,.$$

Hence, there can be at most $2^t \cdot 2^{13} \log^5 n$ such paths q. We conclude with Lemma 3.18 to bound the number of bad events of second type influenced by each q and obtain

$$|\Gamma_2(\mathcal{B}_1(v,t))| \leqslant 2^t \cdot (2^{13} \log^5 n) \cdot (\log n) \leqslant 2^t \cdot (\log^{10} n)\,.$$

□

Lemma 3.20. *For any $p \in L_{i'}$ where $i - \ell - 1 \leqslant i' \leqslant i - 1$ we have*

$$|\Gamma_2(\mathcal{B}_2(p))| \leqslant k^\ell \cdot (\log n)\,. \tag{3.12}$$

Furthermore,

$$|\Gamma_2(\mathcal{B}_3(s))| \leqslant k^\ell \cdot (\log n)\,. \tag{3.13}$$

Proof. The bad event $\mathcal{B}_2(p)$ (or $\mathcal{B}_3(s)$) is entirely determined by the random choices made by the descendants of p (or (s)) in $Q^{(i)}$ that are located in layer L_i. There are at most $(k/16)^\ell \leqslant k^\ell$ such descendants q. By Lemma 3.18, each descendant can influence at most $\log n$ other bad events of the second type. This concludes the proof. □

INSTANTIATING LLL. For the bad events we set

$$x(\mathcal{B}_1(v,t)) = n^{-2}\,, \tag{3.14}$$

$$x(\mathcal{B}_2(p)) = \exp\left(-k^{1/3}\right) \tag{3.15}$$

$$x(\mathcal{B}_3(s)) = 1/2\,. \tag{3.16}$$

Consider now a bad event $\mathcal{B}_1(v,t)$ and recall that $k \geqslant 2^{10}(\log\log n)^8$. Then

$$\begin{aligned}\mathbb{P}[\mathcal{B}_1(v,t)] &\leqslant n^{-10\cdot 2^{(t-1)}} \\ &\leqslant n^{-5\cdot 2^t} \\ &\leqslant n^{-2}\cdot(1-n^{-2})^{n^2}\cdot\left(1-\exp(-k^{1/3})\right)^{2^t\cdot(\log n)^{10}}\cdot(1/2) \\ &\leqslant n^{-2}\cdot\prod_{\mathcal{B}\in\Gamma(\mathcal{B}_1(v,t))}(1-x(\mathcal{B}))\,.\end{aligned}$$

Next, we get

$$\begin{aligned}\mathbb{P}[\mathcal{B}_2(p)] &\leqslant \exp\left(-\sqrt{k}\right) \\ &\leqslant \exp\left(-k^{1/3}\right)\cdot\frac{(1-n^{-2})^{n^2}}{2}\cdot\left(1-\exp(-k^{1/3})\right)^{k^\ell\cdot(\log n)} \\ &\leqslant x(\mathcal{B}_2(p))\cdot\prod_{\mathcal{B}\in\Gamma(\mathcal{B}_2(p))}(1-x(B))\,.\end{aligned}$$

Similarly, for $\mathcal{B}_3(s)$ we have

$$\begin{aligned}\mathbb{P}[\mathcal{B}_3(s)] &\leqslant (\log n)^{-1} \\ &\leqslant (1/2)\cdot(1-n^{-2})^{n^2}\cdot\left(1-\exp(-k^{1/3})\right)^{k^\ell\cdot(\log n)} \\ &\leqslant x(\mathcal{B}_3(s))\cdot\prod_{\mathcal{B}\in\Gamma(\mathcal{B}_3(s))}(1-x(B))\,.\end{aligned}$$

Hence, with positive probability none of the bad events occur and we can compute such a solution in expected quasi-polynomial time (polynomial in $|P'|$).

FINISHING NOTES. We proved that we can round in (expected) quasi-polynomial time from layer 0 to layer h without any bad event ever happening. Next, observe that we get that any sampled path $p \in Q \cap L_i$ has at most

$$\left|\bigcup_{i'=i+1}^{i+\ell+1} M^{(i')}(p)\right| \leqslant \ell\cdot\frac{1}{\ell^3}\cdot\frac{k}{16} \leqslant \frac{k}{32}$$

marked children. We now delete from the solution all the marked paths obtaining a solution Q'. Note that the source remains as it is never marked. This means that any sampled open path retains at least $k/32$ children. For any path p, we denote by $N(p,\ell')$ the number of its descendants at distance ℓ' that have congestion induced by p bigger than $2^{10}\ell^2$. Then if p was never marked at any round, we have that

$$N(p,\ell') \leqslant \frac{1}{\log n}\left(\frac{k}{32}\right)^{\ell'}\,.$$

Indeed, if p belongs to layer i then $\mathrm{cong}_p(q \mid Q^{(i)}) = 1/y(p)\cdot\mathrm{cong}(p,v_q) \leqslant 2$ by Constraint (3.7). Hence the first time we instantiate bad events involving local congestion induced by the path p, it is the case that none of its descendants are bad. Then if p is never marked we have that

$$N^{(i')}(p,\ell') \leqslant \frac{1}{\log^{10} n}\cdot\left(\frac{k}{32}\right)^{\ell'}$$

at every iteration $i \leqslant i' \leqslant i+\ell$, which means that few of the good descendants become bad at each round. Hence we obtain that

$$N(p,\ell') \leqslant \sum_{i'=i}^{i+\ell} N^{(i')}(p,\ell') \leqslant \frac{1}{\log n}\left(\frac{k}{32}\right)^{\ell'} .$$

This ensures that any remaining path has few bad descendants (in terms of local congestion).

We now define a set of paths $R \subseteq Q$, which contains all paths with high local congestion. More precisely, let R contain for every path $p \in Q$ and $\ell' \leqslant \ell$ all paths $q \in D_Q(p,\ell')$ such that $\mathrm{cong}_p(q \mid Q') > 2^{10}\ell^2$. Since we have no more marked paths, each p has at most $(\log n)^{-1} \cdot (k/32)^{\ell'} \leqslant 1/(8\ell)^2 \cdot (k/32)^{\ell'}$ descendants at distance ℓ' in R. Applying Lemma 3.4 we can remove R and obtain a solution $Q' \subseteq Q \setminus R$, where each open path has $k/(64\ell)$ children and the source remains in the solution. Moreover, the solution has no more paths with ℓ-local congestion more than $2^{10}\ell^2$ and the global congestion is at most $2^{11}\log^3 n$. This finishes the proof of Lemma 3.7.

3.5 FROM SINGLE SOURCE TO MULTIPLE SOURCES

We devise an algorithm that solves the problem with multiple sources based on an algorithmic technique first introduced by Haxell in the context of hypergraph matchings and then applied to a range of other problems, including the restricted assignment version of the Santa Claus problem. Haxell's algorithmic technique can be thought of as a (highly non-trivial) generalization of the augmenting path algorithm for bipartite matching. Our algorithm makes only calls to the α-approximation for a single source. The algorithm itself requires quasi-polynomial time as well (even if the α-approximation ran in polynomial time). Our concrete variant relies on the following simple, but powerful subprocedure.

Lemma 3.21. *Let Q be a (single) degree-k' arborescence in a layered graph. Let $R \subseteq Q$ be a set of paths where at most $(k'/4)^i$ many end in each layer i. Then there is a degree-$k'/4$ arborescence $Q' \subseteq Q \setminus R$. Furthermore, Q' can be computed in polynomial time.*

Proof. We start pruning the arborescence from bottom to top and we maintain at every layer i that we remove at most $(k'/2)^i$ paths in it. At the last layer $i = h$ we remove simply the paths that are in R. These are by definition at most $(k'/4)^i \leqslant (k'/2)^i$ many. Then assume we already pruned layers $i, i+1, \ldots, h$ and that we removed at most $(k'/2)^i$ many paths in layer i. Now in layer $i-1$ we again remove the at most $(k'/4)^{i-1}$ many paths in R, but also those open paths where more than $3/4 \cdot k'$ many children were removed in layer i. The number of open paths removed this way is at most $(k'/2)^i/(3/4 \cdot k') \leqslant 2/3 \cdot (k'/2)^{i-1} < (k'/2)^{i-1}$. Clearly, this procedure maintains that every remaining open path has at least $k'/4$ children and the source never gets removed. □

Throughout the section we assume that k is the optimum, which is obtained through a binary search framework. We now assume that we have a degree-$k/256\alpha$ solution that already covers all but one source $s_0 \in S$. We will

augment this solution to one that covers all sources. This is without loss of generality, since we can apply the procedure iteratively $|S|$ times, adding one source at a time.

BLOCKING TREES AND ADDABLE TREES. Let Q be our current solution (which does not cover s_0). On an intuitive level, *blocking trees* are $k/256\alpha$-degree arborescences in our current solution Q, which we would like to remove from the solution. In order to remove them, we need to add other arborescences for their sources instead. The *addable trees* are $k/32\alpha$-degree arborescences not in our solution. Their sources are sources of blocking trees in Q and we would like to add to them solution. The addable trees in turn may be *blocked* by blocking trees, which means they overlap on some non-source vertex with a tree in Q, preventing us from adding them to the solution. We note that addable trees are (by a factor of 8) better arborescences than what we ultimately need. As explained above, addable and blocking trees naturally form an alternating structure, which is usually referred to as layers. In order to avoid conflicts with our other notion of layers, we call these *rings*. The addable and blocking trees in the ith ring will be denoted by A_i and B_i.

THE ALGORITHM. Initially we have no rings. We run our α-approximation to find a k/α-degree arborescence for s_0 (without taking into account our current solution Q). We reduce this to a $k/32\alpha$-degree arborescence and store it as singleton set in A_1. Then we store in B_1 all arborescences in Q that intersect on any vertex with it. If the total intersection on each layer j is at most $(k/128\alpha)^j$, then we can find through Lemma 3.21 a $k/128\alpha$-degree arborescence for s_0 which is disjoint from Q. We reduce it to a $k/256\alpha$-degree arborescence and add it to the solution and terminate. Otherwise we continue the algorithm, but now we indeed have one ring of addable and blocking trees.

Assume now the algorithm in its current state has rings $A_1, B_1, \ldots, A_i, B_i$. Then we intialize $A_{i+1} = \emptyset$ and add addable trees to it in the following Greedy manner. From our (layered) graph we produce a reduced instance by removing any vertices appearing in $A_1, B_1, \ldots, A_i, B_i, A_{i+1}$ (except for the sources). Then we iterate over any source s that is either s_0 or the source of a blocking tree in $B_1 \cup \cdots \cup B_i$. We call our α-approximation for s on the reduced instance. If it outputs a k'-degree solution with $k' \geqslant k/32\alpha$, we add it to A_{i+1} (after reducing to degree exactly $k/32\alpha$). We repeat this until we can no longer add any addable trees to A_{i+1}. Then we construct B_{i+1} by taking any arborescence in Q that intersects with any addable tree in A_{i+1}. As before, we check if any addable tree in A_{i+1} has a total intersection of at most $(k/128\alpha)^j$ on each layer j with solution Q. If so, we collapse: we can compute through Lemma 3.21 a $k/128\alpha$-degree arborescence for the corresponding source s, which is disjoint from Q. We reduce it to a $k/256\alpha$-degree arborescence and add it to Q. Now s has two arborescences in Q. The other arborescence must have appeared as a blocking tree in an earlier ring $B_{i'}$. We remove this blocking tree from Q and delete all rings after i'. Then we revisit the addable trees in $A_{i'}$. Since we removed a blocking tree in $B_{i'}$, one of them may now have a small intersection with all trees in Q (as above). If so, we collapse this as well. We continue so until no more collapse

is possible, leaving us with a new solution Q and a prefix of the previous rings. We then continue the algorithm. Once an addable tree for s_0 is added to Q we terminate.

ANALYSIS. Our analysis relies on two lemmas.

Lemma 3.22. *At any point of time in the local search algorithm we have for every ring that* $|B_i| \geqslant |A_i|$.

Proof. Recall that the addable trees in A_i are vertex disjoint by construction. At the same time, none of them can be collapsed. This means for each of them there must be a layer j such that the addable tree intersects with Q (that is, with B_i) on $(k/128)^j$ many vertices. Let R_j be the set of all intersecting vertices with Q in layer j (across all addable trees in A_i). Then

$$\sum_{j \geqslant 1} \frac{|R_j|}{(k/128)^j} > |A_i| \,.$$

On the other hand, each blocking tree in B_i has at most $(k/256)^j$ many vertices in each layer j. Thus,

$$\sum_{j \geqslant 1} \frac{|R_j|}{(k/128)^j} \leqslant \sum_{j \geqslant 1} |B_i| \cdot \frac{1}{2^j} \leqslant |B_i| \,.$$

□

Lemma 3.23. *After the Greedy procedure to construct* A_{i+1}*, we have that* $|A_{i+1}| \geqslant \sum_{i'=1}^{i} |B_{i'}|$.

Proof. By Lemma 3.22 we have that

$$\sum_{i'=1}^{i} |A_{i'}| + |B_{i'}| \leqslant 2 \sum_{i'=1}^{i} |B_{i'}| \,.$$

Now assume toward constradiction that $|A_{i+1}| < \sum_{i'=1}^{i} |B_{i'}|$ and no more addable trees can be added. Recall that the reduced instance we consider removes all vertices appearing in $A_1, B_1, \ldots, A_i, B_i, A_{i+1}$. By the bounds above we know that this removes a total of at most

$$(k/32)^j \cdot 3 \sum_{i'=1}^{i} |B_{i'}| \tag{3.17}$$

many vertices from each layer j. Now consider the optimal degree-k solution restricted to the sources in $\bigcup_{i'=1}^{i} B_{i'}$. Each of the arborescences in this solution must overlap with the removed vertices on some layer j on at least $(k/4)^j$ many vertices. Otherwise, by Lemma 3.22 there would be a degree-$k/4$ arborescence rooted in one of the sources of a blocking tree which is disjoint from the removed vertices. Consequently, our α-approximation would have found a degree-$k/4\alpha$ arborescence that would have been added as an addable

tree. Let R_j be the intersection of the aforementioned optimal solution with the removed vertices. Then

$$\sum_{j \geqslant 1} \frac{|R_j|}{(k/4)^j} \geqslant \sum_{i'=1}^{i} |B_{i'}| \,.$$

However, from (3.17) it follows that

$$\sum_{j \geqslant 1} \frac{|R_j|}{(k/4)^j} \leqslant \left(3 \sum_{i'=1}^{i} |B_{i'}|\right) \cdot \sum_{j \geqslant 1} \left(\frac{1}{8}\right)^j < \sum_{i'=1}^{i} |B_{i'}| \,,$$

a contradiction. □

From the lemmas above it follows that the local search never gets stuck: at any time it can either collapse or add a new (large) ring. It remains to show that it terminates in quasi-polynomially many steps. First we observe that the algorithm never has more than $O(\log n)$ many rings. By the previous two lemmas we have for every i that

$$\sum_{i'=1}^{i+1} |B_{i'}| = |B_{i+1}| + \sum_{i'=1}^{i} |B_{i'}| \geqslant |A_{i+1}| + \sum_{i'=1}^{i} |B_{i'}| \geqslant 2 \sum_{i'=1}^{i} |B_{i'}| \,.$$

Consequently, the number of blocking trees grows exponentially and the number of rings can be only logarithmic. Now consider the potential

$$(|B_1|, |B_2|, \ldots, |B_i|, \infty) \,,$$

where i is the number of rings. This potential decreases lexicographically with every collapse or newly added ring. Number of possible potentials is bounded by $n^{O(\log n)}$, which implies that the local search terminates in quasi-polynomial time.

3.6 APX-HARDNESS OF MAXMIN ARBORESCENCES

We finish this chapter by showing that the MaxMin arborescence problem is APX-hard, even in a layered graph with 2 layers and a single source (hence a stronger statement than Theorem 2.2). Our proof is based on a reduction from the max-k-cover problem, in which one is given a universe $\mathcal{U}$ of m elements, a family of subsets $S_1, S_2, \ldots, S_n$ of subsets of $\mathcal{U}$ and one has to cover a maximum number of elements of $\mathcal{U}$ using at most k sets. In a seminal result by Feige [38], it is showed that it is NP-hard to approximate max-k-cover within factor better than $\frac{e}{e-1}$. More specifically, it is showed that on instances where all sets have the same cardinality m/k, it is NP-hard to decide whether there exist k disjoint sets that cover the whole universe $\mathcal{U}$ (**YES** instance) or if any k sets cover at most $(1 - 1/e) \cdot m$ elements (**NO** instance).

Theorem 3.24. *For any $\varepsilon > 0$ there is no $(\sqrt{\frac{e}{e-1}} - \varepsilon)$-approximation algorithm to the single source MaxMin Arborescences problem on 2-layered graphs, unless P=NP.*

Proof. From a max-k-cover instance $(\mathcal{U}, S_1, S_2, \ldots, S_n)$ we make a layered instance of max-min arborescence as follows. In layer L_0 there is a single source s. In layer L_1, there are m/k^2 vertices $(v_{i,j})_{1 \leqslant j \leqslant m/k^2}$ for each set S_i. In layer L_2, there are m/k^2 sinks $(s_{a,j})_{1 \leqslant j \leqslant m/k^2}$ for each element $a \in \mathcal{U}$.

Then we connect the source s to all vertices in layer L_1. Each vertex $v_{i,j}$ in L_1 is connected to all sinks $s_{a,j}$ such that $a \in S_i$. Let us denote by OPT the optimum max-min degree of an arborescence on this instance.

Claim 3.25. *If the max-k-cover instance is a **YES** instance, then OPT $\geqslant m/k$. Otherwise, OPT $\leqslant \sqrt{1-1/e} \cdot m/k$.*

Proof. If the max-k-cover instance is a **YES** instance then for the source we select as neighbors in L_1 all the vertices $v_{i,j}$ such that S_i is selected in the k-cover solution. The source gets in this way $k \cdot m/k^2 = m/k$ outgoing neighbors. Each vertex $v_{i,j}$ that was selected then selects all the sinks $s_{a,j}$ such that $a \in S_i$. Since we are in a **YES** instance, all the selected sets are disjoint and each one receives exactly m/k sinks. This is a valid arborescence of value m/k.

In the **NO** case, consider any valid arborescence of value OPT. If there is a j such that the source selects more than $(1+\beta) \cdot k$ vertices (for any $\beta \geqslant 0$) in the set $V_j = (v_{i,j})_{1 \leqslant i \leqslant m}$ then this arborescence cannot have a maxmin degree more than

$$\max_{\beta \geqslant 0} \min \left\{ \frac{1 - 1/e + \beta}{1+\beta}, \frac{1}{1+\beta} \right\} \cdot \frac{m}{k} \leqslant \frac{1}{1+1/e} \cdot \frac{m}{k} \leqslant \left(\sqrt{1-1/e}\right) \cdot \frac{m}{k}$$

To see this, note that in a **NO** instance, any union of k sets contains at most $(1-1/e) \cdot m$ elements, hence any union of $(1+\beta) \cdot k$ sets contains at most $(1-1/e) \cdot m + \beta m$ elements (recall that each set has cardinality m/k), to be shared between $(1+\beta) \cdot k$ vertices. This gives the first ratio. The second comes from the fact that any union of some sets contains at most m elements. Optimizing over all $\beta \geqslant 0$ gives the upper-bound. Hence assume there is not such j.

If the source s has outdegree more than $\sqrt{1-1/e} \cdot m/k$ then it must be that there is a $j \in [1, m/k^2]$ such that s has more than

$$\frac{\sqrt{1-1/e} \cdot m/k}{m/k^2} = \sqrt{1-1/e} \cdot k$$

out-neighbors in the set $V_j = (v_{i,j})_{1 \leqslant i \leqslant m}$. By assumption, the source has also at most k out-neighbors in this set hence the number of sinks connected to vertices in V_j is at most $(1-1/e) \cdot m$ (recall we are in a **NO** case). Hence there must be a vertex $v_{i,j}$ that gets less than

$$\frac{(1-1/e) \cdot m}{\sqrt{1-1/e} \cdot k} = \left(\sqrt{1-1/e}\right) \cdot \frac{m}{k}$$

sinks as neighbors. In all cases, there must be a vertex in the arborescence that gets out-degree at most $(\sqrt{1-1/e}) \cdot \frac{m}{k}$, which ends the proof of the claim. □

By the previous claim, a $(\sqrt{\frac{e}{e-1}} - \varepsilon)$-approximation would be able to distinguish between the **YES** and **NO** instances of the max-k-cover problem, which is NP-hard. □

4

THE RESTRICTED SUBMODULAR SANTA CLAUS

This chapter contains the proof of Theorem 2.3, starting by an introduction to our techniques. This chapter is based on the publication "The Submodular Santa Claus in the Restricted Assignment Case", a joint work with Lars Rohwedder and Paritosh Garg that appeared at the *International Colloquium on Automata, Languages and Programming* (ICALP '21) [11]. Our way to the result is organised as follows. In Section 4.2, we first reduce our problem to a hypergraph matching problem (see next paragraph for a formal definition). We then solve this problem using Lovasz Local Lemma (LLL) in Section 4.3. In [14] the authors also reduce to a hypergraph matching problem which they then solve using LLL, although both parts are substantially simpler. The higher generality of our utility functions is reflected in the more general hypergraph matching problem. Namely, our problem is precisely the weighted variant of the (unweighted) problem in [14]. We will elaborate later why the previous techniques do not easily extend to the weighted variant.

4.1 OVERVIEW OF PREVIOUS TECHNIQUES AND OUR NEW IDEAS

After the reduction in Section 4.2 we arrive at the following problem. There is a hypergraph $\mathcal{H} = (P \cup R, \mathcal{C})$ with hyperedges $\mathcal{C}$ over the vertices P and R. We write $m = |P|$ and $n = |R|$. We will refer to hyperedges as configurations, the vertices in P as players and R as resources[1]. Moreover, a hypergraph is said to be regular if all vertices in P and R have the same degree, that is, they are contained in the same number of configurations. The hypergraph may contain multiple copies of the same configuration. Each configuration $C \in \mathcal{C}$ contains exactly one vertex in P, that is, $|C \cap P| = 1$. Additionally, for each configuration $C \in \mathcal{C}$ the resources $j \in C$ have weights $w_{j,C} \geqslant 0$. We emphasize that the same resource j can be given different weights in two different configurations, that is, we may have $w_{j,C} \neq w_{j,C'}$ for two different configurations C, C'.

We require to select for each player $i \in P$ one configuration C that contains i. For each configuration C that was selected we require to assign a subset of the resources in C which has a total weight of at least $(1/\alpha) \cdot \sum_{j \in C} w_{j,C}$ to the player in C. A resource can only be assigned to one player. We call such a solution an α-relaxed perfect matching. One seeks to minimize α.

We show that every regular hypergraph has an α-relaxed perfect matching for some $\alpha = O(\log \log(n))$ assuming that $w_{j,C} \leqslant (1/\alpha) \cdot \sum_{j' \in C} w_{j',C}$ for all j, C, that is, all weights are small compared to the total weight of the configuration. Moreover, we can find such a matching in randomized polynomial time. In the reduction we use this result to round a certain LP

1 We note that these do not have to be the same players and resources as in the Santa Claus problem we reduced from, but n and m do not increase.

relaxation and α essentially translates to the approximation rate. This result generalizes that of Bansal and Srividenko [14] on hypergraph matching in the following way. They proved the same result for unit weights and uniform hyperedges, that is, $w_{j,C} = 1$ for all j, C and all hyperedges have the same number of resources[2]. In the next paragraph we briefly go over the techniques to prove our result for the hypergraph matching problem.

OUR TECHNIQUES. Already the extension from uniform to non-uniform hypergraphs (assuming unit weights) is highly non-trivial and captures the core difficulty of our result. Indeed, we show with a (perhaps surprising) reduction, that we can reduce our weighted hypergraph matching problem to the unweighted (but non-uniform) version by introducing some bounded dependencies between the choices of the different players. For sake of brevity we therefore focus in this section on the unweighted non-uniform variant, that is, we need to assign to each player a configuration C and at least $|C|/\alpha$ resources in C. We show that for any regular hypergraph there exists such a matching for $\alpha = O(\log\log(n))$ assuming that all configurations contain at least α resources and we can find it in randomized polynomial time. Without the assumption of uniformity the problem becomes significantly more challenging. To see this, we lay out the techniques of Bansal and Srividenko that allowed them to solve the problem in the uniform case. We note that for $\alpha = O(\log(n))$ the statement is easy to prove: We select for each player i one of the configurations containing i uniformly at random. Then by standard concentration bounds each resource is contained in at most $O(\log(n))$ of the selected configurations with high probability. This implies that there is a fractional assignment of resources to configurations such that each of the selected configurations C receives $\lfloor |C|/O(\log(n)) \rfloor$ of the resources in C. By integrality of the bipartite matching polytope, there is also an integral assignment with this property.

To improve to $\alpha = O(\log\log(n))$ in the uniform case, Bansal and Srividenko proceed as follows. Let k be the size of each configuration. First they reduce the degree of each player and resource to $O(\log(n))$ using the argument above, but taking $O(\log(n))$ configurations for each player. Then they sample uniformly at random $O(n\log(n)/k)$ resources and drop all others. This is sensible, because they manage to prove the (perhaps surprising) fact that an α-relaxed perfect matching with respect to the smaller set of resources is still an $O(\alpha)$-relaxed perfect matching with respect to all resources with high probability (when assigning the dropped resources to the selected configurations appropriately). Indeed, the smaller instance is easier to solve: With high probability all configurations have size $O(\log(n))$ and this greatly reduces the dependencies between the bad events of the random experiment above (the event that a resource is contained in too many selected configurations). This allows them to apply Lovász Local Lemma (LLL) in order to show that with positive probability the experiment succeeds for $\alpha = O(\log\log(n))$.

It is not obvious how to extend this approach to non-uniform hypergraphs: Sampling a fixed fraction of the resources will either make the small configura-

2 In fact they get a slightly better ratio of $\alpha = O(\log\log(m)/\log\log\log(m))$.

tions empty—which makes it impossible to retain guarantees for the original instance—or it leaves the big configurations big—which fails to reduce the dependencies enough to apply LLL. Hence it requires new sophisticated ideas for non-uniform hypergraphs, which we describe next.

Suppose we are able to find a set $\mathcal{K} \subseteq \mathcal{C}$ of configurations (one for each player) such that for each $K \in \mathcal{K}$ the sum of intersections $|K \cap K'|$ with smaller configurations $K' \in \mathcal{K}$ is very small, say at most $|K|/2$. Then it is easy to derive a 2-relaxed perfect matching: We iterate over all $K \in \mathcal{K}$ from large to small and reassign all resources to K (possibly stealing them from the configuration that previously had them). In this process every configuration gets stolen at most $|K|/2$ of its resources, in particular, it keeps the other half. However, it is non-trivial to obtain a property like the one mentioned above. If we take a random configuration for each player, the dependencies of the intersections are too complex. To avoid this we invoke an advanced variant of the sampling approach where we construct not only one set of resources, but a hierarchy of resource sets $R_0 \supseteq \cdots \supseteq R_d$ by repeatedly dropping a fraction of resources from the previous set. We then formulate bad events based on the intersections of a configuration C with smaller configurations C', but we write it only considering a resource set R_k of convenient granularity (chosen based on the size of C'). In this way we formulate a number of bad events using various sets R_k. This succeeds in reducing the dependencies enough to apply LLL. Unfortunately, even with this new way of defining bad events, the guarantee that for each $K \in \mathcal{K}$ the sum of intersections $|K \cap K'|$ with smaller configurations $K' \in \mathcal{K}$ is at most $|K|/2$ is still too much to ask. We can only prove some weaker property which makes it more difficult to reconstruct a good solution from it. The reconstruction still starts from the biggest configurations and iterates to finish by including the smallest configurations but it requires a delicate induction where at each step, both the resource set expands and some new small configurations that were not considered before come into play.

A REMARK ON LOCAL SEARCH TECHNIQUES. In our proof, we focus here on an extension of the LLL technique of Bansal and Srividenko. However, another technique proved itself very successful for the Santa Claus problem in the restricted assignment case with a linear utility function. This is a local search technique discovered by Asadpour, Feige, and Saberi [6] who used it to give a non-constructive proof that the integrality gap of the configuration LP of Bansal and Srividenko is at most 4. One may wonder if this technique could also be extended to the submodular case as we did with LLL. Unfortunately, this seems problematic as the local search arguments heavily rely on amortizing different volumes of configurations (i.e., the sum of their resources' weights or the number of resources in the unweighted case). Amortizing the volumes of configurations works well, if each configuration has the same volume, which is the case for the problem derived from linear valuation functions, but not the one derived from submodular functions. If the volumes differ then the amortization arguments break and we believe this is a fundamental problem for this approach.

4.2 REDUCTION TO HYPERGRAPH MATCHING PROBLEM

In this section we give a reduction of the restricted submodular Santa Claus problem to the hypergraph matching problem. As a starting point we solve the configuration LP, a linear programming relaxation of our problem. The LP is constructed using a parameter T which denotes the value of its solution. The goal is to find the maximal T such that the LP is feasible. In the LP we have a variable $x_{i,C}$ for every player $i \in P$ and every configuration $C \in \mathcal{C}(i,T)$. The configurations $\mathcal{C}(i,T)$ are defined as the sets of resources $C \subseteq \Gamma_i$ such that $f(C) \geqslant T$. We require every player $i \in P$ to have at least one configuration and every resource $j \in R$ to be contained in at most one configuration.

$$\begin{aligned} \sum_{C \in \mathcal{C}(i,T)} x_{i,C} &\geqslant 1 && \text{for all } i \in P \\ \sum_{i \in P} \sum_{C \in \mathcal{C}(i,T): j \in C} x_{i,C} &\leqslant 1 && \text{for all } j \in R \\ x_{i,C} &\geqslant 0 && \text{for all } i \in P, C \in \mathcal{C}(i,T) \end{aligned}$$

Since this linear program has exponentially many variables, we cannot directly solve it in polynomial time. We will give a polynomial time constant approximation for it via its dual. This is similar to the linear variant in [14], but requires some more work. In their case they can reduce the problem to one where the separation problem of the dual can be solved in polynomial time. In our case even the separation problem can only be approximated. Nevertheless, this is sufficient to approximate the linear program in polynomial time.

Theorem 4.1. *The configuration LP of the Restricted Submodular Santa Claus problem can be approximated within a factor of* $(1 - 1/e)/2$ *in polynomial time.*

We defer the proof of this theorem to Appendix B.1.1. Given a solution x^* of the configuration LP we want to arrive at the hypergraph matching problem from the introduction such that an α-relaxed perfect matching of that problem corresponds to an $O(\alpha)$-approximate solution of the Restricted Submodular Santa Claus problem. Let T^* denote the value of the solution x^*. We will define a resource $j \in R$ as *fat* if $f(\{j\}) \geqslant T^*/(100\alpha)$.

Resources that are not fat are called *thin*. We call a configuration $C \in \mathcal{C}(i,T)$ thin, if it contains only thin resources and denote by $\mathcal{C}_t(i,T) \subseteq \mathcal{C}(i,T)$ the set of thin configurations. Intuitively in order to obtain an $O(\alpha)$-approximate solution, it suffices to give each player i either one fat resource $j \in \Gamma_i$ or a thin configuration $C \in \mathcal{C}_t(i, T^*/O(\alpha))$. For our next step towards the hypergraph problem we use a technique borrowed from Bansal and Srividenko [14]. This technique allows us to simplify the structure of the problem significantly using the solution of the configuration LP. Namely, one can find a partition of the players into clusters such that we only need to cover one player from each cluster with thin resources. All other players can then be covered by fat resources. Informally speaking, the following lemma

is proved by sampling configurations randomly according to a distribution derived in a non-trivial way from the configuration LP.

Lemma 4.2. *Let $\ell \geqslant 12\log(n)$. Given a solution of value T^* for the configuration LP in randomized polynomial time we can find a partition of the players into clusters $K_1 \cup \cdots \cup K_k \cup Q = P$ and multisets of configurations $\mathcal{C}_h \subseteq \bigcup_{i \in K_h} \mathcal{C}_t(i, T^*/5)$, $h = 1, \ldots, k$, such that*

1. *$|\mathcal{C}_h| = \ell$ for all $h = 1, \ldots, k$ and*
2. *Each small resource appears in at most ℓ configurations of $\bigcup_h \mathcal{C}_h$.*
3. *given any $i_1 \in K_1, i_2 \in K_2, \ldots, i_k \in K_k$ there is a matching of fat resources to players $P \setminus \{i_1, \ldots, i_k\}$ such that each of these players i gets a unique fat resource $j \in \Gamma_i$.*

The role of the players Q in the lemma above is that each one of them gets a fat resource for certain. The proof follows closely that in [14]. For completeness we include it in Appendix B.1.2. We are now ready to define the hypergraph matching instance. The vertices of our hypergraph are the clusters $K_1, \ldots, K_k$ and the thin resources. Let $\mathcal{C}_1, \ldots, \mathcal{C}_k$ be the multisets of configurations as in Lemma 4.2. For each K_h and $C \in \mathcal{C}_h$ there is a hyperedge containing K_h and all resources in C. Let $\{j_1, \ldots, j_m\} = C$ ordered arbitrarily, but consistently. Then we define the weights as normalized marginal gains of resources if they are taken in this order, that is,

$$\begin{aligned} w_{j_i,C} &= \frac{5}{T^*} f(\{j_i\} \mid \{j_1, \ldots, j_{i-1}\}) \\ &= \frac{5}{T^*} (f(\{j_1, \ldots, j_{i-1}, j_i\}) - f(\{j_1, \ldots, j_{i-1}\})) \,. \end{aligned}$$

This implies that $\sum_{j \in C} w_{j,C} \geqslant 5f(C)/T^* \geqslant 1$ for each $C \in \mathcal{C}_h$, $h = 1, \ldots, k$.

Lemma 4.3. *Given an α-relaxed perfect matching to the instance as described by the reduction, one can find in polynomial time an $O(\alpha)$-approximation to the instance of Restricted Submodular Santa Claus.*

Proof. The α-relaxed perfect matching implies that each cluster K_h gets some small resources C' where $C' \subseteq C$ for some $C \in \mathcal{C}_h$ and $\sum_{j \in C'} w_{j,C} \geqslant 1/\alpha$. By submodularity we have that $f(C') \geqslant T^*/(5\alpha)$. Therefore we can satisfy one player in each cluster using thin resources and by Lemma 4.2 all others using fat resources. □

The proof above is the most critical place in the paper where we make use of the submodularity of the valuation function f. We note that since all resources considered are thin resources we have, by submodularity of f, the assumption that

$$w_{j,C} \leqslant \frac{5}{T^*} f(\{j\}) \leqslant \frac{5}{T^*} \frac{T^*}{100\alpha} \leqslant \frac{5}{100\alpha} \sum_{j \in C} w_{j,C}$$

for all j, C such that $j \in C$. This means that the weights are all small enough, as promised in introduction. From now on, we will assume that

$\sum_{j \in C} w_{j,C} = 1$ for all configurations C. This is without loss of generality, since we can just rescale the weights inside each configuration. This does not hurt the property that all weights are small enough.

4.2.1 *Reduction to unweighted hypergraph matching*

Before proceeding to the solution of this hypergraph matching problem, we first give a reduction to an unweighted variant of the problem. We will then solve this unweighted variant in the next section. First, we note that we can assume that all the weights $w_{j,C}$ are powers of 2 by standard rounding arguments. This only loses a constant factor in the approximation rate. Second, we can assume that inside each configuration C, each resource has a weight that is at least a $1/(2n)$. Formally, we can assume that $\min_{j \in C} w_{j,C} \geqslant 1/(2n)$ for all $C \in \mathcal{C}$. If this is not the case for some $C \in \mathcal{C}$, simply delete from C all the resources that have a weight less than $1/(2n)$. By doing this, the total weight of C is only decreased by a factor $1/2$ since it looses in total at most a weight of $n \cdot (1/2n) = 1/2$. (Recall that we rescaled the weights so that $\sum_{j \in C} w_{j,C} = 1$).

Hence after these two operations, an α-relaxed perfect matching in the new hypergraph is still an $O(\alpha)$-relaxed perfect matching in the original hypergraph. From there we reduce to an unweighted variant of the matching problem. Note that each configuration contains resources of at most $\log(n)$ different possible weights (powers of 2 from $1/(2n)$ to $1/\alpha$). We create the following new unweighted hypergraph $\mathcal{H}' = (P' \cup R, \mathcal{C}')$. The resource set R remains unchanged. For each player $i \in P$, we create $\log(n)$ players, which later correspond each to a distinct weight. We will say that the players obtained from duplicating the original player form a *group*. For every configuration C containing player i in the hypergraph $\mathcal{H}$, we add a set $\mathcal{S}_C = \{C_1, \ldots, C_s, \ldots, C_{\log(n)}\}$ of configurations in $\mathcal{H}'$. C_s contains player i_s and all resources that are given a weight $2^{-(s+1)}$ in C. In this new hypergraph, the resources are not weighted. Note that if the hypergraph $\mathcal{H}$ is regular then $\mathcal{H}'$ is regular as well.

