EPFL

Thèse n° 9258

Streaming and Matching Problems with Submodular Functions

Présentée le 14 octobre 2022

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

Paritosh GARG

Acceptée sur proposition du jury

École
polytechnique
fédérale
de Lausanne

2022

Abstract

Submodular functions are a widely studied topic in theoretical computer science. They have found several applications both theoretical and practical in the fields of economics, combinatorial optimization and machine learning. More recently, there have also been numerous works that study combinatorial problems with submodular objective functions. This is motivated by their natural diminishing returns property which is useful in real-world applications. The thesis at hand is concerned with the study of streaming and matching problems with submodular functions.

Firstly, motivated by developing robust algorithms, we propose a new *adversarial injections* model, in which the input is ordered randomly, but an adversary may inject misleading elements at arbitrary positions. We study the maximum matching problem and cardinality constrained monotone submodular maximization. We show that even under this seemingly powerful adversary, it is possible to break the barrier of $1/2$ for both these problems in the streaming setting. Our main result is a novel streaming algorithm that computes a 0.55-approximation for cardinality constrained monotone submodular maximization.

In the second part of the thesis, we study the problem of matroid intersection in the semi-streaming setting. Our main result is a $(2 + \varepsilon)$-approximate semi-streaming algorithm for weighted matroid intersection improving upon the previous best guarantee of $4 + \varepsilon$. While our algorithm is based on the local ratio technique, its analysis differs from the related problem of weighted maximum matching and uses the concept of matroid kernels. We are also able to generalize our results to work for submodular functions by adapting ideas from a recent result by Levin and Wajc (SODA'21) on submodular maximization subject to matching constraints.

Finally, we study the submodular Santa Claus problem in the restricted assignment case. The submodular Santa Claus problem was introduced in a seminal work by Goemans, Harvey, Iwata, and Mirrokni (SODA'09) as an application of their structural result. In the mentioned problem n unsplittable resources have to be assigned to m players, each with a monotone submodular utility function f_i. The goal is to maximize $\min_i f_i(S_i)$ where $S_1, \ldots, S_m$ is a partition of the resources. The result by Goemans et al. implies a polynomial time $O(n^{1/2+\varepsilon})$-approximation algorithm. In the restricted assignment case, each player is given a set of desired resources Γ_i and the individual valuation functions are defined as $f_i(S) = f(S \cap \Gamma_i)$. Our main result is a $O(\log\log(n))$-approximation algorithm for the problem. Our proof is inspired by the approach of Bansal and Srividenko (STOC'06) to the Santa Claus problem. Compared to the more basic linear setting, the introduction of submodularity requires a much more involved analysis and several new ideas.

Keywords— submodular functions, streaming algorithms, matchings, matroids, robust algorithms, approximation algorithms, santa claus problem, hypergraph matchings, combinatorial optimization

Résumé

Les fonctions de type sous-modulaires forment un pan important de l'informatique théorique. Les applications pratiques et théoriques se retrouvent en effet dans plusieurs domaines tels que l'économie, l'optimisation combinatoire et l'apprentissage automatique. Plusieurs travaux récents se sont concentrés sur des problèmes combinatoires où l'objectif est une fonction sous-modulaire. Ceci est motivé par leur propriété intrinsèque de "retour diminuant" qui prend tout son sens dans la vie de tous les jours. Cette thèse se penche sur le streaming ainsi que le problème d'appariement dans le cadre de fonctions sous-modulaires.

Avec comme objectif le développement d'algorithmes robustes, nous commençons par proposer un nouveau modèle d'injection adversariale dans lequel l'ordre du flux de données est aléatoire mais ou un adversaire peut injecter de nouveaux éléments dans le flux à sa guise. Nous étudions le problème d'appariement maximal et de maximisation de fonction sous-modulaire sous contrainte de taille. Nous montrons que même face à cet adversaire qui semble très puissant, il est possible de passer outre à la barrière de facteur d'approximation de 0.5 bien connu pour ces deux problèmes. Notre résultat principal consiste en un nouvel algorithme capable de trouver une solution avec un facteur d'approximation de 0.55 au problème de maximisation de fonction sous-modulaire sous contrainte de taille.

La seconde partie de cette thèse est dévolue à l'étude du problème de l'intersection de deux matroids dans le modèle de semi-streaming. Notre résultat principal est un algorithme garantissant un facteur d'approximation de $2+\varepsilon$ au problème de l'intersection de matroid avec poids dans le modèle de semi-streaming; ceci améliore le précédent algorithme possédant un facteur d'approximation de $4+\varepsilon$. Bien que notre algorithme soit basé sur la technique du ratio local, son analyse diffère du problème de l'appariement maximal avec poids et utilise le concept de matroid à noyau. Nous arrivons à généraliser nos résultats au modèle de fonction sous-modulaire en adaptant les idées de Levin et Wajc (SODA'21) sur la maximisation sous-modulaire avec contrainte d'appariement.

En dernier lieu, nous étudions le problème du Père Noël sous-modulaire dans le cas de l'affectation restreinte. Le problème du Père Noël sous-modulaire fut introduit dans le travail pionnier de Goemans, Harvey, Iwata et Mirrokni (SODA 09) pour exemplifier leur résultat structurel. Ce problème consiste en n resources devant être partagées sans possibilité de decoupe entre m joueurs. Chaque joueur possède une fonction utilitaire f_i de type sous-modulaire; le but étant de maximiser $min_i f_i(s_i)$ ou $S_i, \ldots, S_m$ est une partition des resources. Le résultat obtenu par Goemans et al consiste en un algorithme d'approximation qui obtient en temps polynomial une approximation de facteur $O(n^{1/2+\varepsilon})$. Dans le cas de l'affectation restreinte, chaque joueur possède un ensemble de ressources Γ_i et les valuations individuelles sont définies par $f_i(S) = f(S \cap \Gamma_i)$. Notre résultat principal est un algorithme d'approximation en temps polynomial avec facteur d'approximation $O(\log\log(n))$. Notre preuve s'inspire de l'approche de Bansal et Srividenko (STOC'06) au problème du Père Noël. Par rapport au plus simple cadre linéaire, l'introduction de sous-modularité requiert de nouvelles idées et une analyse plus conséquente.

***Mots-clés*—-** fonctions sous-modulaires, algorithmes de streaming, appariements, matroids, algorithmes robustes, algorithmes d'approximation, problème du Père Noël, appariements sur hypergraphes, optimisation combinatoire

Acknowledgements

My PhD has been a great and memorable experience, and I am grateful to the countless people who have been instrumental in making it such.

I want to thank my advisors, Michael Kapralov and Ola Svensson. Although the time I worked with Michael wasn't very much, I learnt a lot from him. I was constantly impressed by his ability to work on multiple problems. I think of Ola as a child in a playground with many exciting problems to solve. His never-ending energy and enthusiasm have inspired me in many ways. I appreciate his constant support and encouragement throughout my PhD, especially during the pandemic.

I want to thank the jury members of my private defense, Deeparnab Chakrabarty, Friedrich Eisenbrand, Emre Telatar and Rico Zenklusen, for carefully reading my thesis and for the insightful discussions during the defense.

I am grateful to Pauline Raffestin and Chantal Schneeberger for helping me through all the administrative stuff, organizing many fun events in the lab and always being warm and welcoming.

My PhD wouldn't have been successful without my amazing co-authors, Etienne Bamas, Linus Jordan, Sagar Kale, Lars Rohwedder and Ola Svensson. I have learnt a lot from them. I thank my pre-PhD mentors, Kishore Kothapalli, Kannan Srinathan and Srikanth Srinivasan. They played a crucial role in my decision to pursue a PhD.

I have immensely enjoyed being a part of the Theory lab, and I would like to thank everyone. With them, I have done several fun activities like climbing, hiking and biking and also discussed everything from work to life. I would also like to thank the other PhD students in EDIC for sharing many fun times, and all my friends in Lausanne and many roommates throughout the years for making me feel at home. I want to especially thank Gilbert for translating the abstract of this thesis into French on short notice. I would also like to thank Anna for her constant support and for being such a fantastic person to hang out with. I also thank her for proofreading my thesis.

Finally, and most importantly, I would like to thank my family for their unconditional love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Submodular functions are a fundamental object of study in the field of theoretical computer science. They have several applications in the field of combinatorial optimization, algorithmic game theory and combinatorics. Besides being of theoretical interest, they often find use in several real-world applications owing to their natural diminishing returns property. For example, they are very commonly used as utility functions in economics. Naturally, there is a large body of literature that has focussed on the optimization of these functions in different models of computation. More recently, they have also received much attention in the study of combinatorial problems replacing linear objective functions with submodular objective functions.

Set functions are functions that give a value to every possible subset of a ground set of elements. Informally, a submodular function is a set function where the marginal gain by adding an element to a set is non-increasing as the size of the set increases. This is technically not true, but it suffices to provide intuition here. See Section 2.2 for a formal definition. It also captures several well-known functions within combinatorics. For example, cut functions in graphs, rank functions in matroids and covering and cut functions in hypergraphs. Moreover, several objective functions arising in tasks in fields like machine learning can be modelled as being submodular.

In this thesis, we study *matching problems* within combinatorial optimization. Informally speaking, these sorts of problems involve pairing agents with agents or pairing agents with items, where agents might be job seekers and employers, buyers and sellers etc. Matching theory has played an important role in theoretical computer science and has given rise to several important concepts including the polynomial-time computability [19] and important algorithmic techniques such as the primal dual method [46]. It has also found numerous practical applications most notably in economics and market design. To enumerate a few examples, it has been used for associating goods with buyers, scheduling tasks with machines, pairing organ donors with recipients, employers with job applicants etc. Given both the theoretical and practical importance of submodular functions and matching problems, the thesis at hand focuses on developing better algorithms for matching problems with submodular functions.

We first focus on the streaming setting where the algorithms have access to limited memory. This model is motivated by the advent of big data and the need to develop algorithms that are memory efficient. We study the benchmark problems of cardinaltity constrained submodular maximization, unweighted maximum matching and submodular matroid intersection problem. We note that matroid intersection generalizes the problem of maximum matching in bipartite graphs. Later, we study the submodular variant of the Santa Claus problem in the classical model of computation i.e, Random Access Machine (RAM) model. The Santa Claus problem is a basic problem in resource allocation and the special case of restricted assignment has received a lot of attention recently. Interestingly, we reduce the submodular Santa Claus problem to a certain matching problem in hypergraphs. Hence, broadly speaking, our thesis can be considered to deal with the subject of submodular functions and matchings.

1.1 Overveiw of our Contributions

We give a brief overview of our three contributions below.

Adversarial Injections In Part I of this thesis, we develop robust algorithms for cardinality constrained monotone submodular maximization and the unweighted maximum matching problem in the streaming setting. We introduce a new model - adversarial injections - with the motivation of developing robust algorithms. This is based on joint work with Sagar Kale, Lars Rohwedder and Ola Svensson that was published in ICALP 2020.

In the streaming model, an algorithm reads the input sequentially from the input stream while using limited memory. In particular, the algorithm is expected to use memory that is much smaller than the input size, ideally, linear in the size of the solution. A common assumption made in this model to model real-world instances is to assume that the input sequence is chosen uniformly at random. However, many algorithms for problems including cardinality constrained monotone submodular maximization overfit to this assumption. We propose a new model that we call adversarial injections model, in which the input is ordered randomly, but an adversary may inject misleading elements at arbitrary positions. We believe that studying algorithms under this much weaker assumption can lead to new insights and, in particular, more robust algorithms. We investigate two classical combinatorial-optimization problems in this model: Maximum matching and cardinality constrained monotone submodular function maximization. Our main technical contribution is a novel streaming algorithm for the latter that computes a 0.55-approximation. While the algorithm itself is clean and simple, an involved analysis shows that it emulates a subdivision of the input stream which can be used to greatly limit the power of the adversary. For the problem of unweighted maximum matching, we can adapt the existing techniques in literature to beat $1/2$ in our model in the streaming setting. Curiously, we observe due to a recent result by [31] that beating $1/2$ is not possible in our model for the online setting. This makes our model further interesting to study.

Submodular Semi-Streaming Matroid Intersection In Part II of this thesis, we develop an improved approximation algorithm for matroid intersection in the streaming setting and then extend it to work with submodular functions. This is based on joint work with Linus Jordan and Ola Svensson that was published in IPCO 2021.

A matching in a graph is a subgraph in which each vertex has degree at most one. While the basic greedy algorithm gives a semi-streaming algorithm with an approximation guarantee of 2 for the *unweighted* matching problem, it was only recently that Paz and Schwartzman ([60]) obtained an analogous result for weighted instances. Their approach is based on the versatile local ratio technique and also applies to generalizations such as weighted hypergraph matchings. However, the framework for the analysis fails for the related problem of weighted matroid intersection and as a result the approximation guarantee for weighted instances did not match the factor 2 achieved by the greedy algorithm for unweighted instances. Our main result closes this gap by developing a semi-streaming algorithm with an approximation guarantee of $2 + \varepsilon$ for *weighted* matroid intersection, improving upon the previous best guarantee of $4 + \varepsilon$. Our techniques also allow us to generalize recent results by Levin and Wajc [49] on submodular maximization subject to matching constraints to that of matroid-intersection constraints.

While our algorithm is an adaptation of the local ratio technique used in previous works, the analysis deviates significantly and relies on structural properties of matroid intersection, called kernels. Finally, we also conjecture that our algorithm gives a $(k + \varepsilon)$ approximation for the intersection of k matroids but prove that new tools are needed in the analysis as the structural properties we use fail for $k \geqslant 3$.

Submodular Santa Claus problem in the Restricted Assignment Case In Part III of this thesis, we develop an improved approximation algorithm for the submodular Santa Claus problem in the restricted assigment case. This is based on a joint work with Etienne Bamas and Lars Rohwedder that was published in ICALP 2021.

The submodular Santa Claus problem was introduced in a seminal work by Goemans, Harvey, Iwata, and Mirrokni ([33]) as an application of their structural result. In the mentioned problem n unsplittable resources have to be assigned to m players, each with a monotone submodular utility function f_i. The goal is to maximize $\min_i f_i(S_i)$ where $S_1, \dots, S_m$ is a partition of the resources. The result by Goemans et al. implies a polynomial time $O(n^{1/2+\varepsilon})$-approximation algorithm.

Since then progress on this problem was limited to the linear case, that is, all f_i are linear functions. In particular, a line of research has shown that there is a polynomial time constant approximation algorithm for linear valuation functions in the restricted assignment case. This is the special case where each player 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 this thesis we make comparable progress for the submodular variant. Namely, if f is a monotone submodular function, we can in polynomial time compute an $O(\log\log(n))$-approximate solution. Moreover, we also show a certain kind of "equivalence" between a matching problem in hypergraphs and the Santa Claus problem. We show, via a reduction, that a c-approximation for this matching problem would yield a $O((c \log^*(n))^2)$-approximation for the Santa Claus problem with arbitrary linear utility functions. Infact, we reduce the submodular Santa Claus problem in the restricted assignment case to this hypergraph matching problem albeit with additional assumptions on the the size of hyperedges [1].

1.2 Outline of the Thesis

The thesis is outlined as follows. In Chapter 2, we introduce key concepts and formally define the problems that we consider in this thesis.

Part I is devoted to our results on the adversarial injections model. We motivate and introduce our model - adversarial injections - for developing robust algorithms and compare it with related models in Chapter 3. We then show how to beat the factor of $1/2$ for unweighted maximum matching in the streaming setting in our model in Chapter 4. Further, we discuss cardinality constrained monotone submodular maximization in Chapter 5 and present a 0.55-approximate streaming algorithm.

In Part II, we present the results on the problem of submodular semi-streaming matroid intersection. We start with a brief overview of related

[1] Due to the additional assumption on the size of hyperedges, this does not imply an improved approximation algorithm for the Santa Claus problem.

work as well as overview of our results and techniques in Chapter 6. In Chapter 7, we present a $(2 + \varepsilon)$-approximate semi-streaming algorithm for weighted matroid intersection using the local ratio techique. Further in Chapter 8, we extend this algorithm to work for submodular functions. We then discuss the case of more than two matroids in Chapter 9.

Moving on in part III, we consider the submodular Santa Claus problem in the restricted assignment setting. We again start with a brief discussion on related work and discuss our techniques in Chapter 10. Then, in Chapter 11, we reduce the aforementioned problem to a matching problem in hypergraphs. We then show how to obtain an approximation algorithm for this problem in Chapter 12. Further, in Chapter 13 we discuss connections between hypergraph matching and the Santa Claus problem and present reductions in both directions.

Finally, in Chapter 14, we conclude the thesis with a short discussion and some interesting open problems arising out of our work.

Chapter 2

Preliminaries

This chapter covers the preliminaries for the rest of the thesis. First we introduce some notation commonly used, then define some mathematical objects and models of computation studied. Further, we give the definitions of the problems considered in this thesis. Finally, we end with some probability bounds that we require in Part III of our thesis.

2.1 Notation

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}_{\geqslant 0}$ denote the set of non-negative real numbers. We use $\mathbb{N}$ to denote the set of natural numbers.

2.2 Submodular functions

A set function $f : 2^E \to \mathbb{R}$ is submodular if it satisfies that for any two sets $A, B \subseteq E$, $f(A) + f(B) \geqslant f(A \cup B) + f(A \cap B)$. For any two sets $A, B \subseteq E$, let $f(A \mid B) := f(A \cup B) - f(B)$. For $e \in E$ and $S \subseteq E$ we write $S + e$ for the set $S \cup \{e\}$ and $f(e \mid S)$ for $f(S + e) - f(S)$.

An equivalent and more intuitive definition for f to be submodular is that for any two sets $A \subseteq B \subseteq E$, and $e \in E \setminus B$, it holds that $f(e \mid A) \geqslant f(e \mid B)$. The function f is called monotone if for any element $e \in E$ and set $A \subseteq E$, it holds that $f(e \mid A) \geqslant 0$. We say that f is normalized if $f(\emptyset) = 0$.

2.3 Matroids

We define and give a brief overview of the basic concepts related to matroids that we use in Part II of the thesis. For a more comprehensive treatment, we refer the reader to [62]. A *matroid* is a tuple $M = (E, I)$ consisting of a finite ground set E and a family $I \subseteq 2^E$ of subsets of E satisfying:

- if $X \subseteq Y, Y \in I$, then $X \in I$; and
- if $X \in I, Y \in I$ and $|Y| > |X|$, then $\exists\ e \in Y \setminus X$ such that $X \cup \{e\} \in I$.

The elements in I (that are subsets of E) are referred to as the *independent sets* of the matroid and the set E is referred to as the *ground set*. With a matroid $M = (E, I)$, we associate the *rank function* $\text{rank}_M : 2^E \to \mathbb{N}$ and the *span function* $\text{span}_M : 2^E \to 2^E$ defined as follows for every $E' \subseteq E$,

$$\begin{aligned} \text{rank}_M(E') &= \max\{|X| \mid X \subseteq E' \text{ and } X \in I\}, \\ \text{span}_M(E') &= \{e \in E \mid \text{rank}_M(E' \cup \{e\}) = \text{rank}_M(E')\}. \end{aligned}$$

We simply write $\text{rank}(\cdot)$ and $\text{span}(\cdot)$ when the matroid M is clear from the context. In words, the rank function equals the size of the largest independent set when restricted to E' and the span function equals the elements in E' and all elements that cannot be added to a maximum cardinality independent set of E' while maintaining independence. The *rank of the matroid* equals $\text{rank}(E)$, i.e., the size of the largest independent set. In order to gain more intuition, let us consider the example of linear matroids.

Example 2.1 *A matroid $M = (E, I)$ is a linear matroid when it is defined from a matrix A over some field F. Let E be the index set of the columns and for $X \subseteq E$, let A_X be the matrix consisting of the columns indexed by X. Define I by*

$$I = \{X \subseteq E : rank(A_X) = |X|\}$$

For linear matroids, observe that the first matroid axiom is trivially satisfied, as if columns are linearly independent, so is a subset of them. For the second axiom, notice that if A_X has full column rank, its columns span a space of dimension $|X|$ and similarly for Y , and therefore if $|Y| > |X|$, there must exist a column of A_Y that is not in the span of the columns of A_X; adding this column to A_X increases the rank by 1. Furthermore for linear matroids, the definitions rank_M and span_M correspond naturally to those in linear algebra.

2.4 Streaming Model of Computation

In the *streaming* model, the input is revealed in a stream $e_1, e_2, \ldots, e_m$ and at time i the algorithm gets access to e_i and can perform computation based on e_i and its current memory but without knowledge of future elements $e_{i+1}, \ldots, e_m$. In particular, the algorithm is expected to use memory that is much smaller than the input stream size m, ideally, linear in the size of a solution S. Specifically, in the *semi-streaming* setting, the algorithm has a memory that is $O(|S| \operatorname{polylog}(|S|))$. This makes sense in the graph setting, where the input size can be as large as $O(n^2)$ and n denotes the number of vertices. This memory usage is justified, because even storing a solution for example in the case of maximum matching can take $\Omega(n \log(n))$ space ($\Omega(\log(n))$ for each edge identity). This model can also be considered in the *multi-pass* setting when the algorithm is allowed to take several passes over the stream. However, in this thesis we focus on the most basic and widely studied setting in which the algorithm takes a single pass over the stream.

We say that the stream is random order if the input is permuted uniformly at random before being presented to the algorithm. The stream is referred to as worst case or adversarial if the input stream is prepared by an adversary before being presented to the algorithm. Unless explicitly specified, we assume that the stream is worst case.

The difficulty in designing a good streaming algorithm is that the memory requirement is much smaller than the size of the input stream and thus the algorithm must intuitively discard many of the elements without knowledge of the future and without significantly deteriorating the quality of the final solution. The quality of the algorithm is judged by its approximation ratio. The definition that we provide differs between Part I and Part II due to the existing literature. In Part I, we say that an algorithm is α-approximate if no matter what the input, it produces a solution that is at least α times the value of the optimum solution. In the worst-case model, this guarantee should also hold for every possible stream of the input elements whereas in the random-order model this guarantee needs to hold only in expectation over the randomness of the input stream. In Part II, we say that an algorithm is α-approximate if no matter what the input and the order of the stream, it produces a solution that is at least $\frac{1}{\alpha}$ times the value of the optimal solution. Notice that in both cases, we want α to be as close to one as possible.

2.5 Online Model of Computation

In the *online* model, an algorithm sees the input elements one at a time and it has to decide whether to take the element in its solution or not. This decision is irrevocable. This model of computation is used to model problems in which the future is uncertain and one has to make decisions that are irreversible. The quality of an algorithm is judged by its competitive ratio. We define the competitive ratio for maximization problems. An algorithm is said to be α-competitive if it outputs a solution at the end of the stream that is at least α times the value of the solution that an optimal offline (knows the future elements) algorithm would output. Notice that this means α is always less than or equal to one. We deal with online algorithms in Part I of our thesis.

2.6 Approximation Algorithms

Approximation algorithms are polynomial time (efficient) algorithms that output a solution that is close in value to an optimum solution. They are important for several problems as developing an exact algorithm might be infeasible due to unconditional or conditional hardness results. We present an approximation algorithm for an NP-hard problem in resource allocation in the Part III of our thesis.

The quality of the solution that the algorithm produces is judged by its approximation ratio. We consider maximization problems in this thesis and define the approximation ratio accordingly. We say that an algorithm is α-approximate if it outputs a solution that is at least $\frac{1}{\alpha}$ times the value of the optimum solution. Notice that this means that α is always greater than or equal to one. Hence, the goal is to make α as close to one as possible.

2.7 Problems Considered

We define and briefly discuss the problems considered in the thesis below.

2.7.1 The maximum matching problem in the semi-streaming model

We first discuss the (unweighted) *maximum matching* problem. Given a graph $G = (V, E)$, a matching M is a subset of edges such that every vertex has at most one incident edge in M. A matching of maximum cardinality is called a maximum matching, whereas a *maximal* matching is one in which no edge can be added without breaking the property of it being a matching. The goal in the maximum matching problem is to compute a matching of maximum cardinality. Note that a maximal matching is 1/2-approximate. Work on maximum matching has led to several important concepts and new techniques in theoretical computer science [56, 51, 19, 41]. Specifically, in the semi-streaming setting we allow a streaming algorithm to have memory $O(n\,\text{polylog}(n))$ where n is the number of vertices of G. This is usually significantly less than the input size, which can be as large as $O(n^2)$. This memory usage is justified, because even storing a solution can take $\Omega(n \log(n))$ space ($\Omega(\log(n))$ for each edge identity).

2.7.2 Maximizing a monotone submodular function subject to a cardinality constraint

In this problem, we are given a ground set E of n elements and a monotone submodular set function $f : 2^E \to \mathbb{R}_{\geqslant 0}$. The problem we consider is to find a set $S \subseteq E$ with $|S| \leqslant k$ that maximizes $f(S)$. We assume that access to f is via an oracle.

In the offline setting, a simple greedy algorithm that iteratively picks the element with the largest marginal contribution to f with respect to the current solution is $(1-1/e)$-approximate [58]. This is tight: Any algorithm that achieves an approximation ratio of better than $(1-1/e)$ must make $\Omega(n^k)$ oracle calls [57], which is enough to brute-force over all k-size subsets. Even for maximum coverage (which is a special family of monotone submodular functions), it is NP-hard to get an approximation algorithm with ratio better than $1-1/e$ [24].

2.7.3 The matroid Intersection problem in the semi-streaming model

In the *weighted matroid intersection problem*, we are given an oracle access to two matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ on a common ground set E and a non-negative weight function $w : E \to \mathbb{R}_{\geqslant 0}$ on the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $w(X) = \sum_{e \in X} w(e)$ is maximized.

In the *submodular matroid intersection problem*, we are given an oracle access to two matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ on a common ground set E and an oracle access to non-negative submodular function $f : 2^E \to \mathbb{R}_{\geqslant 0}$ on the powerset of the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $f(X)$ is maximized.

In his seminal work [20], Edmonds gave a polynomial-time algorithm for solving the weighted matroid intersection problem to optimality in the classic model of computation when the whole input is available to the algorithm throughout the computation. In contrast, the problem becomes significantly harder and tight results are still eluding us in the semi-streaming model where the memory footprint of the algorithm and its access pattern to the input are restricted. Specifically, in the *semi-streaming* model, the algorithm has an independence-oracle access to the matroids M_1 and M_2 which is restricted to the elements stored in the memory, i.e., for a set of such elements, the algorithm can query whether the set is independent in each matroid. Moreover, the memory usage should be near-linear $O((r_1 + r_2)\,\text{polylog}(r_1 + r_2))$ at any time, where r_1 and r_2 denote the ranks of the input matroids M_1 and M_2, respectively. We remark that the memory requirement $O((r_1 + r_2)\,\text{polylog}(r_1 + r_2))$ is natural as $r_1 + r_2 = |V|$ when formulating a bipartite matching problem as the intersection of two matroids[1].

[1]The considered problem can also be formulated as the problem of finding an independent set in one of the matroids, say M_1, and maximizing a submodular function which would be the (weighted) rank function of M_2. For that problem, [37] recently gave a streaming algorithm with an approximation guarantee of $(2+\varepsilon)$. However, the space requirement of their algorithm is exponential in the rank of M_1 (which would correspond to exponential in

2.7.4 Submodular Santa Claus problem in the Restricted Assignment Case

The submodular Santa Claus problem was introduced in [33] as an application of their structural result. The result by [33] implies a polynomial time $O(n^{1/2+\varepsilon})$-approximation algorithm.

In the submodular Santa Claus problem, n unsplittable resources have to be assigned to m players, each with a monotone submodular utility function f_i. The goal is to maximize $\min_i f_i(S_i)$ where $S_1, \dots, S_m$ is a partition of the resources. Further in the restricted assigment case, each player 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 submodular function f. This can also be interpreted as maximizing $\min_i f(S_i)$ with additional assignment restrictions, i.e., certain resources can only be assigned to certain players.

2.8 Probability Bounds

In this section, we state a few probability bounds that are essential to our analysis in Part III of our thesis. Consider the sum of independent random variables drawn from some distribution. The central limit theorem asserts that such a sum converges to the normal distribution with the right scaling. However, it does not tell us how fast this convergence happens. Chernoff bounds are useful as they provide us with a much more quantitative estimate of how this happens. We state two versions that we will repeatedly use below.

Proposition 2.2 (Chernoff bounds (see e.g. [54])) *Let $X = \sum_i X_i$ be a sum of independent random variables such that each X_i can take values in a range $[0,1]$. Define $\mu = \mathbb{E}(X)$. We then have the following bounds*

$$\mathbb{P}\left(X \geqslant (1+\delta)\mathbb{E}(X)\right) \leqslant \exp\left(-\frac{\min\{\delta, \delta^2\}\mu}{3}\right)$$

for any $\delta > 0$.

$$\mathbb{P}\left(X \leqslant (1-\delta)\mathbb{E}(X)\right) \leqslant \exp\left(-\frac{\delta^2 \mu}{2}\right)$$

for any $0 < \delta < 1$.

The following proposition follows immediately from Proposition 2.2 by applying it with $X' = X/a$.

Proposition 2.3 *Let $X = \sum_i X_i$ be a sum of independent random variables such that each X_i can take values in a range $[0,a]$ for some $a > 0$. Define $\mu = \mathbb{E}(X)$. We then have the following bounds*

$|V|$ in the matching case) and thus it does not provide a meaningful algorithm for our setting.

$$\mathbb{P}\left(X \geqslant (1+\delta)\mathbb{E}(X)\right) \leqslant \exp\left(-\frac{\min\{\delta, \delta^2\}\mu}{3a}\right)$$

for any $\delta > 0$.

$$\mathbb{P}\left(X \leqslant (1-\delta)\mathbb{E}(X)\right) \leqslant \exp\left(-\frac{\delta^2\mu}{2a}\right)$$

for any $0 < \delta < 1$.

In Part III of our thesis, we would like to bound the probability of some bad events happening in many cases. If the bad events were completely independent, then we would be done as there always exists a non zero chance of none of the bad events taking place. But, this is unfortunately not the case for us. However, the Lovász Local Lemma stated below comes in handy as it implies a similar statement for bad events that are not too interdependent.

Proposition 2.4 (Lovász Local Lemma (LLL)) *Let* $B_1, \ldots, B_t$ *be bad events, and let* $G = (\{B_1, \ldots, B_t\}, E)$ *be a dependency graph for them, in which for every* i*, event* B_i *is independent of all events* B_j *for which* $(B_i, B_j) \notin E$*. Let* x_i *for* $1 \leqslant i \leqslant t$ *be such that* $0 < x(B_i) < 1$ *and* $\mathbb{P}[B_i] \leqslant x(B_i) \prod_{(B_i,B_j)\in E}(1 - x(B_j))$*. Then with positive probability no event* B_i *holds.*

Part I

Adversarial Injections

Chapter 3

Introduction

In this part of the thesis, we develop robust streaming algorithms for the problems of unweighted maximum matching and monotone submodular maximization under a cardinality constraint. We first start of by motivating our model in which we develop these robust algorithms.

3.1 Motivation

The most common approach to analyze the quality of an algorithm in the streaming and online models of computation is worst-case analysis. Here, an adversary has full knowledge of the algorithm's strategy and presents a carefully crafted instance to it, trying to make the ratio between the value of the algorithm's solution and that of an optimum solution (the approximation ratio; for online algorithms called the competitive ratio) as small as possible[1]. While worst-case analysis gives very robust guarantees, it is also well-known that such an analysis is often very pessimistic. Not only are good guarantees not possible for many problems, but in many cases worst-case instances appear quite artificial. Hence, the worst-case approximation/competitive ratio does not necessarily represent the quantity that we want to optimize.