Additionally, for a group of player and a set of $\log(n)$ configurations (one for each player in the group), we say that this set of configurations is *consistent* if all the configurations selected are obtained from the same configuration in the original hypergraph $\mathcal{H}$ (i.e. the selected configurations all belong to $\mathcal{S}_C$ for some C in $\mathcal{H}$).

Formally, we focus of the following problem. Given the regular hypergraph $\mathcal{H}'$, we want to select, for each group of $\log(n)$ players, a consistent set of configurations $C_1, \ldots, C_s, \ldots, C_{\log(n)}$ and assign to each player i_s a subset of the resources in the corresponding configuration C_s so that i_s is assigned at least $\lfloor |C_s|/\alpha \rfloor$ resources. No resource can be assigned to more than one player. We refer to this assignment as a consistent α-relaxed perfect matching. Note that in the case where $|C_s|$ is small (e.g. of constant size) we are not required to assign any resource to player i_s.

We finish this section by formally proving that this reduction is sufficient for our purposes. More precisely, we show that we can easily transform a

consistent α-relaxed matching in $\mathcal{H}'$ into a good matching in the original hypergraph $\mathcal{H}$.

Lemma 4.4. *A consistent α-relaxed matching in $\mathcal{H}'$ induces a $O(\alpha)$-relaxed matching in $\mathcal{H}$.*

Proof. Let us consider a group of $\log(n)$ players $i_1, \ldots, i_s, \ldots, i_{\log(n)}$ in $\mathcal{H}'$ corresponding to a player i in $\mathcal{H}$. These players are assigned a consistent set of configurations $C_1, \ldots, C_s, \ldots, C_{\log(n)}$ that correspond to a partition of a configuration in $\mathcal{H}$. Moreover, each player i_s is assigned $\lfloor |C_s|/\alpha \rfloor$ resources from C_s. We have two cases. If $|C_s| \geqslant \alpha$ then we have that i_s is assigned at least

$$\lfloor |C_s|/\alpha \rfloor \geqslant |C_s|/(2\alpha)$$

resources from C_s. On the other hand, if $\lfloor |C_s|/\alpha \rfloor = 0$ then the player i_s might not be assigned anything. However, we claim that that the configurations C_s of cardinality less than α can represent at most a $1/5$ fraction of the total weight of the configuration C in the original weighted hypergraph. To see this note that the total weight they represent is upper bounded by

$$\alpha \left(\sum_{k=\log(100\alpha/5)}^{\infty} \frac{1}{2^k} \right) = \alpha \left(\frac{5}{100\alpha} \sum_{k=0}^{\infty} \frac{1}{2^k} \right) \leqslant \frac{10}{100} = \frac{1}{10} \sum_{j \in C} w_{j,C}.$$

Hence, the consistent α-relaxed matching in $\mathcal{H}'$ induces in a straightforward way a matching in $\mathcal{H}$ where every player gets at least a fraction $1/(2\alpha) \cdot (1 - 1/10) \geqslant 1/(3\alpha)$ of the total weight of the appropriate configuration. This means that the consistent α-relaxed perfect matching in $\mathcal{H}'$ is indeed a (3α)-relaxed perfect matching in $\mathcal{H}$. □

4.3 MATCHINGS IN REGULAR HYPERGRAPHS

In this section we solve the hypergraph matching problem we arrived to in the previous section. For convenience, we give a self contained definition of the problem before formulating and proving our result.

INPUT: We are given $\mathcal{H} = (P \cup R, \mathcal{C})$ a hypergraph with hyperedges $\mathcal{C}$ over the vertices P (players) and R (resources) with $m = |P|$ and $n = |R|$. As in previous sections, we will refer to hyperedges as configurations. Each configuration $C \in \mathcal{C}$ contains exactly one vertex in P, that is, $|C \cap P| = 1$. The set of players is partitioned into groups of size at most $\log(n)$, we will use A to denote a group. These groups are disjoint and contain all players. Finally there exists an integer ℓ such that for each group A there are ℓ consistent sets of configurations. A consistent set of configurations for a group A is a set of $|A|$ configurations such that all players in the group appear in exactly one of these configurations. We will denote by $\mathcal{S}_A$ such a set and for a player $i \in A$, we will denote by $\mathcal{S}_A^{(i)}$ the unique configuration in $\mathcal{S}_A$ containing i. Finally, no resource appears in more than ℓ configurations. We say that the hypergraph is regular (although some resources may appear in less than ℓ configurations).

OUTPUT: We wish to select a matching that covers all players in P. More precisely, for each group A we want to select a consistent set of configurations (denoted by $\{\mathcal{S}_A^{(i)}\}_{i\in A}$). Then for each player $i \in A$, we wish to assign a subset of the resources in $\mathcal{S}_A^{(i)}$ to the player i such that:

1. No resource is assigned to more than one player in total.
2. For any group A and any player $i \in A$, player i is assigned at least $\lfloor|\mathcal{S}_A^{(i)}|/\alpha\rfloor$ resources from $\mathcal{S}_A^{(i)}$.

We call this a consistent α-relaxed perfect matching. Our goal in this section will be to prove the following theorem.

Theorem 4.5. *Let $\mathcal{H} = (P \cup R, \mathcal{C})$ be a regular (non-uniform) hypergraph where the set of players is partitioned into groups of size at most $\log(n)$. Then we can, in randomized polynomial time, compute a consistent α-relaxed perfect matching for $\alpha = O(\log\log(n))$.*

We note that Theorem 4.5 together with the reduction from the previous section will prove our main result (Theorem 2.3) stated in introduction.

4.3.1 *Overview and notations*

To prove Theorem 4.5, we introduce the following notations. Let $\ell \in \mathbb{N}$ be the regularity parameter as described in the problem input (i.e. each group has ℓ consistent sets and each resource appears in no more than ℓ configurations). As we proved in Lemma 4.2 we can assume with standard sampling arguments that $\ell = 300.000\log^3(n)$ at a constant loss. If this is not the case because we might want to solve the hypergraph matching problem by itself (i.e. not obtained by the reduction in Section 4.2), the proof of Lemma 4.2 can be repeated in a very similar way here.

For a configuration C, its size will be defined as $|C \cap R|$ (i.e. its cardinality over the resource set). For each player i, we denote by $\mathcal{C}_i$ the set of configurations that contain i. We now group the configurations in $\mathcal{C}_i$ by size: We denote by $\mathcal{C}_i^{(0)}$ the configurations of size in $[0, \ell^4)$ and for $k \geqslant 1$ we write $\mathcal{C}_i^{(k)}$ for the configurations of size in $[\ell^{k+3}, \ell^{k+4})$. Moreover, define $\mathcal{C}^{(k)} = \bigcup_i \mathcal{C}_i^{(k)}$ and $\mathcal{C}^{(\geqslant k)} = \bigcup_{h\geqslant k} \mathcal{C}^{(h)}$. Let d be the smallest number such that $\mathcal{C}^{(\geqslant d)}$ is empty. Note that $d \leqslant \log(n)/\log(\ell)$. Now consider the following random process.

Random Experiment 4.6. *We construct a nested sequence of resource sets $R = R_0 \supseteq R_1 \supseteq \ldots \supseteq R_d$ as follows. Each R_k is obtained from R_{k-1} by deleting every resource in R_{k-1} independently with probability $(\ell-1)/\ell$.*

In expectation only a $1/\ell$ fraction of resources in R_{k-1} survives in R_k. Also notice that for $C \in \mathcal{C}^{(k)}$ we have that $\mathbb{E}[|R_k \cap C|] = \text{poly}(\ell)$.

The proof of Theorem 4.5 is organized as follows. In Section 4.3.2, we give some properties of the resource sets constructed by Random Experiment 4.6 that hold with high probability. Then in Section 4.3.3, we show that we can find a single consistent set of configurations for each group of players such that for each configuration selected, its intersection with smaller selected configurations is bounded if we restrict the resource set to an appropriate

R_k. Restricting the resource set is important to bound the dependencies of bad events in order to apply Lovasz Local Lemma. Finally in Section 4.3.4, we demonstrate how these configurations allow us to reconstruct a consistent α-relaxed perfect matching for an appropriate assignment of resources to configurations.

4.3.2 *Properties of resource sets*

In this subsection, we give a precise statement of the key properties that we need from Random Experiment 4.6. The first two lemmas have a straightforward proof. The last one is a generalization of an argument used by Bansal and Srividenko [14]. Since the proof is more technical and tedious, we also defer it to Appendix B.2 along with the proof of the first two statements.

We start with the first property which bounds the size of the configurations when restricted to some R_k. This property is useful to reduce the dependencies while applying LLL later.

Lemma 4.7. *Consider Random Experiment 4.6 with $\ell \geqslant 300.000 \log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathcal{C}^{(\geqslant k)}$ we have*

$$\frac{1}{2}\ell^{-k}|C| \leqslant |R_k \cap C| \leqslant \frac{3}{2}\ell^{-k}|C|$$

with probability at least $1 - 1/n^{10}$.

The next property expresses that for any configuration the sum of intersections with configurations of a particular size does not deviate much from its expectation. In particular, for any configuration C, the sum of it's intersections with other configurations is at most $|C|\ell$ as each resource is in at most ℓ configurations. By the lemma stated below, we recover this up to a multiplicative constant factor when we consider the appropriately weighted sum of the intersection of C with other configurations C' of smaller sizes where each configuration $C' \in \mathcal{C}^{(k)}$ is restricted to the resource set R_k.

Lemma 4.8. *Consider Random Experiment 4.6 with $\ell \geqslant 300.000 \log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathcal{C}^{(\geqslant k)}$ we have*

$$\sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C \cap R_k| \leqslant \frac{10}{\ell^k}\left(|C| + \sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C|\right)$$

with probability at least $1 - 1/n^{10}$.

We now define the notion of *good* solutions which is helpful in stating our last property. Let $\mathcal{F}$ be a set of configurations, $\alpha : \mathcal{F} \to \mathbb{N}$, $\gamma \in \mathbb{N}$, and $R' \subseteq R$. We say that an assignment of R' to $\mathcal{F}$ is (α, γ)-good if every configuration $C \in \mathcal{F}$ receives at least $\alpha(C)$ resources of $C \cap R'$ and if no resource in R' is assigned more than γ times in total.

Below we obtain that given a (α, γ)-good solution with respect to resource set R_{k+1}, one can construct an almost $(\ell \cdot \alpha, \gamma)$-good solution with respect to the bigger resource set R_k. Informally, starting from a good solution with respect to the final resource set and iteratively applying this lemma would give us a good solution with respect to our complete set of resources.

Lemma 4.9. *Consider Random Experiment 4.6 with $\ell \geqslant 300.000 \log^3(n)$. Fix $k \geqslant 0$. Conditioned on the event that the bounds in Lemma 4.7 hold for k, then with probability at least $1 - 1/n^{10}$ the following holds for all $\mathcal{F} \subseteq \mathcal{C}^{(\geqslant k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$, and $\gamma \in \mathbb{N}$ such that $\ell^3/1000 \leqslant \alpha(C) \leqslant n$ for all $C \in \mathcal{F}$ and $\gamma \in \{1, \dots, \ell\}$: If there is a (α, γ)-good assignment of R_{k+1} to $\mathcal{F}$, then there is a (α', γ)-good assignment of R_k to $\mathcal{F}$ where*

$$\alpha'(C) \geqslant \ell \left(1 - \frac{1}{\log(n)}\right) \alpha(C)$$

for all $C \in \mathcal{F}$. Moreover, this assignment can be found in polynomial time.

Given the lemmata above, by a simple union bound one gets that all the properties of resource sets hold.

4.3.3 *Selection of configurations*

In this subsection, we give a random process that selects one consistent set of configurations for each group of players such that the intersection of the selected configurations with smaller configurations is bounded when considered on appropriate sets R_k. We will denote $\mathcal{S}_A$ the selected consistent set for group A and for ease of notation we will denote $K_i = \mathcal{S}_A^{(i)}$ the selected configuration for player $i \in A$. For any integer k, we write $\mathcal{K}_i^{(k)} = \{K_i\}$ if $K_i \in \mathcal{C}_i^{(k)}$ and $\mathcal{K}_i^{(k)} = \emptyset$ otherwise. As for the configuration set, we will also denote $\mathcal{K}^{(k)} = \bigcup_i \mathcal{K}_i^{(k)}$ and $\mathcal{K} = \bigcup_k \mathcal{K}^{(k)}$. The following lemma describes what are the properties we want to have while selecting the configurations. For better clarity we also recall what the properties of the sets $R_0, \dots, R_d$ that we need are. These hold with high probability by the lemmata of the previous section.

Lemma 4.10. *Let $R = R_0 \supseteq \dots \supseteq R_d$ be sets of fewer and fewer resources. Assume that for each k and $C \in \mathcal{C}_i^{(k)}$ we have*

$$1/2 \cdot \ell^{k-h} \leqslant |C \cap R_h| \leqslant 3/2 \cdot \ell^{-h}|C| < 3/2 \cdot \ell^{k-h+4}$$

for all $h = 0, \dots, k$. Then there exists a selection of one consistent set $\mathcal{S}_A$ for each group A such for all $k = 0, \dots, d$, $C \in \mathcal{C}^{(k)}$ and $j = 0, \dots, k$ then we have

$$\sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \leqslant \frac{1}{\ell} \sum_{j \leqslant h \leqslant k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + 1000 \frac{d + \ell}{\ell} \log(\ell)|C|.$$

Moreover, this selection of consistent sets can be found in polynomial time.

Before we prove this lemma, we give an intuition of the statement. Consider the sets $R_1, \dots, R_d$ constructed as in Random Experiment 4.6. Then for $C' \in \mathcal{C}^{(h)}$ we have $\mathbb{E}[\ell^h |C' \cap C \cap R_h|] = |C' \cap C|$. Hence

$$\sum_{h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} |K \cap C| = \mathbb{E}[\sum_{h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h|]$$

Similarly for the right-hand side we have

$$\mathbb{E}[\frac{1}{\ell}\sum_{j\leqslant h\leqslant k}\sum_{C'\in\mathcal{C}^{(h)}}\ell^h|C'\cap C\cap R_h|+O(\frac{d+\ell}{\ell}\log(\ell)|C|)]$$
$$=\frac{1}{\ell}\underbrace{\sum_{j\leqslant h\leqslant k}\sum_{C'\in\mathcal{C}^{(h)}}|C'\cap C|}_{\leqslant\ell|C|}+O\left(\frac{d+\ell}{\ell}\log(\ell)|C|\right)$$
$$=O\left(\frac{d+\ell}{\ell}\log(\ell)|C|\right).$$

Hence, the lemma says that each resource in C is roughly covered $O((d+\ell)/\ell\cdot\log(\ell))$ times by smaller configurations.

We now proceed to the proof of Lemma 4.10.

Proof. We perform the following random experiment and show with LLL that there is a positive probability of success.

Random Experiment 4.11. *For each group A, select one consistent set $\mathcal{S}_A$ uniformly at random. Then for each player $i\in A$ set $K_i=\mathcal{S}_A^{(i)}$.*

Given this experiment we can define the following random variables. For all $h=0,\dots,d$ and $i\in P$ we define

$$X_{i,C}^{(h)}=\sum_{K\in\mathcal{K}_i^{(h)}}|K\cap C\cap R_h|\leqslant\min\{3/2\cdot\ell^4,|C\cap R_h|\}.$$

Let $X_C^{(h)}=\sum_{i=1}^m X_{i,C}^{(h)}$. Then

$$\mathbb{E}[X_C^{(h)}]\leqslant\frac{1}{\ell}\sum_{C'\in\mathcal{C}^{(h)}}|C'\cap C\cap R_h|\leqslant|C\cap R_h|.$$

We are now ready to define the bad events on which we will apply the Lovasz Local Lemma. As we will show later, if none of them occur, Lemma 4.10 will hold. For each k, $C\in\mathcal{C}^{(k)}$, and $h\leqslant k$ let $B_C^{(h)}$ be the event that

$$X_C^{(h)}\geqslant\begin{cases}\mathbb{E}[X_C^{(h)}]+63|C\cap R_h|\log(\ell) & \text{if } k-5\leqslant h\leqslant k,\\ \mathbb{E}[X_C^{(h)}]+135|C\cap R_h|\log(\ell)\cdot\ell^{-1} & \text{if } h\leqslant k-6.\end{cases}$$

The intuitive reason as to why we define these two different bad events can be summarized as follows. In the case $h\leqslant k-6$, we are counting how many times C is intersected by configurations that are much smaller than C. Hence the size of this intersection can be written as a sum of independent random variables of value at most $O(\ell^4)$ which is much smaller than the total size of the configuration $|C\cap R_h|$. Since the random variables are in a much smaller range, Chernoff bounds give much better concentration guarantees and we can afford a very small deviation from the expectation. In the other case, we do not have this property hence we need a bigger deviation to maintain a sufficiently low probability of failure. However, this does not hurt the statement of Lemma 4.10 since we sum this bigger deviation only

a constant number of times. One key idea to be able to apply Lovasz Local Lemma here is also to consider intersection of C with smaller configurations but restricted to a set R_h of convenient granularity. One can notice that $|C' \cap R_h| = \text{poly}(\ell)$ if $C' \in \mathcal{C}^{(h)}$ (by the assumption made in Lemma 4.10). This allows to reduce significantly the dependencies between bad events which is crucial to make any use of LLL here.

With this in mind, we claim that the probability of each bad event happening is small.

Claim 4.12. *For each k, $C \in \mathcal{C}^{(k)}$, and $h \leqslant k$ we have*

$$\mathbb{P}[B_C^{(h)}] \leqslant \exp\left(-2\frac{|C \cap R_h|}{\ell^9} - 18\log(\ell)\right).$$

Proof. Consider first the case that $h \geqslant k - 5$. By a Chernoff bound (see full version for the precise formulation) with

$$\delta = 63\frac{|C \cap R_h|\log(\ell)}{\mathbb{E}[X_C^{(h)}]} \geqslant 1$$

we get

$$\begin{aligned}\mathbb{P}[B_C^{(h)}] \leqslant \exp\left(-\frac{\delta\mathbb{E}[X_C^{(h)}]}{3|C \cap R_h|}\right) &\leqslant \exp(-21\log(\ell))) \\ &\leqslant \exp\Bigg(-2\underbrace{\frac{|C \cap R_h|}{\ell^9}}_{\leqslant 3/2} - 18\log(\ell)\Bigg).\end{aligned}$$

Now consider $h \leqslant k - 6$. We apply again a Chernoff bound with

$$\delta = 135\frac{|C \cap R_h|\log(\ell)}{\ell\mathbb{E}[X_C^{(h)}]} \geqslant \frac{1}{\ell}.$$

This implies

$$\begin{aligned}\mathbb{P}[B_C^{(h)}] &\leqslant \exp\left(-\frac{\min\{\delta, \delta^2\}\mathbb{E}[X_C^{(h)}]}{3 \cdot 3/2 \cdot \ell^4}\right) \\ &\leqslant \exp\left(-30\frac{|C \cap R_h|\log(\ell)}{\ell^6}\right) \\ &\leqslant \exp\left(-2\frac{|C \cap R_h|}{\ell^9} - 18\log(\ell)\right).\end{aligned}$$

□

We are now ready to instanciate the Lovász Local Lemma and use it in our setting. Let $k \in \{0, \ldots, d\}$, $C \in \mathcal{C}^{(k)}$ and $h \leqslant k$. For event $B_C^{(h)}$ we set

$$x(B_C^{(h)}) = \exp(-|C \cap R_h|/\ell^9 - 18\log(\ell)).$$

We now analyze the dependencies of $B_C^{(h)}$. The event depends only on random variables $\mathcal{S}_A$ for groups A that contain at least one player i that has

a configuration in $\mathcal{C}_i^{(h)}$ which overlaps with $C \cap R_h$. The number of such configurations (in particular, of such groups) is at most $\ell|C \cap R_h|$ since the hypergraph is regular.

In each of these groups, we count at most $\log(n)$ players, each having ℓ configurations hence in total at most $\ell \cdot \log(n)$ configurations.

Each configuration $C' \in \mathcal{C}^{(h')}$ can only influence those events $B_{C''}^{(h')}$ where $C' \cap C'' \cap R_{h'} \neq \emptyset$. Since $|C' \cap R_{h'}| \leqslant 3/2 \cdot \ell^4$ and since each resource appears in at most ℓ configurations, we see that each configuration can influence at most $3/2 \cdot \ell^5$ events.

Putting everything together, we see that the bad event $B_C^{(h)}$ is independent of all but at most

$$(\ell|C \cap R_h|) \cdot (\ell \cdot \log(n)) \cdot (3/2 \cdot \ell^5) = 3/2 \cdot \ell^7 \cdot \log(n)|C \cap R_h| \leqslant |C \cap R_h|\ell^8$$

other bad events.

We can now verify the condition for LLL (see Lemma 2.5) by calculating

$$\begin{aligned} x(B_C^{(h)}) & \prod_{(B_C^{(h)}, B_{C'}^{(h')}) \in E} (1 - x(B_{C'}^{(h')})) \\ & \geqslant \exp(-|C \cap R_h|/\ell^9 - 18\log(\ell)) \cdot (1 - \ell^{-18})^{|C \cap R_h|\ell^8} \\ & \geqslant \exp(-|C \cap R_h|/\ell^9 - 18\log(\ell)) \cdot \exp(-|C \cap R_h|/\ell^9) \\ & \geqslant \exp(-2|C \cap R_h|/\ell^9 - 18\log(\ell)) \geqslant \mathbb{P}[B_C^{(h)}]. \end{aligned}$$

By LLL we have that with positive probability none of the bad events happen. Let $k \in \{0, \dots, d\}$ and $C \in \mathcal{C}^{(k)}$. Then for $k - 5 \leqslant h \leqslant k$ we have

$$\ell^h X_C^{(h)} \leqslant \ell^h \mathbb{E}[X_C^{(h)}] + 63\ell^h |C \cap R_h| \log(\ell) \leqslant \ell^h \mathbb{E}[X_C^{(h)}] + 95|C| \log(\ell).$$

Moreover, for $h \leqslant k - 6$ it holds that

$$\ell^h X_C^{(h)} \leqslant \ell^h \mathbb{E}[X_C^{(h)}] + 135\ell^{h-1}|C \cap R_h| \log(\ell) \leqslant \ell^h \mathbb{E}[X_C^{(h)}] + 203|C| \log(\ell) \cdot \ell^{-1}.$$

We conclude that, for any $0 \leqslant j \leqslant k$,

$$\begin{aligned} & \sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \\ & \qquad \leqslant \sum_{j \leqslant h \leqslant k} \ell^h \mathbb{E}[X_C^{(h)}] + 1000 \frac{(k - j + 1) + \ell}{\ell} |C| \log(\ell) \\ & \qquad \leqslant \frac{1}{\ell} \sum_{j \leqslant h \leqslant k} \ell^h \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C \cap R_h| + 1000 \frac{d + \ell}{\ell} |C| \log(\ell). \end{aligned}$$

This proves Lemma 4.10. □

Remark 4.13. *Since there are at most* $\mathrm{poly}(n, m, \ell)$ *bad events and each bad event* B *has* $\frac{x(B)}{1 - x(B)} \leqslant 1/2$ *(because* $x(B) \leqslant \ell^{-18}$*), the constructive variant of LLL by Moser and Tardos [70] can be applied to find a selection of configurations such that no bad events occur in randomized polynomial time.*

4.3.4 *Assignment of resources to configurations*

In this subsection, we show how all the previously established properties allow us to find, in polynomial time, a good assignment of resources to the configurations $\mathcal{K}$ chosen as in the previous subsection. We will denote as in the previous subsection $\mathcal{K}_i^{(k)} = \{K_i\}$ if $K_i \in \mathcal{C}_i^{(k)}$ and $\mathcal{K}_i^{(k)} = \emptyset$ otherwise. We also define $\mathcal{K}^{(k)} = \bigcup_i \mathcal{K}_i^{(k)}$ and $\mathcal{K}^{(\geqslant k)} = \bigcup_{h \geqslant k} \mathcal{K}^{(k)}$. Finally we define the parameter

$$\gamma = 100.000 \frac{d+\ell}{\ell} \log(\ell),$$

which will define how many times each resource can be assigned to configurations in an intermediate solution. Note that $d \leqslant \log(n)/\log(\ell)$. By our choice of $\ell = 300.000 \log^3(n)$, we have that $\gamma \leqslant 310.000 \log\log(n)$. Lemma 4.10 implies the following bound.

Claim 4.14. *For any $k \geqslant 0$, any $0 \leqslant j \leqslant k$, and any $C \in \mathcal{K}^{(k)}$*

$$\sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \leqslant 2000 \frac{d+\ell}{\ell} \log(\ell) |C|$$

Proof. By Lemma 4.10 we have that

$$\begin{aligned} &\sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \\ &\qquad \leqslant \frac{1}{\ell} \sum_{j \leqslant h \leqslant k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + 1000 \frac{d+\ell}{\ell} \log(\ell) |C| \,. \end{aligned}$$

Furthermore, by Lemma 4.8, we get

$$\sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| \leqslant \ell^h \frac{10}{\ell^h} \left(|C| + \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C| \right).$$

Finally note that each resource appears in at most ℓ configurations, hence

$$\sum_{j \leqslant h \leqslant k} \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C| \leqslant \ell |C|.$$

Putting everything together we conclude

$$\begin{aligned} &\sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^h |K \cap C \cap R_h| \\ &\qquad \leqslant \frac{1}{\ell} \sum_{j \leqslant h \leqslant k} \sum_{C' \in \mathcal{C}^{(h)}} \ell^h |C' \cap C \cap R_h| + 1000 \frac{d+\ell}{\ell} \log(\ell) |C| \\ &\qquad \leqslant \frac{1}{\ell} \sum_{j \leqslant h \leqslant k} 10 \left(|C| + \sum_{C' \in \mathcal{C}^{(h)}} |C' \cap C| \right) + 1000 \frac{d+\ell}{\ell} \log(\ell) |C| \\ &\qquad \leqslant \frac{k-j}{\ell} 10 |C| + 10 |C| + 1000 \frac{d+\ell}{\ell} \log(\ell) |C| \\ &\qquad \leqslant 20 |C| + 1000 \frac{d+\ell}{\ell} \log(\ell) |C| \\ &\qquad \leqslant 2000 \frac{d+\ell}{\ell} \log(\ell) |C|. \end{aligned}$$

□

We can now proceed to the main technical part of this section which is the following lemma proved by induction.

Lemma 4.15. *For any $j \geqslant 0$, there exists an assignment of resources of R_j to configurations in $\mathcal{K}^{(\geqslant j)}$ such that no resource is taken more than γ times and each configuration $C \in \mathcal{K}^{(k)}$ ($k \geqslant j$) receives at least*

$$\left(1 - \frac{1}{\log(n)}\right)^{2(k-j)} \ell^{k-j}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h|$$

resources from R_k.

Before going through the proof, we give here the intuition of why this is what we want to prove. Note that the term $\ell^{k-j}|C \cap R_k|$ is roughly equal to $\ell^{-j}|C|$ by the properties of the resource sets (precisely Lemma 4.7). The second term

$$\sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h|$$

can be shown to be

$$O\left(\ell^{-j} \frac{d+\ell}{\ell} \log(\ell)|C|\right) = O(\ell^{-j} \log\log(n)|C|)$$

by Claim 4.14. Hence by choosing γ to be $\Theta(\log\log(n))$ we get that the bound in Lemma 4.15 will be $\Theta(\ell^{-j}|C|)$. At the end of the induction, we have $j = 0$ which indeed implies that we have an assignment in which configurations receive

$$\Theta(\ell^{-0}|C|) = \Theta(|C|)$$

resources and such that each resource is assigned to at most $O(\log\log(n))$ configurations. With this in mind, we give the formal proof of Lemma 4.15.

Proof. We start from the biggest configurations and then iteratively reconstruct a good solution for smaller and smaller configurations. Recall d is the smallest integer such that $\mathcal{K}^{(\geqslant d)}$ is empty. Our base case for these configurations in $\mathcal{K}^{(\geqslant d)}$ is vacuously satisfied.

Now assume that we have a solution at level j, i.e. an assignment of resources to configurations in $\mathcal{K}^{(\geqslant j)}$ such that no resource is taken more than γ times and each configuration $C \in \mathcal{K}^{(k)}$ such that $k \geqslant j$ receives at least

$$\left(1 - \frac{1}{\log(n)}\right)^{2(k-j)} \ell^{k-j}|C \cap R_k| - \frac{3}{\gamma} \sum_{j \leqslant h \leqslant k} \sum_{K \in \mathcal{K}^{(h)}} \ell^{h-j}|K \cap C \cap R_h|$$

resources from R_j. We show that this implies a solution at level $j-1$ in the following way. First by Lemma 4.9, this implies an assignment of resources of R_{j-1} to configurations in $\mathcal{K}^{(\geqslant j)}$ such that each $C \in \mathcal{K}^{(k)}$ receives at least

$$\begin{aligned}
&\left(1-\frac{1}{\log(n)}\right)\ell\Bigg[(\ell^{k-j}\left(1-\frac{1}{\log(n)}\right)^{2(k-j)}|C\cap R_k| \\
&\qquad -\frac{3}{\gamma}\sum_{j\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-j}|K\cap C\cap R_h|\Bigg] \\
&=\left(1-\frac{1}{\log(n)}\right)^{2(k-(j-1))-1}\ell^{k-(j-1)}|C\cap R_k| \\
&\qquad -\frac{3}{\gamma}\left(1-\frac{1}{\log(n)}\right)\sum_{j\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-(j-1)}|K\cap C\cap R_h| \\
&\geqslant\left(1-\frac{1}{\log(n)}\right)^{2(k-(j-1))-1}\ell^{k-(j-1)}|C\cap R_k| \\
&\qquad -\frac{3}{\gamma}\sum_{j\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-(j-1)}|K\cap C\cap R_h|
\end{aligned}$$

resources and no resource of R_{j-1} is taken more than γ times. Note that we can apply Lemma 4.9 since we have by Claim 4.14 and Lemma 4.7

$$\begin{aligned}
&\left(1-\frac{1}{\log(n)}\right)^{2(k-j)}\ell^{k-j}|C\cap R_k| \\
&\qquad -\frac{3}{\gamma}\sum_{j\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-j}|K\cap C\cap R_h| \\
&\geqslant\frac{\ell^{k-j}}{e^2}|C\cap R_k|-\frac{3}{\gamma}2000\ell^{-j}\frac{d+\ell}{\ell}\log(\ell)|C| \\
&\geqslant\ell^{-j}|C|\left(\frac{1}{2e^2}-\frac{6000}{\gamma}\frac{d+\ell}{\ell}\log(\ell)\right) \\
&\geqslant\frac{\ell^{-j}|C|}{3e^2}>\frac{\ell^3}{1000}\,.
\end{aligned}$$

Now consider configurations in $\mathcal{K}^{(j-1)}$ and proceed for them as follows. Give to each $C\in\mathcal{K}^{(j-1)}$ all the resources in $C\cap R_{j-1}$ except all the resources that appear in more than γ configurations in $\mathcal{K}^{(j-1)}$. Since each deleted resource is counted at least γ times in the sum $\sum_{K\in\mathcal{K}^{(j-1)}}|K\cap C\cap R_{j-1}|$, we have that each configuration C in $\mathcal{K}^{(j-1)}$ receives at least

$$|C\cap R_{j-1}|-\frac{1}{\gamma}\sum_{K\in\mathcal{K}^{(j-1)}}|K\cap C\cap R_{j-1}|$$

resources and no resource is taken more than γ times by configurations in $\mathcal{K}^{(j-1)}$. Notice that now every resource is taken no more than γ times by configurations in $\mathcal{K}^{(\geqslant j)}$ and no more than γ times by configurations in $\mathcal{K}^{(j-1)}$ which in total can sum up to 2γ times.

Therefore to finish the proof consider an resource $i\in R_{j-1}$. This resource is taken b_i times by configurations in $\mathcal{K}^{(\geqslant j)}$ and a_i times by configurations

in $\mathcal{K}^{(j-1)}$. If $a_i + b_i \leqslant \gamma$, nothing needs to be done. Otherwise, denote by O the set of problematic resources (i.e. resources i such that $a_i + b_i > \gamma$). For every $i \in O$, select uniformly at random $a_i + b_i - \gamma$ configurations in $\mathcal{K}^{(\geqslant j)}$ that currently contain resource i and delete the resource from these configurations. When this happens, each configuration in $C \in \mathcal{K}^{(\geqslant j)}$ that contains i has a probability of $(a_i + b_i - \gamma)/b_i$ to be selected to loose this resource. Hence the expected number of resources that C looses with such a process is

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i}$$

It is not difficult to prove the following claim.

Claim 4.16. *For any $C \in \mathcal{K}^{(\geqslant j)}$,*

$$\frac{1}{\gamma^2} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leqslant \mu \leqslant \frac{2}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|$$

Proof. Note that we can write

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i} \leqslant \max_{i \in O \cap C} \left\{ \frac{a_i + b_i - \gamma}{a_i b_i} \right\} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

The reason for this is that each resource i accounts for an expected loss of $(a_i + b_i - \gamma)/b_i$ while it is counted a_i times in the sum

$$\sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

Similarly,

$$\mu = \sum_{i \in O \cap C} \frac{a_i + b_i - \gamma}{b_i} \geqslant \min_{i \in O \cap C} \left\{ \frac{a_i + b_i - \gamma}{a_i b_i} \right\} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|.$$

Note that by assumption we have that $a_i + b_i > \gamma$. This implies that either a_i or b_i is greater than $\gamma/2$. Assume w.l.o.g. that $a_i \geqslant \gamma/2$. Since by assumption $a_i \leqslant \gamma$ we have that

$$\frac{a_i + b_i - \gamma}{a_i b_i} \leqslant \frac{b_i}{a_i b_i} = \frac{1}{a_i} \leqslant \frac{2}{\gamma}.$$

In the same manner, since $a_i + b_i > \gamma$ and that $a_i, b_i \leqslant \gamma$, we can write

$$\frac{a_i + b_i - \gamma}{a_i b_i} \geqslant \frac{1}{a_i b_i} \geqslant \frac{1}{\gamma^2}.$$

We therefore get the following bounds

$$\frac{1}{\gamma^2} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O| \leqslant \mu \leqslant \frac{2}{\gamma} \sum_{K \in \mathcal{K}^{(j-1)}} |K \cap C \cap R_{j-1} \cap O|,$$

which is what we wanted to prove. □

Assume then that $\mu \leqslant \frac{|C\cap R_k|}{10^{12}\log^3(n)}$. Note that C cannot loose more than $\sum_{K\in\mathcal{K}^{(j-1)}} |K\cap C\cap R_{j-1}\cap O|$ resources in any case. Therefore, by assumption on μ, and since

$$\mu \geqslant \frac{1}{\gamma^2}\sum_{K\in\mathcal{K}^{(j-1)}} |K\cap C\cap R_{j-1}\cap O|\,,$$

we have that

$$\begin{aligned}\sum_{K\in\mathcal{K}^{(j-1)}} |K\cap C\cap R_{j-1}\cap O| &\leqslant \frac{\gamma^2}{10^{12}\log^3(n)}|C\cap R_k| \\ &\leqslant \frac{10^{11}\log^2\log(n)}{10^{12}\log^3(n)}|C\cap R_k| \leqslant \frac{1}{\log(n)}|C\cap R_k|\,.\end{aligned}$$

Therefore C looses at most $|C\cap R_k|/\log(n)$ resources. Otherwise we have that

$$\mu > \frac{|C\cap R_k|}{10^{12}\log^2(n)} \geqslant \frac{\ell^3}{10^{12}\log^3(n)} \geqslant 200\log(n)\,,$$

by Lemma 4.7. Hence noting X the number of deleted resources in C we have that

$$\mathbb{P}\left(X \geqslant \frac{3}{2}\mu\right) \leqslant \exp\left(-\frac{\mu}{12}\right) \leqslant \frac{1}{n^{10}}.$$

With high probability no configuration looses more than

$$\frac{3}{2}\mu \leqslant \frac{3}{\gamma}\sum_{K\in\mathcal{K}^{(j-1)}} |K\cap C\cap R_{j-1}\cap O| \leqslant \frac{3}{\gamma}\sum_{K\in\mathcal{K}^{(j-1)}} |K\cap C\cap R_{j-1}|$$

resources. Hence each configuration $C\in\mathcal{K}^{(\geqslant j)}$ ends with at least

$$\begin{aligned}&\left(1-\frac{1}{\log(n)}\right)^{2(k-(j-1))-1}\ell^{k-(j-1)}|C\cap R_k| \\ &\qquad -\frac{3}{\gamma}\sum_{j\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-(j-1)}|K\cap C\cap R_h| \\ &\qquad -\frac{1}{\log(n)}\left(1-\frac{1}{\log(n)}\right)^{2(k-(j-1))-1}\ell^{k-(j-1)}|C\cap R_k| \\ &\qquad -\frac{3}{\gamma}\sum_{K\in\mathcal{K}^{(j-1)}}|K\cap C\cap R_{j-1}| \\ \geqslant\ &\left(1-\frac{1}{\log(n)}\right)^{2(k-(j-1))}\ell^{k-(j-1)}|C\cap R_k| \\ &\qquad -\frac{3}{\gamma}\sum_{j-1\leqslant h\leqslant k}\sum_{K\in\mathcal{K}^{(h)}}\ell^{h-(j-1)}|K\cap C\cap R_h|\end{aligned}$$

resources which concludes the proof of Lemma 4.15. □

Given Lemma 4.15 and the intuition below it, it is straightforward to prove the following corollary which will complete the proof of Theorem 4.5.