One way to remedy this is to weaken the power of the adversary. A popular model to do so is the random-order model. Here, an adversary may pick the same instance as before, but it is presented in a uniformly-random order to the algorithm. This often allows for significantly better provable guarantees. A prime example is the secretary problem: For the worst-case order it is impossible to get a bounded competitive ratio whereas for the random-order a very simple stopping rule achieves a competitive ratio of $1/e$. Unfortunately, in this model, algorithms tend to overfit and the assumption of a uniformly-random permutation of the input is a strong one. To illustrate this point, it is

[1]We assume that the problem is a maximization problem.

instructive to consider two examples of techniques that break apart when the random-order assumption is slightly weakened:

Several algorithms in the random-order model first read a small fraction of the input, say, the first 1% of the input. Such an algorithm relies on the assumption that around 1% of the elements from an optimum solution are contained in this first chunk. It computes some statistics, summaries, or initial solutions using this chunk in order to estimate certain properties of the optimum solution. Then in the remaining 99% of the input it uses this knowledge to build a good solution for the problem. For examples of such streaming algorithms, see Norouzi-Fard et al. [59] who study submodular maximization and Gamlath et al. [30] who study maximum matching. Also Guruganesh and Singla's [35] online algorithm for maximum matching for bipartite graphs is of this kind. Due to their design, these algorithms are very sensitive to noise at the beginning of the stream.

Another common technique is to split the input into fixed parts and exploit the fact that with high probability the elements of the optimum solution are distributed evenly among these parts, e.g., each part has at most one optimum element. These methods critically rely on the assumption that each part is representative of the whole input or that the parts are in some way homogeneous (the properties of the parts are the same in expectation). Examples of such algorithms include the streaming algorithm for maximum matching [45], and the streaming algorithm for submodular maximization ([1], [50]) that achieves the tight competitive ratio $1 - 1/e$ in the random-order model.

The motivation of this work is to understand whether the strong assumption of uniformly-random order is necessary to allow for better algorithms. More specifically, we are motivated by the following question:

Can we achieve the same guarantees as in the uniform-random order but with algorithms that are more robust against some distortions in the input?

In the next section, we describe our proposed model which is defined so as to avoid overfitting to the random-order model, and, by working within this model, our algorithms for submodular maximization and maximum matching are more robust while maintaining good guarantees.

3.2 The Adversarial Injections Model

Our model—that we call the *adversarial-injections* model—lies in between the two extremes of random-order and adversarial-order. In this model, the input elements are divided into two sets E_{NOISE} and E_{GOOD}. An adversary first picks all elements, puts each element in either E_{NOISE} or E_{GOOD}, and chooses the input order. Then the elements belonging to E_{GOOD} are permuted uniformly at random among themselves. The algorithm does not know if an element is good or noise. We judge the quality of the solution produced by an algorithm by comparing it to the best solution in E_{GOOD}.

An equivalent description of the model is as follows. First, a set of elements is picked by the adversary and is permuted randomly. Then, the adversary injects more elements at positions of his choice without knowing the random permutation of the original stream[2]. Comparing this with the previous definition, the elements injected by the adversary correspond to E_{NOISE} and the elements of the original stream correspond to E_{GOOD}.

We denote by $E_{\text{OPT}} \subseteq E_{\text{GOOD}}$ the elements of a fixed optimum solution of the elements in E_{GOOD}. We can assume without loss of generality that $E_{\text{GOOD}} = E_{\text{OPT}}$, because otherwise elements in $E_{\text{GOOD}} \setminus E_{\text{OPT}}$ can be treated as those belonging to E_{NOISE} (which only strengthens the power of the adversary).

3.3 Related Models

With a similar motivation, Kesselheim, Kleinberg, Niazadeh [43] studied the robustness of algorithms for the secretary problem from a slightly different perspective: They considered the case when the order of the elements is not guaranteed to be uniformly-at-random but still contains "enough" randomness with respect to different notions such as entropy. Recently, Esfandiari, Korula, Mirrokni [22] introduced a model where the input is a combination of stochastic input that is picked from a distribution and adversarially ordered input. Our model is different in the sense that the input is a combination of randomly ordered elements (instead of stochastic input) and adversarially ordered elements.

Two models that are more similar to ours in the sense that the input is initially ordered in a uniformly-random order and then scrambled by an adversary in a limited way are [34] and [10]. First, in the streaming model, Guha and McGregor [34] introduced the notion of a t-bounded adversary that can disturb a uniformly-random stream but has memory to remember and delay at most t input elements at a time. Second, Bradac et al. [10] very recently introduced a new model that they used to obtain robust online

[2]We remark that the assumption that the adversary does not know the order of the elements is important. Otherwise, the model is equivalent to the adversarial order model for "symmetric" problems such as the matching problem. To see this, let E_{OPT} correspond to an optimum matching in any hard instance under the adversarial order. Since a matching is symmetric, the adversary can inject appropriately renamed edges depending on the order of the edges (which he knows if this assumption does not hold) and obtain exactly the hard instance.

algorithms for the secretary problem. Their model, called the *Byzantine* model, is very closely related to ours: the input is split into two sets which exactly correspond to E_{GOOD} and E_{NOISE} in the adversarial-injections model. The adversary gets to pick the elements in both of them, but an algorithm will be compared against only E_{GOOD}. Then—this is where our models differ—the adversary chooses an arrival time in $[0,1]$ for each element in E_{NOISE}. He has no control over the arrival times of the elements in E_{GOOD}, which are chosen independently and uniformly at random in $[0,1]$. The algorithm does not know to which set an element belongs, but it knows the timestamp of each element, as it arrives. While the Byzantine model prevents certain kinds of overfitting (e.g., of the classical algorithm for the secretary problem), it does not tackle the issues of the two algorithmic techniques we discussed earlier: Indeed, by time $t = 0.01$, we will see around 1% of the elements from E_{OPT}. Hence, we can still compute some estimates based on them, but do not lose a lot when dismissing them. Likewise, we may partition the timeline, and thereby the input, into parts such that in each part at most one element of E_{OPT} appears.

Hence, even if our model appears very similar to the Byzantine model, there is a subtle, yet crucial, difference. The adversarial-injections model does not add the additional dimension of time, and hence, does not allow for the kind of overfitting that we discussed earlier. To further emphasize this difference, we now describe why it is strictly harder to devise algorithms in the adversarial-injections model compared to the Byzantine model. We know that it is at least as hard as the Byzantine model, because any algorithm for the former also works for the latter. This holds because the adversarial-injections model can be thought of as the Byzantine model with additional power for the adversary and reduced power for the algorithm: The adversary gets the additional power of setting the timestamps of elements in E_{GOOD}, but not their identities, whereas the algorithm is not allowed to see the timestamp of any element.

To show that it is strictly harder, consider online bipartite matching. We show that one cannot beat $1/2$ in the adversarial-injections model (for further details, see Section 4.3) whereas we observe that the $(1/2 + \delta)$-approximation algorithm [35] for bipartite graphs and its analysis generalizes to the Byzantine model as well. This is the case because the algorithm in [35] runs a greedy algorithm on the first small fraction, say 1% of the input and "augments" this solution using the remaining 99% of the input. The analysis crucially uses the fact that 99% of the optimum elements are yet to arrive in the augmentation phase. This can be simulated in the Byzantine model by using timestamps in the online setting as one sees 1% of E_{OPT} in expectation.

3.4 Our Results

We consider two benchmark problems in combinatorial optimization under the adversarial-injections model in both the streaming and the online settings, namely maximum matching and monotone submodular maximization subject to a cardinality constraint. As we explain next, the study of these classic problems in our new model gives interesting insights: for many settings we can achieve more robust algorithms with similar guarantees as in the random-order model but, perhaps surprisingly, there are also natural settings where the adversarial-injection model turns out to be as hard as the adversarial order model.

The maximum matching problem. We first recall the (unweighted) *maximum matching* problem. Given a graph $G = (V, E)$, a matching M is a subset of edges such that every vertex has at most one incident edge in M. The goal in the maximum matching problem is to compute a matching of maximum cardinality. Work on maximum matching has led to several important concepts and new techniques in theoretical computer science [56, 51, 19, 41]. The combination of streaming and random-order model was first studied by Konrad, Magniez and Mathieu [45], where edges of the input graph arrive in the stream. The question that Konrad et al. answered affirmatively was whether the trivial $1/2$-approximation algorithm that computes a maximal matching can be improved in the random-order model. Since then, there has been some work on improving the constant [31, 23, 9, 4]. The state-of-the-art is an approximation ratio of $2/3 + \varepsilon_0$ for some absolute constant $\varepsilon_0 > 0$ proved by Assadi and Behnezad [4]. We show that beating the ratio of $1/2$ is possible also in the adversarial-injections model by building on the techniques developed for the random-order model.

Theorem 3.1 *There exists an absolute constant $\gamma > 0$ such that there is a semi-streaming algorithm for maximum matching under adversarial-injections with an approximation ratio of $1/2 + \gamma$ in expectation.*

We note that beating $1/2$ in adversarial-order streams is a major open problem. In this regard, our algorithm can be viewed as a natural first step towards understanding this question.

Now we move our attention to the online setting, where the maximum matching problem was first studied in the seminal work of Karp, Vazirani, and Vazirani [42]. They gave a tight $(1 - 1/e)$-competitive algorithm for the so-called one-sided vertex arrival model which is an important special case of the edge-arrival model considered here. Since then, the online matching problem has received significant attention (see e.g. [12, 21, 26, 38, 31]). Unlike the adversarial streaming setting, there is a recent hardness result published in [31] in the adversarial online setting that the trivial ratio of $1/2$ cannot be improved. We also know by [35] that one can beat $1/2$ for bipartite graphs in the random-order online setting. Hence, one might hope at least for bipartite graphs to use existing techniques to beat $1/2$ in the online adversarial-injections setting and get a result analogous to Theorem 3.1. But surprisingly, this is not the case. We observe that the construction used in proving Theorem 3 in [31] also implies that there does not exist an algorithm with a competitive ratio of $1/2 + \varepsilon$ for any $\varepsilon > 0$ in the adversarial-injections model.

Maximizing a monotone submodular function subject to a cardinality constraint. We recall that in this problem, we are given a ground set E of n elements and an oracle access to a monotone submodular set function $f : 2^E \to \mathbb{R}_{\geqslant 0}$. The problem we consider is to find a set $S \subseteq E$ with $|S| \leqslant k$ that maximizes $f(S)$.

In the offline setting, a simple greedy algorithm that iteratively picks the element with the largest marginal contribution to f with respect to the current solution is $(1 - 1/e)$-approximate [58]. This is tight: Any algorithm that achieves an approximation ratio of better than $(1 - 1/e)$ must make $\Omega(n^k)$ oracle calls [57], which is enough to brute-force over all k-size subsets. Even for maximum coverage (which is a special family of monotone submodular functions), it is NP-hard to get an approximation algorithm with a ratio better than $1 - 1/e$ [24].

In the random-order online setting, this problem is called the *submodular secretary* problem, and a exponential time $1/e$-approximation and polynomial-time $(1 - 1/e)/e$-approximation algorithms are the state-of-the-art [44]. In the adversarial online setting, it is impossible to get any bounded approximation ratio for even the very special case of picking a maximum weight element. In this case, $|E_{\text{OPT}}| = 1$ and the adversarial and adversarial-injections models coincide; hence the same hardness holds. In light of this negative result, we focus on adversarial-injections in the streaming setting. Note that to store a solution we only need the space for k element identities. We think of k to be much smaller than n. Hence, it is natural to ask, whether the number of elements in memory can be independent of n.

For streaming algorithms in the adversarial order setting, the problem was first studied by Chakrabarti and Kale [13] where they gave a $1/4$-approximation algorithm. This was subsequently improved to $1/2 - \varepsilon$ by Badanidiyuru et al. [5]. Later, Norouzi-Fard et al. [59] observed that in the random-order model this ratio can be improved to beyond $1/2$. Finally, Agrawal et al. [1] obtained a tight $(1 - 1/e)$-approximation guarantee in the random-order model. Very recently, Liu et al. [50] decreased the memory requirement of the algorithm in [1] from $O(k \exp(\text{poly}(1/\varepsilon))$ to $O(k/\varepsilon)$ by simplifying their algorithm and analysis.

The algorithm of Agrawal et al. [1] and Liu et al. [50] involves partitioning

Table 3.1: Comparison of different models for the two studied problems. Here, $\gamma > 0$, $\varepsilon_0 > 0$ are fixed absolute constants and $\varepsilon > 0$ is any constant.

Maximum matching

	Random order	Adversarial Injections	Adversarial order
Streaming	$\geqslant 2/3 + \varepsilon_0$ [4]	$\geqslant 1/2 + \gamma$	$\leqslant 1 - 1/e + \varepsilon$ [40]
Online	$\geqslant 1/2$ (folklore)	$\leqslant 1/2$	$\leqslant 1/2$ [31]

Submodular function maximization

	Random order	Adversarial Injections	Adversarial order
Streaming	$\geqslant 1 - 1/e - \varepsilon$ [1] $\leqslant 1 - 1/e + \varepsilon$ [53]	$\geqslant 0.55$	$\geqslant 1/2 - \varepsilon$ [5] $\leqslant 1/2$ [28]

the stream as a crucial step in order to isolate the elements of the optimum solution. As discussed earlier, this approach does not work under adversarial-injections. However, we note that the algorithm and analysis by Norouzi-Fard et al. [59] can be easily modified to work under adversarial-injections as well. Their algorithm, however, has an approximation ratio of $1/2 + 8 \cdot 10^{-14}$. In this work, we remedy this weak guarantee.

Theorem 3.2 *There exists a* 0.55*-approximation algorithm that stores a number of elements that is independent of n for maximizing a monotone submodular function with a cardinality constraint k under adversarial-injections in the streaming setting.*

We summarize and compare our results with random-order and adversarial-order models for the problems we study in Table 3.1. It is interesting to see that in terms of beating $1/2$, our model in the streaming setting agrees with the random-order model and in the online setting agrees with the adversarial-order model.

Chapter 4

Matching

In this chapter, we consider the problem of unweighted maximum matching under adversarial injections in both streaming and online settings where the edges of the input graph arrive one after another.

4.1 Streaming Setting

We show that the trivial approximation ratio of $1/2$ can be improved upon. We provide a robust version of existing techniques and prove a statement about robustness of the greedy algorithm to achieve this.

First, let us introduce some notation which we will use throughout this section. We denote the input graph by $G = (V, E)$, and let M^* be a maximum matching. For any matching M, the union $M \cup M^*$ is a collection of vertex-disjoint paths and cycles. When M is clear from the context, a path of length $i \geqslant 3$ in $M \cup M^*$ which starts and ends with an edge of M^* is called an i-augmenting path. Notice that an i-augmenting path alternates between edges of M^* and M and that we can increase the size of M by one by taking all edges from M^* and removing all edges from M along this path. We say that an edge in M is 3-augmentable if it belongs to some 3-augmenting path. Otherwise, we say it is non-3-augmentable. Also, let $M^* = E_{\text{OPT}}$; as described in Section 3.2, this can be done without loss of generality.

As a subroutine for our algorithm we need the following procedure.

Lemma 4.1 (Lemma 3.1 in [30]) *There exists a streaming algorithm* 3-Aug-Paths *with the following properties:*

1. *The algorithm is initialized with a matching M and a parameter $\beta > 0$. Then a set E of edges is given to the algorithm one edge at a time.*

2. *If $M \cup E$ contains at least $\beta|M|$ vertex disjoint 3-augmenting paths, the algorithm returns a set A of at least $(\beta^2/32)|M|$ vertex disjoint 3-augmenting paths. The algorithm uses space $O(|M|)$.*

The Algorithm

We now describe our algorithm Match. It runs two algorithms in parallel and selects the better of the two outputs. The first algorithm simply constructs a maximal matching greedily by updating the variable M_1. The second algorithm also constructs a matching $M_2^{(1)}$ greedily, but it stops once $M_2^{(1)}$ has $|M^*|(1/2 - \varepsilon)$ edges. We call this Phase 1. Then, it finds 3-augmentations using the 3-Aug-Paths algorithm given by Lemma 4.1. Finally, it augments the paths found to obtain a matching M_2. The constant β used in 3-Aug-Paths is optimized for the analysis and will be specified there.

Notice that here we assumed that the algorithm knows $|M^*|$. This assumption can be removed using geometric guessing at the loss of an arbitrary small factor in the approximation ratio. We refer the reader to the appendix for full details.

Overview of the Analysis

Consider the first portion of the stream until we have seen a small constant fraction of the elements in E_{OPT}. If the greedy matching up to this point is already close to a $1/2$-approximation, this is good for the second algorithm as we are able to augment the matching using the remaining edges of M^*. The other case is good for the first algorithm: We will show that the greedy matching formed so far must contain a significant fraction of the edges in M^* which we have seen upto this point. If this happens, the first algorithm outputs a matching of size a constant fraction more than $|M^*|/2$.

A technical challenge and novelty comes from the fact that the two events above are not independent of the random order of E_{OPT}. Hence, when conditioning on one event, we can no longer assume that the order of E_{OPT} is uniformly at random. We get around this by showing that the greedy algorithm is robust to small changes in streams. The intuition is that in the first part of the stream the greedy solution either is large for all permutations of E_{OPT} or it is small for all permutations. Hence, these are not random events depending on the order, but two cases in which we can assume a uniform distribution.

4.2 Analysis of the streaming algorithm for maximum matching

In this section, we prove that MATCH has an approximation ratio of at least $1/2 + \gamma$ in expectation for some fixed constant $\gamma > 0$.

First we prove a lemma on the robustness of the greedy algorithm to small changes in the stream. The lemma is required to deal with correlations that may arise in the case where we need to show that the greedy algorithm picks a significant fraction of edges of $|M^*|$.

Lemma 4.2 *Let σ and σ' be streams of edges in G such that σ can be transformed into σ' by deleting an edge from σ. Let M and M' be the matchings computed by the greedy algorithm on σ and σ' respectively. Then $||M| - |M'|| \leqslant 1$.*

Proof Let $C = (M \setminus M') \cup (M' \setminus M)$, that is, the symmetric difference of M and M'. Notice that C is a collection of disjoint paths and cycles that alternate between edges of M and M'. We claim that in this collection there is at most one path of non-zero length. This implies the statement in the lemma.

We argue that if such a path exists, it must contain the edge that was deleted. Hence, it is the unique path. Assume toward contradiction, that there exists a path in C, which does not contain the deleted edge. We now closely examine the first edge e of this path that arrives. Note that this is the same for σ and σ'. In both runs of the greedy algorithm, the two vertices of e are not incident to a matching edge when e arrives. This means that e should have been taken in both runs and therefore cannot be in the symmetric difference of the matchings, a contradiction.

In our analysis we will use the following lemma by Konrad et al which bounds the number of edges that cannot be augmented by 3-augmenting paths if the size of maximal matching is small.

Lemma 4.3 (Lemma 1 in [45]) *Let $\alpha > 0$, M be a maximal matching in G and M^* be a maximum matching in G with $|M| \leqslant (1/2 + \alpha)|M^*|$. Then the number of 3-augmentable edges in M is at least $(1/2 - 3\alpha)|M^*|$. In particular, the number of non-3-augmentable edges in M is at most $4\alpha|M^*|$.*

We are now ready to prove Theorem 3.1. We restate it for the convenience of the reader.

Theorem (3.1 restated) *There exists an absolute constant $\gamma > 0$ such that there is a semi-streaming algorithm for maximum matching under adversarial-injections with an approximation ratio of $1/2 + \gamma$ in expectation.*

Proof Define $\varepsilon = 1/50$, $\alpha = \varepsilon$, and $\rho = \varepsilon/4$. Without loss of generality, we assume that $|M^*| > 2/\rho$. Otherwise, the algorithm can just store the whole graph as it is sparse i.e., it has a linear number of edges.

Let σ_1 be the smallest prefix of the stream that contains $k = \lceil \rho \cdot |M^*| \rceil$ elements of M^*. Further, let σ_2 be the remainder of the stream. Notice, that $M_2^{(1)} \subseteq M_1$, since the first algorithm starts the same way, but continues even after reaching this threshold. Let $M_2^{(\sigma_1,1)} \subseteq M_2^{(1)}$ be a random variable corresponding to only those edges in $M_2^{(1)}$ taken during σ_1.

Case 1: For all permutations of $E_{\mathbf{opt}}$ it holds that $M_2^{(\sigma_1,1)} = M_2^{(1)}$. Notice that by definition, $|M_2^{(\sigma_1,1)}| = |M_2^{(1)}| \geqslant (1/2 - \varepsilon)|M^*|$.

The basic idea for this case is to show that in σ_2 there are a lot of 3-augmenting paths that can be used to improve $M_2^{(1)}$ via 3-Aug-Paths.

If $|M_1| \geqslant |M^*|(1/2 + \alpha)$, we are already significantly better than 1/2. Hence, assume otherwise. From Lemma 4.3 it follows that the number of non-3-augmentable edges in M_1 is at most $4\alpha|M^*|$. The number of 3-augmentable edges in $M_2^{(\sigma_1,1)}$ is obviously $|M_2^{(\sigma_1,1)}|$ minus the number of non-3-augmentable edges in it. The former is at least $|M^*|(1/2 - \varepsilon)$ while the latter is a subset of the non-3-augmentable edges in M_1 and hence at most $4\alpha|M^*|$. It follows that the number of 3-augmentable edges in $M_2^{(\sigma_1,1)}$ is at least $(1/2 - 4\alpha - \varepsilon)|M^*|$.

We will now restrict our attention to the subgraph of the edges in $M_2^{(\sigma_1,1)}$ and σ_2 and the 3-augmentable edges there. Recall, every 3-augmentable edge corresponds to a 3-augmenting path that has two edges from M^* and one from $M_2^{(\sigma_1,1)}$. If at least one of the edges from M^* appears in σ_1, this edge is no longer 3-augmentable when we restrict ourselves to σ_2. However, by definition only k of the edges from M^* appear in σ_1 and each of them can appear in only one 3-augmenting path. In consequence, the number of 3-augmentable edges in $M_2^{(\sigma_1,1)}$ considering only σ_2 is at least

$$\left(\frac{1}{2} - 4\alpha - \varepsilon\right)|M^*| - k \geqslant \left(\frac{1}{2} - 4\alpha - \varepsilon - \rho - \frac{1}{|M^*|}\right)|M^*|$$
$$\geqslant \left(\frac{1}{2} - 4\alpha - \varepsilon - \frac{3\rho}{2}\right)|M^*|.$$

Here we use that by assumption $|M^*| > 2/\rho$. After constructing the matching $M_2^{(1)}$, the second algorithm proceeds to collect 3-augmenting paths for it using 3-Aug-Paths. We fix its parameter

$$\beta := \left(\frac{1}{2} - 4\alpha - \varepsilon - \frac{3\rho}{2}\right) / \left(\frac{1}{2} - \varepsilon\right).$$

Notice that after completing Phase 1, the second algorithm has at least $\beta|M_2^{(1)}|$ many 3-augmenting paths in the remaining instance.

Hence, by Lemma 4.1 we are guaranteed to find $(\beta^2/32)|M_2^{(1)}|$ many 3-augmenting paths. We conclude that

$$|M_2| \geqslant \left(1+\frac{\beta^2}{32}\right)|M_2^{(1)}| \geqslant \left(1+\frac{\beta^2}{32}\right)(1-\varepsilon)|M^*|.$$

Using the definitions of the constants, we calculate that $\beta = (4 - 43\varepsilon)/(4-8\varepsilon)$.

Case 2: For at least one permutation of $E_{\mathbf{opt}}$ it holds that $M_2^{(\sigma_1,1)} \subsetneq M_2^{(1)}$. Let σ_1^* be the realization of the random variable σ_1 for such a permutation, i.e., we have $|M_2^{(\sigma_1^*,1)}| \leqslant (1/2-\varepsilon)|M^*|$. We will argue that in expectation (not just for σ_1^*) the greedy algorithm will select a considerable number of elements from M^*. This directly improves its guarantee: Let $S = M^* \cap M_1$. Every edge in $M^* \setminus S$ intersects with some edge in $M_1 \setminus S$, but every edge in $M_1 \setminus S$ can only intersect with two edges in $M^* \setminus S$. This implies $2|M_1 \setminus S| \geqslant |M^* \setminus S|$ and consequently

$$|M_1| \geqslant (|M^*| + |S|)/2. \tag{4.1}$$

We bound $\mathbb{E}[|S|]$ from below by examining the elements of M^* in σ_1, denoted by $e_1^*, e_2^*, \ldots, e_k^*$. Let $\sigma^{(1)}, \ldots, \sigma^{(k)}$ correspond to the prefix of σ until right before $e_1^*, \ldots, e_k^*$ arrive. Further, define $M_1^{(1)}, \ldots, M_1^{(k)}$ as the value of M_1 after each prefix $\sigma^{(1)}, \ldots, \sigma^{(k)}$.

Notice that $\sigma^{(k)}$ can be transformed to σ_1^* by adding and deleting at most $2k$ elements. Thus, it follows from Lemma 4.2 that for all $i \leqslant k$ and any σ_1,

$$|M_1^{(i)}| \leqslant |M_1^{(k)}| \leqslant (1/2-\varepsilon)|M^*| + 2k.$$

This implies that the number of edges in M^* not intersecting with edges in $M_1^{(i)}$ is at least $|M^*| - 2|M_1^{(i)}| \geqslant 2\varepsilon|M^*| - 4k$. If e_i^* is one of these edges, then it is taken in M_1. The probability for each element in M^*, which has not arrived yet, to be e_i^* is equal. Hence, conditioning on some choice of $\sigma_1^{(i)}$ the probability that e_i^* is taken in M_1 is at least $(2\varepsilon|M^*| - 4k)/(|M^*| - (i-1)) \geqslant 2\varepsilon - 4k/|M^*|$. Thus,

$$\begin{aligned}\mathbb{E}[S] &\geqslant \sum_{i=1}^{k} \mathbb{P}[e_i^* \in M_1] \geqslant k\left(2\varepsilon - \frac{4k}{|M^*|}\right)\\ &\geqslant \rho|M^*|\left(2\varepsilon - 4\frac{\rho|M^*|+1}{|M^*|}\right) = \left(2\varepsilon\rho - 4\rho^2 - \frac{4\rho}{|M^*|}\right)|M^*|.\end{aligned}$$

With assumption $M^* > 2/\rho$, (4.1) we conclude that

$$\mathbb{E}[|M_1|] \geqslant \left(\frac{1}{2} + 2\varepsilon\rho - 6\rho^2\right)|M^*| = \left(\frac{1}{2} + \frac{\varepsilon^2}{8}\right)|M^*|.$$

Taking the worst of the bounds, we calculate the constant

$$\gamma = \min\left\{\varepsilon, \frac{1}{32}\left(\frac{4-43\varepsilon}{4-8\varepsilon}\right)^2(1-\varepsilon)-\varepsilon, \frac{\varepsilon^2}{8}\right\} = \frac{1}{20000}. \qquad \square$$

4.3 Online Setting

Since we can improve 1/2 for the streaming setting, it is natural to hope that the existing techniques (e.g., the approach of the previous section) can be applied in the online setting as well. Surprisingly, this is not the case. In other words, the competitive ratio of 1/2 is optimal even for bipartite graphs. The technique from the previous section breaks apart, because the algorithm constructs several candidate solutions in parallel by guessing $|M^*|$. This is not a problem for a streaming algorithm, but, an online algorithm can only build one solution.

For a formal proof, we rely on the bipartite construction used in the proof of Theorem 3 from [31]. The authors show that there is no (randomized) algorithm with a competitive ratio of $1/2+\varepsilon$ for any $\varepsilon > 0$. More precisely, they show that not even a good fractional matching can be constructed online. For fractional matchings, randomization does not help and therefore we can assume the algorithm is deterministic. The original proof is with respect to adversarial order, but it is not hard to see that it transfers to adversarial injections.

The authors construct a bipartite instance that arrives in (up to) N rounds. In round i, a matching of size i arrives. The algorithm does not know whether the current round is the last one or not. Hence, it has to maintain a good approximation after each round. This forces the algorithm to take edges that do not belong to the optimal matching and eventually leads to a competitive ratio of 1/2. The same construction works in our model: The edges from the optimal matching arrive in the last round and their internal order does not affect the proof. In fact, the construction works for any order of the elements within a round. Thus, an algorithm cannot exploit the fact that their order is randomized and therefore also cannot do better than 1/2.

Chapter 5

Submodular Maximization

In this chapter, we consider the problem of submodular maximization subject to a cardinality constraint. Recall that in this problem, the algorithm has query access to a monotone, submodular function $f : 2^E \to \mathbb{R}_{\geqslant 0}$ over a ground set E. Moreover, f is normalized with $f(\emptyset) = 0$. The goal is to compute a set S of size at most k that maximizes $f(S)$. We present a 0.55-approximate streaming algorithm in the adversarial-injections model which only needs the memory to store $(O(k))^k$ many elements. Importantly, this number is independent of the length of the stream.

5.1 Notation

We denote by σ the stream of elements E, and by $-\infty$ and ∞ the start and end of the stream. For elements a and b, we write $\sigma[a, b]$ for the interval including a and b and $\sigma(a, b)$ for the interval excluding them. Moreover, we may assume that $f(\emptyset) = 0$, since otherwise, we can otherwise replace the submodular function with $f' : 2^E \to \mathbb{R}_{\geqslant 0}, T \mapsto f(T) - f(\emptyset)$.

Denote the permutation of E_{OPT} by π. Let o_i^π be the i'th element of E_{OPT} in the stream according to the order given by π. Let $O_0^\pi = \emptyset$ and $O_i^\pi = \{o_1^\pi, \dots, o_i^\pi\}$ for all i; hence, $E_{\text{OPT}} = O_k^\pi$ for any π. Finally, let $\text{OPT} = f(O_k^\pi)$.

5.2 The Algorithm

For simplicity we present an algorithm with the assumption that it knows the value OPT. Moreover, for the set of increases in f, that is $I = \{f(e \mid S) : e \in E, S \subseteq E\}$, we assume that $|I| \leqslant O(k)$. These two assumptions can be made at a marginal cost in the approximation ratio and with an insignificant increase in memory. This follows from standard techniques. We refer the reader to Appendix A.2 for details.

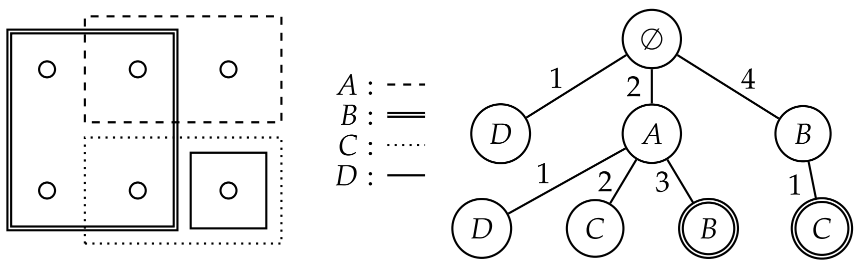

Figure 5.1: In this example, function f counts the dots covered by a set of rectangles. On the right, the tree for stream $\sigma = (A, B, C, D)$ and $k = 2$ is depicted. The labels on the edges correspond to the increase in f. The maximal leaves are highlighted.