Corollary 4.17. *There exists an assignment of resources R to $\mathcal{K}$ such that each configuration $C \in \mathcal{K}$ receives at least $\lfloor |C|/(100\gamma) \rfloor$ resources. Moreover, this assignment can be found in polynomial time.*

Proof. Lemma 4.15 with $k = 0$ and Claim 4.14 together imply that we can assign at least

$$\frac{|C|}{2e^2} - \frac{6000}{100.000}|C| \geqslant \frac{|C|}{100}$$

resources to every $C \in \mathcal{K}$ such that no resource in R is assigned more than γ times. In particular, we can fractionally assign at least $|C|/(100\gamma)$ resources to each $C \in \mathcal{K}$ such that no resource is assigned more than once. By integrality of the bipartite matching polytope, the corollary follows. □

Part II

NETWORK DESIGN PROBLEMS

5 INTRODUCTION

In this part, we focus on the design and analysis of simple algorithms for the Matching Augmentation problem (MAP) and the Steiner Forest problem, two fundamental problems in network design. Designing cheap networks that are robust to edge failures is a basic and important problem in the field of approximation algorithms. The area containing these problems is often referred to as *survivable network design*. Generally, one has to compute the cheapest network that satisfies some connectivity requirements in-between some prespecified set of vertices. Classic examples are for instance the Minimum Spanning Tree problem in which one has to augment the connectivity of a graph from 0 to 1 or related questions such as the Steiner Tree or the Steiner Forest problem. Another type of network design is to build 2-edge connected spanning subgraph (2-ECSS) or multisubgraph (2-ECSM), where one has to augment the connectivity of a graph from 0 to 2. Unfortunately, most of the problems in this area are NP-hard (or even APX-hard), and what one can hope for is generally to compute an approximate solution in polynomial time. In this part, we will focus on two interesting connectivity problems: the *Steiner Forest problem* and the *Matching Augmentation problem* (MAP).

Powerful and versatile techniques such as *primal-dual* [47, 85] or *iterative rounding* [58, 66] guarantee an approximation within factor 2 for many of these problems (including the Steiner Forest problem and the MAP) but improving on this bound for any connectivity problem is often quite challenging. In addition, the mentioned techniques are already fairly involved while some extremely simple heuristics are still poorly understood.

5.1 THE STEINER FOREST PROBLEM

One of the most classic problems in network design is arguably the Steiner Tree problem. Given a weighted graph $G = (V, E, w : E \mapsto \mathbb{R}_+)$ and a set $\mathcal{T} \subseteq V$ of *terminals*, one has to compute the cheapest tree that connects all the terminals. A straightforward generalization of this problem is the so-called Steiner Forest problem in which one is given a set $\mathcal{P}$ of *pairs* of vertices that are required to be connected. One has to buy the cheapest forest that connects all the pairs of terminals. These two problems are also interesting in the study of *online* algorithms, which deal with uncertain future. Indeed, the assumption that we know all the instance upfront might not always apply in real life, and it makes sense to relax this assumption. In the online version of the Steiner Tree (or Forest) problem, the terminals in $\mathcal{T}$ (or $\mathcal{P}$) are revealed one by one (in a possibly adversarial order), and the algorithm has to connect all previously arrived terminals (or pairs) before seeing the next one. The challenge lies in the fact that the algorithm is not allowed to remove edges that were bought before. By opposition, we refer to the standard model of computation where all the information is known upfront as the *offline* case.

In a seminal paper, Imase and Waxman [56] introduced the online version of the Steiner Tree problem and provided tight bounds in this scenario by proving that (1) the natural greedy algorithm which simply connects the latest arrived terminal to the closest previously arrived terminal is $O(\log(|\mathcal{T}|))$-competitive[1] and (2) no algorithm can be better than $\Omega(\log(|\mathcal{T}|))$-competitive. Shortly after, Westbrook and Yan [84] introduced the online Steiner Forest problem. As it is a more general problem, the negative result of Imase and Waxman shows that no algorithm can be better than $\Omega(\log(|\mathcal{P}|))$-competitive. From now on, with a slight abuse of notation, we will use k to denote either the number of terminals $|\mathcal{T}|$ in the online Steiner Tree problem or the number of pairs $|\mathcal{P}|$ in the online Steiner Forest problem. A natural generalization of the greedy algorithm of Imase and Waxman for online Steiner Tree to the case of online Steiner Forest can be described informally as follows:

> Upon the arrival of a new pair $\{s_i, t_i\}$, connect s_i and t_i with the shortest path in the current metric, contract the metric along the chosen path and wait for the next pair.

We mention that there are some subtleties about how the metric is contracted exactly, but for the sake of clarity, we will postpone these details to later in the introduction. The reader might think for now that greedy contracts the edges that it selects (i.e. for any edge e selected by greedy, its weight $w(e)$ is set to 0). For now, an algorithm will be considered as "greedy" if it always buys the shortest path in the current metric, and nothing else. Compared to the algorithm we introduced in Chapter 1, the only difference is that here the algorithm cannot choose the order in which the pairs are revealed.

Westbrook and Yan [84] showed that a wide class of greedy algorithms are $O\left(\sqrt{k}\log(k)\right)$-competitive. This bound was quickly improved by Awerbuch, Azar, and Bartal [8] who showed with an elegant dual fitting argument that greedy algorithms are in fact $O\left(\log^2(k)\right)$-competitive and conjectured that the right bound should be $O(\log(k))$. Since then, their conjecture has remained open. It is quite remarkable that the state-of-the-art for this problem is the same also in the offline case, where greedy can choose the order in which it connects the pairs.

This remark also applies to the lower bound side. The best lower-bound is $\Omega(\log(k))$ both in online and offline settings. In fact, all lower bounds for greedy that appeared so far in the literature (see [3, 8, 23, 56]) are instances where the underlying optimum forest is a single tree. Surprisingly, even in this case, nothing better than the $O\left(\log^2(k)\right)$ upper bound is known. We note that the $O(\log(k))$-competitive analysis of Imase and Waxman for online Steiner Tree does not extend to the online Steiner Forest problem, even if we additionally assume that the optimum is a single tree. Indeed, the solution constructed by greedy may not be a single connected component even if the

1 In the literature on online algorithms, we say that an algorithm is α-competitive if it returns a solution that is never more than α times the optimum. For the purpose of this thesis, one might think of an α-competitive algorithm as an α-approximation algorithm that works in an online setting.

underlying optimum solution is. This problem is highlighted by the lower bounds that appear in [8, 23], which showcase the limitations of the current analysis techniques.

FURTHER RELATED WORK. Apart from the two results [8, 84] mentioned above, many papers are related to the problem of understanding the greedy algorithm for Steiner Forest. The competitive ratio of greedy was first mentioned again in a list of open problems by Fiat and Woeginger [40]. Around the same time, Berman and Coulston [16] designed a more complex (non-greedy) algorithm which they showed to be $O(\log(k))$-competitive in the online setting. However, their algorithm is not greedy because it can buy additional edges that could be helpful in the future but are useless right now. Later, Chen, Roughgarden, and Valiant [23] applied the result of Awerbuch et al. regarding the greedy algorithm to design network protocols for good equilibrium. Interestingly, they mention that the non-greedy algorithm of Berman and Coulston was not possible to apply in their setting. More recently, the performance of greedy algorithms for online Steiner Forest was further raised as an "important open problem" in [36] and also cited in [75].

Finally, we note that there are several other situations where the Steiner Forest problem is significantly harder to the understand than the Steiner Tree problem. In the offline case, giving a better-than-2 approximation for Steiner Forest is a major open problem in approximation algorithms, while such a result already exists for the Steiner Tree problem [19, 77, 86]. Similarly, it was known for a very long time that greedy (i.e. compute a minimum spanning tree of the metric completion on terminals) gives a constant factor approximation to the optimum Steiner Tree in the offline case, but it has only been recently proved by Gupta and Kumar [50] that a simple "gluttonous" algorithm also yields a constant factor approximation in the case of offline Steiner Forest (we emphasize that the gluttonous algorithm cannot apply to the online setting, in constrast with the simpler greedy algorithm).

5.1.1 *Our results*

As announced, there are some subtleties about how the metric is contracted when running greedy. Hence we will first define three variants of greedy that contract the metric in slightly different ways. However, we emphasize that the best current upper bound for all these three variants is the $O(\log^2(k))$ upper bound of Awerbuch et al. Furthermore, these three variants behave *exactly* the same on the problematic examples of [8, 23]. In particular, all the discussion so far applies to any of these three variants and our main technical theorem will apply to all three variants of greedy, but we will be able to obtain more specialized results for some variants. In particular, for one of the variants we show that greedy guarantees a $O(\log(k) \cdot \log\log(k))$-approximation in the offline case. Before getting into the precise definition, it is worthwhile to mention that Gupta and Kumar [50] also discussed some subtleties about contracting the metric and also defined several algorithms based on that. Hence it is not the first time that altering the contraction procedure has been considered.

DEFINITION OF THE GREEDY ALGORITHM. After greedy connected the latest arrived pair $p = \{s, t\}$, it is clear that the distance between s and t can be set to 0 in the current metric. This is the only property of the metric contraction that the proof of Awerbuch et al. uses to obtain a good upper bound. As long as we obtain a new graph G' such that $d_{G'}(s, t) = 0$, the $O(\log^2(k))$ upper bound applies. In our paper, each greedy algorithm will formally maintain a graph $G^{(\tau)} = (V, E^{(\tau)})$ that accurately describes the current metric. In any case, greedy will always take the *shortest path* in the current metric $G^{(\tau)}$ to connect a newly arrived pair. After connecting the τ-th pair, the set of edges will be defined as $E^{(\tau)} = E \cup S^{(\tau)}$ where $S^{(\tau)}$ is a set of edges of weight 0 over vertex set V that we will call *shortcuts*. It will be clear from definition that we will have the natural condition that $\emptyset = S^{(0)} \subseteq S^{(1)} \subseteq \ldots \subseteq S^{(k)}$. The cost incurred by greedy will always be the sum of lengths of all the shortest paths taken for connecting the pairs. We now proceed to define the three contraction rules formally.

Rule 1: When greedy connects a pair $\{s, t\}$ through a path $s = v_0, v_1, v_2, \ldots, v_\ell, t = v_{\ell+1}$, add the following shortcuts:

- For all $0 \leqslant i \leqslant \ell$ add the edge $\{v_i, v_{i+1}\}$ of weight 0.

Rule 1 is what Awerbuch et al. intended in their original paper. It can be seen as simply contracting all the edges on the path taken.

Rule 2: When greedy connects a pair $\{s, t\}$ add an edge $\{s, t\}$ with weight 0.

Rule 2 is actually how the metric is contracted in [50] for their main algorithm. Rule 2 might seem much weaker than Rule 1 as fewer shortcuts are added. One can see that for any $u, v \in V$, the distance between u and v can only be smaller when using Rule 1 over Rule 2. Hence the cost of greedy equipped with Rule 2 is always an upper bound on the cost of greedy equipped with Rule 1. However, the proof of [8] already applies in this case. Hence the greedy algorithm that uses Rule 2 is already $O(\log^2(k))$-competitive.

Rule 3: When greedy connects a pair $\{s, t\}$ through a path $s, v_1, v_2, \ldots, v_\ell, t$, let $s = v'_0, v'_1, \ldots, v'_{\ell'}, t = v'_{\ell'+1}$ be the subsequence of $s, v_1, v_2, \ldots, v_\ell, t$ in which we keep only the vertices that appeared in a previous pair (i.e. previously arrived terminals). Then, add the following shortcuts:

- For all $0 \leqslant i \leqslant \ell'$ add the edge $\{v'_i, v'_{i+1}\}$ of weight 0.

Intuitively, Rule 3 is in-between rules 1 and 2. It is also reminiscent of the contraction rule of the second algorithm in [50]. Again, using this rule, it is clear that we obtain shorter paths than with Rule 2. The $O(\log^2(k))$ upper bound also applies when using Rule 3. The formal definition of Greedy_i follows naturally for any $i \in \{1, 2, 3\}$.

Algorithm 1 Greedy_i

1: Upon arrival of pair $\{s, t\}$, buy the shortest path in the current metric.
2: Update the metric from $G^{(\tau)}$ to $G^{(\tau+1)}$ using Rule i and wait for the next pair.

As a shorthand, we will denote by A the greedy algorithm at hand. If we do not specify which contraction rule we use, it will implicitly mean that the statement that follows holds for any of our three rules.

OUR RESULTS. First, we introduce an intuitive measure of the efficiency of Greedy_i . In general, it might be that the cost incurred by the pair $p = \{s, t\}$ is much smaller than $d_G(s, t) = d_{G^{(0)}}(s, t)$. Indeed, because of additional shortcuts, it might be that the ratio $d_G(s, t)/d_{G^{(\tau)}}(s, t)$ is unbounded when $d_{G^{(\tau)}}(s, t) = 0$. We will define this ratio as the *contraction* of pair p and denote it $\alpha(p)$. We have that

$$1 \leqslant \alpha(p) \leqslant \infty,$$

for all pairs p, in any instance $\mathcal{I}$. Intuitively, a very high contraction means that Greedy_i did a good job at reusing edges bought before. Following this remark, one can note that all known lower bounds in [3, 8, 23, 56] have a contraction of exactly 1 for all pairs in the instance (in the case of Steiner Tree instances in [3, 56], one can always choose one of the two endpoints of each pair so that this is the case). This seems to confirm the intuitive reasoning that a hard instance should have most of the pairs with low contraction.

After this remark, we use the shorthand $\mathrm{A} = \mathrm{Greedy}_i$ to denote the greedy algorithm at hand (using any of our three contraction rules). We denote by $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathcal{P}_{<\alpha})$ the cost incurred by A because of pairs of contraction strictly less than α (i.e. pairs p with $\alpha(p) < \alpha$) when running instance $\mathcal{I}$. Note that we do not count the cost incurred by A because of pairs with contraction higher than α. Furthermore, we will denote $\mathrm{cost}_{\mathrm{A}}(\mathcal{I})$ the total cost incurred by A. Finally, denote by $w(\mathrm{OPT}(\mathcal{I}))$ the cost of the offline optimum. Our main technical result is the following. We remark that it applies regardless of the ordering of the pairs in $\mathcal{P}$.

Theorem 5.1 (Main technical theorem). *Fix a sequence $(\alpha_k)_{k\geqslant 0}$ with $\alpha_k \geqslant 1$ for all k. Let A be a greedy algorithm running on the instance $\mathcal{I}$ containing k pairs. Then,*

$$\mathrm{cost}_{\mathrm{A}}\left(\mathcal{I}, \mathcal{P}_{<\alpha_k}\right) = O(\log(k)) \cdot (\log(\alpha_k) + \log\log(k)) \cdot w(\mathit{OPT}(\mathcal{I})).$$

As mentioned, this theorem applies to greedy with any of our three contraction rules. This already implies the theorem of Awerbuch et al. as it is straightforward to see that pairs with contraction at least k can make greedy pay at most $O(1) \cdot w(\mathrm{OPT}(\mathcal{I}))$. To see this, simply denote by $\mathcal{P}'$ these pairs with contraction more than k and c the greedy cost of the most expensive pair in $\mathcal{P}'$. Then greedy pays for those pairs in $\mathcal{P}'$ a total cost of at most $k \cdot c$ while $\mathrm{OPT}(\mathcal{I})$ must pay at least $k \cdot c$ to connect the most expensive pair in $\mathcal{P}'$. With this observation and plugging in $\alpha_k = k$ in our bound, we obtain an upper bound of $O(\log^2(k))$ on the competitive ratio.

A consequence of our result is that if one wants to have an $\Omega\left(\log^{1+\varepsilon}(k)\right)$ lower bound for any fixed $\varepsilon > 0$, it must be that the lower bound on the cost incurred by greedy comes from pairs with contraction at least $2^{\Omega(\log^{\varepsilon}(k))}$. This already changes the perspective on how to obtain a stronger lower bound (if it exists) and shows that all previous lower bounds (that have contraction $\alpha_k = 1$ for all pairs) cannot give anything stronger than $\Omega(\log(k) \cdot \log\log(k))$. The fact that the cost should come from pairs with high contraction seems quite counter-intuitive, and we believe this is strong evidence that the old conjecture of Awerbuch et al. should be true, even in adversarial order. Unfortunately, it is not clear to us how to formalize such an intuition. However, this result still has a number of additional consequences that are interesting. In the following, $w(T^{\star}(\mathcal{I}))$ denotes the optimum *tree* solution to instance $\mathcal{I}$ (i.e. we restrict the solution to be a single connected component).

Theorem 5.2. *Let* A *be the greedy algorithm using Rule 3. Then we have that*

$$\text{cost}_{\text{A}}(\mathcal{I}) = O(\log(k) \cdot \log\log(k)) \cdot w(T^{\star}(\mathcal{I})),$$

even in adversarial order.

As mentioned in the introduction, even in the case of a single tree spanning the whole graph, nothing better than the general $O(\log^2(k))$ was known, and all lower bounds in the literature are instances where the optimum is a single component. We also obtain the following result, which was highlighted in the introduction of this thesis (Chapter 1).

Theorem 5.3 (Main theorem)**.** *Let* A *be the greedy algorithm using contraction rule 3. Denote by c_i the cost paid by* A *when connecting pair $p_i = \{s_i, t_i\}$ and define $d_i = d_G(s_i, t_i)$ (note that we take the distance in the original graph G). If either of the two sequences $(c_1, c_2, \ldots, c_k)$ or $(d_1, d_2, \ldots, d_k)$ is non-increasing then*

$$\text{cost}_{\text{A}}(\mathcal{I}) = O(\log(k) \cdot \log\log(k)) \cdot w(OPT(\mathcal{I})).$$

This last result can be derived by using our main theorem, combined with a potential function argument from [50]. This result proves that greedy is an $O(\log(k) \cdot \log\log(k))$-approximation in the offline setting (for instance, we can choose to order the pairs so that the sequence $(d_1, d_2, \ldots, d_k)$ is non-increasing).

These results are proven in Chapter 6. This chapter is organized as follows. Section 6.1, we present the main obstacles in previous approaches and the key idea underlying our new result. In Section 6.2 we present the proof of our main technical result, Theorem 5.1. In Section 6.3, we present our proof of Theorem 5.2 and Theorem 5.3.

5.2 THE MATCHING AUGMENTATION PROBLEM

As mentioned earlier, an important network design problem is the 2-edge connected spanning subgraph (2-ECSS) where has to select the cheapest 2-edge connected subgraph of a given graph (note that taking two copies of the same edge is not allowed in this problem. This problem is closely related to the famous Traveling Salesman Problem (TSP). As for the Steiner Forest

problem, iterative rounding gives a factor 2 approximation for this problem [58], and it is known that the problem is APX-hard (see for instance [34]). In the case of 2-ECSS, a $118/89+\varepsilon < 1.321$-approximation is known [44] if the underlying graph G is unweighted (this improves on a the previous bound of $4/3$ [55, 79]). However, a similar result for the weighted case has remained elusive, and the best approximation algorithm only guarantee a factor 2 approximation. A prominent special case of the weighted 2-ECSS problem is the so-called Forest Augmentation Problem (FAP). In such instances of 2-ECSS all edge weights are either 0 or 1 (we will refer to edges of cost 0 as *light edges* and edges of cost 1 as *heavy edges*). The name stems from the fact that one can assume that the light edges form a forest F, and the goal is to find the smallest set of heavy edges E' such that $F \cup E'$ is 2-edge connected. Independently of our work, a very recent breakthrough [48] gives an 1.9973-approximation algorithm for FAP, breaching for the first time the barrier of 2 in that setting.

A famous special case of FAP is the Tree Augmentation Problem (TAP) which has been extensively studied for decades. In this problem, the forest F is a single spanning tree, and one has to find the smallest set of edges to make the tree 2-edge connected. For this problem, several better-than-2 approximations were designed in a long line of research [1, 20, 28–30, 32, 37, 41, 43, 49, 61, 63, 64, 72, 74, 81, 82]. One can see TAP as an extreme case of FAP where the forest is a single component. Another interesting special case is the Matching Augmentation Problem (MAP), in which the forest of light edges forms a matching M and one has to find the smallest set of heavy edges E' such that $M \cup E'$ is 2-edge connected. It can be seen as the other extreme case in which the forest forms as many components as possible. We also remark that MAP generalizes the unweighted 2-ECSS problem, which can be viewed as an instance of MAP with an empty matching. For MAP, only recently several better-than-2 approximation [26, 27, 45, 48] were given. These works culminate in a $13/8 \approx 1.625$-approximation for MAP.

For many of these network design problems, there is a simple linear programming relaxation called the *cut LP*. In the case of FAP, for a given graph $G = (V, E)$, forest $F \subseteq E$ the cut LP is written as follows, with a variable x_e to decide to take each edge e or not. Recall that $\delta(S)$ denotes the edges with exactly one endpoint in S.

$$
\begin{aligned}
LP(G,F): \quad \min & \sum_{e \in E \setminus F} x_e \\
& \sum_{e \in \delta(S)} x_e \geqslant 2, \quad \text{for all } S, \emptyset \subsetneq S \subsetneq V \\
& 0 \leqslant x_e \leqslant 1, \qquad \forall e \in E.
\end{aligned}
$$

The integrality gap of this linear program is an interesting question by itself. Recently, in the case of TAP (i.e. F is a spanning tree), Nutov [74] showed that the integrality gap is at most $2 - 2/15 \approx 1.87$. Cheriyan et al. [30] showed that the integrality gap is at least $3/2$ in the case of TAP while in the case of MAP, the best upper bound on the integrality gap is 2, and the best lower bound is $9/8$ [2, 79]. We note that the recent works on MAP [26,

27, 45, 48] do not compare against the cut LP, and therefore do not show an integrality gap better than 2 for MAP. Moreover, these algorithms and their analysis are quite involved.

5.2.1 *Our results*

In Chapter 7 of this thesis, we give a simple algorithm that guarantees an approximation ratio $2 - c$ (for some absolute constant $c > 0$) with respect to the best fractional solution of the cut LP. In contrast to the other better-than-2 approximation algorithms, our algorithm is very simple. We note that some of our techniques are reminiscent of the algorithm of Mömke and Svensson [69] for the travelling salesman problem (see also [71, 73] for follow-up works). The algorithm is the following.

The LP-based algorithm:

1. Compute an optimal extreme point solution x^* to $LP(G, M)$.
2. Let $E' = \{e \in E, x^*_e > 0\}$ be the *support* of x^*, and run a DFS on the support graph $G' = (V, E')$ which always give priority first to an available light edge and second to the available heavy edge e maximizing x^*_e.
3. Compute an optimum augmentation A to the TAP problem with respect to the DFS tree T computed in the previous step and return $H = T \cup A$.

We note that the LP-based algorithm indeed runs in polynomial time. Step 2 computes a DFS in which some edges are explored in priority (if possible). Step 3 can also be done in polynomial time because the tree T is a DFS tree. This implies that all non-tree edges are back-edges (i.e. one endpoint is an ancestor of the other). In the language of TAP, these edges are often referred to as "uplinks", and it is well-known that TAP instances in which the edges are only "uplinks" are solvable in polynomial time [32, 42]. Finally, the solution given by the algorithm is feasible since Step 2 increases connectivity from 0 to 1 and Step 3 from 1 to 2. One can check that no edge is taken twice in the process since A and T are disjoint.

In this part of the thesis, our main result shows that this simple algorithm guarantees an approximation within factor strictly better than 2 with respect to the cut LP relaxation.

Theorem 5.4. *The LP-based algorithm returns a feasible solution to any MAP instance of cost at most $2 - c$ times the cost of the fractional solution x^*, for some absolute constant $c > 0$.*

For the sake of exposition, we did not try to optimize the constant c but we believe that improving the ratio of $13/8$ in [45] (that holds with respect to the optimum *integral* solution) would require new techniques in the analysis. Since Nutov [74] proved the integrality gap of the cut LP to be strictly better than 2 for TAP, the cut LP seems a promising relaxation for the general FAP. Additionally, we prove the following simple theorem.

Theorem 5.5. *The integrality gap of the cut LP for MAP is at least* $4/3$.

Proof. Consider the example given in Figure 5.1, which is a simple adaptation of a classic example for the related TSP problem. One can check that the

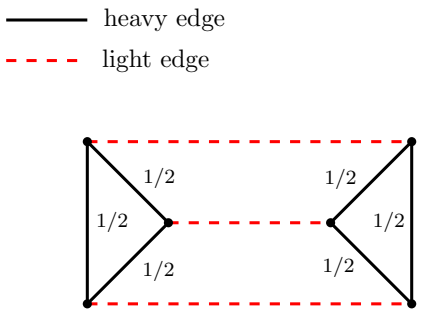

Figure 5.1: An integrality gap example.

fractional solution that gives $1/2$ fractional value to all heavy edges and value 1 to all light edges is feasible for a total cost of $6/2 = 3$. However, any integral solution costs at least 4. □

6 IMPROVED BOUNDS FOR GREEDY STEINER FOREST

In this chapter, we give the proof of Theorem 5.1 in Section 6.2, Theorem 5.2, and Theorem 5.3 in Section 6.3. We first give an overview of the techniques in Section 6.1 before providing the proofs. All the results presented in this chapter are based on the paper "An Improved Analysis of Greedy for Online Steiner Forest", a joint work with Marina Drygala and Andreas Maggiori. It appeared at the *ACM-SIAM Symposium on Discrete Algorithms* (SODA '22) [9].

6.1 THE IDEA BEHIND THE PROOF

In this section, we give the necessary technical details to illustrate our techniques but keeping it rather informal for clarity.

PREVIOUS TECHNIQUES. Before going into our techniques, we will briefly mention the main ingredients of previous proofs. For ease of notation, we will denote by $\mathcal{T}$ the set of terminals that appear in at least one pair of $\mathcal{P}$. For a terminal $u \in \mathcal{T}$, denote by $\overline{u}$ its *mate* which is the terminal that u should be connected to. Note that we can assume without loss of generality that all terminals have exactly one mate, by duplicating vertices if this is not the case. For ease of presentation, we will assume until the end of the section that for each pair $\{u, \overline{u}\}$ greedy pays exactly the distance of u and $\overline{u}$ in the initial graph, i.e. no previously arrived pair helps greedy in paying less for the newly arrived terminal pair $\{u, \overline{u}\}$. Put otherwise, we assume that the contraction of all the pairs is equal to 1.

By standard arguments, we can assume that $w(\text{OPT}(\mathcal{I})) = k$ and that for each pair of terminal greedy pays a cost that belongs to the set $\{k/2^i\}_{1 \leqslant i \leqslant \log(k)}$, with only the loss of a constant factor. Indeed, we can rescale all the edges in the graph G by the factor $k/w(\text{OPT}(\mathcal{I}))$ and for the second assumption, note that by standard geometric grouping, we can assume that greedy pays a cost that belongs to the set $\{k/2^i\}_{1 \leqslant i \leqslant \infty}$. It is then straightforward to see that the total cost incurred by greedy for pairs cheaper than $k/2^{\log(k)} = 1$ is at most $O(w(\text{OPT}(\mathcal{I}))$. Based on this observation, we will partition $\mathcal{P}$ into disjoint sets $\mathcal{P}^{(i)}$ where each set contains pairs of terminals for which greedy paid $k/2^i$. These sets will be called *cost classes*. Moreover, in order to introduce the first observation let $B(v, r) = \{u \in V \mid d_G(v, u) < r\}$ be an open ball with center the terminal u and radius r. A classic observation is the following.

Observation 6.1.1. *Let $\mathcal{B} = \{B_1, B_2, \ldots, B_\ell\}$ be a collection of balls around terminals, such that (1) all the balls are pairwise disjoint and (2) each ball B_i is centered as some terminal $u \in \mathcal{T}$ and its radius r_i satisfies $r_i < d_G(u, \overline{u})$. Then $\sum_i r_i \leqslant w(OPT(\mathcal{I}))$.*

The proof of this observation is straightforward, and we will come back to it later in Section 6.2. These balls can be viewed as a solution to the dual

of the natural linear programming relaxation of the Steiner Forest problem; hence we will refer to these balls as dual balls. We continue by restating (informally) a key lemma in the analysis of [8].

Lemma 6.1 ([8]). *For any cost class $\mathcal{P}^{(i)}$ (associated with cost $c_i = k/2^i$), it is possible to place disjoint dual balls in G such that these balls are centered around terminals that belong to $\mathcal{P}^{(i)}$ and they all have a radius of*

$$\frac{k}{2^i \cdot \log(|\mathcal{P}^{(i)}|)}.$$

Moreover, at the cost of losing a constant factor we can assume that every pair $p = \{s, t\}$ in $\mathcal{P}^{(i)}$ has at least one dual ball centered around s or t.

By taking Lemma 6.1 together with Observation 6.1.1, we obtain that greedy pays at most $O(\log(|\mathcal{P}^{(i)}|)) = O(\log(k))$ times the dual solution for each cost class. This proves that greedy is $O(\log(k))$-competitive for each cost class $\mathcal{P}_i$. Hence we see that the previous proof technique has mainly two ingredients:

(1) Partition the set $\mathcal{P}$ into disjoint cost classes $\mathcal{P}^{(i)}$ such that $\mathcal{P} = \bigcup_{i=1}^{\log(k)} \mathcal{P}^{(i)}$ and for each pair in $\mathcal{P}^{(i)}$ greedy paid $k/2^i$.

(2) Prove that greedy is $O(\log(k))$-competitive for each cost class separately by building a dual solution.

By (1) and (2) we get that greedy is $O(\log(k) \cdot \log(k)) = O(\log^2(k))$-competitive. Interestingly, in the case of Steiner Tree, it is possible to improve the second step by showing that greedy is $O(1)$-competitive for each cost class, hence the $O(\log(k))$ competitive ratio in general (see [3]). Unfortunately, this is impossible in the case of Steiner Forest. Even if all the pairs have contraction 1, it might be that greedy is already $\Omega(\log(k))$-competitive for a single cost class (see [8, 23] for an example). We also note that the Berman-Coulston algorithm [16] is designed so that the algorithm is $O(1)$-competitive for each cost class, so the analysis of this more complex algorithm cannot apply to greedy.

OUR NEW APPROACH. The first step of our new proof relies in a different partitioning of the set $\mathcal{P}$. Indeed we will partition $\mathcal{P}$ into $N = \Theta(\log\log(k))$ classes $\tilde{\mathcal{P}}^{(j)}$ such that each class is defined as follows.

$$\tilde{\mathcal{P}}^{(j)} = \bigcup_{i \equiv j \bmod(N)} \mathcal{P}^{(i)}.$$

Note that we have $\Theta(\log\log(k))$ groups with this partitioning, each containing $\Theta\left(\frac{\log(k)}{\log\log(k)}\right)$ cost classes. Inside each group, the cost classes have the nice property that they are *well separated*, that is, the multiplicative gap between two consecutive costs is polylog(k). We will make good use of this property to disentangle the interactions between pairs that have different costs. Using these techniques we prove that the competitive ratio of greedy for each set $\tilde{\mathcal{P}}^{(j)}$ is $O(\log(k))$ ending up with a competitive ratio of

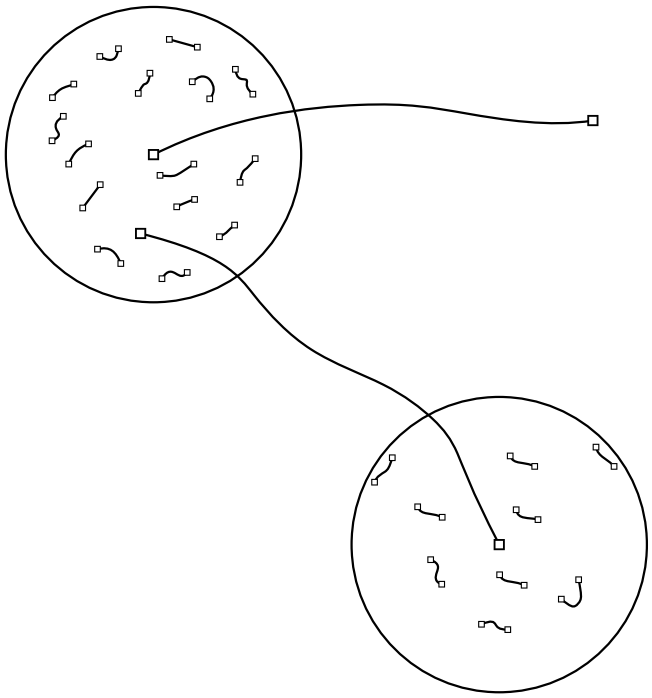

Figure 6.1: An example with two cost classes before charging or clustering.

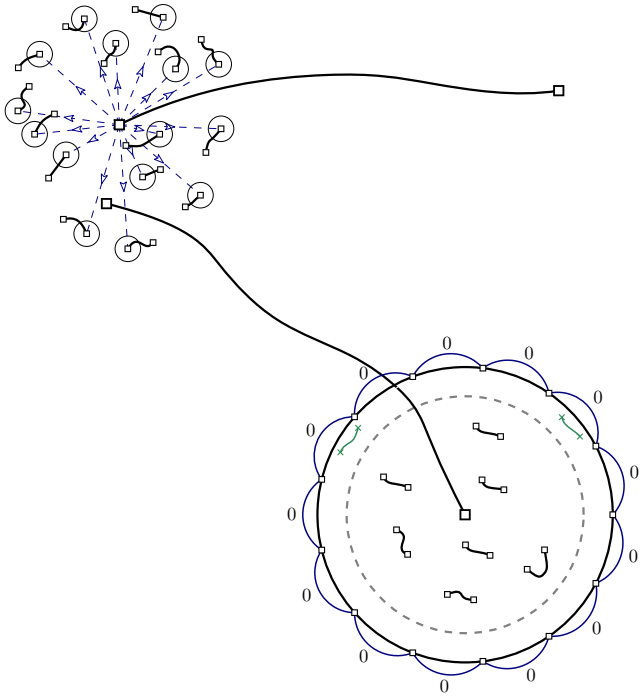

Figure 6.2: An example with two cost classes after charging or clustering. In the top left corner, smaller pairs are much more expensive than the pairs that created the big dual ball while we have the opposite situation in the bottom right corner.

$O(\log(k) \cdot \log\log(k))$ overall. The main technical challenge lies in proving such a result.

If we use Lemma 6.1 to place dual balls around pairs in each cost class $\mathcal{P}^{(i)} \subseteq \tilde{\mathcal{P}}^{(j)}$ (hence creating several collections $\mathcal{B}^{(i)}$ of balls) it might be that two dual balls B, B' that belong to different sets $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(i')}$ overlap. This is the critical issue in the previous proofs, and we will proceed differently. For simplicity, assume we have two cost classes in our set $\tilde{\mathcal{P}}^{(j)}$. Let c_i be the cost of the larger class and $c_{i'}$ the cost of the smaller class (hence $i' > i$). We place the dual balls only for the biggest cost class (using Lemma 6.1). Intuitively, the worst case in the analysis will be when *all* pairs of the smaller cost lie inside the dual balls from the bigger cost class (as depicted in Figure 6.1). If this happens, it will be impossible to place dual balls for the small cost class without intersecting the bigger balls already placed. To overcome this issue, we consider a ball B from the big cost class, and we look at the number of pairs from the small cost class that lie inside this ball. Let the number of such pairs be k'. We have two cases.

(1) $k' \cdot c_{i'} \geqslant \text{polylog}(k) \cdot c_i$ which is the easy case. In this case, instead of charging the cost of the big pairs to the dual ball B we can instead charge this cost to the smaller pairs inside B. By Lemma 6.1, the cost that was initially charged to the ball B was $O(c_i)$. Hence if we evenly distribute this cost among all the small pairs inside B, each pair will get a cost of roughly.

$$O\left(\frac{c_i}{k'}\right) \leqslant O\left(\frac{c_{i'} \cdot c_i}{\text{polylog}(k) \cdot c_i}\right) = O\left(\frac{c_{i'}}{\text{polylog}(k)}\right).$$

Since the cost was transferred to smaller pairs, we can also safely delete the big dual ball B, hence making this space available to place the smaller balls. Note that smaller pairs can be charged at most once in this way because the balls in the dual solution $\mathcal{B}^{(i)}$ for big pairs are pairwise disjoint. This case is depicted in the top left corner of Figure 6.2.

(2) $k' \cdot c_{i'} < \text{polylog}(k) \cdot c_i$. This is the most challenging case and is depicted in the bottom right corner of Figure 6.2. We cannot proceed as in the previous case as we cannot guarantee that small pairs do not get charged too much. Here lies the crux of our proof. First, by re-scaling slightly the ball B, we can assume that *almost all* the small pairs in B are far from the border of B (a pair $\{s, t\}$ is *far* from the border if one of s or t is at a distance much bigger than $c_{i'}$ from the border of B). For simplicity we assume that all the small pairs are far from the border of B. From here we construct an instance $\mathcal{I}'$ as follows. Consider the graph G' that is induced by vertices inside B. The instance $\mathcal{I}'$ will be composed of the set $\mathcal{P}'$ containing all the pairs of small cost that are inside B, and the metric will be the graph G'. Here the assumption that the contraction is 1 implies that both endpoints of each pair in $\mathcal{P}'$ should be inside B (one endpoint well inside the ball and one outside would cost too much). It also implies that greedy behaves *exactly* the same for the pairs in $\mathcal{P}'$ in instance $\mathcal{I}'$ as it was behaving for these

pairs $\mathcal{P}'$ in instance $\mathcal{I}$, that is, for each pair $p \in \mathcal{P}'$ greedy buys exactly the same path to connect p whether instance $\mathcal{I}$ or $\mathcal{I}'$ is running. Recall that we assumed

$$k' \cdot c_{i'} < \text{polylog}(k) \cdot c_i$$

hence we have

$$k' < \frac{\text{polylog}(k) \cdot c_i}{c_{i'}}.$$

We know by previous results that greedy is $O(\log(k))$-competitive on a single cost class; hence the competitive ratio of greedy on instance $\mathcal{I}'$ will be bounded by

$$\begin{aligned} O(\log(k')) &= O\left(\log\left(\frac{\text{polylog}(k) \cdot c_i}{c_{i'}}\right)\right) \\ &= O(\log\log(k)) + O\left(\log\left(\frac{c_i}{c_{i'}}\right)\right). \end{aligned}$$

Now the crucial question: What is the value of $w(\text{OPT}(\mathcal{I}'))$? As we defined the graph G' now, it is not clear. But because we assumed all the small pairs are far from the border of B, we can allow ourselves to modify the metric on the border of B *without* changing the behavior of greedy. If we consider V' the set of vertices that lie *exactly* on the border of B we will add an edge of length 0 between any pair of vertices in V'. This does not change the behavior of greedy on instance $\mathcal{I}'$ because these extra edges are already too far from the pairs in $\mathcal{I}'$ to be used (see Figure 6.2, bottom right corner). The interesting fact is now that

$$w(\text{OPT}(\mathcal{I}')) \leqslant w(\text{OPT}(\mathcal{I}) \cap B),$$

where we denote by $w(\text{OPT}(\mathcal{I}) \cap B)$ the cost of edges bought by $\text{OPT}(\mathcal{I})$ inside the ball B.

These observations suggest a Top-Down approach where we first try to place dual balls around big pairs. Then proceed by the case distinction described above. Then we move to the next cost class but ignoring all the pairs that got into case (2). We repeat this until we reach the bottom of the cost hierarchy. We end up with dual balls that have different radii but are all pairwise disjoint (because we ignored the pairs that were in case (2) of any iteration). During this process, each pair got into case (1) at most $\log(k)$ times, hence the total additional charge is $O(\frac{\log(k)}{\text{polylog}(k)}) = O(1)$. It remains to handle all the pairs that were ignored. The idea is now that these pairs can be partitioned into disjoint instances with not too many pairs (recall that we have an upper bound on k' in case (2)) and such that the optimum solution is at most what $\text{OPT}(\mathcal{I})$ pays inside the ball that created this instance. For instance in the case of two consecutive cost classes $\mathcal{P}^{(i)}, \mathcal{P}^{(i')}$ (hence $\frac{c_i}{c_{i'}} = \text{polylog}(k)$), the total cost of ignored pairs would be equal to:

$$\begin{aligned} &\sum_{B \in \mathcal{B}^{(i)}} \left(O(\log\log(k)) + O\left(\log\left(\frac{c_i}{c_{i'}}\right)\right)\right) \cdot w(\text{OPT}(\mathcal{I}) \cap B) \\ &\qquad = \sum_{B \in \mathcal{B}^{(i)}} O(\log\log(k)) \cdot w(\text{OPT}(\mathcal{I}) \cap B)\,. \end{aligned}$$

But because the pairs in $\mathcal{B}^{(i)}$ are pairwise disjoint we have

$$\sum_{B \in \mathcal{B}^{(i)}} w(\mathrm{OPT}(\mathcal{I}) \cap B) \leqslant w(\mathrm{OPT}(\mathcal{I})) \ .$$

Hence in total the ignored pairs cost at most $O(\log\log(k)) \cdot w(\mathrm{OPT}(\mathcal{I}))$ to greedy. Because we have $\Theta(\frac{\log(k)}{\log\log(k)})$ cost classes inside a set $\tilde{\mathcal{P}}^{(j)}$, it feels that a $\log(k)$ competitive ratio for this set is now possible. Of course we took two consecutive cost classes so that $\frac{c_i}{c_{i'}} = \mathrm{polylog}(k)$ but this is intuitively the worst case in the analysis. All this is formally handled via a delicate induction that is done in Section 6.2.

6.2 PROOF OF THEOREM 5.1

This section is devoted to the proof of Theorem 5.1. Recall that this theorem applies to the three variants of greedy as defined in the introduction. Hence in this section, A will denote Greedy_i for any $i \in \{1, 2, 3\}$. This section is organized as follows. In Subsection 6.2.1, we introduce some basic definitions that will be needed. In Subsection 6.2.2, we detail some results implied by previous work as well as some pre-processing of the instance needed for the rest of the proof. Namely, we recall the concept of dual fitting used by [8] . In addition, we pre-process the instance so that the different costs greedy pays upon the arrival of different pairs is well-structured (i.e. there is a geometric grouping and a big gap in-between two consecutive cost classes). In Subsection 6.2.3, we give an overview of the main body of the proof, and finally, in Subsections 6.2.4 and 6.2.5, we finish the proof.

6.2.1 *Problem definition and notation*

We will consider a slightly more general problem than Online Steiner Forest. Formally, we are given a weighted graph $G = (V, E, w)$ with weight function $w : E \mapsto \mathbb{R}_{\geqslant 0}$. Along with graph G we are given an ordered sequence of pairs of vertices $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ revealed one by one, and an ordered sequence of sets $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$ of additional weighted edges. These edges will be made available to the online algorithm A over time as follows. Before revealing the first pair p_1, the set of edges in E_1 is added in the graph G to form the graph $G_1 = (V, E \cup E_1)$. These edges in E_1 will remain available to the greedy algorithm A until the end. Then A buys some path and contracts the metric according to its contraction rule as defined in the Introduction to obtain graph G'_1. Next, before revealing p_2, we add the edge set E_2 into the graph G'_1 to obtain the graph G'_2 (hence A updates the metric accordingly). Then A sees the pair p_2 and so on. In general, if $G^{(\tau)}$ denotes the current metric available to greedy after reading pairs $p_1, p_2, \ldots, p_\tau$, we first add the edges of $E_{\tau+1}$ to the graph $G^{(\tau)}$ and after this A connects the pair $p_{\tau+1}$ via the shortest path contracting the metric according to the chosen contraction rule. We call this variant *online Steiner Forest in decreasing metrics*. This generalizes the classic Online Steiner Forest which is the special case where $E_i = \emptyset$ for all i.

The goal is to compare the cost incurred by A on the instance $\mathcal{I} = (G, \mathcal{P}, \mathcal{S})$ to the cost of the optimum Steiner forest in the graph G with pairs $\mathcal{P}$. We insist that the offline optimum is not allowed to use edges from $\mathcal{S}$ while the algorithm A can use these edges in $\mathcal{S}$ after they are revealed to it. We will denote the optimum cost by $w(\mathrm{OPT}(\mathcal{I}))$. The *size* of an instance $\mathcal{I}$ is the number of pairs in $\mathcal{P}$. It will be denoted k in the following (hence $k = |\mathcal{P}|$). For each pair $p = \{s, t\}$, we will naturally call the *endpoints* of the pair p the two vertices s, t.

For any subset $S \subseteq \mathcal{P}$, we will denote by $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, S)$ the cost incurred by algorithm A on instance $\mathcal{I}$ because of pairs in S. By a slight abuse of notation, for a single pair p, we will denote by $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \{p\})$ the cost that A pays upon arrival of p.

The *contraction* of a pair with respect to instance $\mathcal{I}$ and algorithm A will be the ratio of the shortest path distance in-between the two endpoints of the pair in G and the actual cost paid by A for this pair when running instance $\mathcal{I}$. Note that the shortest path is taken in the original graph G, without help of edges in $\mathcal{S}$. Formally, if we denote by $\alpha(p)$ the contraction of pair $p = \{s, t\}$, we have

$$\alpha(p) = \frac{d_G(s, t)}{\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p)},$$

with the convention that $\alpha(p) = \infty$ if $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p) = 0$. Given a fixed $\alpha \geqslant 1$ and an instance $\mathcal{I} = (G, \mathcal{P}, \mathcal{S})$, we denote by $\mathcal{P}_{<\alpha}$ the set of pairs of $\mathcal{P}$ that have contraction less than α when running A (i.e. the pairs p with $\alpha(p) < \alpha$).

For simplicity we will assume that every edge is of weight exactly η for some arbitrarily small $\eta > 0$. If the graph G does not satisfy this, we subdivide all the edges into chains of smaller edges. Of course, this increases the number of edges and vertices in the graph, but since our competitive ratio is only a function of the number of pairs of terminals, this subdivision will not hurt our analysis. It is also clear that subdividing edges changes neither the optimum solution nor the behavior of the greedy algorithm. It also does not change any of the parameters we just defined above. This assumption will be used for simplicity when constructing balls in the graph; we will assume that no edge in the graph G has an endpoint inside and the other endpoint outside the balls (i.e. edges do not "jump over" the border of any ball).

6.2.2 *Preliminary results and preprocessing of the instance*

We describe here some key concepts that will be useful in the rest of the proof. We first introduce the following definition that gives much more structure to the instance $\mathcal{I}$.

Definition 6.2 (Canonical instance)**.** *For any $\alpha, \delta \geqslant 1$, any instance $\mathcal{I} = (G, \mathcal{P}, \mathcal{S})$ of online Steiner Forest in decreasing metrics is said to be (α, δ)-canonical with respect to a greedy algorithm* A *if the following holds:*

(a) There exist some real number $m > 0$ such that for any pair $p \in \mathcal{P}$, there exists an integer $1 \leqslant j \leqslant \log(k)/\delta$ such that

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p) = m/2^{j \cdot (\delta + 10)}.$$

(i.e. we have some geometric grouping of costs and two consecutive cost classes are separated by a multiplicative factor of at least $2^{\delta+10}$). We will say that cost classes are well separated, and define $c_j = m/2^{j\cdot(\delta+10)}$.

(b) All pairs in $\mathcal{P}$ have contraction at most α when running A on instance $\mathcal{I}$. We say that all pairs have low contraction.

(c) For any $i \geqslant 1$, the set of additional edges E_i contains exactly one edge with the same endpoints of the pair p_i and whose weight is exactly $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p_i)$ (i.e. we can assume A connected the pair p_i by simply using the single edge in E_i).

This definition suggests that we partition the set of pairs $\mathcal{P}$ into *cost classes* $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(j)}, \ldots, \mathcal{P}^{(\log(k)/(\delta+10))}$ where $\mathcal{P}^{(j)}$ is the subset of pairs in $\mathcal{P}$ that cost exactly $m/2^{(\delta+10)\cdot j}$. Note that there are at most $M \leqslant \log(k)/(\delta + 10)$ distinct cost classes. Given this definition, we first claim the following lemma. Intuitively, the lemma states that worst-case instances can be reduced to (α, δ)-canonical instances (for some big δ) at a multiplicative loss of $O(\log\log(k) + \log(\alpha))$.

Lemma 6.3. *For any instance $\mathcal{I}$ of size k, any greedy algorithm A and any $\alpha \geqslant 1$, there exists an (α, δ)-canonical instance $\mathcal{I}'$ of size $k' \leqslant k$ such that*

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}') \geqslant \frac{\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathcal{P}_{<\alpha})}{O(\log\log(k) + \log(\alpha))},$$

$$\delta \geqslant 100 \cdot (\log(\alpha) + \log\log(k)), \text{ and}$$

$$w(OPT(\mathcal{I}')) \leqslant w(OPT(\mathcal{I})).$$

Proof. By standard geometric grouping arguments we can assume that there are at most $\log(k)$ cost classes $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(\log(k))}$ such that the greedy algorithm A pays a cost of $w(\mathrm{OPT}(\mathcal{I}))/2^{i-1}$ for all pairs in $\mathcal{P}^{(i)}$. This first transformation already appeared in [75] and loses a constant factor. Then we consider these cost classes but we keep only the pairs of contraction at most α. Fix $\delta = 100 \cdot (\log\log(k) + \log(\alpha))$ We partition the pairs as follows:

$$\tilde{\mathcal{P}}^{(j)} = \bigcup_{i \equiv j \bmod(\delta+10)} \mathcal{P}^{(i)}$$

for all $0 \leqslant j < (\delta + 10)$. Since there are $(\delta + 10)$ groups, one of them represents at least a fraction $1/(\delta + 10)$ of the total cost. Keep only this group and transform the instance by adding additional edges to $\mathcal{S}$ as follows. Assume that we kept the group $\tilde{\mathcal{P}}^{(j)}$ then index the pairs in $\tilde{\mathcal{P}}^{(j)}$ by order of arrival i.e. $\tilde{\mathcal{P}}^{(j)} = \{p_1, \ldots, p_i, \ldots, p_{k'}\}$. For each pair $p_i \in \tilde{\mathcal{P}}^{(j)}$, we define the set of additional edges $E^{(i)}$ as a single edge whose endpoints are exactly the endpoints of the pair p_i and whose length is exactly what A paid for this pair in the original instance $\mathcal{I}$. This formally describes the instance $\mathcal{I}' = (G, \tilde{\mathcal{P}}^{(j)}, \mathcal{S})$. We claim that for any pair selected, the greedy algorithm A pays exactly the same cost for this pair regardless of which instance $\mathcal{I}$ or $\mathcal{I}'$ is running. We can prove this simple fact by induction of the number of pairs already arrived in $\mathcal{I}'$. If no pair has arrived this is clear. Now consider the

next pair p_i to arrive. Note that when running instance $\mathcal{I}'$, a path of length exactly $\text{cost}_{\text{A}}(\mathcal{I}, p_i)$ is available to connect p_i since we added an edge in $E^{(i)}$ of exactly this length connecting the endpoints of p_i. We claim that there cannot be a shorter path. Indeed, by induction we assumed that A paid the same in instance $\mathcal{I}$ and $\mathcal{I}'$ for previously arrived pairs hence it must be that the greedy algorithm A used the additional edges in $\mathcal{S}$ to connect previously arrived pairs. Because A is Greedy_i for some $i \in \{1, 2, 3\}$, it must be that the shortcuts added by A on instance $\mathcal{I}'$ so far are exactly edges of length 0 with endpoints at the endpoints of pairs arrived before p_i. Note that when running greedy on instance $\mathcal{I}$, the endpoints of previously arrived pairs must be at distance 0 when the new pair p_i arrives. Hence all the shortcuts available to A when receiving the pair p_i in instance $\mathcal{I}'$ are also available when receiving the pair p_i in instance $\mathcal{I}$. In particular, the shortest path taken by A for pair p_i in instance $\mathcal{I}'$ can only be longer than the path taken for pair p_i in instance $\mathcal{I}$.

One can see that in total we lose a multiplicative factor of at most $O(\delta)$ during the reduction. Finally, it is also clear that $w(\text{OPT}(\mathcal{I}')) \leqslant w(\text{OPT}(\mathcal{I}))$ and $k' \leqslant k$ since the graph G has not changed and we keep in $\mathcal{I}'$ only a subset of the pairs in $\mathcal{I}$. This ends the proof of the lemma. □

The rest of the section will be devoted to the proof of the following theorem.

Theorem 6.4. *Let* A *be any greedy algorithm that uses one of our 3 contraction rules. Let* $\mathcal{I}$ *be an* (α, δ)*-canonical instance (of size* k*) of online Steiner Forest in decreasing metrics. Assume* $\delta \geqslant 100 \cdot (\log(\alpha) + \log\log(k))$*. Then,*

$$\text{cost}_{\text{A}}(\mathcal{I}) \leqslant O(\log(k)) \cdot w(\mathit{OPT}(\mathcal{I})).$$

Note that Theorem 6.4 together with Lemma 6.3 imply Theorem 5.1. To see this, consider any instance $\mathcal{I}$. By losing a multiplicative factor of $O(\log\log(k) + \log(\alpha))$ and only considering the pairs in $\mathcal{P}_{<\alpha}$, we transform the instance $\mathcal{I}$ into an (α, δ)-canonical instance $\mathcal{I}'$ using Lemma 6.3. Then we apply Theorem 6.4 on instance $\mathcal{I}'$ and the total competitive ratio for pairs in $\mathcal{P}_{<\alpha}$ will be $O(\log(k)) \cdot O(\log\log(k) + \log(\alpha))$ which is exactly what we wanted to prove.

DUAL FITTING. A key technical ingredient in the proof of Theorem 6.4 will be dual fitting, which was also used in [3, 8] and is a common technique in competitive analysis. In the case of Steiner Forest, a natural way to do dual fitting without explicitly writing a linear program is to consider a set of balls in the graph $G = (V, E)$. For some vertex $v \in V$ and some radius $r > 0$, the ball $B(v, r)$ is the open ball of center v and radius r, i.e.

$$B(v, r) = \{u \in V \mid d_G(v, u) < r\}.$$

Denote by $\mathcal{T}$ the set of *terminals* which are the vertices that appear in at least one pair. For a terminal $u \in \mathcal{T}$, denote by $\overline{u}$ its *mate* which is the terminal that u should be connected to. Note that we can assume without loss of generality that all terminals have a only one mate, by duplicating vertices if this is not the case. Assume we have a collection $\mathcal{B} = \{B_1, B_2, \ldots, B_\ell\}$ of balls such that:

- All of the balls in $\mathcal{B}$ are pairwise disjoint, and
- Each ball B_i is centered at some terminal $u \in \mathcal{T}$ and its radius r_i satisfies $r < d_G(u, \overline{u})$.

Then if we define $y = \sum_i r_i$ the sum of radii of these balls it must be that

$$w(\mathrm{OPT}(\mathcal{I})) \geqslant y.$$

A reason for this is that any feasible solution to the Steiner Forest instance must connect u to $\overline{u}$. If we look at any ball $B_i \in \mathcal{B}$, then at its center lies at a terminal u, and we know that $\overline{u}$ is not in B_i. Therefore, to connect u to $\overline{u}$, a feasible solution needs to buy at least a path from the center to the border of B_i, which will have length at least r_i. Since all balls are pairwise disjoint, we know that these paths will be disjoint, and we can sum the lower bounds on each ball.

An alternative view of this is that the dual balls can be seen as a feasible solution (because the balls are pairwise disjoint) to the dual of the natural LP relaxation of the Steiner Forest problem. Then by weak duality, we know that any feasible solution has cost at least the cost of the dual. Hence we will also refer to a collection of balls as above as a *dual solution*. A dual solution is *feasible* if all the corresponding balls are pairwise disjoint and are all centered around the endpoints of some pairs. Using previous works, we obtain the following lemma, which is a formal and slightly more general version of Lemma 6.1.

Lemma 6.5. *Let $\mathcal{I}$ be an instance of online Steiner Forest in decreasing metrics. Consider a cost class $\tilde{\mathcal{P}} \subseteq \mathcal{P}$ of pairs. Let $\mathcal{P}' \subseteq \tilde{\mathcal{P}}$ be an arbitrary subset of $\tilde{\mathcal{P}}$. Let c be the constant such that*

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p) = c$$

for all $p \in \tilde{\mathcal{P}}$ and A a greedy algorithm. Fix an arbitrary radius

$$r \leqslant \frac{c}{8 \log(|\tilde{\mathcal{P}}|)}.$$

Then for any such radius r it is possible to construct a feasible dual solution $\mathcal{B}$ such that:

(a) All the balls $B \in \mathcal{B}$ have a radius equal to r,

(b) $|\mathcal{P}'| \leqslant 5 \cdot |\mathcal{B}|$,

(c) each pair $p \in \mathcal{P}'$ has at most one ball $B \in \mathcal{B}$ (denoted $B(p)$) centered around one of its endpoints, and

(d) all balls in $\mathcal{B}$ are centered around endpoints of pairs in $\mathcal{P}'$.

Proof. We start the proof by stating Moore's bound. We recall that the girth of a graph is the length of the shortest cycle in that graph.

Theorem 6.6 (Moore's bound, see [4, 18])**.** *Every graph with at least $2n^{1+\frac{1}{p}}$ edges has girth at most $2p$.*

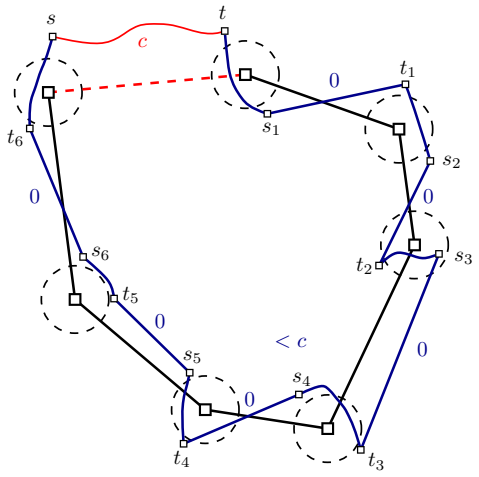

Figure 6.3: The girth argument.

We are interested in a subset of pairs $\mathcal{P}'$ that all belong to the same cost class $\mathcal{P}'$. That is, all the pairs in $\mathcal{P}'$ cost the same value c. We will place dual balls as desired in the statement of Lemma 6.5. To do this we will try to place a dual ball around the endpoint of a pair p as it arrives. If this is not possible without intersecting previously placed balls, we then skip the pair p. It will be clear by construction that the dual solution will be feasible. As we construct the feasible dual solution $\mathcal{B}$, we will maintain an auxiliary graph $G' = (V', E')$ which will be unweighted. The quantity of interest will be the number of edges in E'. The vertex set V' will correspond to a subset of terminals that appear in the pairs in $\mathcal{P}'$.

We proceed as follows. When a pair $p = \{s, t\}$ belonging to $\mathcal{P}'$ arrives, we try to place a ball of radius

$$r = \frac{c}{8 \log(|\mathcal{P}'|)}$$

around either one of s or t. If one of these two balls can be placed without intersecting balls previously placed in $\mathcal{B}$ we place it. We also add its center (either s or t) to the vertex set V' of the auxiliary graph G'. Otherwise, if none of these balls can be placed without intersecting balls already placed in $\mathcal{B}$ we do not add any ball. However, we identify one ball $B \in \mathcal{B}$ that intersects with the ball of radius r around s and another ball $B' \in \mathcal{B}$ for t. Let s' and t' be their centers. By construction we have that $s', t' \in V'$. We can add the edge $\{s', t'\}$ in the auxiliary graph. We have the following claim.

Claim 6.7. *The girth of G' is at least $2 \log(|\mathcal{P}'|)$.*

Proof of Claim 6.7. Suppose that the claim does not hold, and consider a cycle of length $\ell < 2 \log(|\mathcal{P}'|)$ in G' as in Figure 6.3. Consider the last edge that was added to the cycle. Suppose it was added because of terminal pair $\{s, t\}$ (in red on Figure 6.3). Each other edge in the cycle must have been created by a previous pair $\{s_m, t_m\}$ for $1 \leqslant m < \ell$. By triangle inequality we have, for all $m < \ell$ that

$$d_G(t_m, s_{m+1}) \leqslant d_G(t_m, u) + d_G(u, s_{m+1}) \leqslant 4r,$$

where u is the center of the ball that was close to both t_m and s_{m+1}. Similarly, $d_G(t, s_1) \leqslant 4r$ and $d_G(t_{m-1}, s) \leqslant 4r$. Since all pairs $\{s_m, t_m\}_{m \leqslant \ell-1}$ arrived before $\{s, t\}$, it must be that the shortest path in the current metric G' (when the pair $\{s, t\}$ arrives) satisfies

$$d_{G'}(s_m, t_m) = 0$$

(this holds for all three contraction rules). Hence, by going through the terminals $t, s_1, t_1, s_2, t_2, \ldots, s_{\ell-1}, t_{\ell-1}, s$, greedy could have paid at most

$$(4r) \cdot \ell < (8r) \cdot \log(|\mathcal{P}'|) \leqslant c,$$

which is a contradiction on the fact that greedy should always take the shortest path available. Hence the girth in G' is at least $2\log(|\mathcal{P}'|)$. We note that this whole argument also holds if we are in the case of Steiner Forest in decreasing metrics where additional edges are revealed over time. Indeed, these additional edges cannot make a shortest path longer. □

Applying Moore's bound to the graph G' (which has at most $|\mathcal{P}'|$ vertices by construction) gives that $|E'| < 4|V'| \leqslant 4|\mathcal{P}'|$. Since the cardinality of $\mathcal{P}'$ equals the number of vertices $|V'|$ in G' plus the number of edges $|E'|$ in G', and $|V'| = |\mathcal{B}|$ property (b) of Lemma 6.5 follows. Properties (a), (c) and (d) follow by construction. □

We are now ready to start the overview of our main proof.

6.2.3 *Overview of the proof*

Recall that we aim to prove Theorem 6.4. By assumption we have that all pairs have contraction at most α and that the cost classes are well separated. Recall that this means the multiplicative gap between two consecutive cost classes is at least 2^δ for some $\delta \geqslant 100 \cdot (\log\log(k) + \log(\alpha))$. We use M to denote the number of cost classes where class j is denoted by $\mathcal{P}^{(j)}$ for $1 \leqslant j \leqslant M$.

The goal will be to construct a feasible dual solution $\mathcal{B}$ that has some special properties. This dual solution will be constructed by taking a subset of the dual balls in each of the $\log(k)$ dual solutions constructed with the technique of Awerbuch, Azar, and Bartal [8]. In the end we will charge a portion of the cost that A pays to the dual solution $\mathcal{B}$. The remaining portion of the cost A pays will be handled by an inductive argument. To this end we will have a charging scheme, $\mathrm{charge} : \mathcal{P} \mapsto \mathbb{R}_+$ that will redistribute $cost_A(\mathcal{I})$ amongst the terminal pairs.

Precisely, the total cost that a pair p carries will be

$$\mathrm{charge}(p) \cdot \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p).$$

Hence we see the charge as an additional multiplicative factor on the cost of a given pair (initially, the charge is set to 1 for all pairs).

Note that we might sometimes decrease or increase the charge of a pair or transfer the cost, but we will always make sure that when a charge of a pair p is decreasing, the charge of some other pairs are increasing accordingly so that

no cost is lost. For any set S of terminal pairs, we will let $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, S, \mathrm{charge})$ denote the total *charged* cost carried by pairs in S. Formally,

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, S, \mathrm{charge}) = \sum_{p \in S} \mathrm{charge}(p) \cdot \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p).$$

In the proof the pairs in $\mathcal{P}$ will be classified into three types: *surviving* pairs, *charged* pairs, and *dangerous* pairs. The *surviving* pairs will contain pairs p such that there is a ball $B(p) \in \mathcal{B}$ centered around one of the endpoints. Intuitively these pairs are good for us since we can charge their cost directly to the dual ball $B(p)$. The other pairs will be by default classified as *non-surviving*. Non-surviving pairs are further partitioned into two subsets, charged or dangerous pairs. *Charged* pairs are those pairs that have their charge set to o (i.e. $\mathrm{charge}(p) = 0$). Intuitively, they are also in an excellent situation for us since it means that we were able to transfer their cost entirely to other pairs. We do not need to count them in our total cost anymore. Finally, the *dangerous* pairs are those pairs have neither a charge equal to 0 nor a dual ball in $\mathcal{B}$ centered at one of the endpoints. These pairs will be handled carefully via an inductive argument since we cannot charge them to the dual solution nor to some other pair. To keep careful track of all these elements, we will store a triple $(\mathcal{B}, \mathrm{charge}, D)$ where charge is the charge function as described above, $\mathcal{B}$ is a feasible dual solution and $D \subseteq \mathcal{P}$ a set of *dangerous* pairs. The family of dual balls will be a union of subsets of dual balls $\mathcal{B}^{(j)}$ for $1 \leqslant j \leqslant M$. Each of the balls in $\mathcal{B}^{(j)}$ will account for a subset of pairs in $\mathcal{P}^{(j)}$ and have some radius of roughly

$$r_j = \frac{c_j}{8 \log(k_j)},$$

where c_j is the cost of pairs in $\mathcal{P}^{(j)}$ and $k_j = |\mathcal{P}^{(j)}|$. This choice of radius is coming from previous work, summarized in Lemma 6.5.

Note that in our procedure, it might be that some pairs are not yet classified into one of the three categories (surviving, charged, or dangerous). However, at the beginning of iteration j, all pairs in cost classes $j' < j$ will be classified. The procedure contains two main steps.

STEP 1. In this step, we start taking into account interactions in-between cost classes. Informally, we do an iterative procedure from $j = 1$ to M, where we try to build the dual solution from top to bottom. When we start iteration j of this procedure we have a feasible dual solution composed of $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(j-1)}$ (of radius $\Theta(r_j)$ with r_j specified above) centered around pairs in $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(j-1)}$. All the pairs in $\mathcal{P}^{(j)}$ that are not yet classified are guaranteed to be far from the dual balls already in place. We then look at pairs in $\mathcal{P}^{(j)}$ that are not yet classified, and build a dual solution $\mathcal{B}^{(j)}$ around these pairs using Lemma 6.5. Because unclassified pairs are far from previously placed balls in $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(j-1)}$, it is guaranteed that this new dual solution $\mathcal{B}^{(j)}$ will not overlap with the previous dual solution. We then proceed as follows. For any ball $B \in \mathcal{B}^{(j)}$ that we just added, we let $B_{\mathcal{P}}$ denote the set

of pairs of $\bigcup_{j'>j} \mathcal{P}^{(j')}$ (note that we only consider pairs of smaller cost) such that one of its endpoints is at a distance at most

$$r \cdot \left(1 + \frac{1}{200 \cdot \log^2(K)}\right)$$

from the center of B (denoted $p(B)$). Here we note that for a technical reason, K is only an upper bound on the real number of pairs k (i.e. $K \geqslant k$). We will let $\partial B_{\mathcal{P}}$ denote the set of pairs of $B_{\mathcal{P}}$ that have one endpoint at a distance of at least

$$r \cdot \left(1 - \frac{1}{200 \cdot \log^2(K)}\right)$$

from the center of B. These pairs are on the border of B hence the choice of notation. With a similar analogy, we will denote the interior of $B_{\mathcal{P}}$ by $\mathring{B}_{\mathcal{P}}$. This is the complement of $\partial B_{\mathcal{P}}$ in $B_{\mathcal{P}}$, i.e.

$$\mathring{B}_{\mathcal{P}} = B_{\mathcal{P}} \setminus \partial B_{\mathcal{P}}.$$

Then for the current ball $B \in \mathcal{B}^{(j)}$ at hand, centered at an endpoint of p, we look at the total charged cost of the pairs in $\mathring{B}_{\mathcal{P}}$ and make a case distinction based on this value. If this cost is more than $\text{polylog}(K)$ times the charged cost of the pair p, then we can safely set the charge of p to 0, delete the dual ball B, and increase the charge of pairs in $B_{\mathcal{P}}$ to account for this lost cost. Note that the charges of pairs in $\mathring{B}_{\mathcal{P}}$ increase by at most a multiplicative $(1 + \frac{1}{\text{polylog}(K)})$). This is pictured in the top left corner of Figure 6.2. Since the number of cost classes is at most $M < \log(K)$ such accumulation of charges is not a problem (note that the charge of each pair can increase at most once per cost class in this way, since we only charge pairs inside a dual ball and the dual balls in a single cost class are pairwise disjoint).

On the contrary, if the charged cost of pairs in $\mathring{B}_{\mathcal{P}}$ is less than $\text{polylog}(K)$ times the charged cost of the pair p, we first halve the radius of B to get B'. If the charged cost of the pairs inside $B'_{\mathcal{P}}$ is at most a constant factor times the charged cost of p, we classify all the pairs in $B'_{\mathcal{P}}$ as *charged* and charge their cost to the pair p. Note that the charge of p only increases by a constant factor when doing this, and this happens at most once per pair p (when we place the dual ball around p). In addition we update B to be B', and we classify the pair p as *surviving*. If on the other hand, the charge of the pairs inside $B'_{\mathcal{P}}$ is larger than a specified constant factor times the charge of B, we scale the radius of B' up until we reach a point where most of the cost in $B'_{\mathcal{P}}$ is carried by $\mathring{B}'_{\mathcal{P}}$ and not $\partial B'_{\mathcal{P}}$. Then we mark all the pairs in $B'_{\mathcal{P}}$ as *dangerous* and add them to the set D. We update B to be the ball B' and classify p as *surviving*. This case is pictured in the bottom right corner of Figure 6.2. Note that if the ball B is not deleted, then all the pair in $B_{\mathcal{P}}$ will be classified as either charged or dangerous. In particular, we will never try to place a dual ball around these pairs in the following iterations. We do this procedure for all the balls $B \in \mathcal{B}^{(j)}$ and then move to iteration $(j+1)$. This step is handled in Subsection 6.2.4.

STEP 2. After Step 1, we end with a feasible dual solution $\mathcal{B}$ consisting of the balls placed around surviving pairs, a charge function , and a set D of dangerous pairs. Additionally, we guarantee that no pair is overcharged.

The pairs that are not dangerous are easily accounted for by the dual solution $\mathcal{B}$. Indeed, the surviving pairs still have a dual ball around an endpoint, and the charged pairs have their cost entirely redistributed to other pairs. The only problem might come from dangerous pairs. However, because of how we constructed the dual solution $\mathcal{B}$ and the set D, we will be able to cluster the dangerous pairs into disjoint sub-instances that are contained in dual balls corresponding to bigger cost classes. These instances are disjoint, and the crux of the argument is to show a statement of the form:

If the greedy algorithm were to run separately on each sub-instance, then the cost greedy would pay for these pairs would be the same cost that it was paying for these pairs in the bigger instance $\mathcal{I}$.

Hence we can argue that the total cost incurred for dangerous pairs is at most the sum of costs paid by greedy on each sub-instance separately. This helps because we only put pairs in D in the case that their charged cost was bounded by $\text{polylog}(K)$ times the charged cost of the pair that created the ball B that contains them. As a result, we have a strong upper bound on the number of pairs k' in each smaller sub-instance $\mathcal{I}'$. To finish the proof, we need to bound the cost of the offline optimum for each sub-instance $\mathcal{I}'$. We note that because all the pairs in D are in the *interior* of B (i.e. far from the border of B), we can modify the metric of the graph G at the border of B. This will not change the behavior of greedy for the pairs in D because the border is way too far from the interior of B for greedy to be tempted to use the modified metric (recall that greedy always takes the shortest path). We will define a new graph G', which is the graph induced by vertices in B. We also say that all the vertices *exactly* on the border of B are all at a distance 0 from each other (see bottom right corner of Figure 6.2). With this modification, it becomes clear that the offline optimum cost on instance $\mathcal{I}'$ is at most the cost paid by $\text{OPT}(\mathcal{I})$ inside B, which we will denote by $w(\text{OPT}(\mathcal{I}) \cap B)$. Hence the offline optimum cost for each sub-instance is at most what the global optimum pays locally inside the ball that created the sub-instance. Since all balls in $\mathcal{B}$ are disjoint, these areas never overlap; hence the *sum* of all local optima is at most the global optimum of instance $\mathcal{I}$. Using this observation, we handle the cost incurred by pairs in D via a delicate induction hypothesis on the number of cost classes in the instance. This induction is described formally in Subsection 6.2.5.

6.2.4 *Building a balanced dual solution*

We formalize here Step 1 of the previous subsection. We give a formal definition of all the properties that our triple $(\mathcal{B}, \text{charge}, D)$ should satisfy. Note that we are also given an upper bound K on the real number of pairs k (this is for technical reasons for handling the induction in the next subsection). Recall that for a ball $B \in \mathcal{B}^{(j)}$ we denote by $B_{\mathcal{P}}$ the set of pairs of $\bigcup_{j'>j} \mathcal{P}^{(j')}$ such that one of its endpoint is at distance at most

$$r \cdot \left(1 + \frac{1}{200 \cdot \log^2(K)}\right)$$

from the center of B. We also have similar definitions for $\partial B_{\mathcal{P}}$ and $\mathring{B}_{\mathcal{P}}$. Now we can state the main definition of this subsection. Intuitively, conditions (a)

and (b) state that $\mathcal{B}$ is a feasible dual solution whose dual balls have radii large enough. Condition (c) states that the total charged cost of dangerous pairs inside the ball B is never much more than $\mathrm{polylog}(K)$ times the charged cost of the pair that created the ball B. Similarly, condition (d) states that the charged cost of dangerous pairs on the border of B is not more than $1/\log(K)$ times the charged cost of dangerous pairs strictly inside B. The last condition (e) states that no pair was charged too many times.