As a central data-structure, the algorithm maintains a rooted tree T of height at most k. Every node except for the root stores a single element from E. The structure resembles a prefix tree: Each node is associated with the solution where the elements on the path from the root to the node is selected. The nodes can have at most $|I|$ children, that is, one for each increase. The basic idea is that for some partial solution $S \subseteq E$ (corresponding to a node) and two elements e, e' with $f(e \mid S) = f(e' \mid S)$ we only consider one of the solutions $S \cup \{e\}$ and $S \cup \{e'\}$. More precisely, the algorithm starts with a tree consisting only of the root. When it reads an element e from the stream, it adds e as a child to every node where (1) the distance of the node to the root is smaller than k and (2) the node does not have a child with increase $f(e \mid S)$, where S is the partial solution corresponding to this particular node.

Because of (1), the solutions are always of cardinality at most k. When the stream is read completely, the algorithm selects the best solution among all leaves. An example of the algorithm's behavior is given in Figure 5.1.

5.2.1 Overview of the Analysis

For analyzing the algorithm, we will use a sophisticated strategy to select one of the leaves and only compare this leaf to the optimum. We emphasize that this selection does not have to be computed by the algorithm. In particular, it does not need to be computable by a streaming algorithm and it can rely on knowledge of E_{OPT} and E_{NOISE}, which the algorithm does not have. Since the algorithm always takes the best leaf, we only need to give a lower bound for one of them. Before we describe this strategy, we analyze the tree algorithm in two educational corner cases.

The first one shows that by a careful selection of a leaf the algorithm appears to take elements based on the location of the E_{OPT}, although it does not know them. Let $r_i^{\pi} = \operatorname{argmax}_{e \in \sigma(-\infty, o_1^{\pi}]} f(e)$, that is, the most valuable element until the arrival of the first element from E_{OPT}. Here argmax breaks ties in favor of the first element in σ. We do not know when o_1^{π} arrives, but we know that the algorithm will have created a node (with the root as its parent) for r_1^π by then. We define iteratively $R_i^\pi = \{r_1^\pi, \ldots, r_i^\pi\}$ and $r_{i+1}^\pi = \operatorname{argmax}_{e \in \sigma(r_i^\pi, o_{i+1}^\pi]} f(e \mid R_i^\pi)$ for all i. Again, we can be sure that r_{i+1}^π, which yields the best increase for R_i^π until the arrival of o_{i+1}^π, is appended to the path $r_1^\pi \to \cdots \to r_i^\pi$.

This selection is inspired by the following idea. Suppose we could partition the stream into k intervals such that in each exactly one element from E_{OPT} appears. Then a sensible approach would be to start with an empty solution and greedily add the element that yields the maximal increase to our partial solution in each interval. Clearly one such partition would be $\sigma(o_i^\pi, o_{i+1}^\pi]$, $i = 1, \ldots, k$. We note that while the selection detailed in the previous paragraph is similar, it does not completely capture this idea. Although r_{i+1}^π is an element that arrives before o_{i+1}^π, we cannot be certain that it arrives after o_i^π. We only know that it arrives after r_i^π.

Next, we prove that the solution R_k^π is a 1/2-approximation. This already shows that the tree algorithm is 1/2-approximate even in the adversarial order model. By definition of R_i^π and r_i^π, we have

$$\begin{aligned} f(R_k^\pi) = \sum_{i=1}^{k} f(r_i^\pi \mid R_{i-1}^\pi) \geqslant \sum_{i=1}^{k} f(o_i^\pi \mid R_{i-1}^\pi) \\ = \sum_{i=1}^{k} [f(o_i^\pi \mid R_{i-1}^\pi) - f(o_i^\pi \mid R_k^\pi)] + \sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi). \end{aligned}$$

Notice that due to submodularity the term $f(o_i^\pi \mid R_{i-1}^\pi) - f(o_i^\pi \mid R_k^\pi)$ is always non-negative. Moreover, if $o_i^\pi = r_i^\pi \in R_k^\pi$, it collapses to $f(o_i^\pi \mid R_{i-1}^\pi)$. Thus, we can bound the right term of the equation and thereby $f(R_k^\pi)$ with

$$f(R_k^\pi) \geqslant \sum_{\substack{i=1 \\ r_i^\pi = o_i^\pi}}^{k} f(o_i^\pi \mid R_{i-1}^\pi) + \sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi).$$

From submodularity and monotonicity of f it follows that

$$\sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi) \geqslant f(O_k^\pi \mid R_k^\pi) = f(O_k^\pi \cup R_k^\pi) - f(R_k^\pi) \geqslant f(O_k^\pi) - f(R_k^\pi).$$

Hence, we conclude that

$$2f(R_k^\pi) \geqslant f(O_k^\pi) + \sum_{\substack{i=1 \\ r_i^\pi = o_i^\pi}}^{k} f(o_i^\pi \mid R_{i-1}^\pi).$$

This shows that R_k^π is 1/2-approximate, because $O_k^\pi = E_{\text{OPT}}$. Indeed, if a significant value of the elements in E_{OPT} are taken, then R_k^π is even better than 1/2-approximate.

Recall that the elements E_{OPT} are ordered randomly in the adversarial-injections model. Hence, the worst-case in the analysis above is that R_k^π is disjoint from E_{OPT} for all realizations of π. However, by a different analysis we can see that this case is in fact well-behaved. This is because the algorithm would select the same elements $r_1^\pi, \dots, r_k^\pi$ for every realization of π. Hence, we can safely drop the superscript π in R_i^π and r_i^π. Since for every element $o \in E_{\text{OPT}}$ there is some realization of π where $o_i^\pi = o$, yet the algorithm does not pick o_i^π, we can bound the increase of each r_i by

$$f(r_i \mid R_{i-1}) \geqslant \max_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}) \geqslant \frac{1}{k} \sum_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}).$$

By submodularity and monotonicity we get

$$\frac{1}{k} \sum_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}) \geqslant \frac{1}{k} f(E_{\text{OPT}} \mid R_{i-1}) \geqslant \frac{1}{k}(\text{OPT} - f(R_{i-1})).$$

This is the same recurrence formula as in the classic greedy algorithm and by simple calculations we get the closed form

$$f(R_k) \geqslant \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \text{OPT} \geqslant \left(1 - \frac{1}{e}\right) \text{OPT}.$$

In other words, the algorithm is $(1 - 1/e)$-approximate even in this case. In our main proof we will use a more involved strategy for selecting a leaf. This is to be able to combine the two approaches discussed above.

5.2.2 Analysis

Let us first define the selection of the leaf we are going to analyze. The elements on the path to this leaf will be denoted by $s_1^\pi, \dots, s_k^\pi$ and we write S_i^π for $\{s_1^\pi, \dots s_i^\pi\}$. The elements are defined inductively, but in contrast to the previous section we also need indices $n_1, \dots, n_k$. Recall that we previously defined the $(i+1)$'th element r_{i+1}^π as the best increase in $\sigma(r_i^\pi, o_{i+1}^\pi]$. Here, we use n_{i+1} to describe the index of the element from E_{OPT} which constitutes the end of this interval. It is not necessarily o_{i+1}^π anymore. We always start with $n_1 = 1$, but based on different cases we either set $n_{i+1} = n_i + 1$ or $n_{i+1} = n_i$. We underline that n_i is independent of the realization of π. In the following, $t \in [0, 1]$ denotes a parameter that we will specify later.

The element s_i^π will be chosen from two candidates u_i^π and v_i^π. The former is the best increase of elements excluding $o_{n_i}^\pi$, that is,

$$u_i^\pi = \begin{cases} \operatorname{argmax}_{e \in \sigma(-\infty, o_{n_1}^\pi)} f(e) & \text{if } i = 1, \\ \operatorname{argmax}_{e \in \sigma(s_i^\pi, o_{n_i}^\pi)} f(e \mid S_{i-1}^\pi) & \text{otherwise.} \end{cases}$$

The latter is defined in the same way, except it includes $o^\pi_{n_i}$ in the choices, that is,

$$v^\pi_i = \begin{cases} \operatorname{argmax}_{e \in \sigma(-\infty, o^\pi_{n_1}]} f(e) & \text{if } i = 1, \\ \operatorname{argmax}_{e \in \sigma(s^\pi_{i-1}, o^\pi_{n_i}]} f(e \mid S^\pi_{i-1}) & \text{otherwise.} \end{cases}$$

We now define the choice of s^π_i and n_{i+1} based on the following two cases. Note that the cases are independent from the realization of π.

Case 1: $\mathbb{E}_\pi f(u^\pi_i \mid S^\pi_{i-1}) \geqslant t \cdot \mathbb{E}_\pi f(o^\pi_{n_i} \mid S^\pi_{i-1})$. In this case, we set $s^\pi_i = u^\pi_i$ and $n_{i+1} = n_i$. Notice that this means s^π_i is chosen independently from $o^\pi_{n_i}$. In other words, we did not see $o^\pi_{n_i}$, yet. The element $o^\pi_{n_i}$ is still each of the remaining elements in E_{OPT} with equal probability. In the analysis this is beneficial, because the distribution of $o^\pi_{n_i}, \dots, o^\pi_k$ remains unchanged. This is similar to the second case in the previous section.

Case 2: $\mathbb{E}_\pi f(u^\pi_i \mid S^\pi_{i-1}) < t \cdot \mathbb{E}_\pi f(o^\pi_{n_i} \mid S^\pi_{i-1})$. Here, set $s^\pi_i = v^\pi_i$ and $n_{i+1} = n_i + 1$. Now the distribution of $o^\pi_i, \dots, o^\pi_k$ can change. However, a considerable value of s^π_i over different π comes from taking $o^\pi_{n_i}$. As indicated by the first case in the previous section this will improve the guarantee of the algorithm.

The solution S^π_k corresponds to a leaf in the tree algorithm. Clearly, u^π_1 and v^π_1 are children of the root. Hence, s^π_1 is also a child. Then for induction we assume s^π_i is a node, which implies u^π_{i+1} and v^π_{i+1} are also nodes: The elements u^π_{i+1} and v^π_{i+1} are the first elements after s^π_i with the respective gains $(f(u^\pi_{i+1} \mid S^\pi_i)$ and $f(v^\pi_{i+1} \mid S^\pi_i))$. Hence, s^π_{i+1} is a child of s^π_i.

In order to bound $\mathbb{E}_\pi f(S^\pi_k)$, we will study more broadly all values of $\mathbb{E}_\pi f(S^\pi_h)$ where $h \leqslant k$. To this end, we define a recursive formula $R(k,h)$ and prove that it bounds $\mathbb{E}_\pi f(S^\pi_h)/\text{OPT}$ from below. Then, using basic calculus we will show that $R(k,k) \geqslant 0.5506$ for all k. Initialize $R(k,0) = 0$ for all k. Then let $R(k,h)$, $h \leqslant k$, be defined by

$$R(k,h) = \min\left\{ \frac{t}{k} + \left(1 - \frac{t}{k}\right) R(k,h-1),\ \frac{1}{k} + \left(1 - \frac{1+t}{k}\right) R(k-1,h-1),\ \frac{1}{1+t} \right\}.$$

Lemma 5.1 *For all instances of the problem and $h \leqslant k$, the solution S^π_h as defined above satisfies $\mathbb{E}_\pi f(S^\pi_h) \geqslant R(k,h)\text{OPT}$.*

Proof The proof is by induction over h. For $h = 0$, the statement holds as $R(k,0)\text{OPT} = 0 = \mathbb{E}_\pi f(S^\pi_0)$. Let $h > 0$ and suppose the statement of the lemma holds with $h-1$ for all instances of the problem. Suppose we are given an instance with $k \geqslant h$. We distinguish the two cases $s^\pi_1 = u^\pi_1$ and $s^\pi_1 = v^\pi_1$.

First, consider $\mathbb{E}_\pi f(u^\pi_1) \geqslant t \cdot \mathbb{E}_\pi f(o^\pi_1)$, which implies that $s^\pi_1 = u^\pi_1$. Note that u^π_1 is the best element in $\sigma(-\infty, o^\pi_1)$, consequently, its choice is independent from the realization of π. Let us drop the superscript in u_1^π and s_1^π for clarity. We construct a new instance mimicking the subtree of s_1. Formally, our new instance still has the same k elements from E_{OPT}, i.e., $k' = k$. The stream is $\sigma' = \sigma(s_1^\pi, \infty)$ and, the submodular function $f' : 2^U \to \mathbb{R}$, $f'(T) \mapsto f(T \mid s_1)$. In this instance we have $\text{OPT}' = f'(E_{\text{OPT}}) = f(E_{\text{OPT}} \mid s_1) \geqslant \text{OPT} - f(s_1)$. It is easy to see that the elements $s_1'^\pi, \dots, s_{h-1}'^\pi$ chosen in the new instance correspond exactly to the elements $s_2^\pi, \dots, s_h^\pi$. Hence, with the induction hypothesis we get

$$\begin{aligned}\mathbb{E}_\pi f(S_h^\pi) = f(s_1) + \mathbb{E}_\pi f(S_h^\pi \mid s_1) = f(s_1) + \mathbb{E}_\pi f'(S_{h-1}'^\pi)\\ \geqslant f(s_1) + R(k, h-1)(\text{OPT} - f(s_1)).\end{aligned}$$

By assumption we have $f(s_1) \geqslant t \cdot \mathbb{E}_\pi f(o_i^\pi) \geqslant t \cdot \text{OPT}/k$. Together with $R(k, h-1) \leqslant 1/(1+t) \leqslant 1$ we calculate

$$f(s_1) + R(k, h-1)(\text{OPT} - f(s_1)) \geqslant \frac{t}{k}\text{OPT} + R(k, h-1)\left(1 - \frac{t}{k}\right)\text{OPT}.$$

The right-hand side is by definition at least $R(k, h)\text{OPT}$.

Now we turn to the case $\mathbb{E}_\pi f(u_1^\pi) < t \cdot \mathbb{E}_\pi f(o_1^\pi)$, which means $s_1^\pi = v_1^\pi$ is chosen. Similar to the previous case, we construct a new instance. After taking s_1^π, our new instance has $k' = k - 1$ elements $E'_{\text{OPT}} = E_{\text{OPT}} \setminus \{o_1^\pi\}$, the stream $\sigma' = \sigma(s_1, \infty)$, and the submodular function $f' : 2^E \to \mathbb{R}$, $f(T) \mapsto f(T \mid s_1^\pi)$. Thus, $\text{OPT}' = f'(E'_{\text{OPT}}) = f(E_{\text{OPT}} \setminus \{o_1^\pi\} \mid s_1^\pi) \geqslant \text{OPT} - f(s_1^\pi \cup o_1^\pi)$. We remove o_1^π from E_{OPT}, because $s_1^\pi = v_1^\pi$ depends on it. The distribution of $o_2^\pi, \dots, o_k^\pi$ when conditioning on the value of o_1^π (and thereby the choice of s_1^π) is still a uniformly random permutation of E'_{OPT}. Like in the previous case, we can see that $S_{h-1}'^\pi = S_h^\pi \setminus \{s_1^\pi\}$ and we can apply the induction hypothesis. First, however, let us examine $\mathbb{E}_\pi f(s_1^\pi \cup o_1^\pi)$. Since we know that whenever $s_1^\pi \neq o_1^\pi$ we have $s_1^\pi = u_1^\pi$, it follows that

$$\mathbb{P}_\pi[s_1^\pi \neq o_1^\pi] \cdot \mathbb{E}_\pi[f(s_1) \mid s_1^\pi \neq o_1^\pi] \leqslant \mathbb{E}_\pi f(u_1^\pi) < t \cdot \mathbb{E}_\pi f(o_1^\pi) \leqslant t \cdot \mathbb{E}_\pi f(s_1^\pi).$$

Hence, we deduce

$$\begin{aligned}\mathbb{E}_\pi f(s_1^\pi \cup o_1^\pi) \leqslant \mathbb{E}_\pi f(o_1^\pi) + \mathbb{P}_\pi[s_1^\pi \neq o_1^\pi] \cdot \mathbb{E}_\pi[f(s_1) \mid s_1^\pi \neq o_1^\pi]\\ \leqslant \mathbb{E}_\pi f(o_1^\pi) + t \cdot \mathbb{E}_\pi f(s_1^\pi).\end{aligned}$$

We are ready to prove the bound on $\mathbb{E}_\pi f(S_h^\pi)$. By induction hypothesis, we get

$$\begin{aligned}\mathbb{E}_\pi f(S_h^\pi) &= \mathbb{E}_\pi f(s_1^\pi) + \mathbb{E}_\pi f'(S_{h-1}'^\pi)\\ &\geqslant \mathbb{E}_\pi f(s_1^\pi) + R(k-1, h-1)(\text{OPT} - \mathbb{E}_\pi f(s_1^\pi \cup o_1^\pi)).\end{aligned}$$

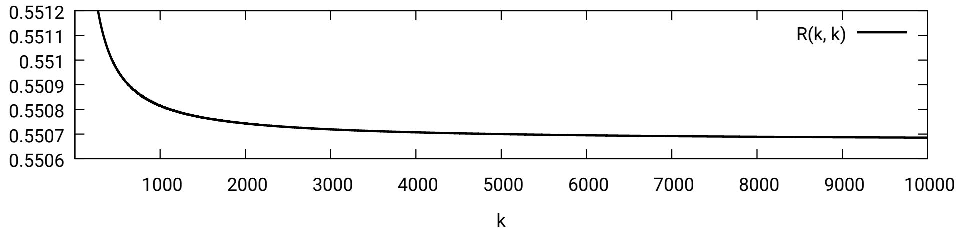

Figure 5.2: Values of the recurrence formula for $t = 0.8$.