Definition 6.8 (Balanced dual solution)**.** *A balanced dual solution for an (α, δ)-canonical instance $\mathcal{I}$ with respect to algorithm* A *is a quadruple $(\mathcal{B}, \mathrm{charge}, D, K)$ such that:*

(a) All balls $B \in \mathcal{B}$ are pairwise disjoint and $D \subseteq \bigcup_{B\in\mathcal{B}} B_{\mathcal{P}}$. Moreover, $\mathcal{B}$ is partitioned into M sub-collections of balls $\mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(M)}$ such that,

(b) For every $j \geqslant 1$, each ball in $\mathcal{B}^{(j)}$ has a radius r' that satisfies $r_j/2 \leqslant r' \leqslant r_j = c_j/8\log(k_j)$, with c_j the cost associated to cost class $\mathcal{P}^{(j)}$ and $k_j = |\mathcal{P}^{(j)}|$.

(c) For every ball $B \in \mathcal{B}^{(j)}$

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}} \cap D, \mathrm{charge}) \leqslant 10 \cdot \mathrm{charge}(p(B)) \cdot c_j \cdot \log^{10}(K),$$

(d) For any ball $B \in \mathcal{B}^{(j)}$

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \partial B_{\mathcal{P}} \cap D, \mathrm{charge}) \leqslant \frac{10 \cdot \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}} \cap D, \mathrm{charge})}{\log(K)},$$

(e) For any $j > 0$, any pair $p \in \mathcal{P}^{(j)}$,

$$\mathrm{charge}(p) \leqslant \begin{cases} 55 \cdot e^5 \text{ if } p \text{ is a surviving pair,} \\ 0 \text{ if } p \text{ is a charged pair,} \end{cases}$$

$\mathrm{charge}(p) \geqslant 1$ *if p is a surviving or dangerous pair, and finally if p is a dangerous pair that belongs to $B_{\mathcal{P}}$ for some $B \in \mathcal{B}^{(j)}$ then*

$$\mathrm{charge}(p) \leqslant \left(1 + \frac{5}{\log(K)}\right)^{j-1}.$$

The main result of this subsection will be that it is always possible to find a balanced dual solution.

Lemma 6.9. *Given an (α, δ)-canonical instance $\mathcal{I}$ with respect to* A*, a balanced dual solution $(\mathcal{B}, \mathrm{charge}, D, K)$ always exists provided that $\delta \geqslant 100 \cdot (\log(\alpha) + \log\log(K))$, the number of cost classes M satisfies $M \leqslant \log(K)$, and the number of pairs k satisfies $k \leqslant K$.*

Proof. We build the solution quadruple $(\mathcal{B}, \mathrm{charge}, D, K)$ with an iterative procedure from $j = 1$ to $j = M$ (recall that M is the number of cost classes in $\mathcal{I}$) that will maintain the following invariants at the beginning of any iteration j:

(i) For any $j' < j$, the dual balls in $\mathcal{B}^{(j')}$ are already fixed and all pairs in $\mathcal{P}^{(j')}$ are already classified as either surviving, charged, or dangerous. The dual balls of $\mathcal{B}^{(j')}$ satisfy conditions (b), (c), (d) of Definition 6.8.

(ii) For any $j' < j$, the charge of pairs in $\mathcal{P}^{(j')}$ satisfy condition (e) of Definition 6.8.

(iii) For any $j' \geqslant j$, the pairs in $\mathcal{P}^{(j')}$ can be classified as either dangerous or charged, or not be classified yet. No pair of $\mathcal{P}^{(j')}$ is classified as surviving yet. We also have $\mathcal{B}^{(j')} = \emptyset$.

(iv) For any $j' \geqslant j$, all pairs in $\mathcal{P}^{(j')}$ classified as either dangerous or charged satisfy condition (e) in Definition 6.8. The pairs p that are not yet classified satisfy

$$1 \leqslant \text{charge}(p) \leqslant \left(1 + \frac{5}{\log(K)}\right)^{j-1}.$$

(v) For any $j' \geqslant j$, all pairs in $\mathcal{P}^{(j')}$ not yet classified do not belong to any set $B_{\mathcal{P}}$ for some already existing ball (i.e. unclassified pairs are far from the balls already placed).

We start iteration j by considering the pairs in $\mathcal{P}^{(j)}$ that are not yet classified. To avoid confusion, let us denote by $\mathcal{P}'^{(j)} \subseteq \mathcal{P}^{(j)}$ this set of unclassified pairs. Using Lemma 6.5, we get a dual solution $\mathcal{B}^{(j)}$ of balls all of radius

$$r_j = \frac{c_j}{8\log(k_j)}$$

that are all centered around endpoints of pairs in $\mathcal{P}'^{(j)}$ and such that no pair in $\mathcal{P}'^{(j)}$ has more than one ball centered around an endpoint. All the pairs in $\mathcal{P}'^{(j)}$ that do not have a dual ball can already be classified as charged. We decrease the charge of these pairs to 0 increase the charge of the other pairs in $\mathcal{P}'^{(j)}$ to compensate. Since we have that $|\mathcal{P}'^{(j)}| \leqslant 5 \cdot |\mathcal{B}^{(j)}|$ and invariant (iv), we have that the charge of the remaining pairs in $\mathcal{P}'^{(j)}$ will be at most

$$5 \cdot \left(1 + \frac{5}{\log(K)}\right)^{(j-1)} \leqslant 5 \cdot \left(1 + \frac{5}{\log(K)}\right)^{M} \leqslant 5 \cdot e^5 \tag{6.1}$$

after this step (we use the assumption that $M \leqslant \log(K)$ and $(1+x) \leqslant e^x$). Then for each new ball $B \in \mathcal{B}^{(j)}$ that we created around the endpoint of a pair p we consider the following process with two cases. We consider the pairs of $\mathring{B}_{\mathcal{P}}$ (i.e. pairs of smaller cost in the *interior* of B) and proceed to a case distinction on the value of their total charged cost

$$\text{cost}_{\text{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}}, \text{charge})$$

that we denote Σ.

CASE 1 (TOP LEFT CORNER IN FIGURE 6.2). If

$$\Sigma > 10 \cdot \text{charge}(p) \cdot c_j \cdot \log^{10}(K), \tag{6.2}$$

that we can rephrase intuitively as "B does not satisfy condition (c)" then we simply erase the dual ball B and charge all the cost $\text{charge}(p) \cdot c_j$ to the pairs in $\mathring{B}_{\mathcal{P}}$ proportionally to their weight. Note that by doing this, all the pairs in $\mathring{B}_{\mathcal{P}}$ see their charge increasing by a multiplicative factor of $\left(1 + \frac{1}{(10 \log^{10}(K))}\right) < \left(1 + \frac{5}{\log(K)}\right)$. The pair p is now classified as charged. Note that the pairs in $\mathcal{P}^{(j'>j)}$ have now a charge that is at most $\left(1 + \frac{5}{\log(K)}\right)^j$ by applying invariant (iv). This will be the only place where their charge can increase during iteration j and note that this happens only once per iteration since only the pairs inside a ball $B \in \mathcal{B}^{(j)}$ are charged and the balls in $\mathcal{B}^{(j)}$ are pairwise disjoint. Therefore invariant (iv) will hold at the beginning of iteration $j+1$ for these pairs.

CASE 2. In this case we assume that $\Sigma \leqslant 10 \cdot \text{charge}(p(B)) \cdot c_j \cdot \log^{10}(K)$. We consider the ball B' with the same center as B but with only half the radius of B (i.e. B' is a ball centered at $p(B)$ with radius $r = r_j/2$). We look at the total charged cost of pairs in $B'_{\mathcal{P}}$ and proceed to a case distinction on this value (note that we do not consider only the strict interior of B but also its border). Denote by Σ' this new value, i.e.

$$\Sigma' = \text{cost}_{\text{A}}(\mathcal{I}, B'_{\mathcal{P}}, \text{charge}).$$

We proceed by a sub-case distinction of the value of Σ'.

SUB-CASE 2(A). If $\Sigma' \leqslant 10 \cdot \text{charge}(p(B)) \cdot c_j$ then we simply charge the cost of all the pairs in in $B'_{\mathcal{P}}$ to the pair p. In this case the pairs in $B'_{\mathcal{P}}$ become charged pairs. Note that the charge of p in that case is multiplied by at most 11. We also set the dual ball B to B' (i.e. we scale down the radius of B by a factor 2). By Equation 6.1, we get that the charge of p will be at most

$$\text{charge}(p) \leqslant 11 \cdot 5 \cdot e^5 = 55e^5.$$

This will be the final charge of this pair hence invariant (ii) will be satisfied.

SUB-CASE 2(B) (BOTTOM RIGHT CORNER IN FIGURE 6.2). If $\Sigma' > 10 \cdot \text{charge}(p(B)) \cdot c_j$, then we will try to increase the radius r of B' by an increment of

$$\Delta r = \frac{r_j}{200 \cdot \log^2(K)},$$

until we find a radius which satisfies condition (d) for the ball B'. We denote by $B^{(t)}$ the ball obtained after increasing the radius of B' t times by Δr. Hence the radius of $B^{(t)}$ is $r_t = r_j/2 + t \cdot (\Delta r)$. Each time we increase r, we check if

$$\text{cost}_{\text{A}}(\mathcal{I}, \partial B^{(t)}_{\mathcal{P}}, \text{charge}) \leqslant \frac{10 \cdot \text{cost}_{\text{A}}(\mathcal{I}, \mathring{B}^{(t)}_{\mathcal{P}}, \text{charge})}{\log(K)}. \tag{6.3}$$

If this is the case we stop and add all the pairs in $B_{\mathcal{P}}^{(t)}$ to D. The pairs in $B_{\mathcal{P}}^{(t)}$ become in this case dangerous pairs. If not we continue increasing the radius r until Equation 6.3 holds. We claim that this process will stop before reaching $r_t = r_j$. To see this we first prove that $B_{\mathcal{P}}^{(t)} \cap \partial B_{\mathcal{P}}^{(t+6)} = \emptyset$ for any $t \geqslant 0$. Indeed if a pair $p_i \in \mathcal{P}^{(j')}$ with $j' > j$ belongs to $B_{\mathcal{P}}^{(t)} \cap \partial B_{\mathcal{P}}^{(t+6)}$ then it must be that one of the two endpoints of $p_i = \{s_i, t_i\}$ belongs to $B^{(t)}$ while the other belongs to the border $\partial B^{(t+6)}$ since $B^{(t)} \cap \partial B^{(t+6)} = \emptyset$. Hence the distance between the two endpoints of $p_i = \{s_i, t_i\}$ satisfies:

$$\begin{aligned} d_G(s_i, t_i) &\geqslant (r_t + 6\Delta r) \cdot \left(1 - \frac{1}{200 \cdot \log^2(K)}\right) - r_t \cdot \left(1 + \frac{1}{200 \cdot \log^2(K)}\right) \\ &\geqslant 5\Delta r - \frac{2r_t}{200 \cdot \log^2(K)} \geqslant \frac{r_j}{200 \cdot \log^2(K)}. \end{aligned}$$

where the last inequality comes from the fact that $r_t \leqslant r_j$. However, we assumed that the instance $\mathcal{I}$ is (α, δ)-canonical with $\delta \geqslant 100 \cdot (\log(\alpha) + \log\log(K))$ and this implies that all pairs have contraction at most α which means that the pair p_i should have costed A at least

$$\begin{aligned} \frac{r_j}{\alpha(200 \cdot \log^2(K))} \geqslant \frac{c_j}{\alpha(8 \cdot 200 \cdot \log^3(K))} &> \frac{c_j}{2^{10} \cdot \alpha^{100} \log^{100}(K)} \\ &> \frac{c_j}{2^{\delta+10}} = c_{j+1}\,, \end{aligned}$$

which is a contradiction. Thus $B_{\mathcal{P}}^{(t)} \cap \partial B_{\mathcal{P}}^{(t+6)} = \emptyset$. Since we assume the process does not stop, we have that

$$\begin{aligned} \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \partial B_{\mathcal{P}}^{(t+6)}, \mathrm{charge}) &> \frac{10 \cdot \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}}^{(t+6)}, \mathrm{charge})}{\log(K)} \\ &\geqslant \frac{10 \cdot \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \mathrm{charge})}{\log(K)}. \end{aligned}$$

Together with the fact that $B_{\mathcal{P}}^{(t)} \cap \partial B_{\mathcal{P}}^{(t+6)} = \emptyset$ we get

$$\begin{aligned} \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t+6)}, \mathrm{charge}) &\geqslant \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \mathrm{charge}) + \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \partial B_{\mathcal{P}}^{(t+6)}, \mathrm{charge}) \\ &\geqslant \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \mathrm{charge}) \cdot \left(1 + \frac{10}{\log(K)}\right)\,. \end{aligned}$$

This implies that, for all $t \geqslant 0$,

$$\begin{aligned} \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \mathrm{charge}) &\geqslant \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, B_{\mathcal{P}}^{(0)}, \mathrm{charge}) \cdot \left(1 + \frac{10}{\log(K)}\right)^{\lfloor t/6 \rfloor} \\ &= \Sigma' \cdot \left(1 + \frac{10}{\log(K)}\right)^{\lfloor t/6 \rfloor}. \end{aligned}$$

However, we have by assumption in our case that

$$\frac{\Sigma}{\Sigma'} \leqslant \log^{10}(K),$$

thus for $t = 60\log(K)\log\log(K)$, we can write simultaneously

$$\text{cost}_{\text{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \text{charge}) \geqslant \Sigma' \cdot \left(1 + \frac{10}{\log(K)}\right)^{10\log(K)\log\log(K)} \tag{6.4}$$

$$> \Sigma' \cdot \log^{10}(K) \geqslant \Sigma, \tag{6.5}$$

and

$$r_t = r_j/2 + t \cdot (\Delta r) = r_j \cdot \left(\frac{1}{2} + \frac{60\log(K)\log\log(K)}{200 \cdot \log^2(K)}\right) < r_j. \tag{6.6}$$

This is a contradiction since by Equation 6.6 we should have $B_{\mathcal{P}}^{(t)} \subseteq B_{\mathcal{P}}$ and by Equation 6.5 we have $\text{cost}_{\text{A}}(\mathcal{I}, B_{\mathcal{P}}^{(t)}, \text{charge}) > \text{cost}_{\text{A}}(\mathcal{I}, B_{\mathcal{P}}, \text{charge})$. Thus the process must stop before reaching $t = 60\log(K)\log\log(K)$, hence before reaching $r_t = r_j$.

CORRECTNESS. We note that Sub-case 2(b) is the only case in which we create dangerous pairs. By construction, properties (c) and (d) will hold for any ball. This is because if a ball B remains around a pair p at the end of the procedure, either all the pairs inside are charged to p, or they become dangerous pairs. In the case where the pairs inside become dangerous the procedure described in Sub-case 2(b) stops exactly when properties (c) and (d) are satisfied. Property (b) is also satisfied since the only place where a radius can be modified is in Sub-case 2(b). In this case, we show that after halving the radius, it cannot grow for too long before the number of pairs in the border is less concentrated than in the interior. This means that since the balls in $\mathcal{B}^{(j)}$ were disjoint when taken with radius r_j, they also have to be disjoint if the radius is slightly smaller. Condition (a) is also satisfied. To see this, note that the balls in $\mathcal{B}^{(j)}$ cannot intersect balls installed before since in Case 2, which is the only case where a ball survives, for any ball $B \in \mathcal{B}^{(j'<j)}$, all the pairs in $B_{\mathcal{P}}$ become classified as either dangerous or charged (invariant (v)). In particular, we will never try to place a ball around these pairs since we only place dual balls around pairs that are not classified yet. Since $B_{\mathcal{P}}$ encompasses a region slightly bigger than the ball B, the smaller dual balls will not intersect B. We also have that $D \subseteq \bigcup_{B\in\mathcal{B}} B_{\mathcal{P}}$ since dangerous pairs are only created in Sub-case 2(b) where we consider pairs inside a ball B. It remains to show that condition (e) holds. Note that for charged pairs, this is clear. For surviving pairs, we showed that our process maintains invariant (ii); hence after the end of the procedure, condition (e) is satisfied. For dangerous pairs, note that once a pair becomes dangerous, its charge will never increase anymore. Hence this shows that invariant (ii) also holds for these pairs, and in particular, at the end of the process, condition (e) holds. □

6.2.5 *Inductive proof using balanced dual solutions*

Given the previous subsection, we are ready to state the main induction. Recall that in a balanced dual solution $(\mathcal{B}, \text{charge}, D)$, all the pairs except the dangerous pairs (the set D) are accounted for by the dual balls in $\mathcal{B}$. Our induction will precisely take advantage of this. In the following, for any ball

B and instance $\mathcal{I}$, we denote by $w\left(\mathrm{OPT}(\mathcal{I}) \cap B\right)$ the total cost of edges that are contained in B and bought by $\mathrm{OPT}(\mathcal{I})$ (note that we assume that edges in G are arbitrarily small, so no edge is crossed by the border of B). Note that we also have the straightforward bound $r \leqslant w\left(\mathrm{OPT}(\mathcal{I}) \cap B\right)$ (with r being the radius of B) since the optimum solution needs to connect at least the center of the ball B to its border. For a collection of disjoint balls $\mathcal{B}$, we define

$$w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}\right) = \sum_{B \in \mathcal{B}} w\left(\mathrm{OPT}(\mathcal{I}) \cap B\right).$$

Lemma 6.10. *Let $\mathcal{I}$ be an (α, δ)-canonical instance with respect to algorithm* A. *Assume that its size is $k \leqslant K$, and $\delta \geqslant 100 \cdot (\log(\alpha) + \log\log(K))$. Assume there are $M \leqslant \log(K)$ distinct cost classes when running* A *on instance $\mathcal{I}$. Let $\left(\mathcal{B} = \bigcup_{j=1}^{M} \mathcal{B}^{(j)}, \mathrm{charge}, D, K\right)$ be a balanced dual solution. Then we have that*

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}) \leqslant (880e^5) \cdot \left(\sum_{j=1}^{M} \log(k_j) \cdot w\left(\mathit{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right) + e^{200+20M/\log(K)} \cdot \left(\sum_{j=1}^{M} \delta \cdot (M-j) \cdot w\left(\mathit{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right),$$

where k_j is the number of terminals in cost class j for $j = 1, \ldots, M$.

Intuitively, the first term of the right-hand side of the inequality corresponds to the pairs that are either surviving or charged and can be charged to the dual. The second term corresponds to the cost paid by A because of dangerous pairs in D. The proof of Lemma 6.10 will be done by induction on the number of cost classes M.

Proof. The base case of the induction is for $M = 0$ in which case the statement is vacuously true (the instance is empty). Hence assume $M > 0$ and that the statement is true for any (α, δ)-canonical instance with $M' < M$ cost classes, and satisfying the conditions of the lemma.

Recall that in a balanced dual solution $(\mathcal{B} = \bigcup_{j=1}^{M} \mathcal{B}^{(j)}, \mathrm{charge}, D, K)$ (that exists by Lemma 6.9) we have that $\mathrm{charge}(p) \leqslant 55e^5$ for all surviving pairs p, that $\mathcal{B}$ is a feasible dual solution, and that the radius of each ball $B \in \mathcal{B}^{(j)}$ is at least $1/(16\log(k_j))$ times the cost of corresponding pair. Hence we have that the total cost of surviving or charged pairs (denote this set $\mathcal{P}'$) is at most

$$\begin{aligned}\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathcal{P}') &\leqslant (16 \cdot 55e^5) \cdot \left(\sum_{j=1}^{M} \log(k_j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right) \\ &= (880e^5) \cdot \left(\sum_{j=1}^{M} \log(k_j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right).\end{aligned}$$

All that remains to do is to upper bound the total cost incurred by A because of pairs in D. By property (a) of Definition 6.8, it suffices to consider each

ball $B \in \mathcal{B}$ and the dangerous pairs it contains. Hence fix a ball $B \in \mathcal{B}^{(j)}$ of radius r. We consider the instance $\mathcal{I}' = (G', \mathcal{P}', \mathcal{S}')$ that is defined as follows (see bottom right corner of Figure 6.2).

- The graph G' is the graph induced by G on the vertex set B, in which we add an edge of cost 0 between any pair of vertices at a distance exactly r from the center of B. We denote by E' this set of additional edges of weight o (see Figure 6.2 in the bottom right corner).
- $\mathcal{P}'$ is the set of pairs in $\mathring{B}_{\mathcal{P}} \cap D$, given in the same relative order to algorithm A. Note that these pairs must have both endpoints inside B because all pairs have a low contraction, and the cost paid by these pairs is much smaller than the radius of the ball. Hence traveling from the interior of B to the border is already way too far. We also emphasize that we ignore pairs in $\partial B_{\mathcal{P}} \cap D$.
- The set of additional edges $\mathcal{S}'$ contains all the edges in $\mathcal{S}$ that we revealed just before reading a pair in $\mathcal{P}'$. Moreover, the edges in $\mathcal{S}'$ are revealed in the same way to A, that is, if an edge $e \in \mathcal{S}$ is revealed just before a pair $p \in \mathcal{P}'$, then it is also revealed just before p in instance $\mathcal{I}'$.

We then make the following claim.

Claim 6.11. *Given instance $\mathcal{I}'$ we have that:*

(a) For all $p \in \mathcal{P}'$, $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}', p) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p)$. In particular, the cost incurred by A because of pairs in $\mathcal{P}'$ is exactly the same as the cost that A would pay when running on instance $\mathcal{I}'$.

(b) $w(OPT(\mathcal{I}')) \leqslant w(B \cap OPT(\mathcal{I}))$.

Proof. To see (b), let us buy the edge set $(\mathrm{OPT}(\mathcal{I}) \cap B) \cup E'$. It is clear that this costs at most $w(B \cap \mathrm{OPT}(\mathcal{I}))$ since all edges in E' have cost o. We claim that this is a feasible solution for the instance $\mathcal{I}'$. Consider any pair $p = \{s, t\} \in \mathcal{P}'$. It must be that $\mathrm{OPT}(\mathcal{I})$ contains a path between s and t. If this path does not leave the ball B then it is contained in the edge set $(\mathrm{OPT}(\mathcal{I}) \cap B) \cup E'$. Otherwise, denote this path by its sequence of vertices: $s, v_1, v_2, \ldots, v_k, t$. Denote by v_f and v_ℓ the first and last vertices on this path that are outside B. Then it is clear that the edge set $(\mathrm{OPT}(\mathcal{I}) \cap B) \cup E'$ contains the path $s, v_1, \ldots, v_{f-1}, v_{\ell+1}, \ldots, v_k, t$. In both case, s and t are connected and $(\mathrm{OPT}(\mathcal{I}) \cap B) \cup E'$ is a feasible solution to instance $\mathcal{I}'$.

To see (a), we prove by induction on the number of pairs $\mathcal{P}'$ arrived so far that the claim holds. If no pair arrived yet, this is vacuously true. Otherwise, consider the arrival of pair p and assume (a) holds for all pairs that arrived before. First note that the set of shortcuts added by A when connecting any pair $p' = \{s, t\}$ in instance $\mathcal{I}$ is just one edge $e = \{s, t\}$ with weight o. This is because we assumed that just before p' arrived, an edge of $\mathcal{S}$ between s and t arrived with weight exactly $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p')$. Hence we can assume that A went through this edge to connect p'. Here we also use the fact that when A goes through only one edge to connect a pair, the three contraction rules behave exactly the same.

Now by induction hypothesis, the set of shortcuts bought by A so far on instance $\mathcal{I}'$ have exactly the same property. Hence this set of shortcuts is a subset of the shortcuts added by A on instance $\mathcal{I}$. We claim that when pair $p = \{s, t\}$ arrives, the shortest path is again through the corresponding edge of $\mathcal{S}'$. Assume this is not the case. As a shorthand, define $c = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p)$. Denote by $\mathring{B}$ the set of vertices in B that are at distance at most

$$r \cdot \left(1 - \frac{1}{100 \cdot \log^2(K)}\right)$$

from the center of B. We first claim that all the shortcuts added so far by A in instance $\mathcal{I}'$ have both endpoints in $\mathring{B}$ (i.e. far from the border of B). To see this, note that all the pairs in $\mathcal{P}'$ are in $\mathring{B}_{\mathcal{P}}$ hence they have one endpoint at distance at most

$$r \cdot \left(1 - \frac{1}{200 \cdot \log^2(K)}\right)$$

from the center of B. If the other endpoint was not in $\mathring{B}$, the distance d between the two endpoints would be at least

$$d \geqslant \frac{r}{200 \cdot \log^2(K)},$$

and since pairs have contraction at most α, we would have that the cost paid for this pair in $\mathcal{I}$ is at least

$$\frac{d}{\alpha} \geqslant \frac{r}{200 \cdot (\alpha \log^2(K))}$$

but we assume that cost classes would be separated by at least a multiplicative $2^{\delta+10} > 2^{10} \cdot \alpha^{100} \cdot \log^{100}(K)$ by assumption on δ. Hence we would have a contradiction. Hence all pairs in $\mathcal{P}'$ have both endpoints in $\mathring{B}$, but this also implies that all shortcuts added so far by A in instance $\mathcal{I}'$ have both endpoints in $\mathring{B}$.

Now remark that because the instance is (α, δ)-canonical, we have that $d_G(s, t) \leqslant \alpha c$, and since the distance between a point in $\mathring{B}$ and the exact border of B is much bigger than αc, it cannot be that the shortest path uses some edges in E' (just to reach them is already too expensive because all previously bought shortcuts have endpoints in $\mathring{B}$). Hence the shortest path only uses edges that were available to A when connecting p in instance $\mathcal{I}$. Therefore the shortest path available in $\mathcal{I}'$ can only be longer than the shortest path for the same pair in $\mathcal{I}$. Since the corresponding edge of $\mathcal{S}'$ of weight exactly c is available, the shortest path is in fact exactly the same. □

By Claim 6.11, we know that the cost incurred by A because of pairs in $\mathring{B}_{\mathcal{P}}$ is equal to the cost that A would pay on the instance $\mathcal{I}'$. Denote by k'_i the number of pairs in $\mathcal{P}'^{(i)}$ in $\mathcal{I}'$. Note that by Claim 6.11 we have $\mathcal{P}'^{(i)} = \mathcal{P}^{(i)} \cap \mathring{B}_{\mathcal{P}} \cap D$. Because of properties (c) and (e) of Definition 6.8, it must be that the following inequalities hold

$$\begin{aligned} \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}}, \mathrm{charge}) &\leqslant 10 \cdot \mathrm{charge}(p(B)) \cdot c_j \cdot \log^{10}(K) && (6.7) \\ &\leqslant (550e^5) \cdot c_j \cdot \log^{10}(K) && (6.8) \end{aligned}$$

where c_j is the cost of the pair that created the ball B, and

$$k'_i \cdot c_i \leqslant \mathrm{cost}_{\mathrm{A}}(\mathcal{I}, \mathring{B}_{\mathcal{P}}, \mathrm{charge}). \tag{6.9}$$

Putting Equations 6.8 and 6.9 together we obtain

$$k'_i \leqslant (550e^5)\cdot\log^{10}(K)\cdot\frac{c_j}{c_i} = (550e^5)\cdot\log^{10}(K)\cdot 2^{(\delta+10)\cdot(i-j)} \leqslant 2^{3(\delta+10)\cdot(i-j)}, \tag{6.10}$$

for all $i > j$.

Note that in instance $\mathcal{I}'$, we have at most $M' = M - j < M$ cost classes. Finally, it is clear that $\mathcal{I}'$ is still an (α, δ)-canonical instance , A behaves the same on the relevant pairs, and all conditions of Lemma 6.10 are satisfied (we keep the same upper bound K on the number of pairs). Hence we can apply the induction hypothesis on the sub-instance $\mathcal{I}'$ to obtain the following upper bound on the cost of dangerous pairs inside B.

$$\begin{aligned}\mathrm{cost}_{\mathrm{A}}(\mathcal{I}') \leqslant (880e^5) &\left(\sum_{i=1}^{M'} \log(k'_i)\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\right)\\ &+ e^{200+20M'/\log(K)}\cdot\left(\sum_{i=1}^{M'}\delta\cdot(M'-i)\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\right),\end{aligned}$$

with $k'_i \leqslant 2^{3(\delta+10)\cdot i}$ by re-indexing cost classes from 1 to M' in Equation 6.10. The first term on the right-hand side is

$$\begin{aligned}&(880e^5)\cdot\left(\sum_{i=1}^{M-j}\log(k'_i)\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\right)\\ &\leqslant (880e^5)\cdot\left(\sum_{i=1}^{M-j}\log\left(2^{3(\delta+10)\cdot i}\right)\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\right)\\ &\leqslant (2640e^5)\cdot(\delta+10)\cdot\sum_{i=1}^{M-j} i\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\\ &\leqslant \left(\delta e^{200+20M'/\log(K)}\right)\cdot\sum_{i=1}^{M-j} i\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right).\end{aligned}$$

The second term in the right-hand side is less than

$$\left(e^{200+20M'/\log(K)}\right)\cdot\left(\sum_{i=1}^{M-j}\delta\cdot(M-j-i)\cdot w\left(\mathrm{OPT}(\mathcal{I}')\cap\mathcal{B}'^{(i)}\right)\right).$$

By summing both terms, we obtain a cost of at most

$$\left(e^{200+20M'/\log(K)}\right) \cdot \sum_{i=1}^{M-j} (i\delta) \cdot w\left(\mathrm{OPT}(\mathcal{I}') \cap \mathcal{B}'^{(i)}\right) +$$
$$\left(e^{200+20M'/\log(K)}\right) \cdot \left(\sum_{i=1}^{M-j} \delta \cdot (M-j-i) \cdot w\left(\mathrm{OPT}(\mathcal{I}') \cap \mathcal{B}'^{(i)}\right)\right)$$
$$\leqslant \left(\delta e^{200+20M'/\log(K)}\right) \cdot \left(\sum_{i=1}^{M-j} (M-j) \cdot w\left(\mathrm{OPT}(\mathcal{I}') \cap \mathcal{B}'^{(i)}\right)\right).$$

Now recall that in a balanced dual solution, all balls are pairwise disjoint hence we obtain the upper bound

$$\left(\delta e^{200+20M'/\log(K)}\right) \cdot (M-j) \cdot w(\mathrm{OPT}(\mathcal{I}')) \tag{6.11}$$

on the cost of pairs in $\mathring{B}_{\mathcal{P}} \cap D$ when summing on all i in the previous inequality.

By Claim 6.11, we also have $w(\mathrm{OPT}(\mathcal{I}')) \leqslant w(B \cap \mathrm{OPT}(\mathcal{I}))$. Now we need to take into account the charges that were put on dangerous pairs in $\mathring{B}_{\mathcal{P}} \cap D$ by the balanced dual solution in the big instance $\mathcal{I}$. By property (d) of Definition 6.8, the total charged cost incurred by A for pairs in $\partial B_{\mathcal{P}} \cap D$ is a most $10/\log(K)$ fraction of the cost of pairs in $\mathring{B}_{\mathcal{P}} \cap D$. Additionally, the charge of each pair in $p \in \mathring{B}_{\mathcal{P}} \cap D$ is at most (by property (e))

$$\left(1 + \frac{5}{\log(K)}\right)^{j-1} \leqslant e^{5(j-1)/\log(K)},$$

if the ball that contains this pair is in $\mathcal{B}^{(j)}$. Hence we only need to increase the upper bound given by Equation 6.11 by a multiplicative term $e^{(5(j-1)+10)/\log(K)} \leqslant e^{20j/\log(K)}$ to get a final upper bound of

$$\left(\delta e^{200+(20(M-j)+20j)/\log(K)}\right) \cdot (M-j) \cdot w(\mathrm{OPT}(\mathcal{I}'))$$
$$\leqslant \left(\delta e^{200+20M/\log(K)}\right) \cdot (M-j) \cdot w(\mathrm{OPT}(\mathcal{I}')) \tag{6.12}$$

on the total charged cost incurred by A (in instance $\mathcal{I}$) because of pairs in $B_{\mathcal{P}} \cap D$. To finish the proof, we sum these upper bounds over all $B \in \mathcal{B}^{(j)}$ and all j to obtain indeed the second term of the induction hypothesis. □

Given Lemma 6.10, it is now straightforward to prove Theorem 6.4. Lemma 6.10 applied to the main canonical instance $\mathcal{I}$ states that A pays at most

$$\begin{aligned}
&(880e^5) \cdot \left(\sum_{j=1}^{M} \log(k_j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right) \\
&+ e^{200+20M/\log(K)} \cdot \left(\sum_{j=1}^{M} \delta \cdot (M-j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \right) \\
&\leqslant O\Bigg[\sum_{j=1}^{M} \log(k_j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \\
&\qquad\qquad + \sum_{j=1}^{M} \delta \cdot (M-j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \Bigg] .
\end{aligned}$$

For the first inequality, we use that $M \leqslant \log(K)$. We obtain that the first term is at most

$$\log(k) \cdot \sum_{j=1}^{M} w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) \leqslant \log(k) \cdot w(\mathrm{OPT}(\mathcal{I})) ,$$

since all balls in $\mathcal{B}$ are pairwise disjoint. The second term, is at most

$$\begin{aligned}
\sum_{j=1}^{M} \delta \cdot (M-j) \cdot w\left(\mathrm{OPT}(\mathcal{I}) \cap \mathcal{B}^{(j)}\right) &\leqslant (\delta M) \cdot w(\mathrm{OPT}(\mathcal{I})) \\
&\leqslant \log(k) \cdot w(\mathrm{OPT}(\mathcal{I})),
\end{aligned}$$

since we have that $M \leqslant \log(k)/\delta$ (choosing the upper bound $K = k$ is a valid choice). Hence, if $\mathcal{I}$ is an (α, δ)-canonical instance, we indeed have that the cost paid by A on this instance is at most $O(\log(k)) \cdot w(\mathrm{OPT}(\mathcal{I}))$, which proves Theorem 6.4, and Theorem 5.1, combining it with Lemma 6.3.

6.3 PROOF OF THEOREM 5.2 AND THEOREM 5.3

The aim of this section is to establish Theorem 5.2 and Theorem 5.3. To this end we prove Lemma 6.12. We recall that in this section, all results apply only to Greedy_3 so A will be a shorthand for Greedy_3 in this whole section, unlike in Section 6.2 where A would mean that we could use any of the contraction rules.

Lemma 6.12. *Suppose we are given an instance $\mathcal{I}$ of Steiner Forest, with k pairs of terminals. Then we can construct an instance $\mathcal{I}'$ of Steiner Forest that satisfies the following, where $\mathrm{A} = \mathrm{Greedy}_3$:*

(a) $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}')$.

(b) The contraction of all pairs in instance $\mathcal{I}'$ when running A *is exactly equal to* 1.

(c) $\mathcal{I}'$ has $k' = O(k^2)$ terminal pairs.

(d) The set of terminals (vertices that appear in a pair) is the same for instance $\mathcal{I}$ and $\mathcal{I}'$.

Proof. The idea is to subdivide each pair p_i in instance $\mathcal{I}$ into $O(k)$ pairs that are made of pairs of previously arrived terminals that lie on the path P that is used to connect p_i. It will be clear from construction that Property (d) holds. Formally, we will construct an instance $\mathcal{I}'$ of Steiner Forest as follows. Beginning with the empty set of terminal pairs, for each pair $\{s_i, t_i\}$ that arrives in $\mathcal{I}$ we will construct a list of pairs $\mathcal{P}(\{s_i, t_i\})$ and add them to our instance $\mathcal{I}'$. The order in which we add pairs to $\mathcal{I}'$ determines the arrival order. We next describe how to construct $\mathcal{P}(\{s_i, t_i\})$. Suppose that when $\{s_i, t_i\}$ appears in $\mathcal{I}$ the algorithm A connects the pair using the path $P = s_i, v_1, \ldots, v_\ell, t_i$. Let $s_i, v'_1, \ldots, v'_{\ell'}, t_i$ be the sub-sequence of P specified by the contraction rule 3. Recall that this sub-sequence contains only s_i, t_i and previously arrived terminals. We add to $\mathcal{P}(\{s_i, t_i\})$ the subset of $\{\{s_i, v'_1\}, \{v'_1, v'_2\}, \ldots, \{v'_{\ell'-1}, v'_{\ell'}\}, \{v'_{\ell'}, t_i\}\}$ containing all the pairs such that their distance in the contracted metric is not yet 0 (here we consider the contracted metric just before connecting the pair $p = \{s_i, t_i\}$ in instance $\mathcal{I}$). For any pair $\{s', t'\}$ in $\mathcal{P}(\{s_i, t_i\})$ we say that $\{s_i, t_i\}$ is the parent of $\{s', t'\}$.