Inserting the bound on $\mathbb{E}_\pi f(s_1^\pi \cup o_1^\pi)$ we know that the right-hand side is at least

$$\mathbb{E}_\pi f(s_1^\pi) + R(k-1, h-1)(\mathrm{OPT} - \mathbb{E}_\pi f(o_1^\pi) - t \cdot \mathbb{E}_\pi f(s_1^\pi)).$$

Using that $f(s_1^\pi) \geqslant f(o_1^\pi)$ for all π and $R(k-1, h-1) \cdot t \leqslant t/(1+t) \leqslant 1$ we bound the previous term from below by

$$\mathbb{E}_\pi f(o_1^\pi) + R(k-1, h-1)(\mathrm{OPT} - (1+t)\mathbb{E}_\pi f(o_1^\pi)).$$

Finally, we use that $\mathbb{E}_\pi f(o_1^\pi) \geqslant \mathrm{OPT}/k$ and $R(k-1, h-1)(1+t) \leqslant 1$ to arrive at

$$\frac{1}{k}\mathrm{OPT} + R(k-1, h-1)\left(\mathrm{OPT} - \frac{1+t}{k}\mathrm{OPT}\right) \geqslant R(k, h)\mathrm{OPT},$$

which concludes the proof. □

With $t = 0.8$ we are able to show that for sufficiently large k the minimum in the definition of $R(k, k)$ is always attained by the first term. Then, after calculating a lower bound on $R(k, k)$ for small values, we can easily derive a general bound.

Lemma 5.2 *With $t = 0.8$ for all positive integers k it holds that $R(k, k) \geqslant 0.5506$.*

Figure 5.2 contains a diagram (generated by computer calculation), which shows that the formula tends to a value between 0.5506 and 0.5507 for $k \in \{0, \ldots, 10000\}$. The proof requires tedious and mechanical calculations and hence is omitted here. We refer the reader to Appendix A.2 for complete details.

Part II

Submodular Matroid Intersection

Chapter 6

Introduction

In this second part of the thesis, we consider the problem of weighted matroid intersection in the semi-streaming setting. We refer the reader to Section 2.3 for the definition of a matroid. Recall that in our problem, we are given an oracle access to two matroids $M_1 = (E, I_1), M_2 = (E, I_2)$ on a common ground set E, and a non-negative weight function $w : E \to \mathbb{R}_{\geqslant 0}$ on the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $w(X) = \sum_{e \in X} w(e)$ is maximized. Our main result is a $(2 + \varepsilon)$-approximate semi-streaming algorithm for this problem. We then extend this algorithm to also work for submodular functions by borrowing some ideas from [49].

6.1 Literature Review

For the unweighted maximum matching problem (See Subsection 2.7.1 for a definition), the best known semi-streaming algorithm is the basic greedy approach:

Initially, let $M = \emptyset$. Then for each edge e in the stream, add it to M if $M \cup \{e\}$ is a feasible solution, i.e., a matching; otherwise the edge e is discarded.

The algorithm uses space $O(|V| \log |V|)$ and a simple proof shows that it returns a 2-approximate solution in the *unweighted* case, i.e, a matching of size at least half the size of a maximum matching. However, this basic approach fails to achieve any approximation guarantee for *weighted graphs*.

Indeed, for weighted matchings, it is non-trivial to get even a small constant-factor approximation. One way to do so is to replace edges if we have a much heavier edge. This is formalized in [27] whose authors get a 6-approximation.

Later, [52] improved this algorithm to find a 5.828-approximation; and, with a more involved technique, [17] provided a $(4+\varepsilon)$-approximation.

It was only in the recent breakthrough work [60] that the gap in the approximation guarantee between unweighted and weighted matchings was closed. Specifically, [60] gave a semi-streaming algorithm for weighted matchings with an approximation guarantee of $2+\varepsilon$ for every $\varepsilon > 0$. Shortly after, [32] came up with a simplified analysis of their algorithm, reducing the memory requirement from $O_\varepsilon(|V|\log^2|V|)$ to $O_\varepsilon(|V|\log|V|)$. These results for weighted matchings are tight (up to ε) in the sense that any improvement would also improve the state-of-the-art in the unweighted case, which is a long-standing open problem.

The algorithm by [60] is an elegant use of the local ratio technique in [8] and [7] in the semi-streaming setting. While this technique is very versatile and it readily generalizes to weighted hypergraph matchings, it is much harder to use for the related problem of weighted matroid intersection. This is perhaps surprising as many of the prior results for the matching problem also applied to the matroid intersection problem in the semi-streaming model. Indeed, the greedy algorithm still returns a 2-approximate solution in the unweighted case and the algorithm in [17] returns a $(4+\varepsilon)$-approximate solution for weighted instances. So, prior to our work, the status of the matroid intersection problem was that of the matching problem *before* [60].

6.2 Overview of Results and Techniques

We now describe at a high-level the reason that the techniques from [60] are not easily applicable to matroid intersection and our approach for dealing with this difficulty. The approach in [60] works in two parts, first certain elements of the stream are selected and added to a set S, and then at the end of the stream a matching M is computed by the greedy algorithm that inspects the edges of S in the reverse order in which they were added. This way of constructing the solution M greedily by going backwards in time is a standard framework for analyzing algorithms based on the local ratio technique. Now in order to adapt the algorithm in [60] to matroid intersection, recall that the bipartite matching problem can be formulated as the intersection of two partition matroids. We can thus reinterpret their algorithm and analysis in this setting. Furthermore, after this reinterpretation, it is not too hard to define an algorithm that works for the intersection of any two matroids. However, bipartite matching is a *special* case of matroid intersection which captures a rich set of seemingly more complex problems. This added expressiveness causes the analysis and the standard framework for analyzing local ratio algorithms to fail. Specifically, we prove that a solution formed by running the greedy algorithm on S in the reverse order (as done for the matching problem) fails to give any constant-factor approximation guarantee for the matroid intersection problem. To overcome this and to obtain our main result, we make a connection to a concept called matroid kernels (see [29] for more details about kernels), which allows us to identify a subset of S with an approximation guarantee of $2+\varepsilon$ in a more complex way.

Finally, for the intersection of more than two matroids, the approach used in the analysis does not work, because the notion of matroid kernel does not generalize to more than two matroids. However, we conjecture that the subset S generated for the intersection of k matroids still contains a $(k+\varepsilon)$-approximation. Currently, the best approximation results are a $(k^2+\varepsilon)$-approximation from [17] and a $(2(k+\sqrt{k(k-1)})-1)$-approximation from [13]. For $k=3$, the former is better, giving a $(9+\varepsilon)$-approximation. For $k>3$, the latter is better, giving an $O(k)$-approximation.

Generalization to submodular functions. Recently, Levin and Wajc [49] obtained improved approximation ratios for matching and b-matching problems in the semi-streaming model with respect to submodular functions. Specifically, they get a $(3+2\sqrt{2})$-approximation for monotone submodular b-matching, a $(4+3\sqrt{2})$-approximation for non-monotone submodular matching, and a $(3+\varepsilon)$-approximation for maximum weight (linear) b-matching. In our work, we are able to extend our algorithm for weighted matroid intersection to work with submodular functions by combining our and their ideas. In fact, we are able to generalize all their results to the case of matroid intersection with better or equal[1] approximation ratios: we get a $(3+2\sqrt{2}+\delta)$-approximation for monotone submodular matroid intersection, a $(4+3\sqrt{2}+\delta)$-approximation for non-monotone submodular matroid intersection and a $(2+\varepsilon)$-approximation for maximum weight (linear) matroid intersection.

[1]One can get rid of the δ factor if we assume that the function value is polynomially bounded by $|E|$, an assumption made by [49].

Chapter 7

The Local Ratio Technique for Weighted Matroid Intersection

In this chapter, we first present the local ratio algorithm for the weighted matching problem that forms the basis of the semi-streaming algorithm in [60]. We then adapt it to the weighted matroid intersection problem. While the algorithm is fairly natural to adapt to this setting, we give an example in Section 7.2 that shows that the same techniques as used for analyzing the algorithm for matchings does not work for matroid intersection. Instead, our analysis, which is presented in Section 7.3, deviates from the standard framework for analyzing local ratio algorithms and it heavily relies on a structural property of matroid intersection known as kernels. We remark that the algorithms considered in the aforementioned sections do not have a small memory footprint. We deal with this in Section 7.4 to obtain our semi-streaming algorithm.

7.1 Local Ratio Technique for Weighted Matching

The local ratio algorithm for the weighted matching problem is given in Algorithm 1. The algorithm maintains vertex potentials $w(u)$ for every vertex u, a set S of selected edges, and an auxiliary weight function $g : S \to \mathbb{R}_{\geqslant 0}$ of the selected edges. Initially the vertex potentials are set to 0 and the set S is empty. When an edge $e = \{u, v\}$ arrives, the algorithm computes how much it gains compared to the previous edges, by taking its weight minus the weight/potential of its endpoints ($g(e) = w(e) - w(u) - w(v)$). If the gain is positive, then we add the edge to S, and add the gain to the weight of the endpoints, that is, we set $w(u) = w(u) + g(e)$ and $w(v) = w(v) + g(e)$.

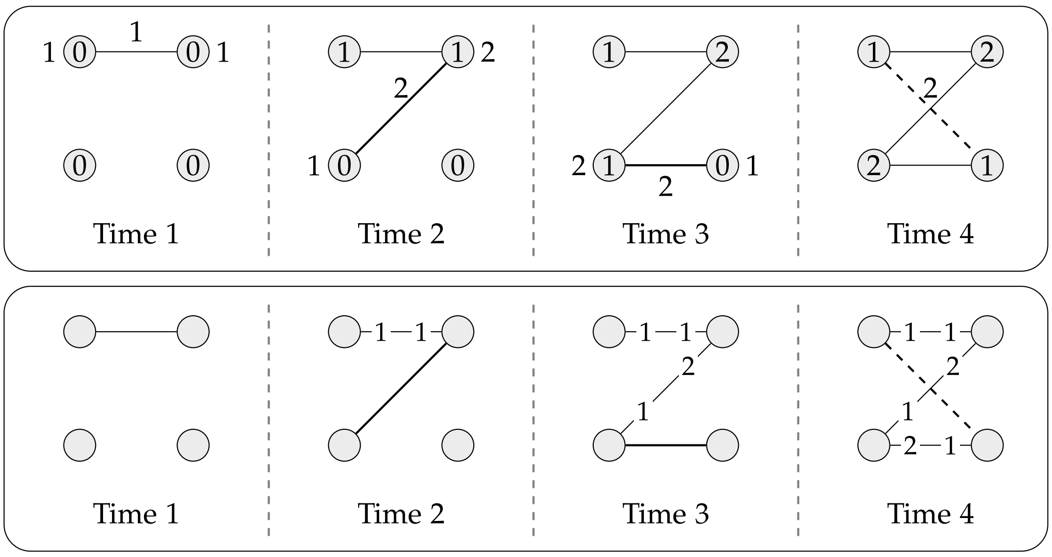

Figure 7.1: The top part shows an example execution of the local ratio technique for weighted matchings (Algorithm 1). The bottom part shows how to adapt this (bipartite) example to the language of weighted matroid intersection (Algorithm 2).

Algorithm 1 Local ratio algorithm for weighted matching

Input: A stream of the edges of a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geqslant 0}$.

Output: A matching M.

```
 1: $S \leftarrow \emptyset$
 2: $\forall u \in V, w(u) \leftarrow 0$
 3: for edge $e = (u, v)$ in the stream do
 4:     if $w(u) + w(v) < w(e)$ then
 5:         $g(e) \leftarrow w(e) - w(u) - w(v)$
 6:         $w(u) \leftarrow w(u) + g(e)$
 7:         $w(v) \leftarrow w(v) + g(e)$
 8:         $S \leftarrow S \cup \{e\}$
 9:     end if
10: end for
11: return a maximum weight matching $M$ among the edges stored on the
    stack $S$
```

For a better intuition of the algorithm, consider the example depicted on the top of Figure 7.1. The stream consists of four edges e_1, e_2, e_3, e_4 with weights $w(e_1) = 1$ and $w(e_2) = w(e_3) = w(e_4) = 2$. At each time step i, we depict the arriving edge e_i in thick along with its weight; the vertex potentials before the algorithm considers this edge is written on the vertices, and the updated vertex potentials (if any) after considering e_i are depicted next to the incident vertices. The edges that are added to S are solid and those that are not added to S are dashed.

At the arrival of the first edge of weight $w(e_1) = 1$, both incident vertices have potential 0 and so the algorithm adds this edge to S and increases the incident vertex potentials with the gain $g(e_1) = 1$. For the second edge of weight $w(e_2) = 2$, the sum of incident vertex potentials is 1 and so the gain of e_2 is $g(e_2) = 2 - 1$, which in turn causes the algorithm to add this edge to S and to increase the incident vertex potentials by 1. The third time step is similar to the second. At the last time step, edge e_4 of weight $w(e_4) = 2$ arrives. As the incident vertex potentials sum up to 2 the gain of e_4 is not strictly positive and so this edge is *not* added to S and no vertex potentials are updated. Finally, the algorithm returns the maximum weight matching in S which in this case consists of edges $\{e_1, e_3\}$ and has weight 3. Note that the optimal matching of this instance had weight 4 and we thus found a 4/3-approximate solution.

In general, the algorithm has an approximation guarantee of 2. This is proved using a common framework to analyze algorithms based on the local ratio technique: We ignore the weights and greedily construct a matching M by inspecting the edges in S in reverse order, i.e., we first consider the edges that were added last. An easy proof (see e.g. [32]) then shows that the matching M constructed in this way has weight at least half the optimum weight.

In the next section, we adapt the above described algorithm to the context of matroid intersections. We also give an example that the above framework for the analysis fails to give any constant-factor approximation guarantee. Our alternative (tight) analysis of this algorithm is then given in Section 7.3.