Now that we have constructed $\mathcal{I}'$, we argue that it satisfies the properties of the lemma. To show that $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}')$ we will prove the following stronger statement. For any pair $p_i = \{s_i, t_i\}$,

1. A spends exactly the same cost for the pairs in $\mathcal{P}(\{s_i, t_i\})$ in instance $\mathcal{I}'$ that it was spending for the pair $p_i = \{s_i, t_i\}$ in instance $\mathcal{I}$, i.e.

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p_i) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}', \mathcal{P}(\{s_i, t_i\})).$$

2. The contracted metric $G^{(i)}$ after revealing all the pairs up to pair p_i in instance $\mathcal{I}$ is exactly the same as the contracted metric $G'^{(\tau)}$ after revealing all the pairs $\bigcup_{i' \leqslant i} \mathcal{P}(\{s_{i'}, t_{i'}\})$ in instance $\mathcal{I}'$.

We will prove this statement by induction on i. It is vacuously true for $i = 0$ since no pair appeared so far in both instances. Assume this is true up to pair $i-1$, and let us prove the statement for step i. Let $v'_0, v'_1, \ldots, v'_{\ell'}, v'_{\ell'+1}$ be the sub-sequence of previously arrived terminals in the path P that A uses to connect s_i and t_i in instance $\mathcal{I}$, where $v'_0 = s_i$ and $v'_{\ell'+1} = t_i$. Suppose that for some $0 \leqslant j \leqslant \ell'$ the pair $\{v'_j, v'_{j+1}\}$ is not in $\mathcal{P}(\{s_i, t_i\})$. Then by construction we know that v'_j and v'_{j+1} were already at distance 0 in the contracted metric before the arrival of $\{s_i, t_i\}$ during the execution of A on $\mathcal{I}$, i.e.

$$d_{G^{(i-1)}}(v'_j, v'_{j+1}) = 0.$$

As a result the portion of the path P between v'_j and v'_{j+1} contributes 0 to the value $\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p_i)$ hence no cost is lost. Now suppose that the pair $\{v'_j, v'_{j+1}\}$ is in $\mathcal{P}(\{s_i, t_i\})$, and that when presented with the pair $\{v'_j, v'_{j+1}\}$ in instance $\mathcal{I}'$, A uses a path P' to connect the pair. Denote by $P_{[v'_j, v'_{j+1}]}$ the

restriction of the path P to vertices that appear in-between vertices v'_j, v'_{j+1} (included). We claim that $P' = P_{[v'_j, v'_{j+1}]}$. To see this, let us consider the first time this is not the case, this would mean that the path P' costs less to A in instance $\mathcal{I}'$ than what $P_{[v'_j, v'_{j+1}]}$ costed to A in instance $\mathcal{I}$. But since we assumed that the contracted metrics were identical up to pair p_{i-1}, A could have replaced the path $P_{[v'_j, v'_{j+1}]}$ by P' to pay less. This is a contradiction to the fact that A always takes the shortest path. Hence the total cost for the pair p_i is preserved and we have that

$$\mathrm{cost}_{\mathrm{A}}(\mathcal{I}, p_i) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}', \mathcal{P}(\{s_i, t_i\})).$$

To see why we have the second property, now note that we have that $P' = P_{[v'_j, v'_{j+1}]}$, in particular on the path P' there is no previously arrived terminal other than v'_j, v'_{j+1}. By contraction rule 3, it means that A simply adds an edge $\{v'_j, v'_{j+1}\}$ of weight 0 in the contracted metric. But by the definition of contraction rule 3, this edge was also added by A when connecting the pair p_i in instance $\mathcal{I}$. Reciprocally, it is clear that if a shortcut was added by A when connecting the pair p_i in instance $\mathcal{I}$, it must be between two previously arrived terminals $v'_{j'}, v'_{j'+1}$ that are consecutive on the path P. If these two previously arrived terminals were already at distance 0 in the contracted metric, then the shortcut does not change the metric. Otherwise if $v'_{j'}$ and $v'_{j'+1}$ were not at distance 0 then we have that $\mathcal{P}(\{s_i, t_i\})$ also contains the pair $\{v'_{j'}, v'_{j'+1}\}$ hence this shortcut will also be added in the metric when running instance $\mathcal{I}'$. This proves that the contracted metrics are indeed the same after step i. We also have Property (b) since, by construction, there is no previously arrived terminal on the path P'. Hence it must be that A pays the cost of the path P' in the original metric (the only way to pay less than the length in the original metric is to use a shortcut that was added before, but such a shortcut must connect previously arrived terminals).

Lastly, we argue that the number of terminals k' in $\mathcal{I}'$ is $O(k^2)$. Since there are k pairs in instance $\mathcal{I}$, we know that there are at most $2k$ terminals in total. Thus there are at most $2k$ terminals along any path A takes to connect a terminal pair in $\mathcal{I}$ (going twice through the same terminal cannot happen since A takes the shortest path). Summing this bound over all terminal pairs in $\mathcal{I}$, we get that $\mathcal{I}'$ has at most $2k^2$ terminals. □

With Lemma 6.12, we are able to prove Theorem 5.2.

Theorem 5.2. Construct $\mathcal{I}'$ from $\mathcal{I}$ as in Lemma 6.12. Then we can write

$$\begin{aligned}\mathrm{cost}_{\mathrm{A}}(\mathcal{I}) = \mathrm{cost}_{\mathrm{A}}(\mathcal{I}') &\leqslant O(\log(k') \cdot \log\log(k')) \cdot w(\mathrm{OPT}(\mathcal{I}')) \\ &\leqslant O(\log(k') \cdot \log\log(k')) \cdot w(T^\star(\mathcal{I})) \\ &\leqslant O(\log(k) \cdot \log\log(k)) \cdot w(T^\star(\mathcal{I})),\end{aligned}$$

where the first equality follows by Property (a) of Lemma 6.12, the second inequality from Theorem 5.1 applied to instance $\mathcal{I}'$ (using that contraction of all pairs is 1 by Property (b) of Lemma 6.12), the third inequality from the fact that the optimum tree solution to instance $\mathcal{I}$ denoted $T^\star(\mathcal{I})$ is also a feasible solution to instance $\mathcal{I}'$ (because by Property (d) of Lemma 6.12 the set of terminals does not change) and the last inequality from the fact that $k' = O(k^2)$ by Property (c) of Lemma 6.12. □

Next we prove Theorem 5.3. For this, we require some additional notation. Let $\mathcal{I}$ be an instance of Steiner Forest and $\mathcal{F}$ a feasible solution consisting of trees $T_1, \ldots, T_m$. Then the width of a tree T_j with respect to an instance $\mathcal{I}$, denoted $\text{width}(T_j, \mathcal{I})$, is defined to be the largest distance between any terminal pair in T_j in the original metric, i.e.

$$\text{width}(T_j, \mathcal{I}) = \max_{u \in \mathcal{T} \cap T_j} d_G(u, \overline{u}).$$

where $\overline{u}$ is the terminal that u should be connected to in instance $\mathcal{I}$, and $\mathcal{T}$ is the set of all terminals in instance $\mathcal{I}$. This is exactly the same definition of width as in [50]. For such a forest $\mathcal{F} = (T_1, \ldots, T_q)$ we define the potential

$$\Phi(\mathcal{F}, \mathcal{I}) = w(\mathcal{F}) + \sum_{j=1}^{q} \text{width}(T_j, \mathcal{I}).$$

Where $w(\mathcal{F})$ is the cost of the forest. Note that $w(\mathcal{F}) \leqslant \Phi(\mathcal{F}, \mathcal{I}) \leqslant 2w(\mathcal{F})$ for any forest $\mathcal{F}$ that is a feasible solution to instance $\mathcal{I}$. We will use this property to prove Theorem 5.3.

Theorem 5.3. We construct the instance $\mathcal{I}'$ from instance $\mathcal{I}$ exactly as how we transformed the instance in Lemma 6.12. We show that if the costs of shortest paths are non-increasing over time, it must be that

$$w(\text{OPT}(\mathcal{I}')) = O(w(\text{OPT}(\mathcal{I})).$$

Once this is established, the result follows as we have

$$\begin{aligned} \text{cost}_{\text{A}}(\mathcal{I}) = \text{cost}_{\text{A}}(\mathcal{I}') &= O(\log(k') \cdot \log\log(k')) \cdot w(\text{OPT}(\mathcal{I}')) \\ &= O(\log(k) \cdot \log\log(k)) \cdot w(\text{OPT}(\mathcal{I})) \end{aligned}$$

again by Lemma 6.12 and Theorem 5.1 (recall that subdividing pairs to create instance $\mathcal{I}'$ as in Lemma 6.12 guarantees that the contraction is 1 for all pairs in the instance $\mathcal{I}'$).

We argue that $w(\text{OPT}(\mathcal{I}')) = O(w(\text{OPT}(\mathcal{I}))$, using essentially the same potential function argument than that of [50]. The idea is to begin with the solution $\text{OPT}(\mathcal{I})$ and add additional connections to produce a feasible solution $\mathcal{F}$ to $\mathcal{I}'$, where $w(\mathcal{F}) \leqslant 2w(\text{OPT}(\mathcal{I}))$. Since $w(\text{OPT}(\mathcal{I}')) \leqslant w(\mathcal{F})$ if we succeed the proof is complete.

As previously stated, we initialize $\mathcal{F}$ to be $\text{OPT}(\mathcal{I})$. We will only add edges to $\mathcal{F}$ hence it will be clear that $\mathcal{F}$ will always be a feasible solution to instance $\mathcal{I}$. Now for each terminal pair $\{s_i, t_i\}$ that arrives in $\mathcal{I}$ we construct $\mathcal{F}'$ from $\mathcal{F}$ as follows. Suppose that all the pairs in $\mathcal{P}(\{s_i, t_i\})$ are already connected by the current solution $\mathcal{F}$, then nothing needs to be done.

Suppose otherwise that the solution $\mathcal{F}$ does not connect all pairs in $\mathcal{P}(\{s_i, t_i\})$. By re-indexing let $T_1, \ldots, T_q$ be the components of $\mathcal{F}$ that contain the terminals that appear in the pairs of $\mathcal{P}(\{s_i, t_i\})$, ordered so that

$$\text{width}(T_1) \geqslant \ldots \geqslant \text{width}(T_q).$$

Construct $\mathcal{F}'$ from $\mathcal{F}$ by adding the edges that A uses to connect the pairs in $\mathcal{P}(\{s_i, t_i\})$ in instance $\mathcal{I}'$. We claim that the total cost of these edges (which is exactly equal to $\mathrm{cost}(\mathcal{I}, p_i)$) is at most the width of T_q (with respect to instance $\mathcal{I}$), i.e.

$$\mathrm{cost}(\mathcal{I}, p_i) \leqslant \mathrm{width}(T_q, \mathcal{I}).$$

To see this first note that the tree T_q either contains the pair $\{s_i, t_i\}$ or a pair that appeared even before $\{s_i, t_i\}$ in instance $\mathcal{I}$ (here we use that $\mathcal{F}$ is a feasible solution to instance $\mathcal{I}$). Hence if the sequence of shortest paths $d_1 = d_G(s_1, t_1), \ldots, d_k = d_G(s_k, t_k)$ is non-increasing, then

$$\mathrm{width}(T_q, \mathcal{I}) \geqslant \min_{i' \leqslant i} d_{i'} = d_i \geqslant c_i = \mathrm{cost}(\mathcal{I}, p_i).$$

For the same reason if the sequence of costs $c_1 = \mathrm{cost}(\mathcal{I}, p_1), \ldots, c_k = \mathrm{cost}(\mathcal{I}, p_k)$ is non-increasing, then

$$\mathrm{width}(T_q, \mathcal{I}) \geqslant \min_{i' \leqslant i} d_{i'} \geqslant \min_{i' \leqslant i} c_{i'} = c_i = \mathrm{cost}(\mathcal{I}, p_i).$$

We use here that $d_i \geqslant c_i$ for all i. This proves our claim.

We now note that, if we needed to add edges because the current solution $\mathcal{F}$ did not connect all the pairs in $\mathcal{P}(\{s_i, t_i\})$, then we obtained a new solution $\mathcal{F}'$ such that

$$w(\mathcal{F}') \leqslant w(\mathcal{F}) + \mathrm{cost}(\mathcal{I}, p_i),$$

and at least two of the components in $T_1, T_2, \ldots, T_q$ were merged into one component. Hence

$$\begin{aligned}\mathrm{width}(\mathcal{F}', \mathcal{I}) &\leqslant \mathrm{width}(\mathcal{F}, \mathcal{I}) - \min_{1 \leqslant i \leqslant q} \mathrm{width}(T_i, \mathcal{I}) \\ &\leqslant \mathrm{width}(\mathcal{F}, \mathcal{I}) - \mathrm{width}(T_q, \mathcal{I}).\end{aligned}$$

In particular we obtain

$$\Phi(\mathcal{F}', \mathcal{I}) - \Phi(\mathcal{F}, \mathcal{I}) \leqslant \mathrm{cost}(\mathcal{I}, p_i) - \mathrm{width}(T_q, \mathcal{I}) \leqslant 0,$$

by the above remarks. We then update $\mathcal{F}$ to be $\mathcal{F}'$.

After completing the process we have that $\mathcal{F}$ connects all pairs in $\mathcal{I}'$, as required and that the potential function did not increase. Therefore we have that

$$w(\mathcal{F}') \leqslant \Phi(\mathcal{F}', \mathcal{I}) \leqslant \Phi(\mathcal{F}, \mathcal{I}) \leqslant 2w(\mathcal{F}),$$

as desired. □

7

A SIMPLE APPROXIMATION ALGORITHM FOR MAP

This chapter contains the proof of Theorem 5.4, which is arguably much simpler than in other works. We start by a brief explanation of the overall idea in Section 7.1 before providing the complete proof in Section 7.2. The results presented in this chapter are based on the paper entitled "A Simple LP-based Approximation Algorithm for the Matching Augmentation Problem", which is a joint work with Marina Drygala and Ola Svensson. It appeared at the *International Conference on Integer Programming and Combinatorial Optimization* (IPCO '22) [10].

7.1 OUR PROOF TECHNIQUE

The proof of Theorem 5.4 relies on several crucial observations that we sketch here. The first observation is that the total cost of the DFS tree T is always at most the cost of x^* (denoted $c(x^*)$). This follows because T must contain all the light edges since they are given priority over any other edge (note that since we assume that M is a matching, it cannot happen that two distinct light edges want priority at the same time). Therefore, the total cost of the tree T is exactly equal to $n - 1 - |M|$, while it is easy to show that $c(x^*) \geqslant (n - |M|)$.

Another interesting fact is that if one considers the LP solution x^* restricted to the edges *not* in the tree T (denote this solution by $x^*_{E\setminus T}$), then this is a feasible solution to the cut LP of the TAP instance with respect to the tree T (i.e. $x^*_{E\setminus T}$ is a feasible solution to $LP(G,T)$). Hence, if we denote by y^* the optimum fractional solution to $LP(G,T)$, we have that $c(y^*) \leqslant c(x^*_{E\setminus T})$.

Because T is a DFS tree, the TAP instance with respect to the tree T contains only "uplinks" and therefore $LP(G,T)$ is known to be integral [1]. We note that this already gives a simple proof that the integrality gap of $LP(G,M)$ is at most 2. To get better than 2, we only need to show that

$$(n - |M| - 1) + c(y^*) \leqslant (2 - \delta)c(x^*) \ ,$$

for some $\delta > 0$. Conceptually, we distinguish between two cases. If $c(x^*) > (1+\delta)(n-|M|)$ (i.e. the LP solution is expensive), then the DFS tree is significantly cheaper than $c(x^*)$ and it is easy to conclude that the cost of our solution $T \cup A$ is better than $2c(x^*)$. Otherwise, assume that the LP value is close to the trivial lower bound of $(n - |M|)$. In this case, we show that $c(y^*) \leqslant (1-\delta)c(x^*)$.

To show this, we consider two possibilities. We can prove that either we can scale down a significant portion of $x^*_{E\setminus T}$ to obtain a cheaper feasible solution to $LP(G,T)$, or that $c(x^*_{E\setminus T})$ itself is already significantly smaller than $c(x^*)$. When a lot of the tree cuts in T (i.e. the cuts defined by removing an edge from T to obtain two trees and taking the edges with one endpoint in each tree) have some slack in the TAP solution $x^*_{E\setminus T}$ (that is when a lot of tree

cuts S satisfy $x^*_{E\setminus T}(\delta(S)) > 1+\delta$), the first case is realized. Otherwise, when almost all of the tree cuts are nearly tight (i.e. satisfy $x^*_{E\setminus T}(\delta(S)) \leqslant 1+\delta$), we can show that the DFS must have captured a good fraction of the value of $c(x^*)$ *inside* the tree T. This step uses some crucial properties of extreme point solutions as well as our choice of DFS. Therefore the cost of $x^*_{E\setminus T}$ is significantly smaller than the cost of x^* completing the argument.

Before proceeding to the proof, it is worthwhile to mention that we are not aware of any example on which our algorithm has a ratio worse than $4/3$ times the cost of x^*. It remains open to give a tighter analysis of this algorithm. We also note that [62] also makes use of DFS for the related problem of unweighted 2-ECSS. They obtain a ratio of $3/2$ for the unweighted 2-ECSS problem. However, their DFS is not LP-based and we remark that if we do not guide the DFS with the LP solution, the approximation ratio can be arbitrarily close to 2. We give an example in Figure 7.1. One can see that the DFS tree (rooted at r) contains all the matching edges, and the tree augmentation problem requires us to take all but one of the back-edges. However, the optimum solution to the MAP instance is to take a Hamiltonian tour containing all the light edges. Generalizing the same example by simply increasing the depth of the tree leads to an approximation arbitrarily close to 2.

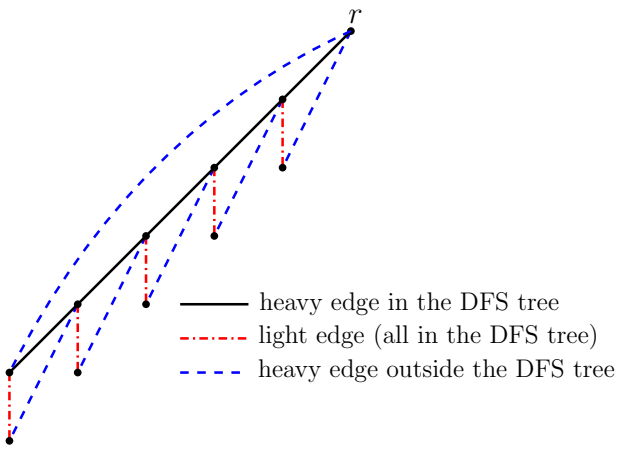

Figure 7.1: An example of a bad DFS tree.

7.2 THE ANALYSIS OF THE LP-BASED ALGORITHM

In this section, we prove Theorem 5.4. It is organized as follows. In subsection 7.2.1, we introduce some basic definitions. In the subsequent subsection, we proceed via a case distinction to prove the theorem.

7.2.1 *Preliminaries*

We will use T to refer to the DFS tree computed by the algorithm, and we will list edges in G as uv, where u is an ancestor of v in T. Since T is a DFS tree, all edges in G must have the property that one endpoint is an ancestor of the other in T. We will let $B = E \setminus T$ denote the set of *back-edges* of G.

As in the introduction, we will call an edge of weight 1 a *heavy edge* and an edge of weight 0 a *light edge*. For every edge e in the DFS tree T computed, we let $T(e)$ denote the tree cut corresponding to the edge e in the tree T. Formally, $T(e) = \delta(T_v)$, where $e = uv$ and T_v is the sub-tree rooted at v. We call an edge $e \in T$ α-*tight* if we have

$$x^*(T(e)) - x^*_e < 1 + \alpha.$$

Implicitly, if we call an edge e α-tight, this will mean that e belongs to the tree T. In addition, we denote by $N_t^{(\alpha)}$ the number of α-tight edges in the tree T. For a tree T, we denote by x^*_T the restriction of x^* to the edges in the tree T. We note that for any instance of the MAP, it must be that $c(x^*) \geqslant (n - |M|)$. This follows by a simple double counting argument on the fractional degree of each component (precisely we have $n - |M|$ components that must have fractional degree 2 each). It is also clear that the DFS tree T must contain all the light edges in M since they are given priority. Hence the cost of T is at most $n - |M| - 1 \leqslant c(x^*)$. In the following, we will fix two parameters $\varepsilon = 10^{-1}, \gamma = 10^{-3}$.

7.2.2 *The analysis of the algorithm*

We note that if $c(x^*) \geqslant (1 + \gamma)(n - |M|)$, it is easy to show that the cost of the returned solution $T \cup A$ is at most

$$(n - |M| - 1) + c(x^*) \leqslant \frac{c(x^*)}{1 + \gamma} + c(x^*) = c(x^*)\left(2 - \frac{\gamma}{1 + \gamma}\right). \quad (7.1)$$

However, if $c(x^*) < (1 + \gamma)(n - |M|)$ and $N_t^{(\gamma)} \leqslant (1 - \gamma)(n - |M|)$ (i.e. there are few γ-tight tree cuts), then we proceed as follows. We partition the set of back edges in our graph B into $B_t^{(\gamma)} \cup B_s^{(\gamma)}$, where $B_t^{(\gamma)}$ contains all edges $e \in B$ that are contained in $T(e')$ for some γ-tight edge $e' \in T$. Then x', defined by

$$x'(e) = \begin{cases} x^*(e) & e \in B_t^{(\gamma)} \\ \frac{x^*(e)}{1+\gamma} & e \in B_s^{(\gamma)} \\ 1 & \text{otherwise} \end{cases}$$

is also a feasible solution to $LP(G, T)$. The total fractional value represented by edges in $B_t^{(\gamma)}$ is at most $(1 + \gamma)N_t^{(\gamma)}$. Hence, $c(x')$ can be upper bounded as follows.

$$c(x') \leqslant \frac{c(x^*) - (1 + \gamma)N_t^{(\gamma)}}{1 + \gamma} + (1 + \gamma)N_t^{(\gamma)} = \frac{c(x^*)}{1 + \gamma} + \gamma N_t^{(\gamma)}.$$

Since the cost of $T \cup A$ is at most $c(x^*) + c(x')$ and we assume that

$$N_t^{(\gamma)} \leqslant (1 - \gamma)(n - |M|),$$

it is easy to get the upper-bound of

$$c(x^*)\left(1 + \frac{1}{1 + \gamma}\right) + \gamma(1 - \gamma)(n - |M|) \leqslant c(x^*)\left(2 - \frac{\gamma^3}{1 + \gamma}\right), \quad (7.2)$$

where the last inequality follows because $n - |M| \leqslant c(x^*)$. Since these two cases clearly give a better-than-2 approximation, we assume in the rest of the analysis that

$$(n - |M|) \leqslant c(x^*) < (1 + \gamma)(n - |M|), \tag{7.3}$$

and

$$N_t^{(\gamma)} > (1 - \gamma)(n - |M|). \tag{7.4}$$

We will show that $c(x_T^*)$ is at least a constant fraction times $c(x^*)$. Since the cost of the returned solution $T \cup A$ is at most $2c(x^*) - c(x_T^*)$, this will conclude the proof. First, we partition the γ-tight tree cuts into two sets of cuts $\mathcal{S}_0$ and $\mathcal{S}_1$ containing the tight tree cuts associated with light edges and heavy edges, respectively. We can then distinguish between two sub-cases. For each edge $e = uv \in T$, we say that e is a *leaf* edge if v is a leaf in the tree T (recall that we always write an edge e as $e = uv$ such that v is a descendant of u in T). We denote $\mathcal{S}_0^+$ the non-leaf edges in $\mathcal{S}_0$ and $\mathcal{S}_0^-$ the leaf edges in $\mathcal{S}_0$. We have two main cases.

7.2.2.1 *Suppose that $|\mathcal{S}_1| \geqslant \gamma(n - |M|)$ or that $|\mathcal{S}_0^+| \geqslant \gamma(n - |M|)$.*

By feasibility of x^* at least 2 units of x^* must cross any tree cut. Hence $x^*(\delta(T_v)) \geqslant 2$, for any $v \in V$. By definition of γ-tightness we know that for any γ-tight edge $e = uv$ we have $x^*(e) \geqslant x^*(\delta(T_v)) - (1 + \gamma) \geqslant 1 - \gamma$.

Hence if $|\mathcal{S}_1| \geqslant \gamma(n - |M|)$, we have that

$$c(x_T^*) \geqslant \gamma(1 - \gamma)(n - |M|) \geqslant \frac{\gamma(1 - \gamma)}{1 + \gamma} c(x^*),$$

which concludes the case when $|\mathcal{S}_1|$ is large. In the following we use some properties of extreme point solutions. We say that an edge e is *fractional* (with respect to the fractional solution x^*) if $0 < x_e^* < 1$. A vertex v is said to be *α-fractional* if it has more than $1/\alpha$ incident fractional edges in the support of x^* (for any $\alpha > 0$). We claim the following lemma, the proof of which relies on standard techniques that we repeat here for completeness. We note that a similar result was used in [69].

Lemma 7.1. *If x^* is an extreme point solution of the cut LP, then there are at most $2n - 1$ fractional edges in G. Moreover, for any $\alpha > 0$, there are at most $4\alpha n$ α-fractional vertices with respect to x^*.*

Proof. An extreme point x^* can be defined as the unique solution to the following system of $|E|$ linearly independent constraints, for some $\mathcal{S} \subseteq 2^V$ and $E_0 \cup E_1 \subseteq E$.

$$\begin{aligned} &\sum_{e \in \delta(S)} x_e = 2, \quad \text{for all } S \in \mathcal{S} \\ &x_e = 0 \qquad \forall e \in E_0 \\ &x_e = 1 \qquad \forall e \in E_1 \end{aligned}$$

We then use Theorem 4.9 from [33] that proves that we can select $\mathcal{S}$ to be a laminar family.

Lemma 7.2 (Theorem 4.9 in [33]). *Let x^* be an extreme point of the MAP cut LP then the family of equations $\mathcal{S}$ can be chosen to be a laminar family.*

It is well known that any laminar family has size at most $2n-1$. Using this fact together with Lemma 7.2, we obtain that the number of fractional edges is at most $|E| - |E_0| - |E_1| = |\mathcal{S}| \leqslant 2n-1$. To finish the proof of Lemma 7.1, fix some $\alpha > 0$. Each vertex that is α-fractional accounts for at least $1/\alpha$ fractional edges. By a simple double-counting argument, there can be at most $\alpha \cdot (4n-2) \leqslant 4\alpha n$ vertices that are α-fractional. □

Using Lemma 7.1 with $\alpha = \gamma/16$, we get that if $|\mathcal{S}_0^+| \geqslant \gamma(n - |M|)$, then (recall that $n - |M| \geqslant n/2$) there are at least

$$\gamma(n-|M|) - (\gamma/4)n \geqslant (\gamma/2)(n-|M|)$$

edges $uv \in \mathcal{S}_0^+$ such that v is not $\gamma/16$-fractional. We then claim the following simple lemma.

Lemma 7.3. *Fix any $\alpha, \alpha' > 0$. Suppose that $e = uv$ is an α-tight light edge, such that v is not a leaf in T. Then if v is not α'-fractional there exists some edge $e' = vw$ in T such that $x^*(e') \geqslant (1-\alpha)\alpha'$.*

Proof. By feasibility of x^* we know that $x^*(\delta(T_v \setminus v)) \geqslant 2$. Since e is α-tight and T is a DFS tree we have that $x^*(\delta(T_v)) - x^*(e) \leqslant 1+\alpha$. We know that $E(T_v \setminus v, v) = \delta(T_v \setminus v) \setminus \delta(T_v)$, and as a result $x^*(E(T_v \setminus v, v)) \geqslant 1-\alpha$. Since v is not α'-fractional there must be an edge $e' \in E(T_v \setminus v, v)$ with value at least $x^*(E(T_v \setminus v, v))\alpha' \geqslant (1-\alpha)\alpha'$. Since our DFS selects always the highest possible fractional value if there is no light edge to explore, the first edge selected after exploring v must be of fractional value at least $(1-\alpha)\alpha'$. □

Combining Lemma 7.3 with the previous observation, if $|\mathcal{S}_0^+| \geqslant \gamma(n-|M|)$ we get that

$$c(x_T^*) \geqslant (\gamma/2)(n-|M|)(1-\gamma)(\gamma/16) \geqslant c(x^*)\frac{\gamma^2(1-\gamma)}{32(1+\gamma)}.$$

Combining these two cases we get that if $|\mathcal{S}_1| \geqslant \gamma(n-|M|)$ or $|\mathcal{S}_0^+| \geqslant \gamma(n-|M|)$ then

$$c(x_T^*) \geqslant c(x^*) \cdot \min\left(\frac{\gamma^2(1-\gamma)}{32(1+\gamma)}, \frac{\gamma(1-\gamma)}{1+\gamma}\right),$$

hence the cost of $T \cup A$ is upper bounded by

$$2c(x^*) - c(x_T^*) \leqslant c(x^*)\left(2 - \frac{\gamma^2(1-\gamma)}{32(1+\gamma)}\right), \tag{7.5}$$

which is clearly better than 2. Hence we are left with the last case, in which

$$|\mathcal{S}_0^-| > (1-\gamma)(n-|M|) - |\mathcal{S}_1| - |\mathcal{S}_0^+| > (1-3\gamma)(n-|M|).$$

7.2.2.2 *Suppose* $|\mathcal{S}_0^-| > (1-3\gamma)(n-|M|)$.

This is the most interesting case. Note that for each edge $e = uv \in \mathcal{S}_0^-$, the fractional degree of v restricted to heavy edges must be at least 1, and all of this fractional degree is carried by backedges in T. Denote by B' this subset of backedges. Next we define $B'' \subseteq B'$ to be the subset of B' containing only edges with fractional value at least $\varepsilon = 10^{-1}$. We claim that

$$|B''| \geqslant n/10. \tag{7.6}$$

Assume the contrary, since the fractional value of any edge is at most 1 then the total value carried by edges in $B' \setminus B''$ must be at least

$$|\mathcal{S}_0^-| - (n/10) > (1-3\gamma)(n-|M|) - n/10 > 3n/8 - n/10.$$

(Recall that $(n-|M|) \geqslant n/2$ and $(1-3\gamma) > 3/4$). Since all the edges in $B' \setminus B''$ have fractional value at most ε, there must be at least $(3n/8 - n/10)/\varepsilon = 30n/8 - n > 2n-1$ such edges, contradicting Lemma 7.1. Hence $|B''| \geqslant n/10$.

For completeness we consider the case when E contains heavy edges that are parallel to light edges. Partition B'' into $B_1'' \cup B_2''$, where B_2'' is the set of edges in B'' parallel to an edge in $\mathcal{S}_0^-$. We define B_1'' to be the remaining edges in B''.

We claim that $|B_2''| \leqslant n/100$, and thus loosely $|B_1''| \geqslant n/20$.

To see this note that,

$$c(x^*) - (n-|M|) \geqslant |B_2''|\varepsilon. \tag{7.7}$$

Equation (7.7) holds as the lower bound of $(n-|M|)$ on $c(x^*)$ is obtained only by counting the fractional degree of each component in M. Since those parallel edges are not counted in this bound (they are only *within* a single component), they directly count in the value of $c(x^*) - (n-|M|)$, which counts the surplus of $c(x^*)$ above $(n-|M|)$.

Then as $c(x^*) - (n-|M|) \leqslant \gamma(n-|M|)$, by choice of ε and γ we obtain that $|B_2''| \leqslant n/100$.

Consider the set of vertices X that contains the ancestor vertices of the edges in B_1''. We claim that

$$|X| \geqslant n/500. \tag{7.8}$$

To prove this, we first claim that

$$c(x^*) - (n-|M|) \geqslant |B_1''|\varepsilon - 2|X|. \tag{7.9}$$

To see this, note again that the value $c(x^*) - (n-|M|)$ represents the surplus value of $c(x^*)$ above the lower bound that gives fractional degree 2 to every vertex. This trivial lower bound gives a fractional value—which is the fractional degree restricted to heavy edges—of at most 2 to every vertex, hence a fractional value of at most $2|X|$ to the set of vertices X. Since every edge in B_1'' has fractional value of at least ε and is adjacent to a single vertex in X, we get that the surplus value of $c(x^*)$ above the trivial lower bound is at least $|B_1''|\varepsilon - 2|X|$ which proves Equation (7.9).

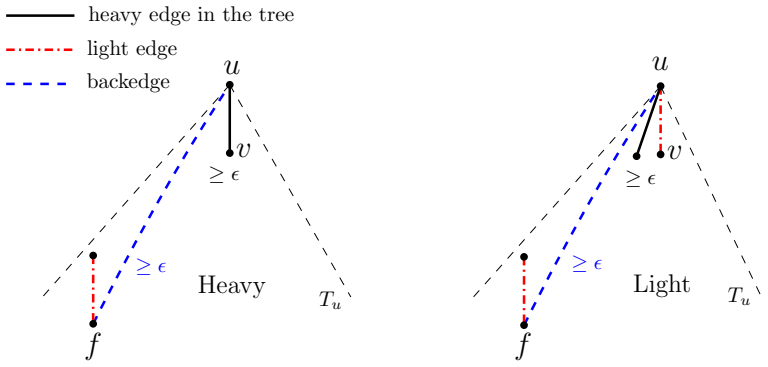

Figure 7.2: On the left side, the case when the first edge selected out of u is heavy. On the right the case when the first edge selected out of u is light.

Since by assumption we have $c(x^*) < (1+\gamma)(n-|M|)$ we conclude with Equation (7.9) that

$$\gamma(n-|M|) > c(x^*) - (n-|M|) \geqslant |B_1''|\varepsilon - 2|X|$$

which implies, by our lower bound on $|B_1''|$ and our choice of γ and ε,

$$|X| \geqslant \frac{|B_1''|\varepsilon - \gamma(n-|M|)}{2} \geqslant \frac{n/200 - n/10^3}{2} = n/500. \tag{7.10}$$

For each vertex $u \in X$, denote by e_u the first edge selected by the DFS after reaching u. Denote $X' \subseteq X$ the subset of X containing only vertices $u \in X$ such that e_u does *not* belong to $\mathcal{S}_0^+$. Then, we have by assumption,

$$|X'| \geqslant |X| - |\mathcal{S}_0^+| \geqslant n/500 - \gamma(n-|M|) \geqslant n/500 - n/10^3 = n/10^3.$$

We finally claim the following, which crucially uses how the DFS selects the edges to explore in priority.

Claim 7.4.

$$c(x_T^*) \geqslant \varepsilon|X'|.$$

Proof. There are two cases to consider (depicted in Figure 7.2).

If $u \in X'$ is such that $e_u = uv$ is a heavy edge, by definition of X' there must be an edge $e' = uf$ coming from a leaf f in the tree T to u of fractional value $x_{e'} \geqslant \varepsilon$. At the first time the DFS visits the vertex u, the leaf f was not explored yet hence the edge e' was a valid choice of edge to explore. Since our DFS always takes the highest fractional value, it must be that

$$x_{e_u} \geqslant x_{e'} \geqslant \varepsilon.$$

If $u \in X'$ is such that $e_u = uv$ is a light edge, recall that by definition of X', v must be a leaf in T. Then when the DFS arrived at v, it must be that all reachable vertices from v were already visited. Hence the DFS must have backtracked to u. Now note that by our construction of B_1'', we know that there must be another leaf f such that $e' = uf$ is a back-edge in the tree of fractional value $x_{uf} \geqslant \varepsilon$ (recall that uf is not parallel to the edge e_u). Since

f is a leaf, u must have been explored before f therefore, after backtracking from v to u the edge uf was a valid edge to take. Therefore the DFS must have selected a second edge e'' in the tree from u such that

$$x_{e'} \geqslant x_{uf} \geqslant \varepsilon.$$

Hence we proved that all vertices u in X must be adjacent to at least one heavy edge of fractional value ε that belongs to the tree T and goes to a child of u. Hence the proof of the claim. □ □

By the previous claim, we have $c(x_T^*) \geqslant \varepsilon|X'|$ hence the cost of the returned solution $T \cup A$ is at most

$$2c(x^*) - c(x_T^*) \leqslant 2c(x^*) - n/10^4 \leqslant c(x^*)\left(2 - 10^{-4}\right), \tag{7.11}$$

which ends the proof of Theorem 5.4.

8

CONCLUSION AND OPEN PROBLEMS

In the first part of this thesis, we have given improved approximation algorithms for the Santa Claus problem in various settings. For the MaxMin Arborescences problem, we gave an $O(\mathrm{poly}(\log\log n))$-approximation running in quasi-polynomial time. As we highlighted in the introduction of this thesis, the ultimate goal would be to design a constant factor approximation for the Santa Claus problem with linear valuation functions (if it exists). Here we give three open problems that are necessary intermediate steps towards this goal.