7.2 Adaptation to Weighted Matroid Intersection

When adapting Algorithm 1 to matroid intersection to obtain Algorithm 2, the first problem we encounter is the fact that matroids do not have a notion of vertices, so we cannot keep a weight/potential for each vertex. To understand how we overcome this issue, it is helpful to consider the case of bipartite matching and in particular the example depicted in Figure 7.1. It is well known that the weighted matching problem on a bipartite graph with edge set E and bipartition V_1, V_2 can be modelled as a weighted matroid intersection problem on matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ where for $i \in \{1,2\}$

$$I_i = \{E' \subseteq E \mid \text{each vertex } v \in V_i \text{ is incident to at most one vertex in } E'\}\,.$$

Instead of keeping a weight for each vertex, we will maintain two weight functions w_1 and w_2, one for each matroid. These weight functions will be

Algorithm 2 Local ratio for matroid intersection

```
Input: A stream of the elements of the common ground set of matroids
  $M_1 = (E, I_1), M_2 = (E, I_2)$.
Output: A set $X \subseteq E$ that is independent in both matroids.
  $S \leftarrow \emptyset$
  for element $e$ in the stream do
    calculate $w_i^*(e) = \max\left(\{0\} \cup \{\theta : e \in \mathrm{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\right)$
    for $i \in \{1, 2\}$.
    if $w(e) > w_1^*(e) + w_2^*(e)$ then
      $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$
      $w_1(e) \leftarrow w_1^*(e) + g(e)$
      $w_2(e) \leftarrow w_2^*(e) + g(e)$
      $S \leftarrow S \cup \{e\}$
    end if
  end for
  return a maximum weight set $T \subseteq S$ that is independent in $M_1$ and $M_2$
```

set so that the following holds in the special case of bipartite matching: on the arrival of a new edge e, let $T_i \subseteq S$ be an independent set in I_i of selected edges that maximizes the weight function w_i. Then we have that

$$\min_{f \in T_i : T_i \setminus \{f\} \cup \{e\} \in I_i} w_i(f) \qquad \text{if } T_i \cup \{e\} \notin I_i \text{ and } 0 \text{ otherwise} \tag{7.1}$$

equals the vertex potential of the incident vertex V_i when running Algorithm 1. It is well-known (e.g. by the optimality of the greedy algorithm for matroids) that the cheapest element f to remove from T_i to make $T_i \setminus \{f\} \cup \{e\}$ an independent set equals the largest weight θ so that the elements of weight at least θ spans e. We thus have that (7.1) equals

$$\max\left(\{0\} \cup \{\theta : e \in \mathrm{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\right).$$

It follows that the quantities $w_1^*(e)$ and $w_2^*(e)$ in Algorithm 2 equal the incident vertex potentials in V_1 and V_2 of Algorithm 1 in the special case of bipartite matching. To see this, let us return to our example in Figure 7.1 and let V_1 be the two vertices on the left and V_2 be the two vertices on the right. In the bottom part of the figure, the weight functions w_1 and w_2 are depicted (at the corresponding side of the edge) after the arrival of each edge. At time step 1, e_1 does not need to replace any elements in any of the matroids and so $w_1^*(e_1) = w_1^*(e_2) = 0$. We therefore have that its gain is $g(e_1) = 1$ and the algorithm sets $w_1(e_1) = w_2(e_1) = 1$. At time 2, edge e_2 of weight 2 arrives. It is not spanned in the first matroid whereas it is spanned by edge e_1 of weight 1 in the second matroid. It follows that $w_1^*(e_2) = 0$ and $w_2^*(e_2) = w_2(e_1) = 1$ and so e_2 has positive gain $g(e_2) = 1$ and it sets

$w_1(e_2) = 1$ and $w_2(e_2) = w_2(e_1) + 1 = 2$. The third time step is similar to the second. At the last time step, e_4 of weight 2 arrives. However, since it is spanned by e_1 with $w_1(e_1) = 1$ in the first matroid and by e_3 with $w_2(e_3) = 1$ in the second matroid, its gain is 0 and it is thus not added to the set S. Note that throughout this example, and in general for bipartite graphs, Algorithm 2 is identical to Algorithm 1. One may therefore expect that the analysis of Algorithm 1 also generalizes to Algorithm 2. We explain next why this is not the case for general matroids.

Counter Example to Same Approach in Analysis

We give a simple example showing that the greedy selection (as done in the analysis for Algorithm 1 for weighted matching) does not work for matroid intersection. Still, it turns out that the set S generated by Algorithm 2 always contains a 2-approximation but the selection process is more involved.

Lemma 7.1 *There exist two matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ on a common ground set E and a weight function $w : E \to \mathbb{R}_{\geqslant 0}$ such that a greedy algorithm that considers the elements in the set S in the reverse order of when they were added by Algorithm 2 does not provide any constant-factor approximation.*

Proof The example consists of the ground set $E = \{a, b, c, d\}$ with weights $w(a) = 1, w(b) = 1 + \varepsilon, w(c) = 2\varepsilon, w(d) = 3\varepsilon$ for a small $\varepsilon > 0$ (the approximation guarantee will be at least $\Omega(1/\varepsilon)$). The matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ are defined by

- a subset of E is in I_1 if and only if it does not contain $\{a, b\}$; and
- a subset of E is in I_2 if and only if it contains at most two elements.

To see that M_1 and M_2 are matroids, note that M_1 is a partition matroid with partitions $\{a, b\}, \{c\}$ and $\{d\}$, and M_2 is the 2-uniform matroid (alternatively, one can easily check that M_1 and M_2 satisfy the definition of a matroid).

Now consider the execution of Algorithm 2 when given the elements of E in the order a, b, c, d:

- Element a has weight 1, and $\{a\}$ is independent both in M_1 and M_2, so we set $w_1(a) = w_2(a) = g(a) = 1$ and a is added to S.
- Element b is spanned by a in M_1 and not spanned by any element in M_2. So we get $g(b) = w(b) - w_1^*(b) - w_2^*(b) = 1 + \varepsilon - 1 - 0 = \varepsilon$. As $\varepsilon > 0$, we add b to S, and set $w_1(b) = w_1(a) + \varepsilon = 1 + \varepsilon$ and $w_2(b) = \varepsilon$.
- Element c is not spanned by any element in M_1 but is spanned by $\{a, b\}$ in M_2. As b has the smallest w_2 weight, $w_2^*(c) = w_2(b) = \varepsilon$. So we have $g(c) = 2\varepsilon - w_1^*(c) - w_2^*(c) = 2\varepsilon - 0 - \varepsilon = \varepsilon > 0$, and we set $w_1(c) = \varepsilon$ and $w_2(c) = 2\varepsilon$ and add c to S.

- Element d is similar to c. We have $g(d) = 3\varepsilon - 0 - 2\varepsilon = \varepsilon > 0$ and so we set $w_1(d) = \varepsilon$ and $w_2(d) = 3\varepsilon$ and add d to S.

As the algorithm selected all the elements, we have $S = E$. It follows that the greedy algorithm on S will select d and c (in the reverse order of when elements were added), after which the set is a maximal independent set in M_2. This gives a weight of 5ε, even though a and b both have weight at least 1, which shows that this algorithm does not guarantee any constant factor approximation. □

7.3 Analysis of Algorithm 2

We prove that Algorithm 2 has an approximation guarantee of 2.

Theorem 7.2 *Let S be the subset generated by Algorithm 2 on a stream E of elements, matroids $M_1 = (E, I_1), M_2 = (E, I_2)$ and weight function $w : E \to \mathbb{R}_{\geqslant 0}$. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 whose weight $w(T)$ is at least $w(S^*)/2$, where S^* denotes an optimal solution to the weighted matroid intersection problem.*

Throughout the analysis we fix the input matroids $M_1 = (E, I_1), M_2 = (E, I_2)$, the weight function $w : R \to \mathbb{R}_{\geqslant 0}$, and the order of the elements in the stream. While Algorithm 2 only defines the weight functions w_1 and w_2 for the elements added to the set S, we extend them in the analysis by, for $i \in \{1, 2\}$, letting $w_i(e) = w_i^*(e)$ for the elements e not added to S.

We now prove Theorem 7.2 by showing that $g(S) \geqslant w(S^*)/2$ (Lemma 7.4) and that there is a solution $T \subseteq S$ such that $w(T) \geqslant g(S)$ (Lemma 7.5). In the proof of both these lemmas, we use the following properties of the computed set S.

Lemma 7.3 *Let S be the set generated by Algorithm 2 and $S' \subseteq S$ any subset. Consider one of the matroids M_i with $i \in \{1, 2\}$. There exists a subset $T' \subseteq S'$ that is independent in M_i, i.e., $T' \in I_i$, and $w_i(T') \geqslant g(S')$. Furthermore, the maximum weight independent set in M_i with respect to w_i over the whole ground set E can be selected to be a subset of S, i.e. $T_i \subseteq S$, and it satisfies $w_i(T_i) = g(S)$.*

Proof Consider matroid M_1 (the proof is identical for M_2) and fix $S' \subseteq S$. The set $T_1' \subseteq S'$ that is independent in M_1 and that maximizes $w_1(T_1')$ satisfies

$$w_1(T_1') = \int_0^\infty \operatorname{rank}(\{e \in T_1' \mid w_1(e) \geqslant \theta\})\, d\theta = \int_0^\infty \operatorname{rank}(\{e \in S' \mid w_1(e) \geqslant \theta\})\, d\theta\,.$$

The second equality follows from the fact that the greedy algorithm that considers the elements in decreasing order of weight is optimal for matroids and thus we have $\operatorname{rank}(\{e \in T_1' \mid w_1(e) \geqslant \theta\}) = \operatorname{rank}(\{e \in S' \mid w_1(e) \geqslant \theta\})$ for any $\theta \in \mathbb{R}$.

Now index the elements of $S' = \{e_1, e_2, \dots, e_\ell\}$ in the order they were added to S by Algorithm 2 and let $S'_j = \{e_1, \dots, e_j\}$ for $j = 0, 1, \dots, \ell$ (where $S'_0 = \emptyset$). By the above equalities and by telescoping,

$$w_1(T'_1) = \sum_{i=1}^{\ell} \int_0^\infty \left(\text{rank}(\{e \in S'_i \mid w_1(e) \geqslant \theta\}) - \text{rank}(\{e \in S'_{i-1} \mid w_1(e) \geqslant \theta\})\right) d\theta \,.$$

We have that $\text{rank}(\{e \in S'_i \mid w_1(e) \geqslant \theta\}) - \text{rank}(\{e \in S'_{i-1} \mid w_1(e) \geqslant \theta\})$ equals 1 if $w(e_i) \geqslant \theta$ and $e_i \notin \text{span}(\{e \in S'_{i-1} \mid w_1(e) \geqslant \theta\})$ and it equals 0 otherwise. Therefore, by the definition of $w_1^*(\cdot)$, the gain $g(\cdot)$ and $w_1(e_i) = w_1^*(e_i) + g(e_i)$ in Algorithm 2 we have

$$\begin{aligned} w_1(T'_1) &= \sum_{i=1}^{\ell} \left[w_1(e_i) - \max\left(\{0\} \cup \{\theta : e_i \in \text{span}\left(\{f \in S'_{i-1} \mid w_i(f) \geqslant \theta\}\right)\}\right)\right] \\ &\geqslant \sum_{i=1}^{\ell} g(e_i) = g(S') \,. \end{aligned}$$

The inequality holds because S'_{i-1} is a subset of the set S at the time when Algorithm 2 considers element e_i. Moreover, if $S' = S$, then S'_{i-1} equals the set S at that point and so we then have

$$w_1^*(e_i) = \max\left(\{0\} \cup \{\theta : e_i \in \text{span}\left(\{f \in S'_{i-1} \mid w_i(f) \geqslant \theta\}\right)\}\right)$$

which implies that the above inequality holds with equality in that case. We can thus also conclude that a maximum weight independent set $T_1 \subseteq S$ satisfies $w_1(T_1) = g(S)$. Finally, we can observe that T_1 is also a maximum weight independent set over the whole ground set since we have $\text{rank}(\{e \in S \mid w_1(e) \geqslant \theta\}) = \text{rank}(\{e \in E \mid w_1(e) \geqslant \theta\})$ for every $\theta > 0$, which holds because, by the extension of w_1, an element $e \notin S$ satisfies $e \in \text{span}(\{f \in S : w_1(f) \geqslant w_1(e)\})$. $\square$

We can now relate the gain of the elements in S to the weight of an optimal solution.

Lemma 7.4 *Let S be the subset generated by Algorithm 2. Then $g(S) \geqslant w(S^*)/2$.*

Proof We first observe that $w_1(e) + w_2(e) \geqslant w(e)$ for every element $e \in E$. Indeed, for an element $e \in S$, we have by definition $w(e) = g(e) + w_1^*(e) + w_2^*(e)$, and $w_i(e) = g(e) + w_i^*(e)$, so $w_1(e) + w_2(e) = 2g(e) + w_1^*(e) + w_2^*(e) = w(e) + g(e) > w(e)$. In the other case, when $e \notin S$ then $w_1^*(e) + w_2^*(e) \geqslant w(e)$, and $w_i(e) = w_i^*(e)$, so automatically, $w_1(e) + w_2(e) \geqslant w(e)$.

The above implies that $w_1(S^*) + w_2(S^*) \geqslant w(S^*)$. On the other hand, by Lemma 7.3, we have $w_i(T_i) \geqslant w_i(S^*)$ (since T_i is a max weight independent set in M_i with respect to w_i) and $w_i(T_i) = g(S)$, thus $g(S) \geqslant w_i(S^*)$ for $i = 1, 2$. $\square$

We finish the proof of Theorem 7.2 by proving that there is a $T \subseteq S$ independent in both M_1 and M_2 such that $w(T) \geqslant g(S)$. As described in Section 7.2, we cannot select T using the greedy method. Instead, we select T using the concept of kernels studied in [29].

Lemma 7.5 *Let S be the subset generated by Algorithm 2. Then for any subset $S' \subseteq S$, there exists a subset $T \subseteq S'$ independent in M_1 and in M_2 such that $w(T) \geqslant g(S')$.*

Proof Consider one of the matroids M_i with $i \in \{1, 2\}$ and define a total order $<_i$ on E such that $e <_i f$ if $w_i(e) > w_i(f)$, or if $w_i(e) = w_i(f)$ and e appeared later in the stream than f. The pair $(M_i, <_i)$ is known as an ordered matroid. We further say that a subset E' of E dominates element e of E if $e \in E'$ or there is a subset $C_e \subseteq E'$ such that $e \in \text{span}(C_e)$ and $c < e$ for all elements c of C_e. The set of elements dominated by E' is denoted by $D_{M_i}(E')$. Note that if E' is an independent set, then the greedy algorithm that considers the elements of $D_{M_i}(E')$ in the order $<_i$ selects exactly the elements E'.

Theorem 2 in [29] says that for two ordered matroids $(M_1, <_1), (M_2, <_2)$ there always is a set $K \subseteq E$, which is referred to as a M_1M_2-kernel, such that

- K is independent in both M_1 and in M_2; and
- $D_{M_1}(K) \cup D_{M_2}(K) = E$.

We use the above result on M_1 and M_2 restricted to the elements in S'. Specifically we select $T \subseteq S'$ to be the kernel such that $D_{M_1}(T) \cup D_{M_2}(T) = S'$. Let $S_1 = D_{M_1}(T)$ and $S_2 = D_{M_2}(T)$. By Lemma 7.3, there exists a set $T' \subseteq S_1$ independent in M_1 such that $w_1(T') \geqslant g(S_1)$. As noted above, the greedy algorithm that considers the element of S_1 in the order $<_i$ (decreasing weights) selects exactly the elements in T. It follows by the optimality of the greedy algorithm for matroids that T is a maximum weight independent set in S_1 for M_1 with weight function w_1, which in turn implies $w_1(T) \geqslant g(S_1)$. In the same way, we also have $w_2(T) \geqslant g(S_2)$. By definition, for any $e \in S'$, we have $w(e) = w_1(e) + w_2(e) - g(e)$. Together, we have $w(T) = w_1(T) + w_2(T) - g(T) \geqslant g(S_1) + g(S_2) - g(T)$. As elements from T are in both S_1 and S_2, and all other elements are in at least one of both sets, we have $g(S_1) + g(S_2) \geqslant g(S') + g(T)$, and thus $w(T) \geqslant g(S')$. □

7.4 Making the Algorithm Memory Efficient

We now modify Algorithm 2 to only select elements with a significant gain, parametrized by $\alpha > 1$, and delete elements if we have too many in memory, parametrized by a real number y. If α is close enough to 1 and y is large enough, then Algorithm 3 is very close to Algorithm 2, and allows for a similar analysis. This method is very similar to the one used in [60] and [32], but our analysis is quite different.