Open Question 8.1. *Can the $O(\mathrm{poly}(\log\log n))$-approximation guarantee be generalized to the Santa Claus problem with linear valuations?*

A big hurdle to extend our proof to the general case seems to be our use of the Lovasz Local Lemma. Indeed, in the general version of the problem, one needs to deal with a slightly more general version of the MaxMin Arborescences problem, in which two vertices in a arborescence can be connected by a long path instead of a single edge. This seems to introduce more dependencies between bad events. Nevertheless, we believe that many of our ideas will be very useful to generalize to the Santa Claus with linear valuations.

Open Question 8.2. *Is there an $O(1)$-approximation for MaxMin Arborescences (possibly in quasi-polynomial time)?*

It is worthwhile to recall that the MaxMin Arborescences problem is a special case of the Santa Claus problem. Hence obtaining a constant factor for MaxMin Arborescences is necessary if one aims at a constant factor for the Santa Claus problem. We mention that even in the special case of a single source in a layered graph, we do not know how to solve Open Question 8.2. This seems to be an interesting starting point.

Open Question 8.3. *Is an $O(\mathrm{poly}(\log n))$-approximation in polynomial time possible for MaxMin Arborescences?*

The best guarantee that is known to be possible in polynomial time is only polynomial [15, 21]. This state of affairs in reminiscent of the Directed Steiner Tree problem, for which an $O(n^{\varepsilon})$-approximation is achievable in polynomial time for every fixed $\varepsilon > 0$ [22]. Designing an $O(\mathrm{poly}(\log n))$-approximation in polynomial time for the Directed Steiner Tree problem is a notorious open problem in approximation algorithms. Hence we believe Open Question 8.3 to be very challenging.

As our second result, we showed an $O(\log\log n)$-approximation algorithm (in polynomial time) for the Restricted Submodular Santa Claus. The main open question here is the following.

Open Question 8.4. *Is there an $O(1)$-approximation for the Restricted Submodular Santa Claus?*

In the second part of this thesis, we provided new results for fundamental network design problems. For the Steiner Forest problem, we show that offline greedy is an $O(\log(k) \cdot \log\log(k))$-approximation. Even if our result has important consequences also for the online version, it is still an interesting open question to generalize our result to online greedy. Furthermore, improving the bound to a simple $O(\log(k))$ would be very interesting. Both these questions can be summarized in a single open question.

Open Question 8.5. *Is the greedy algorithm an $O(\log(k))$-approximation in the online setting?*

As a starting point, one might assume that all pairs have contraction 1 and the cost of each pair belongs to the interval $[\mathrm{OPT}/\log(k), \mathrm{OPT}]$. As far as we know, this case seems to retain all the difficulty and we are not aware of any better bounds in this setting that the ones in this thesis.

Finally, for the Matching Augmentation Problem, we gave a simple polynomial time algorithm that guarantees a better-than-2 approximation when compared to the cut LP. Tha Matching Augmentation problem can be seen as an instance of the cheapest 2-edge connected subgraph problem in a graph with edge-weights 0 or 1, where the edges of cost 0 form a matching. A natural direction would be to drop the assumption on the structure of edges of cost 0, in which case we obtain the *Forest Augmentation problem*. A natural research direction is the following question.

Open Question 8.6. *Is the integrality gap of the cut LP for the Forest Augmentation problem strictly less than 2?*

It was shown that it is better than 2 for the (unweighted) Tree Augmentation problem ([74]), which is the case where we assume the edges of cost 0 to form a spanning tree. Our result shows a similar statement for the Matching Augmentation problem. Since these two cases are the two "extreme" cases of the Forest Augmentation problem, it is believable that the cut LP is also better than 2 in the more general Forest Augmentation case.

Part III

APPENDIX

A

DEFERRED PROOFS FOR MAXMIN ARBORESCENCES

A.1 A CHALLENGING INSTANCE FOR RANDOMIZED ROUNDING

In this section, we discuss a specific limitation of the rounding algorithm in previous works (see [21] and [15]). Essentially, we explain why their techniques cannot give a better than $\Theta(\frac{\log(n)}{\log\log(n)})$ approximation. This argument is based on an instance that we build below, which also illustrates why the bottom-to-top pruning cannot fix the issue by itself. We reuse here the notation defined in Section 3.2. For clarity, we restate here the LP relaxation of the problem (for a single source s), called the path LP.

$$\begin{aligned}
\sum_{q \in C(p)} x(q) &= k \cdot x(p) && \forall p \in P \setminus T \\
\sum_{q \in I(v) \cap D(p)} x(q) &\leqslant x(p) && \forall p \in P, \forall v \in V \\
x((s)) &= 1 && \forall s \in S \\
x &\geqslant 0
\end{aligned}$$

Below we give a construction of a random instance inspired by [51]. The graph will have $h+1$ layers labeled from 0 to h and a single source s located in L_0. n will be the number of vertices in the last layer L_h and we choose $h = \Theta\left(\frac{\log n}{\log\log n}\right)$ so that $(\log^{10} n)^h = n$. The set of sinks corresponds exactly to the vertices in the last layer L_h. The graph until layer $h-1$ is a complete and regular tree where every internal vertex has exactly $\log^{20} n$ children. Vertices in layer L_{h-1} will be called *leaves*. Next, we describe how we connect layer $h-1$ to layer h.

Each vertex $v \in L_\ell$ (note that v is a sink) runs the following sampling process, independently of other sinks. Start at the source s. From the source select each vertex u in L_1 independently at random with probability $1/\log^{10} n$. All the vertices in L_2 whose parent was selected is then selected independently at random with probability $1/\log^{10} n$. We repeat layer by layer, each vertex in layer L_i whose parent was selected is then selected independently at random with probability $1/\log^{10} n$.

Finally, connect the sink $v \in L_h$ to all vertices in layer L_{h-1} that were selected by the above random process initiated by v.

In the following we denote by T_u for any $u \in \cup_{0 \leqslant i \leqslant h-1} L_i$ the subtree rooted at vertex u that is contained in layers $\cup_{0 \leqslant i \leqslant h-1} L_i$. We first claim the following.

Claim A.1. *With high probability, for any $j \leqslant h-1$ no sink $v \in L_h$ is connected to more than*

$$\left(1 + \frac{1}{\log(n)}\right) \cdot (\log^{10}(n))^{h-1-j}$$

leaves in layer $\ell - 1$ that belong to the subtree rooted at a vertex $u \in L_j$.

Proof. To see this, note that during the random process initiated by the sink v, every internal node u that was selected selects in expectation $\log^{20}(n)/\log^{10}(n) = \log^{10}(n)$ of its children. By a standard Chernoff bound the probability that this internal nodes selects more than

$$\left(1 + \frac{1}{\log^2(n)}\right) \cdot \log^{10}(n)$$

of its children is at most

$$\exp\left(-\frac{1}{3\log^4(n)} \cdot \log^{10}(n)\right) \leqslant \exp\left(-\log^6(n)/3\right).$$

Therefore with high probability, no internal node selects more than $\left(1 + \frac{1}{\log^2 n}\right) \cdot \log^{10}(n)$ of its children which implies that the total number of selected nodes at layer $h-1$ that belong to the subtree T_u (which are exactly the neighbors of v in that subtree) is at most

$$\left(1 + \frac{1}{\log^2 n}\right)^{h-1-j} (\log^{10} n)^{h-1-j} \leqslant \left(1 + \frac{1}{\log(n)}\right) (\log^{10} n)^{h-1-j}.$$

□

Let u be a vertex in layer L_{h-1}. Then u is connected to each vertex $v \in L_h$ independently with probability $1/(\log^{10} n)^{h-1}$, therefore with high probability each vertex $u \in L_{h-1}$ has at least

$$\frac{n}{\left(1 + \frac{1}{\log(n)}\right) \cdot (\log^{10}(n))^{h-1}} = \frac{\log^{10}(n)}{1 + \frac{1}{\log(n)}}$$

many neighbors in layer h.

With this observation and Claim A.1, it is easy to prove the following.

Claim A.2. *With high probability, the path LP is feasible on this instance for*

$$k = \frac{\log^{10}(n)}{1 + \frac{1}{\log(n)}}.$$

Proof. We set the following fractional values.

$$x(p) = \begin{cases} (1 + 1/\log n)^{-(h-1)} \cdot (\log^{10} n)^{-(h-1)} & \text{if } p \text{ ends at layer } h, \text{ and} \\ (1 + 1/\log n)^{-i} \cdot (\log^{10} n)^{-i} & \text{if } p \text{ ends at layer } i \leqslant h-1. \end{cases}$$

Clearly, $x((s)) = 1$. We can easily verify the demand constraints: let p be a path that ends at layer $i \leqslant h-2$, then

$$\sum_{q \in C(p)} x(q) = \frac{\log^{20} n}{(1 + 1/\log n)^{i+1} \cdot (\log^{10} n)^{i+1}} = \frac{k}{(1 + 1/\log n)^i \cdot (\log^{10} n)^i} = k \cdot x(p).$$

If p ends at layer $h-1$, then with high probability we have

$$\sum_{q \in C(p)} x(q) \geqslant \frac{\log^{10}(n)}{1+\frac{1}{\log n}} \cdot \frac{1}{(1+1/\log n)^{h-1} \cdot (\log^{10} n)^{h-1}} = k \cdot x(p) \, .$$

The capacity constraints are also easily verified with high probability. Let u be a vertex in layer $i \leqslant h-1$ and p a path ending in layer $j \leqslant i$ then

$$\begin{aligned} \sum_{q \in I(v) \cap D(p)} x(q) &\leqslant \frac{1}{(1+1/\log n)^{i} \cdot (\log^{10} n)^{i}} \\ &\leqslant \frac{1}{(1+1/\log n)^{j} \cdot (\log^{10} n)^{j}} = x(p) \, . \end{aligned}$$

If v is a vertex in layer h and p a path ending in layer $i \leqslant h-2$ then, using Claim A.1, we obtain

$$\begin{aligned} &\sum_{q \in I(v) \cap D(p)} x(q) \\ &\quad \leqslant \left(1+\frac{1}{\log n}\right) \cdot (\log^{10} n)^{h-1-i} \cdot \frac{1}{(1+1/\log n)^{h-1} \cdot (\log^{10} n)^{h-1}} \\ &\quad \leqslant \frac{1}{(1+1/\log n)^{h-2} \cdot (\log^{10} n)^{i}} \\ &\quad \leqslant x(p) \, . \end{aligned}$$

If v is a vertex in layer h and p a path ending in layer $i = h-1$ then

$$\sum_{q \in I(v) \cap D(p)} x(q) \leqslant \frac{1}{(1+1/\log n)^{h-1} \cdot (\log^{10} n)^{h-1}} = x(p) \, .$$

□

Given Claim A.2, one might be tempted to run the intuitive randomized rounding as in previous works. It is easy to show that this rounding will select a tree in which every internal node will have between $(1-2/\log n) \cdot \log^{10} n$ and $(1+2/\log n) \cdot \log^{10} n$ children with high probability. Note that this implies that this tree will have at least $(1-2/\log n)^{h-1} \cdot (\log^{10} n)^{h-1} = \Omega(\log^{10(h-1)} n)$ leaves in layer L_{h-1}. These leaves will each demand $k = \Omega(\log^{10}(n))$ sinks. Hence the total number of sinks needed will be $\Omega(\log^{10h} n) = \Omega(n)$. We prove the following last claim.

Claim A.3. *With high probability, any subtree T rooted at s with leaves at layer $h-1$ and in which every internal node has between*

$$\left(1-\frac{2}{\log n}\right) \log^{10}(n),$$

and

$$\left(1+\frac{2}{\log n}\right) \log^{10}(n)$$

many children is connected to at most

$$O\left(\frac{(\log^{10} n)^h}{h}\right) = O\left(\frac{n}{h}\right)$$

many sinks in layer L_h.

Before giving a proof sketch of this, we explain why this shows that some kind of pruning is needed. As explained above, the intuitive randomized rounding will select a tree in which every node has roughly $\log^{10}(n)$ children with high probability. This tree needs to be connected to $\Omega(n)$ sinks in layer L_h in order to give $\Omega(k)$ children to every leaf in L_{h-1}. By Claim A.3, only a $O(1/h)$ fraction of these sinks are available to these leaves, yielding an approximation factor of $\Omega(h) = \Omega(\log n/\log\log n)$ on the vertices in L_{h-1}. Hence without any post-processing that can modify the tree, it seems impossible to break through this $\tilde{\Omega}(\log n)$ approximation factor. Moreover, this instance shows that the post-processing cannot be only local: a simple post-processing one could think of is to remove vertices with high congestion and hope that this does not remove too many children of any other vertex. However, it does not suffice to remove only children of L_{h-1} in this example. Instead one needs to remove vertices from many layers. A careful reader may notice that if instead of sampling k children for every node (as it is the case in previous works) we allow ourselves to sample only $k/2$ children (which is equivalent to sample k children and then perform our top-to-bottom pruning); then the necessary number of sinks in L_h drops by a factor of 2^{-h}. This factor seems to be enough to overcome the issue highlighted by our construction.

We finish this discussion by giving a proof sketch of Claim A.3.

Proof sketch for Claim A.3. Let us call d the number of children of each node in a fixed tree T and assume for simplicity that $d = \log^{10}(n)$ for all internal nodes in the tree. For any vertex $u \in L_j \cap T$ with $j \leqslant h-1$, we will denote by T_u the subtree of T rooted at u. We say that a vertex $v \in L_h$ *survives* in T_u if at least one leaf of T_u is connected to v. We define

$$p_j := \mathbb{P}(v \text{ survives in } T_u \mid v \text{ selected } u),$$

for any $u \in L_j \cap T$. Note that by symmetry this probability does not depend on which vertex $u \in T \cap L_j$ and which sink v we select, hence the simplified notation. We will now upper bound p_j depending on the depth j. We note that for a vertex v to survive in T_u, it must be that there is at least one child u' of u such that v selects u' and v survives in $T_{u'}$. The probability that v survives in T_u conditioned by the fact that v selected u can be written as

$$\begin{aligned}&\mathbb{P}[v \text{ survives in } T_u \mid v \text{ selected } u]\\ &= 1 - \prod_{u' \text{ child of } u \text{ in the tree } T_u} \left(1 - \frac{P[v \text{ survives in } T_{u'} \mid v \text{ selected } u']}{\log^{10} n}\right)\end{aligned}$$

which by symmetry is equivalent to

$$p_j = 1 - \left(1 - \frac{p_{j+1}}{\log^{10} n}\right)^d.$$

When n goes to infinity, this is roughly equal to

$$1 - e^{-\frac{dp_{j+1}}{\log^{10} n}} \approx 1 - e^{-p_{j+1}}.$$

From there, it is easy to prove by induction that

$$p_j \leqslant \frac{O(1)}{h-j}.$$

Hence the expected number of vertices of the last layer that survive in any fixed tree T rooted at the source is at most

$$p_0 \cdot n = O\left(\frac{n}{h}\right).$$

By independence and a standard Chernoff bound, one can show that with probability at most

$$\exp\left(-\Omega\left(\frac{n}{h}\right)\right),$$

more than $\frac{10n}{h}$ sinks survive in a fixed tree T. To upper bound the number of possible trees, note that it suffices so select the set of $(\log^{10}(n))^{h-1}$ leaves among the $(\log^{20}(n))^{h-1}$ possible leaves. Hence there are at most

$$(\log^{20}(n))^{(h-1)\cdot(\log^{10} n)^{h-1}} = \exp\left(\Theta\left(\frac{n \cdot h \cdot \log\log(n)}{\log^{10} n}\right)\right) = \exp\left(o\left(\frac{n}{h}\right)\right)$$

possible trees with internal degree $\log^{10}(n)$. By a standard union bound, this happens for no such tree. □

A.2 PREPROCESSING THE PATH LP

We recall the goal is to obtain, from a feasible fractional solution x to the path LP a multiset of paths P' containing (s) for each $s \in S$ and satisfying

$$\sum_{q \in C_{P'}(p)} y(q) = \frac{k}{4} \cdot y(p) \qquad \forall p \in P' \setminus T' \qquad \text{(A.1)}$$

$$\sum_{q \in I_{P'}(v) \cap D_{P'}(p)} y(q) \leqslant 2y(p) \qquad \forall p \in P', v \in V \qquad \text{(A.2)}$$

$$y(p) = \frac{1}{(4\log^2 n)^i} \qquad \forall i \leqslant h, \forall p \in L_i \qquad \text{(A.3)}$$

The value of y is fixed by the distance to the sources; hence, the only challenge in this part is in fact to select the multiset P'.

Towards this, consider a randomized rounding algorithm, that proceeds layer by layer starting from the sources. We will produce partial solutions $P^{(i)}$ that correspond to our final solution for all paths in $L_{\leqslant i}$. Recall that this is a multiset and there might be several times the same copy of a path $p \in P$, but every copy will have a unique parent. Initially, we add a single copy of the trivial path (s) for each $s \in S$ to $P^{(0)}$. Assume now that we defined the solutions up to $P^{(i)}$, that is, we sampled up to layer i for some $i \geqslant 0$. For

each path p that is open and belongs to $P^{(i)} \cap L_i$, we sample $(k/4) \cdot (4 \log^2 n)$ many children paths where the j-th child equals path $q \in C_P(p) \cap L_{i+1}$ with probability

$$\frac{x(q)}{\sum_{q' \in C_P(p)} x(q')} .$$

For each path q that is sampled, we set the parent of q to be p (which is a unique copy). Let Q_q be the multiset of all copies of a path $q \in P$ that were sampled in this way. Then we set

$$P^{(i+1)} \leftarrow P^{(i)} \cup (\cup_{q \in L_{i+1}} Q_q).$$

We repeat this process until reaching the last layer h and we set $P' = P^{(h)}$. We argue below that we obtain the desired properties with high probability. To this end, we notice that each open path $p \in P^{(i)} \cap L_i$ samples exactly $(k/4) \cdot (4 \log^2 n)$ children paths q and hence we obtain

$$\sum_{q \in C_{P'}(p)} y(q) = \frac{(k/4) \cdot (4 \log^2 n)}{(4 \log^2 n)^{i+1}} = \frac{k}{4} y(p) .$$

Therefore constraint (A.1) is satisfied with probability 1. It remains to verify that constraint (A.2) holds with high probability. For this, we consider the following quantity, that keeps track of the (possibly fractional) congestion induced by the descendants of a path $p \in P^{(i)}$ on a vertex v. If $v \in L_{\leqslant i}$, then this induced congestion cannot change anymore so we will only keep track of this congestion for $v \in L_{\geqslant i+1}$. For ease of notation, let us define for all $q \in P$,

$$\mathrm{cong}(q, v) := \sum_{q' \in D_P(q) \cap I_P(v)} x(q') .$$

Then, we define for any $p \in P^{(i)}$

$$\mathrm{cong}(p, v \mid P^{(i)}) := \sum_{q \in D_{P^{(i)}}(p) \cap L_i} \frac{y(q)}{x(q)} \mathrm{cong}(q, v) ,$$

which intuitively is the fractional congestion induced by descendants of p on vertex v but where we condition by what happened until layer i. Note that if $v \in L_j$ the quantity $\mathrm{cong}(p, v \mid P^{(j)})$ is the final quantity we need to bound in order to ensure constraint (A.2). Indeed, in this case we obtain that

$$\mathrm{cong}(p, v \mid P^{(j)}) = \sum_{q \in D_{P^{(j)}}(p) \cap I_{P^{(j)}}(v)} \frac{y(q)}{x(q)} \cdot x(q) = \sum_{q \in D_{P^{(j)}}(p) \cap I_{P^{(j)}}(v)} y(q) .$$

Claim A.4. *For any $p \in P^{(i)} \cap L_i$, we have that*

$$\mathrm{cong}(p, v \mid P^{(i)}) \leqslant y(p),$$

with probability 1.

Proof. If $p \in P^{(i)} \cap L_i$, then we have that

$$\operatorname{cong}(p, v \mid P^{(i)}) = y(p)\frac{\operatorname{cong}(p,v)}{x(p)} \leqslant y(p) \ ,$$

using constraint (3.3) of the original path LP. □

Claim A.5. *For any open $p \in P^{(i)}$ and any $i \geqslant 0$, we have that*

$$\mathbb{P}\left[\operatorname{cong}(p, v \mid P^{(i+1)}) \geqslant \frac{\operatorname{cong}(p, v \mid P^{(i)})}{2} + y(p)\right] \leqslant \frac{1}{n^{2\log(n)}} \ ,$$

where the probability is taken over the randomness to round layer L_{i+1}.

Proof. Denote by N_q the number of copies of $q \in L_{i+1} \cap D_P(p)$ that are sampled as descendants of p. We also denote $P(q)$ the parent of q in the set P. Accordingly, $N_{P(q)}$ is the number of copies of the path $P(q)$ that are chosen in $P^{(i)}$ as descendants of p. First, we compute

$$\begin{aligned}
\mathbb{E}\left[\operatorname{cong}(p, v \mid P^{(i+1)})\right] &= \sum_{q \in D_P(p) \cap L_{i+1}} \mathbb{E}[N_q] \cdot \frac{1}{(4\log^2 n)^{i+1} \cdot x(q)} \cdot \operatorname{cong}(q, v) \\
&= \sum_{q \in D_P(p) \cap L_{i+1}} N_{P(q)} \cdot \frac{(k/4) \cdot (4\log^2 n)}{(4\log^2 n)^{i+1}} \cdot \frac{x(q)}{\sum_{q' \in C_P(P(q))} x(q')} \cdot \frac{\operatorname{cong}(q,v)}{x(q)} \\
&= \sum_{q \in D_P(p) \cap L_{i+1}} N_{P(q)} \cdot (k/4) \cdot \frac{1}{(4\log^2 n)^{i}} \cdot \frac{1}{\sum_{q' \in C_P(P(q))} x(q')} \cdot \operatorname{cong}(q, v) \\
&= \sum_{q \in D_P(p) \cap L_{i+1}} N_{P(q)} \cdot (k/4) \cdot y(P(q)) \cdot \frac{1}{k \cdot x(P(q))} \cdot \operatorname{cong}(q, v) \\
&= \frac{1}{4} \sum_{q \in D_P(p) \cap L_{i+1}} N_{P(q)} \cdot \frac{y(P(q))}{x(P(q))} \cdot \operatorname{cong}(q, v) \\
&= \frac{1}{4} \sum_{q' \in D_P(p) \cap L_i} N_{q'} \cdot \frac{y(q')}{x(q')} \cdot \sum_{q \in C_P(q')} \operatorname{cong}(q, v) \\
&= \frac{1}{4} \sum_{q' \in D_P(p) \cap L_i} N_{q'} \cdot \frac{y(q')}{x(q')} \cdot \operatorname{cong}(q', v) = \frac{\operatorname{cong}(p, v \mid P^{(i)})}{4}.
\end{aligned}$$

where we used constraint (3.2) of the original path LP to obtain the fourth line, and standard algebraic manipulations for the rest. Second, we note that $\operatorname{cong}(p, v \mid P^{(i+1)})$ can be written as a sum of independent random variables of value

$$\frac{y(q)}{x(q)} \cdot \operatorname{cong}(q, v)$$

for some $q \in L_{i+1}$. By constraint (3.3) of the path LP, the absolute value of these variables is never more than $1/(100\log^2 n)^{i+1}$. By a standard Chernoff bound, we can conlude that

$$\begin{aligned}
\mathbb{P}\left[\operatorname{cong}(p, v \mid P^{(i+1)}) \geqslant \frac{\operatorname{cong}(p, v \mid P^{(i)})}{2} + y(p)\right] &\leqslant \exp\left(-\frac{y(p)(4\log^2 n)^{i+1}}{2}\right) \\
&\leqslant n^{-2\log(n)} \ .
\end{aligned}$$

□

There are at most $n^h \leqslant n^{\log(n)+1}$ constraints to maintain to ensure constraint (A.2) over at most h rounds of rounding. Hence by Claim A.5, we have $\operatorname{cong}(p, v \mid P^{(i+1)}) \leqslant \operatorname{cong}(p, v \mid P^{(i)})/2 + y(p)$ for all paths p, vertices v, and rounds i with probability at least $1 - n^{-\log(n)+1}$. Using Claim A.4 with this fact we obtain that for any path p and vertex v

$$\sum_{q \in I_{P'}(v) \cap D_{P'}(p)} y(q) \leqslant y(p) \sum_{j=0}^{\infty} \frac{1}{2^j} \leqslant 2y(p).$$

This proves the desired result.

B

DEFERRED PROOFS FOR THE RESTRICTED SUBMODULAR SANTA CLAUS

B.1 REDUCTION TO HYPERGRAPH MATCHING PROBLEM

B.1.1 *Solving the Configuration LP*

The goal of this section is to prove Theorem 4.1. We consider the dual of the configuration LP (after adding an artificial minimization direction $\min 0^T x$).

$$\begin{aligned} \max & \sum_{i \in P} y_i - \sum_{j \in R} z_j \\ & \sum_{j \in C} z_j \geqslant y_i \quad \text{for all } i \in P, C \in \mathcal{C}(i,T) \\ & y_j, z_i \geqslant 0 \end{aligned}$$

Observe that the optimum of the dual is either 0 obtained by $y_i = 0$ and $z_j = 0$ for all i, j or it is unbounded: If it has any solution with $\sum_{i \in P} y_i - \sum_{j \in R} z_j > 0$, the variables can be scaled by an arbitrary common factor to obtain any objective value. If it is unbounded, this can therefore be certified by providing a feasible solution y, z with

$$\sum_{i \in P} y_i - \sum_{j \in R} z_j \geqslant 1. \tag{$*$}$$

We approximate the dual in the variant with constraint $(*)$ instead of a maximization direction using the ellipsoid method. The separation problem of the dual is as follows. Given z_j, y_i find a player i and set C with $g(C \cap \Gamma_i) \geqslant T$ such that $\sum_{j \in C} z_j < y_i$.

To this end, consider the related problem of maximizing a monotone submodular function subject to knapsack constraints. In this problem we are given a monotone submodular function g over a ground set E and the goal is to maximize $g(E')$ over all $E' \subseteq E$ with $\sum_{j \in E'} a_j \leqslant b$. Here $a_j \geqslant 0$ is a weight associated with $j \in E$ and b is a capacity. For this problem Srividenko gave a polynomial time $(1 - 1/e)$-approximation algorithm [80]. It is not hard to see that this can be used to give a constant approximation for the variation where strict inequality is required in the knapsack constraint: Assume w.l.o.g. that $0 < a_j < b$ for all j. Then run Srivideko's algorithm to find a set E' with $\sum_{j \in E'} a_j \leqslant b$. Notice that $g(E')$ is at least $(1 - 1/e)$OPT, also when OPT is the optimal value with respect to strict inequality. If E' contains only one element then equality in the knapsack constraint cannot hold and we are done. Otherwise, split E' into two arbitrary non-empty parts E'' and E'''. It follows that $\sum_{j \in E''} a_j < b$ and $\sum_{j \in E'''} a_j < b$. Moreover, either $g(E'') \geqslant g(E')/2$ or $g(E'') \geqslant g(E')/2$. Hence, this method yields a c-approximation for $c = (1 - 1/e)/2$. We now demonstrate how to use this to find a c-approximation to the configuration LP.

Let OPT be the optimum of the configuration LP. It suffices to solve the problem of finding for a given T either a solution of value cT or deciding that $T >$ OPT. This can then be embedded into a standard dual approximation framework. We run the ellipsoid method on the dual of the configuration LP with objective value cT and constraint $(*)$. This means we have to solve the separation problem. Let z, y be the variables at some state. We first check whether $(*)$ is satisfied, that is $\sum_{i\in P} y_i - \sum_{j\in R} z_j \geqslant 1$. If not, we return this inequality as a separating hyperplane. Hence, assume $(*)$ is satisfied and our goal is to find a violated constraint of the form $\sum_{j\in C} z_j < y_i$ for some $i \in P$ and $C \in \mathcal{C}(i,T)$. For each player i we maximize f over all $S \subseteq \Gamma_i$ with $\sum_{j\in S} z_j < y_i$. We use the variant of Srividenko's algorithm described above to obtain a c-approximation for each player. If for one player i the resulting set S satisfies $f(S) \geqslant cT$, then we have found a separating hyperplane to provide to the ellipsoid method. Otherwise, we know that $f(S) < T$ for all players i and $S \subseteq \Gamma_i$ with $\sum_{j\in S} z_j < y_i$. In other words, for all players i and all $C \in \mathcal{C}(i,T)$ it holds that $\sum_{j\in C} z_j \geqslant y_i$, i.e., z, y is feasible for objective value T and hence OPT $> T$. If the ellipsoid method terminates without concluding that OPT $> T$, we can derive a feasible primal solution with objective value cT: The configurations constructed for separating hyperplanes suffice to prove that the dual is bounded. These configurations can only be polynomially many by the polynomial running time of the ellipsoid method. Hence, when restricting the primal to these configurations it must remain feasible. To obtain the primal solution we now only need to solve a polynomial size linear program. This concludes the proof of Theorem 4.1.

B.1.2 *Clusters*

This section is devoted to prove Lemma 4.2. The arguments are similar to those used in [14].

Lemma B.1. *Let x^* be a solution to the configuration LP of value T^*. Then x^* can be transformed into some $x'_{i,C} \geqslant 0$ for $i \in P, C \in \mathcal{C}_t(i,T^*)$ which satisfies the following. There is a partition of the players into clusters $K_1 \cup \cdots \cup K_k \cup Q = P$ that satisfy the following.*

1. *any thin resource j is fractionally assigned at most once, that is,*

$$\sum_{i\in P} \sum_{C\in\mathcal{C}_t(i,T^*):j\in C} x'_{i,C} \leqslant 1$$

   We say that the congestion on resource j is at most 1.

2. *every cluster K_j gets at least $1/2$ thin configurations in x', that is,*

$$\sum_{i\in K_j} \sum_{C\in\mathcal{C}_t(i,T^*)} x'_{i,C} \geqslant 1/2;$$

3. *given any $i_1 \in K_1, i_2 \in K_2, \ldots, i_k \in K_k$ there is a matching of fat resources to players $P \setminus \{i_1, \ldots, i_k\}$ such that each of these players i gets a unique fat resource $j \in \Gamma_i$.*

The role of the set of players Q in the lemma above is that each of them gets one fat resource for certain.

Proof. We first transform the solution x^* as follows. For every configuration C (for player i) that contains at least one fat resource and such that $x^*_{i,C} > 0$, we select arbitrarily one of these fat resources j and we set $x^*_{i,\{j\}} = x^*_{i,C}$ and then we set $x^*_{i,C} = 0$. It is clear that this does not increase the congestion on resources and now every configuration that has non-zero value is either a thin configuration or a singleton containing one fat resource. Therefore we can consider the bipartite graph G formed between the players and the fat resources where there is an edge between player i and fat resource j if the corresponding configuration $C = \{j\}$ is of non zero value (i.e. $x^*_{i,C} > 0$). The value of such an edge will be exactly the value $x^*_{i,C}$. We now make G acyclic by doing the following operation until there exists no cycle anymore. Pick any cycle (which must have even length since the graph is bipartite) and increase the coordinate of x^* corresponding to every other edge in the cycle by a small constant. Decrease the value corresponding to the remaining edges of the cycle by the same constant. This ensures that fat resources are still (fractionally) taken at most once and that the players still have one unit of configurations fractionally assigned to them. We continue this until one of the edge value becomes 0 or 1. If an edge becomes 0, delete that edge and if it becomes 1, assign the corresponding resource to the corresponding player forever. Then delete the player and the resource from the graph and add the player to the cluster Q. By construction, every added player to Q is assigned a unique fat resource. Notice that when we stop, each remaining player still has at least 1 unit of configurations assigned to him and every fat resource is still (fractionally) taken at most once. Hence we get a new assignment vector where the assignments of fat resources to players form a forest. We also note that the congestion on thin resources did not increase during this process (it actually only decreased either when we replace fat configurations by a singleton and when players are put into the set Q and deleted from the instance). We show below how to get the clusters for any tree in the forest.

1. If the tree consists of a single player, then it trivially forms its own cluster. By feasibility of the original solution x^*, condition 2 of the lemma holds.

2. If there is a fat resource that has degree 1, assign it to its player, add the player to Q and delete both the player and resource. Continue this until every resource has a degree of at least 2. This step adds players to cluster Q. By construction, every added player is assigned a unique fat resource.

3. While there is a resource of degree at least 3, we perform the following operation. Root the tree containing such a resource at an arbitrary player. Consider a resource j of degree at least 3 such that the subtree rooted at this resource contains only resources of degree 2. Because this resource must have at least 2 children in the tree $i_1, i_2, \ldots$ (which are players) and because

$$\sum_{i \in P} \sum_{C : j \in C} x^*_{i,C} \leqslant 1,$$

it must be that one of the children (say i_1) satisfies $x^*_{i_1,\{j\}} \leqslant 1/2$. We then delete the edge (j, i_1) in the tree and set $x^*_{i_1,\{j\}}$ to 0.

4. Every resource now has degree exactly 2. We form a cluster for each tree in the forest. The cluster will contain the players and fat resources in the tree. We note that in every tree, only the player at the root lost at most $1/2$ unit of a fat resource by the previous step in the construction. By the degree property of resources and because the graph contains no cycle, it must be that in each cluster K we have $|R(K)| = |P(K)| - 1$ where $|R(K)|$ is the number of resources in the cluster and $|P(K)|$ the number of players. Because each resource is assigned at most once, and because only one player in the cluster lost at most $1/2$ unit of a fat resource, it must be that the cumulative amount of thin configurations assigned to players in K is at least

$$|P(K)| - |R(K)| - 1/2 = 1/2.$$

This gives the second property of the lemma. For the third property, notice that for any choice of player $i \in K$, we can root the tree corresponding to the cluster K at the player i and assign all the fat resources in K to their only child in the tree (they all have degree 2). This gives the third property of the lemma.

As each of these steps individually maintained maintained a congestion of at most 1 on every thin resource, we indeed get a new solution x' and the associated clusters with the required properties.

□

Lemma B.1 implies that for each cluster we need to cover only one player with a thin configuration. Then the remaining players can be covered with fat resources. We will now replace x' by a solution x'' which takes slightly worse configurations $\mathcal{C}_t(i, T^*/5)$, but satisfies (2) in Lemma B.1 with 2 instead of $1/2$. This can be achieved by splitting each configuration $C \in \mathcal{C}_t(i, T^*)$ in 4 disjoint parts $C_1, C_2, C_3, C_4 \in \mathcal{C}_t(i, T^*/5)$. Let $C_1 \subseteq C$ with $f(C_1) \geqslant T^*/5$ minimal in the sense that $f(C_1 \setminus \{j\}) < T^*/5$ for all $j \in C_1$. Let $j_1 \in C_1$. By submodularity and because j_1 is thin it holds that

$$f(C \setminus C_1) \geqslant f(C) - f(C_1 \setminus \{j_1\}) - f(\{j_1\}) \geqslant 4T^*/5 - T^*/100.$$

Hence, in the same way we can select $C_2 \subseteq C \setminus C_1$, $C_3 \subseteq C \setminus (C_1 \cup C_2)$ and $C_4 \subseteq C \setminus (C_1 \cup C_2 \cup C_3)$. We now augment x' to x'' by initializing x'' with 0 and then for each i and $C \in \mathcal{C}(i, T^*)$ increasing x''_{i,C_1}, x''_{i,C_2}, x''_{i,C_3}, and x''_{i,C_4} by $x'_{i,C}$. Here $C_1, C_2, C_3, C_4 \in \mathcal{C}(i, T^*/5)$ are the configurations derived from C by splitting it as described above.

Finally, we sample for each cluster some $\ell \geqslant 12\log(n)$ many configurations with the distribution of x'' to obtain the statement of Lemma 4.2 which we restate for convenience.

Lemma (Lemma 4.2 restated)**.** *Let $\ell \geqslant 12\log(n)$. Given a solution of value T^* for the configuration LP in randomized polynomial time we can find a partition of the players into clusters $K_1 \cup \cdots \cup K_k \cup Q = P$ and multisets of configurations $\mathcal{C}_h \subseteq \bigcup_{i \in K_h} \mathcal{C}_T(i, T^*/5)$, $h = 1, \ldots, k$, such that*

1. $|\mathcal{C}_h| = \ell$ *for all* $h = 1, \dots, k$ *and*
2. *Each small resource appears in at most* ℓ *configurations of* $\bigcup_h \mathcal{C}_h$.
3. *given any* $i_1 \in K_1, i_2 \in K_2, \dots, i_k \in K_k$ *there is a matching of fat resources to players* $P \setminus \{i_1, \dots, i_k\}$ *such that each of these players* i *gets a unique fat resource* $j \in \Gamma_i$.

Proof. We start with the clusters obtained with Lemma B.1 and the solution x'' described above. Recall that

$$\sum_{i \in K_h} \sum_{C \in C_t(i, T^*/5)} x''_{i,C} \geqslant 2$$

for each cluster K_h. We assume w.l.o.g. that equality holds by reducing some variables $x''_{i,C}$. Clearly then each resource is still contained in at most one configuration in total.

For each cluster K_h, we sample a configuration that contains a player in this cluster according to the probability distribution given by the values $\{x''_{i,C}/2\}_{i \in K_h, C \in \mathcal{C}_t(i, T^*/5)}$. By the assumption of equality stated above this indeed defines a probability distribution. We repeat this process ℓ times. We first note that for one iteration, each resource is in expectation contained in

$$\sum_{i \in P} \sum_{C \in \mathcal{C}(i, T^*/5) : j \in C} x''_{i,C}/2 \leqslant 1/2$$

selected configurations. Hence in expectation all the resource are contained in $\ell/2$ selected configurations after ℓ iterations. By a standard Chernoff bound (see Proposition 2.4), we have that with probability at most

$$\exp(-\ell/6) \leqslant 1/n^2$$

a resource is contained in more than ℓ configurations. By a union bound, it holds that all resources are contained in at most ℓ selected configurations with high probability. □

B.2 PROPERTIES OF RESOURCE SETS

Lemma B.2 (Lemma 4.7 restated)**.** *Consider Random Experiment 4.6 with* $\ell \geqslant 300.000 \log^3(n)$. *For any* $k \geqslant 0$ *and any* $C \in \mathcal{C}^{(\geqslant k)}$ *we have*

$$\frac{1}{2}\ell^{-k}|C| \leqslant |\mathcal{I}_k \cap C| \leqslant \frac{3}{2}\ell^{-k}|C|$$

with probability at least $1 - 1/n^{10}$.

Proof. The lemma trivially holds for $k = 0$. For $k > 0$, by assumption $C \in \mathcal{C}^{(\geqslant k)}$ hence $|C| \geqslant \ell^{k+3}$. Since each resource of $R = R_0$ survives in R_k with probability ℓ^{-k} we clearly have that in expectation

$$\mathbb{E}(|R_k \cap C|) = \ell^{-k}|C|$$

Hence the random variable $X = |R_k \cap C|$ is a sum of independent variables of value either 0 or 1 and such that $\mathbb{E}(X) \geqslant \ell^3$. By a standard Chernoff bound (see Proposition 2.4), we get

$$\mathbb{P}\left(X \notin \left[\frac{\mathbb{E}(X)}{2}, \frac{3\mathbb{E}(X)}{2}\right]\right) \leqslant 2\exp\left(-\frac{\mathbb{E}(X)}{12}\right) \leqslant 2\exp\left(-\frac{300.000\log^3(n)}{12}\right) \leqslant \frac{1}{n^{10}},$$

since by assumption $\ell \geqslant 300.000\log^3(n)$. □

Lemma B.3 (Lemma 4.8 restated)**.** *Consider Random Experiment 4.6 with $\ell \geqslant 300.000\log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathcal{C}^{(\geqslant k)}$ we have*

$$\sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C \cap R_k| \leqslant \frac{10}{\ell^k}\left(|C| + \sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C|\right)$$

with probability at least $1 - 1/n^{10}$.

Proof. The expected value of the random variable $X = \sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C \cap R_k|$ is equal to

$$\mathbb{E}(X) = \frac{1}{\ell^k} \sum_{C' \in \mathcal{C}^{(k)}} |C' \cap C|.$$

Since each resource is in at most ℓ configurations, X is a sum of independent random variables that take value in a range $[0, \ell]$. Then by a standard Chernoff bound (see Lemma 2.4), we get

$$\mathbb{P}\left(X \geqslant 10\left(\frac{|C|}{\ell^k} + \mathbb{E}(X)\right)\right) \leqslant \exp\left(-\frac{3|C|}{\ell^{k+1}}\right) \leqslant \frac{1}{n^{10}},$$

since by assumption, $|C| \geqslant \ell^{k+3}$ and $\ell \geqslant 300.000\log^3(n)$.

□

We finish by the proof of the last property. As mentioned in Chapter 4, this statement is a generalization of some ideas that already appeared in [14]. However, in [14], the situation is simpler since they need to sample down the resource set only once (i.e. there are only two sets $R_1 \subseteq R$ and not a full hierarchy of resource sets $R_d \subseteq R_{d-1} \subseteq \cdots \subseteq R_1 \subseteq R$). Given the resource set R_1, they want to select configurations and give to each selected configuration K all of its resource set $|K \cap R_1|$ so that no resource is assigned too many times. In our case the situation is also more complex than that since at every step the selected configurations receive only a fraction of their current resource set. Nevertheless, we extend the ideas of Bansal and Srividenko to our more general setting. We recall the main statement before proceeding to its proof.

Lemma B.4 (Lemma 4.9 restated). *Consider Random Experiment 4.6 with $\ell \geqslant 300.000 \log^3(n)$. Fix $k \geqslant 0$. Conditioned on the event that the bounds in Lemma 4.7 hold for k, then with probability at least $1-1/n^{10}$ the following holds for all $\mathcal{F} \subseteq \mathcal{C}^{(\geqslant k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$, and $\gamma \in \mathbb{N}$ such that $\ell^3/1000 \leqslant \alpha(C) \leqslant n$ for all $C \in \mathcal{F}$ and $\gamma \in \{1, \ldots, \ell\}$: If there is a (α, γ)-good assignment of R_{k+1} to $\mathcal{F}$, then there is a (α', γ)-good assignment of R_k to $\mathcal{F}$ where*

$$\alpha'(C) \geqslant \ell \left(1 - \frac{1}{\log(n)}\right) \alpha(C) \tag{B.1}$$

for all $C \in \mathcal{F}$. Moreover, this assignment can be found in polynomial time.

We first provide the definitions of a flow network that allows us to state a clean condition whether a good assignment of resources exists or not. We then provide the high probability statements that imply the lemma.

For any subset of configurations $\mathcal{F} \subseteq \mathcal{C}^{(\geqslant k+1)}$, resource set R_k, $\alpha : \mathcal{F} \to \mathbb{N}$, and any integer γ, consider the following directed network (denoted by $\mathcal{N}(\mathcal{F}, R_k, \alpha, \gamma)$). Create a vertex for each configuration in $\mathcal{F}$ as well as a vertex for each resource. Add a source s and sink t. Then add a directed arc from s to the vertex $C \in \mathcal{F}$ with capacity $\alpha(C)$. For every pair of a configuration C and a resource i such that $i \in C$ add a directed arc from C to i with capacity 1. Finally, add a directed arc from every resource to the sink of capacity γ. See Figure B.1 for an illustration.

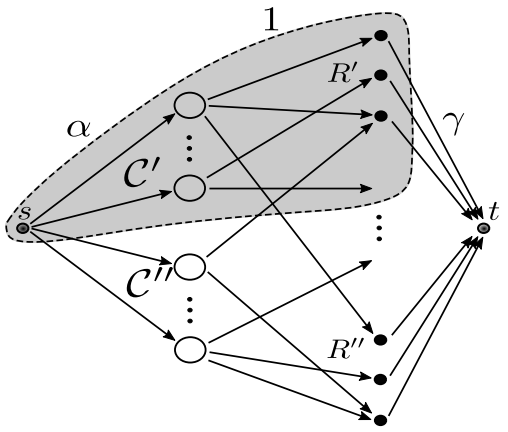

Figure B.1: The directed network and an s-t cut.

We denote by

$$\text{maxflow}\left(\mathcal{N}(\mathcal{F}, R_k, \alpha, \gamma)\right)$$

the value of the maximum s-t flow in $\mathcal{N}(\mathcal{F}, R_k, \alpha, \gamma)$.

Before delving into the technical lemmas, we provide a brief road map for the proof. First, we argue that for any subset of configurations, in the two networks induced on this subset and the consecutive resource sets (which are R_k and R_{k+1}), the value of the maximum flow differs by approximately a factor ℓ (this is Lemma B.6 stated below). Then by a union bound over all possible subsets of configurations, we say that the above argument consecutively holds with good probability. This helps us conclude that a good

assignment of the resource set R_{k+1} implies that there is a good assignment of the resource set R_k. Notice that if one does not have the above argument with respect to all subsets of configurations at once, it is not necessary that a good assignment of resources must exist. In particular, we need Lemma B.5 to show that if on *all* subsets of configurations the maximum flow is multiplied by *approximately* ℓ when we expand the resource set from R_{k+1} to R_k, then an (α, γ)-good assignment of R_{k+1} implies an (α', γ)-good assignment of R_k, where α' is almost equal to $\ell\alpha$.

Lemma B.5. *Let $\mathcal{F}$ be a set of configurations, $R' \subseteq R$, $\alpha : \mathcal{F} \to \mathbb{N}$ a set of resources, $\gamma \in \mathbb{N}$, and $\varepsilon \geqslant 0$. Define*

$$\alpha'(C) = \lfloor (1-\varepsilon)\alpha(C) \rfloor.$$

There is an (α', γ)-good assignment of R' to $\mathcal{F}$ if and only if for every $\mathcal{F}' \subseteq \mathcal{F}$, the maximum flow in the network $\mathcal{N}(\mathcal{F}', R', \alpha, \gamma)$ is of value at least $\sum_{C \in \mathcal{F}'} \alpha'(C)$. Moreover, this assignment can be found in polynomial time.

Proof. First assume there is such an (α', γ)-good assignment. Then send a flow of $\alpha'(C)$ from s to each $C \in \mathcal{F}$. If resource i is assigned to C, send a flow of 1 from C to i. Finally ensure that flow is preserved at every vertex corresponding to a resource by sending the correct amount of flow to t. Since no resource is taken more than γ times, this flow is feasible.

We prove the other direction by contradiction. Denote by $\mathcal{N}$ the network $\mathcal{N}(\mathcal{F}, R', \alpha', \gamma)$. If there is no good assignment satisfying the condition of the lemma then the maximum flow in $\mathcal{N}$ must be strictly less than $\sum_{C \in \mathcal{F}} \alpha'(C)$ (otherwise consider the maximum flow, which can be taken to be integral, and give to every configuration C all the resources to which they send a flow of 1). Then by the max-flow min-cut theorem, there exists an s-t cut S that has value strictly less than $\sum_{C \in \mathcal{F}} \alpha'(C)$. Let $\mathcal{C}'$ be the set of configurations on the side of the source in S. Notice that $\mathcal{C}'$ cannot be empty by assumption on the value of the cut.

Consider the induced network $\mathcal{N}(\mathcal{C}', R', \alpha', \gamma)$ and the cut S in it. It has a value strictly lower than $\sum_{C \in \mathcal{C}'} \alpha'(C)$. This, in turn implies that the cut S in $\mathcal{N}(\mathcal{C}', R', \alpha, \gamma)$ has a value strictly lower than $\sum_{C \in \mathcal{C}'} \alpha'(C)$, since this cut does not contain any edge from the source s to some configuration. Hence the maximum flow in $\mathcal{N}(\mathcal{C}', R', \alpha, \gamma)$ has a value strictly less than $\sum_{C \in \mathcal{C}'} \alpha'(C)$, a contradiction to the assumption in the premise. □

Lemma B.6. *Let $\mathcal{F} \subseteq \mathcal{C}^{\geqslant (k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$ such that $\ell^3/1000 \leqslant \alpha(C) \leqslant n$ for all $C \in \mathcal{F}$, and $1 \leqslant \gamma \leqslant \ell$. Denote by $\mathcal{N}$ the network $\mathcal{N}(\mathcal{F}, R_k, \ell \cdot \alpha, \gamma)$ and by $\tilde{\mathcal{N}}$ the network $\mathcal{N}(\mathcal{F}, R_{k+1}, \alpha, \gamma)$. Then*

$$\operatorname{maxflow}(\mathcal{N}) \geqslant \frac{\ell}{1 + 0.5/\log(n)} \operatorname{maxflow}\left(\tilde{\mathcal{N}}\right)$$

with probability at least $1 - 1/(n\ell)^{20|\mathcal{F}|}$.

Proof. We use the max-flow min-cut theorem that asserts that the value of the maximum flow in a network is equal to the value of the minimum s-t cut in the network. Consider a minimum cut S of network $\mathcal{N}$ with $s \in S$

and $t \notin S$. Denote by $c(S)$ the value of the cut. We will argue that with high probability this cut induces a cut of value at most $c(S)/\ell \cdot (1 + 0.5/\log(n))$ in the network $\tilde{\mathcal{N}}$. This directly implies the lemma.

Denote by $\mathcal{C}'$ the set of configurations of $\mathcal{F}$ that are in S, i.e., on the source side of the cut, and $\mathcal{C}'' = \mathcal{F} \setminus \mathcal{C}'$. Similarly consider R' the set of resources in the s side of the cut and $R'' = R_k \setminus R'$. With a similar notation, we denote $\tilde{R}' = R' \cap R_{k+1}$ the set of resources of R' surviving in R_{k+1}; and $\tilde{R}'' = R'' \cap R_{k+1}$. Finally, denote by $\tilde{S}$ the cut in $\tilde{\mathcal{N}}$ obtained by removing resources of R' that do not survive in R_{k+1} from S, i.e., $\tilde{S} = \{s\} \cup \mathcal{C}' \cup R'$. The value of the cut S of $\mathcal{N}$ is

$$c(S) = \sum_{C \in \mathcal{C}''} \ell \cdot \alpha(C) + e(\mathcal{C}', R'') + \gamma |R'|$$

where $e(X, Y)$ denotes the number of edges from X to Y. The value of the cut $\tilde{S}$ in $\tilde{\mathcal{N}}$ is

$$c(\tilde{S}) = \sum_{C \in \mathcal{C}''} \alpha(C) + e(\mathcal{C}', \tilde{R}'') + \gamma |\tilde{R}'|$$

We claim the following properties.

Claim B.7. *For every $C \in \mathcal{F}$, the outdegree of the vertex corresponding to C in $\mathcal{N}$ is at least $\ell^4/2$.*

Since $C \in \mathcal{C}^{(\geqslant k+1)}$ and by Lemma 4.7, we clearly have that $|C \cap R_k| \geqslant \ell^4/2$.

Claim B.8. *It holds that*

$$c(S) \geqslant \frac{|\mathcal{F}|\ell^3}{1000}.$$

We have by assumption on $\alpha(C)$

$$\begin{aligned} c(S) = \sum_{C \in \mathcal{C}''} \ell \cdot \alpha(C) + e(\mathcal{C}', R'') + \gamma |R'| &\geqslant \sum_{C \in \mathcal{C}''} \frac{\ell^3}{1000} + e(\mathcal{C}', R'') + \gamma |R'| \\ &\geqslant \frac{|\mathcal{C}''|\ell^3}{1000} + e(\mathcal{C}', R'') + \gamma |R'| \,. \end{aligned}$$

Now consider the case where $e(\mathcal{C}', R'') \leqslant |\mathcal{C}'|\ell^3/1000$. Since each vertex in $\mathcal{C}'$ has outdegree at least $\ell^4/2$ in the network $\mathcal{N}$ (by Claim B.7) it must be that $e(\mathcal{C}', R') \geqslant |\mathcal{C}'|\ell^4/2 - |\mathcal{C}'|\ell^3/1000 > |\mathcal{C}'|\ell^4/3$. Using that each vertex in R' has indegree at most ℓ (each resource is in at most ℓ configurations), this implies $|R'| \geqslant |\mathcal{C}'|\ell^3/3$. Since $\gamma \geqslant 1$ we have in all cases that $e(\mathcal{C}', R'') + \gamma |R'| \geqslant |\mathcal{C}'|\ell^3/1000$. Hence

$$c(S) \geqslant \frac{|\mathcal{C}''|\ell^3}{1000} + \frac{|\mathcal{C}'|\ell^3}{1000} = \frac{|\mathcal{F}|\ell^3}{1000}.$$

This proves Claim B.8. We can now finish the proof of the lemma. Denote by X the value of the random variable $e(\mathcal{C}', \tilde{R}'') + \gamma |\tilde{R}'|$. We have that

$$\mathbb{E}[X] = \frac{1}{\ell}(e(\mathcal{C}', R'') + \gamma |R'|).$$

Moreover, X can be written as a sum of independent variables in the range $[0, \ell]$ since each vertex is in at most ℓ configurations and $\gamma \leqslant \ell$ by assumption. By a Chernoff bound (see Lemma 2.4) with

$$\delta = \frac{0.5c(S)}{\log(n) \cdot (c(S) - \sum_{C \in \mathcal{C}''} \alpha(C))} \geqslant \frac{0.5}{\log(n)} ,$$

we have that

$$\begin{aligned} \mathbb{P}\left(X \geqslant \mathbb{E}(X) + \frac{0.5c(S)}{\ell \log(n)}\right) &\leqslant \exp\left(-\frac{\min\{\delta, \delta^2\}\mathbb{E}(X)}{3\ell}\right) \\ \leqslant \exp\left(-\frac{c(S)}{12\ell^2 \log^2(n)}\right) &\leqslant \exp\left(-\frac{|\mathcal{F}|\ell^3}{12.000\ell^2 \log^2(n)}\right) \leqslant \frac{1}{(n\ell)^{20|\mathcal{F}|}}, \end{aligned}$$

where the third inequality comes from Claim B.8 and the last one from the assumption that $\ell \geqslant 300.000 \log^3(n)$. Hence with probability at least $1 - 1/(n\ell)^{20|\mathcal{F}|}$, we have that

$$c(\tilde{S}) = \sum_{C \in \mathcal{C}''} \alpha(C) + e(\mathcal{C}', \tilde{R}'') + \gamma|\tilde{R}'| \leqslant \frac{1}{\ell} c(S) + \frac{0.5}{\ell \log(n)} c(S) .$$

□

We are now ready to prove Lemma 4.9. Note that Lemma B.6 holds with probability at least $1 - 1/(n\ell)^{20|\mathcal{F}|}$. Given the resource set R_k and a cardinality $s = |\mathcal{F}|$ there are $O((n\ell)^{2s})$ ways of defining a network satisfying the conditions from Lemma B.6 ($(m\ell)^s \leqslant (n\ell)^s$ choices of $\mathcal{F}$, n^s choices for α and ℓ choices for γ). By a union bound, we can assume that the properties of Lemma B.6 hold for every possible network with probability at least $1 - 1/n^{10}$. Assume now there is a (α, γ)-good assignment of R_{k+1} to some family $\mathcal{F}$. Then by Lemma B.5 the $\text{maxflow}(\mathcal{N}(\mathcal{F}', R_{k+1}, \alpha, \gamma))$ is exactly $\sum_{C \in \mathcal{F}'} \alpha(C)$ for any $\mathcal{F}' \subseteq \mathcal{F}$. By Lemma B.6, this implies that $\text{maxflow}(\mathcal{N}(\mathcal{F}', R_k, \ell \cdot \alpha, \gamma))$ is at least $\ell/(1 + 0.5/\log(n)) \sum_{C \in \mathcal{F}'} \alpha(C)$. By Lemma B.5, this implies a (α', γ)-good assignment from R_k to $\mathcal{F}$, where

$$\begin{aligned} \alpha'(C) = \lfloor \ell/(1 + 0.5/\log(n)) \rfloor \alpha(C) &\geqslant \ell/(1 + 1/\log(n))\alpha(C) \\ &\geqslant \ell(1 - 1/\log(n))\alpha(C) . \end{aligned}$$

BIBLIOGRAPHY

[1] David Adjiashvili. "Beating approximation factor two for weighted tree augmentation with bounded costs." In: *ACM Transactions on Algorithms* 15.2 (2018), pp. 1–26.

[2] Anthony Alexander, Sylvia Boyd, and Paul Elliott-Magwood. *On the integrality gap of the 2-edge connected subgraph problem.* Tech. rep. Citeseer, 2006.

[3] Noga Alon and Yossi Azar. "On-line Steiner trees in the Euclidean plane." In: *Proceedings of SoCG* (1992), pp. 337–343.

[4] Noga Alon, Shlomo Hoory, and Nathan Linial. "The Moore bound for irregular graphs." In: *Graphs and Combinatorics* 18.1 (2002), pp. 53–57.

[5] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. "Combinatorial algorithm for restricted max-min fair allocation." In: *ACM Transactions on Algorithms* 13.3 (2017), pp. 1–28.

[6] Arash Asadpour, Uriel Feige, and Amin Saberi. "Santa claus meets hypergraph matchings." In: *ACM Transactions on Algorithms* 8.3 (2012), 24:1–24:9.

[7] Arash Asadpour and Amin Saberi. "An approximation algorithm for max-min fair allocation of indivisible goods." In: *Proceedings of STOC* (2007), pp. 114–121.

[8] Baruch Awerbuch, Yossi Azar, and Yair Bartal. "On-line generalized Steiner problem." In: *Theoretical Computer Science* 324.2-3 (2004), pp. 313–324.

[9] Étienne Bamas, Marina Drygala, and Andreas Maggiori. "An Improved Analysis of Greedy for Online Steiner Forest." In: *Proceedings of SODA* (2022), pp. 3202–3229.

[10] Étienne Bamas, Marina Drygala, and Ola Svensson. "A Simple LP-Based Approximation Algorithm for the Matching Augmentation Problem." In: *Proceedings of IPCO* (2022), pp. 57–69.

[11] Etienne Bamas, Paritosh Garg, and Lars Rohwedder. "The Submodular Santa Claus Problem in the Restricted Assignment Case." In: *Proceedings of ICALP* (2021), 22:1–22:18.

[12] Étienne Bamas and Lars Rohwedder. "Better Trees for Santa Claus." In: *arXiv preprint arXiv:2211.14259* (2022).

[13] Nikhil Bansal. *Scheduling: Open problems old and new.* Presentation at MAPSP (2017).

[14] Nikhil Bansal and Maxim Sviridenko. "The santa claus problem." In: *Proceedings of STOC* (2006), pp. 31–40.

[15] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. "Maxmin allocation via degree lower-bounded arborescences." In: *Proceedings of STOC* (2009), pp. 543–552.

[16] Piotr Berman and Chris Coulston. "On-line algorithms for Steiner tree problems." In: *Proceedings of STOC* (1997), pp. 344–353.

[17] Ivona Bezáková and Varsha Dani. "Allocating indivisible goods." In: *ACM SIGecom Exchanges* 5.3 (2005), pp. 11–18.

[18] Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.

[19] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. "Steiner tree approximation via iterative randomized rounding." In: *Journal of the ACM (JACM)* 60.1 (2013), pp. 1–33.

[20] Federica Cecchetto, Vera Traub, and Rico Zenklusen. "Bridging the gap between tree and connectivity augmentation: unified and stronger approaches." In: *Proceedings of STOC* (2021), pp. 370–383.

[21] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. "On allocating goods to maximize fairness." In: *Proceedings of FOCS* (2009), pp. 107–116.

[22] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. "Approximation algorithms for directed Steiner problems." In: *Journal of Algorithms* 33.1 (1999), pp. 73–91.

[23] Ho-Lin Chen, Tim Roughgarden, and Gregory Valiant. "Designing network protocols for good equilibria." In: *SIAM Journal on Computing* 39.5 (2010), pp. 1799–1832.

[24] Siu-Wing Cheng and Yuchen Mao. "Restricted Max-Min Fair Allocation." In: *Proceedings of ICALP* 107 (2018), 37:1–37:13.

[25] Siu-Wing Cheng and Yuchen Mao. "Restricted Max-Min Allocation: Approximation and Integrality Gap." In: *Proceedings of ICALP* (2019), 38:1–38:13.

[26] Joe Cheriyan, Jack Dippel, Fabrizio Grandoni, Arindam Khan, and Vishnu V Narayan. "The matching augmentation problem: a 7/4-approximation algorithm." In: *Mathematical Programming* 182.1 (2020), pp. 315–354.

[27] Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu. "An improved approximation algorithm for the matching augmentation problem." In: *SIAM Journal on Discrete Mathematics* 37.1 (2023), pp. 163–190.

[28] Joseph Cheriyan and Zhihan Gao. "Approximating (unweighted) tree augmentation via lift-and-project, part I: stemless TAP." In: *Algorithmica* 80.2 (2018), pp. 530–559.

[29] Joseph Cheriyan and Zhihan Gao. "Approximating (unweighted) tree augmentation via lift-and-project, part II." In: *Algorithmica* 80.2 (2018), pp. 608–651.

[30] Joseph Cheriyan, Howard Karloff, Rohit Khandekar, and Jochen Könemann. "On the integrality ratio for tree augmentation." In: *Operations Research Letters* 36.4 (2008), pp. 399–401.

[31] Herman Chernoff. "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations." In: *The Annals of Mathematical Statistics* (1952), pp. 493–507.

[32] Nachshon Cohen and Zeev Nutov. "A (1+ ln2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius." In: *Theoretical Computer Science* 489 (2013), pp. 67–74.

[33] Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef. "The traveling salesman problem on a graph and some related integer polyhedra." In: *Mathematical programming* 33.1 (1985), pp. 1–27.

[34] Artur Czumaj and Andrzej Lingas. "On approximability of the minimum-cost k-connected spanning subgraph problem." In: *Proceedings of SODA* (1999), pp. 281–290.

[35] Sami Davies, Thomas Rothvoss, and Yihao Zhang. "A tale of Santa Claus, hypergraphs and matroids." In: *Proceedings of SODA* (2020), pp. 2748–2757.

[36] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. "Greedy algorithms for online survivable network design." In: *Proceedings of ICALP* (2018), 152:1–152:14.

[37] Guy Even, Jon Feldman, Guy Kortsarz, and Zeev Nutov. "A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2." In: *ACM Transactions on Algorithms* 5.2 (2009), pp. 1–17.

[38] Uriel Feige. "A threshold of ln n for approximating set cover." In: *Journal of the ACM (JACM)* 45.4 (1998), pp. 634–652.

[39] Uriel Feige. "On allocations that maximize fairness." In: *Proceedings of SODA* (2008), pp. 287–293.

[40] Amos Fiat and Gerhard J Woeginger. *Online algorithms: The state of the art*. Vol. 1442. Springer, 1998.

[41] Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. "Approximating weighted tree augmentation via Chvátal-Gomory cuts." In: *Proceedings of SODA* (2018), pp. 817–831.

[42] Greg N Frederickson and Joseph Ja'ja. "On the relationship between the biconnectivity augmentation and travelling salesman problems." In: *Theoretical Computer Science* 19.2 (1982), pp. 189–201.

[43] Greg N Frederickson and Joseph Ja'Ja'. "Approximation algorithms for several graph augmentation problems." In: *SIAM Journal on Computing* 10.2 (1981), pp. 270–283.

[44] Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. "Improved Approximation for Two-Edge-Connectivity." In: *Proceedings of SODA* (2023), pp. 2368–2410.

[45] Mohit Garg, Felix Hommelsheim, and Nicole Megow. "Matching Augmentation via Simultaneous Contractions." In: *arXiv preprint arXiv:2211.01912* (2022).

[46] Michel X Goemans, Nicholas JA Harvey, Satoru Iwata, and Vahab Mirrokni. "Approximating submodular functions everywhere." In: *Proceedings of SODA* (2009), pp. 535–544.

[47] Michel X Goemans and David P Williamson. "A general approximation technique for constrained forest problems." In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317.

[48] Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. "Breaching the 2-approximation barrier for the forest augmentation problem." In: *Proceedings of STOC* (2022), pp. 1598–1611.

[49] Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. "Improved approximation for tree augmentation: saving by rewiring." In: *Proceedings of STOC* (2018), pp. 632–645.

[50] Anupam Gupta and Amit Kumar. "Greedy algorithms for steiner forest." In: *Proceedings of STOC* (2015), pp. 871–878.

[51] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. "Integrality ratio for group Steiner trees and directed Steiner trees." In: *SIAM Journal on Computing* 36.5 (2007), pp. 1494–1511.

[52] Eran Halperin and Robert Krauthgamer. "Polylogarithmic inapproximability." In: *Proceedings of STOC* (2003), pp. 585–594.

[53] Penny Haxell and Tibor Szabó. "Improved Integrality Gap in Max-Min Allocation: or Topology at the North Pole." In: *Proceedings of SODA* (2023), pp. 2875–2897.

[54] Dorit S Hochbaum and David B Shmoys. "Using dual approximation algorithms for scheduling problems theoretical and practical results." In: *Journal of the ACM (JACM)* 34.1 (1987), pp. 144–162.

[55] Christoph Hunkenschröder, Santosh Vempala, and Adrian Vetta. "A 4/3-Approximation Algorithm for the Minimum 2-Edge Connected Subgraph Problem." In: *ACM Transactions on Algorithms* 15.4 (2019). ISSN: 1549-6325.

[56] Makoto Imase and Bernard M Waxman. "Dynamic Steiner tree problem." In: *SIAM Journal on Discrete Mathematics* 4.3 (1991), pp. 369–384.

[57] AI Impacts. *Global computing capacity*. 2015. URL: https://aiimpacts.org/global-computing-capacity (visited on 12/12/2022).

[58] Kamal Jain. "A factor 2 approximation algorithm for the generalized Steiner network problem." In: *Combinatorica* 21.1 (2001), pp. 39–60.

[59] Richard M Karp. "Reducibility among combinatorial problems." In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[60] L.G. Khachiyan. "Polynomial algorithms in linear programming." In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72.

[61] Samir Khuller and Ramakrishna Thurimella. "Approximation algorithms for graph augmentation." In: *Journal of algorithms* 14.2 (1993), pp. 214–225.

[62] Samir Khuller and Uzi Vishkin. "Biconnectivity Approximations and Graph Carvings." In: *Journal of the ACM (JACM)* 41.2 (1994), 214–235.

[63] Guy Kortsarz and Zeev Nutov. "A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2." In: *ACM Transactions on Algorithms* 12.2 (2015), pp. 1–20.

[64] Guy Kortsarz and Zeev Nutov. "LP-relaxations for tree augmentation." In: *Discrete Applied Mathematics* 239 (2018), pp. 94–105.

[65] Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. "Simultaneous placement and scheduling of sensors." In: *Proceedings of IPSN* (2009), pp. 181–192.

[66] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press, 2011.

[67] Benny Lehmann, Daniel Lehmann, and Noam Nisan. "Combinatorial auctions with decreasing marginal utilities." In: *Games Econ. Behav.* 55.2 (2006), pp. 270–296.

[68] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. "Approximation algorithms for scheduling unrelated parallel machines." In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[69] Tobias Mömke and Ola Svensson. "Removing and adding edges for the traveling salesman problem." In: *Journal of the ACM (JACM)* 63.1 (2016), pp. 1–28.

[70] Robin A Moser and Gábor Tardos. "A constructive proof of the general Lovász local lemma." In: *Journal of the ACM (JACM)* 57.2 (2010), pp. 1–15.

[71] Marcin Mucha. "13/9 -Approximation for Graphic TSP." In: *Theory of Computing Systems* 55.4 (2014), pp. 640–657.

[72] Hiroshi Nagamochi. "An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree." In: *Discrete Applied Mathematics* 126.1 (2003), pp. 83–113.

[73] Alantha Newman. "An Improved Analysis of the Mömke-Svensson Algorithm for Graph-TSP on Subquartic Graphs." In: *SIAM Journal on Discrete Mathematics* 34.1 (2020), pp. 865–884.

[74] Zeev Nutov. "On the tree augmentation problem." In: *Algorithmica* 83.2 (2021), pp. 553–575.

[75] Debmalya Panigrahi. *COMPSCI 638: Graph Algorithms*. 2019. URL: https://www2.cs.duke.edu/courses/fall19/compsci638/fall19_notes/lecture16.pdf (visited on 02/20/2023).

[76] Lukáš Poláček and Ola Svensson. "Quasi-polynomial local search for restricted max-min fair allocation." In: *ACM Transactions on Algorithms* 12.2 (2015), pp. 1–13.

[77] Gabriel Robins and Alexander Zelikovsky. "Improved Steiner tree approximation in graphs." In: *Proceedings of SODA* (2000), pp. 770–779.

[78] Petra Schuurman and Gerhard J Woeginger. "Polynomial time approximation algorithms for machine scheduling: Ten open problems." In: *Journal of Scheduling* 2.5 (1999), pp. 203–213.

[79] András Sebö and Jens Vygen. "Shorter tours by nicer ears: 7/5-Approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs." In: *Combinatorica* 34.5 (2014), pp. 597–629.

[80] Maxim Sviridenko. "A note on maximizing a submodular set function subject to a knapsack constraint." In: *Oper. Res. Lett.* 32.1 (2004), pp. 41–43.

[81] V Traub and R Zenklusen. "A better-than-2 approximation for weighted tree augmentation." In: *Proceedings of FOCS* (2021), pp. 1–12.

[82] Vera Traub and Rico Zenklusen. "Local search for weighted tree augmentation and Steiner tree." In: *Proceedings of SODA* (2022), pp. 3253–3272.

[83] Jan Vondrák. "Optimal approximation for the submodular welfare problem in the value oracle model." In: *Proceedings of STOC* (2008). Ed. by Cynthia Dwork, pp. 67–74.

[84] Jeffery Westbrook and Dicky C. K. Yan. "The performance of greedy algorithms for the on-line Steiner tree and related problems." In: *Mathematical systems theory* 28.5 (1995), pp. 451–468.

[85] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[86] Alexander Z Zelikovsky. "An 11/6-approximation algorithm for the network Steiner problem." In: *Algorithmica* 9.5 (1993), pp. 463–470.

CURRICULUM VITAE

EDUCATION

- (2018-2023) **EPFL**, Lausanne, Switzerland.
  PhD student in Theoretical Computer Science, Advisor: Ola Svensson.
- (2017-2018) **Université Paris Diderot - École Normale Supérieure**, Paris, France.
  M.Sc. in Theoretical Computer Science (MPRI) (*summa cum laude*); GPA: 18.81/20 (rank: 2/63).
- (2014-2018) **École polytechnique (Engineering diploma)**, Paris, France.
- (2012-2014) **Lycée Louis-Le-Grand, classe préparatoire MPSI/MP***, Paris, France.
- (2012) French "Baccalauréat série S" (*summa cum laude*), Paris, France.

WORK EXPERIENCE

- (Nov.-Dec. 2022) **Academic visit at the Simons Institute**, Berkeley, USA.
- (Apr.-Aug. 2018) **Research internship at the G-SCOP lab**, Grenoble, France.
- (Mar.-Jul. 2017) **Research and software engineering internship, INRIA**, Grenoble, France.
- (Summer 2016) **Internship at Surrey Satellite Technology Ltd**, Guildford, England.
- (Sep. 2014-Apr. 2015) **Military Service**.

AWARDS AND HONORS

- (2020) NeurIPS oral presentation ($\approx 1\%$ acceptance rate).
- (2020) NeurIPS spotlight presentation ($\approx 4\%$ acceptance rate).
- (2018) EPFL IC School 1-year Fellowship.
- (2015) Citation for outstanding service in my unit during military service.

PUBLICATIONS

- Etienne Bamas and Lars Rohwedder, *Better Trees for Santa Claus,* to appear in STOC ‘23.
- Etienne Bamas, Marina Drygala, and Ola Svensson, *A Simple LP-Based Approximation Algorithm for the Matching Augmentation Problem*, IPCO ‘22.
- Etienne Bamas, Marina Drygala, and Andreas Maggiori, *An Improved Analysis of Greedy for Online Steiner Forest*, SODA ‘22.
- Etienne Bamas, Paritosh Garg, and Lars Rohwedder, *The Submodular Santa Claus Problem in the Restricted Assignment Case*, ICALP ‘21.
- Etienne Bamas, Andreas Maggiori, and Ola Svensson, *The Primal-Dual method for Learning Augmented Algorithms*, NeurIPS ‘20.
- Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson, *Learning Augmented Energy Minimization via Speed Scaling*, NeurIPS ‘20.
- Etienne Bamas and Louis Esperet, *Local Approximation of the Maximum Cut in Regular Graphs*, WG ‘19.
- Etienne Bamas and Louis Esperet, *Distributed Coloring of Graphs with an Optimal Number of Colors*, STACS ‘19.

TEACHING

- (since 2021) Supervision of semester projects at EPFL. Supervised students: Alexandre Reynaud (master student), Taha El Ghazi (master student).
- (2019-2023) Teaching Assistant at EPFL ("Algorithms" (head TA), "Theory of Computation" (head TA), "Information, Calcul, Communication").
- (2017-2018) Teaching Assistant at Lycée Janson-de-Sailly for oral exams in mathematics.

ACADEMIC SERVICE AND TALKS

- I co-organized the workshop ALPS 2022 on algorithms with predictions (~ 45 international participants).
- Reviewer for the conferences: MFCS ‘19, WAOA ‘20, ITCS ‘20, NeurIPS ‘21, SODA ‘22, ICML ‘22, NeurIPS ‘22, SODA ‘23, STOC ‘23, ICALP ‘23.
- Reviewer for the journals: Algorithmica (2021).
- I gave a talk at the following conferences: WG ‘19, STACS ‘19, NeurIPS ‘20, ICALP ‘21, Operations Research Bern ‘21, SODA ‘22.