More precisely, we take an element e only if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ instead of $w(e) > w_1^*(e) + w_2^*(e)$, and we delete elements if the ratio between two g weights becomes larger than y ($\frac{g(e)}{g(e')} > y$). For technical purposes, we also need to keep independent sets T_1 and T_2 which maximize the weight functions w_1 and w_2 respectively. If an element with small g weight is in T_1 or T_2, we do not delete it, as this would modify the w_i-weights and selection of coming elements. We show that this algorithm is a semi-streaming algorithm with an approximation guarantee of $(2 + \varepsilon)$ for an appropriate selection of the parameters (see Lemma 7.7 for the space requirement and Theorem 7.8 for the approximation guarantee).

Lemma 7.6 *Let S be the subset generated by Algorithm 3 with $\alpha \geqslant 1$ and $y = \infty$. Then $w(S^*) \leqslant 2\alpha g(S)$.*

Proof We define $w_\alpha : E \to \mathbb{R}$ by $w_\alpha(e) = w(e)$ if $e \in S$ and $w_\alpha(e) = \frac{w(e)}{\alpha}$ otherwise. By construction, Algorithm 3 and Algorithm 2 give the same set S, and the same weight function g for this modified weight function. By Lemma 7.4, $w_\alpha(S^*) \leqslant 2g(S)$. On the other hand, $w(S^*) \leqslant \alpha w_\alpha(S^*)$. □

Lemma 7.7 *Let S be the subset generated by Algorithm 3 with $\alpha = 1 + \varepsilon$ and $y = \frac{\min(r_1,r_2)}{\varepsilon^2}$ and S^* be a maximum weight independent set, where r_1 and r_2 are the ranks of M_1 and M_2 respectively. Then $w(S^*) \leqslant 2(1 + 2\varepsilon + o(\varepsilon))g(S)$. Furthermore, at any point of time, the size of S is at most $r_1 + r_2 + \min(r_1, r_2)\log_\alpha(\frac{\alpha y}{\varepsilon})$.*

Proof We first prove that the generated set S satisfies $w(S^*) \leqslant 2(1 + 2\varepsilon + o(\varepsilon))g(S)$ and we then verify the space requirement of the algorithm, i.e., that it is a semi-streaming algorithm.

Let us call S' the set of elements selected by Algorithm 3, including the elements deleted later. By Lemma 7.6, we have $2\alpha g(S') \geqslant w(S^*)$, so if we prove that $g(S') - g(S) \leqslant \alpha\varepsilon g(S) = (\varepsilon + o(\varepsilon))g(S)$, we are done. We set $i \in \{1, 2\}$ to be the index of the matroid with smaller rank.

In our analysis, it will be convenient to think that the algorithm maintains the maximum weight independent set T_i of M_i throughout the stream. At the arrival of an element e that is added to S, we have that the set T_i is updated as follows. If $T_i \cup \{e\} \in I_i$ then e is simply added to T_i. Otherwise, before updating T_i, there is an element $e^* \in T_i$ such that $w_i(e^*) = w_i^*(e)$ and $T_i \setminus \{e^*\} \cup \{e\}$ is maximum weight independent set in M_i with respect to w_i. Thus we can speak of elements which are *replaced* be another element in T_i. By construction, if e replaces f in T_i, then $w_i(e) > \alpha w_i(f)$.

We can now divide the elements of S' into stacks in the following way: If e replaces an element f in T_i, then we add e on top of the stack containing f, otherwise we create a new stack containing only e. At the end of the stream, each element $e \in T_i$ is in a different stack, and each stack contains exactly one element of T_i, so let us call S'_e the stack containing e whenever $e \in T_i$. We define S_e to be the restriction of S'_e to S. In particular, each element from S' is in exactly one S'_e stack, and each element from S is in exactly one S_e stack. For each stack S'_e, we set $e_{del}(S'_e)$ to be the highest weight element of S'_e which was removed from S. By construction, $g(S'_e) - g(S_e) \leqslant w_i(e_{del}(S'_e))$. On the other hand, $w_i(f) < \frac{1+\varepsilon}{\varepsilon} g(f)$ for any element $f \in S'$ (otherwise we would not have selected it), so $g(S'_e) - g(S_e) < \frac{1+\varepsilon}{\varepsilon} g(e_{del}(S'_e))$. As $e_{del}(S'_e)$ was removed from S, we have $g(e_{del}(S'_e)) < \frac{g_{max}}{y}$ where $g_{max} = \max_{e \in S} g(e)$. As there are exactly r_i stacks, we get $g(S') - g(S) < r_i \frac{g_{max}\varepsilon^2(1+\varepsilon)}{r_i \varepsilon} = \varepsilon(1+\varepsilon)g_{max} \leqslant (\varepsilon + o(\varepsilon))g(S)$.

We now have to prove that the algorithm fits the semi-streaming criteria. In fact, the size of S never exceeds $r_1 + r_2 + r_i \log_\alpha(\frac{\alpha y}{\varepsilon})$. By the pigeonhole principle, if S has at least $r_i \log_\alpha(\frac{\alpha y}{\varepsilon})$ elements, then there is at least one stack S_e which has at least $\log_\alpha(\frac{\alpha y}{\varepsilon})$ elements. By construction, the w_i weight increases by a factor of at least α each time we add an element on the same stack, so the w_i weight difference between the lowest and highest element on the biggest stack would be at least $\frac{\alpha y}{\varepsilon}$. As $w_i(f) < \frac{1+\varepsilon}{\varepsilon} g(f)$, the g weight difference would be at least y, and we would remove the lowest element, unless it was in T_1 or T_2.

Theorem 7.8 *Let S be the subset generated by running Algorithm 3 with $\alpha = 1 + \varepsilon$ and $y = \frac{\min(r_1, r_2)}{\varepsilon^2}$. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 such that $w(T) \geqslant g(S)$. Furthermore, T is a $2(1 + 2\varepsilon + o(\varepsilon))$-approximation for the intersection of two matroids.*

Proof Let S^* be a maximum weight independent set. By Lemma 7.7, we have $2(1 + 2\varepsilon + o(\varepsilon)g(S) \geqslant w(S^*)$. Let S' be the set of elements selected by Algorithm 3, including the elements deleted later. As long as we do not delete elements from T_1 or T_2, Algorithm 2 restricted to S' will select the same elements, with the same weights, so we can consider S' to be generated by Algorithm 2. Since $S \subseteq S'$, we now observe that by Lemma 7.5, we can find an independent set $T \subseteq S$ such that $w(T) \geqslant g(S)$. □

Remark 7.9 *It is easy to construct examples where the set S only contains a 2α-approximation (for an example, see Figure 7.2 involving a bipartite graph), so our analysis is tight up to ε.*

Remark 7.10 *The techniques of this section can also be used in the case when the ranks of the matroids are unknown. Specifically, the algorithm can maintain the stacks created in the proof of Theorem 7.7 and allow for an error $\varepsilon/2$ in the first two stacks created, an error of $\varepsilon/4$ in the next 4 stacks, and in general an error of $\varepsilon/2^i$ in the next 2^i stacks by having a y value specific to each stack. The idea of constructing*

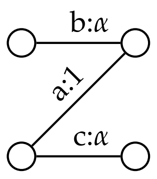

Figure 7.2: Consider the example on a bipartite graph where edges arrive in the order e_a, e_b, e_c. It is easy to see that the set S formed by Algorithm 3 contains only the edge e_a of weight 1 whereas the optimal matching consists of taking edges e_b, e_c of combined weight 2α.

such a geometric sequence is to have a total error of at most ε. We explain this in detail in Appendix B.1.

Algorithm 3 Semi-streaming adaptation of Algorithm 2

```
Input: A stream of the elements and 2 matroids (which we call $M_1, M_2$) on
  the same ground set $E$, a real number $\alpha > 1$ and a real number $y$.
Output: A set $X \subseteq E$ that is independent in both matroids.
  Whenever we write an assignment of a variable with subscript $i$, it means
  we do it for $i = 1, 2$.
  $S \leftarrow \emptyset$
  for element $e$ in the stream do
    calculate $w_i^*(e) = \max\Big(\{0\} \cup \{\theta : e \in \text{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\Big)$.
    if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ then
      $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$
      $S \leftarrow S \cup \{e\}$
      $w_i(e) \leftarrow g(e) + w_i^*(e)$
      Let $T_i$ be a maximum weight independent set of $M_i$ with respect to $w_i$.

      Let $g_{max} = \max_{e \in S} g(e)$
      Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin T_1 \cup T_2$
      from S.
    end if
  end for
  return a maximum weight set $T \subseteq S$ that is independent in $M_1$ and $M_2$
```

Chapter 8

Extension to Submodular Functions

In this chapter, we consider the problem of submodular matroid intersection in the semi-streaming model. We refer the reader to Section 2.3 to recall the definition of matroids and Section 2.2 for the definition of submodular functions. In this problem, we are given an oracle access to two matroids $M_1 = (E, I_1), M_2 = (E, I_2)$ on a common ground set E and an oracle access to non-negative submodular function $f : 2^E \to \mathbb{R}_{\geqslant 0}$ on the powerset of the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $f(X)$ is maximized.

Our Algorithm 4 is a straightforward generalization of Algorithm 2 and Algorithm 1 of [49]. Since, the weight of an element e now depends on the underlying set that it would be added to, we (naturally) define the weight of e to be the additional value e provides after adding it to set S, i.e. $w(e) = f(e \mid S)$. If e provides S a good enough value, i.e, $f(e \mid S) \geqslant \alpha(w_1^*(e) + w_2^*(e))$, we add it to set S but now with a probability q. This probability q is the most important difference between Algorithm 3 and Algorithm 4. This is a trick that we borrow from the Algorithm 1 of [49] which is useful when f is non-monotone because of the following Lemma 2.2 from [11].

Lemma 8.1 (Lemma 2.2 in [11]) *Let $h : 2^E \to \mathbb{R}_{\geqslant 0}$ be a non-negative submodular function, and let S be a a random subset of E containing every element of M with probability at most q (not necessarily independently), then $E[h(S)] \geqslant (1-q)h(\varnothing)$.*

In our proof, we can relate the weight of the set that we pick to the value $f(S^* \cup S_f)$, where S_f denotes the elements in the stack when the algorithm stops and S^* denotes the set of optimum elements. If the function f is monotone, this is sufficient as $f(S^* \cup S_f) \geqslant f(S^*)$. If, however, function f is non-monotone, one can use Lemma 8.1 with the function $h(T) = f(T \cup S^*)$. This enables us to conclude that $\mathbb{E}[f(S^* \cup S_f)] \geqslant (1-q)f(S^*)$.

Algorithm 4 Extension of Algorithm 3 to submodular functions

```
Input: A stream of the elements and 2 matroids (which we call $M_1, M_2$) on
  the same ground set $E$, a submodular function $f : 2^E \mapsto \mathbb{R}$, a real number
  $\alpha \geqslant 1$, a real number $q$ such that $0 \leqslant q \leqslant 1$ and a real number $y$.
Output: A set $X \subseteq E$ that is independent in both matroids.
  Whenever we write an assignment of a variable with subscript $i$, it means
  we do it for $i = 1, 2$.
  $S \leftarrow \emptyset$
  for element $e$ in the stream do
    calculate $w_i^*(e) = \max\Big(\{0\} \cup \{\theta : e \in \text{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\Big)$.
    if $f(e \mid S) > \alpha(w_1^*(e) + w_2^*(e))$ then
      with probability $1 - q$, continue;  {//skip $e$ with probability $1 - q$.}
      $g(e) \leftarrow f(e \mid S) - w_1^*(e) - w_2^*(e)$
      $S \leftarrow S \cup \{e\}$
      $w_i(e) \leftarrow g(e) + w_i^*(e)$
      Let $T_i$ be a maximum weight independent set of $M_i$ with respect to $w_i$.

      Let $g_{max} = \max_{e \in S} g(e)$
      Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin T_1 \cup T_2$
      from S.
    end if
  end for
  return a maximum weight set $T \subseteq S$ that is independent in $M_1$ and $M_2$
```

8.1 Analysis of Algorithm 4

We extend the analysis of Section 7.4 by using ideas from [49] to analyze our algorithm. Before going into the technical details, we give a brief overview of our analysis. For the sake of intuition, we assume that the Algorithm 4 does not delete elements and also does not skip elements with probability $1 - q$. Then, due to the fact that the weight of an element e is the additional value it provides to the current set S, one can relate the weight of the independent set picked to the weight of the optimal solution given by the set S_f i.e., $f(S^* \mid S_f)$ by basically using the analysis of the previous chapter. However, this is not enough as the weight of the optimal solution is $f(S^*)$. But, we can still relate the gain of S_f to $f(S_f)$ similar to [49], which helps us relate $f(S^* \cup S_f)$ to the weight of our solution. In order to extend it to the case when elements are skipped with probability $1 - q$, we show the above to hold in expectation similar to [49], which is helpful for dealing with non-monotone functions because of Lemma 8.1. Finally, we remark that one can use an analysis similar to Section 7.4, to show that the effect of deleting elements does not affect the weight of the solution by a lot.

Let S_f denote the set S generated when the algorithm stops and S'_f denote the union of S_f and the elements that were deleted by the algorithm. For the sake of the analysis, we define the weight function $w : E \to \mathbb{R}$ of an element e to be the additional value it provided to the set S when it appeared in the stream, i.e., $w(e) = f(e \mid S)$. Like before, we extend the definition of weight functions w_1 and w_2 for an element e that is not added to S as $w_i(e) = w_i^*(e)$ for $i \in \{1, 2\}$. We note here that all the functions defined above are random variables which depend on the internal randomness of the algorithm. Unless we explicitly mention it, we generally talk about statements with respect to any fixed realization of the internal random choices of the algorithm.

In our analysis, we will prove properties about our algorithm that are already proven for Algorithms 2 and 3 in the previous chapter. Our proof strategy will be simply running Algorithm 2 or 3 with the appropriate weight function which will mimick running our original algorithm. Hence, we will prove these statements in a black-box fashion. A weight function that we will use repeatedly in our proofs is $w' : E \to \mathbb{R}_{\geqslant 0}$ where $w'(e) = w(e)$ if $e \in S'_f$, otherwise $w'(e) = 0$. This basically has the effect of discarding elements not in S'_f i.e, elements that were never picked by the algorithm either because they did not provide a good enough value or because they did but were still skipped.

Lemma 8.2 *Consider the set S'_f which is the union of S_f generated by Algorithm 4 and the elements it deletes. Then a maximum weight independent set in M_i for $i \in \{1, 2\}$ over the whole ground set E can be selected to be a subset of S'_f, i.e. $T_i \subseteq S'_f$ and satisfies $w_i(T_i) = g(S'_f)$.*

Proof Consider running Algorithm 2 with weight function w'. Notice that doing this generates a stack containing exactly the elements in the set S'_f and exactly the same functions w_1, w_2 and g. Now by applying Lemma 7.3, we get our result. $\square$

We prove the following lemma similar to [49] which relates the gain of elements in S'_f to the weight of the optimal solution given the set S'_f i.e, $f(S^* \mid S'_f)$. Notice that the lemma below holds only in expectation for $q \neq 1$.

Lemma 8.3 *Denote the set S'_f, which is the union of S_f generated by the Algorithm 4 with $q \in \{1/(2\alpha + 1), 1\}$ and the elements it deletes. Then, $\mathbb{E}[f(S^* \mid S'_f)] \leqslant 2\alpha\mathbb{E}[g(S'_f)]$.*

Proof We first prove the lemma for $q = 1$ as the proof is easier than that for $q = 1/(2\alpha + 1)$. Consider running Algorithm 2 with the weight function $w'' : E \to \mathbb{R}_{\geqslant 0}$ which is defined as follows. If $e \in S'_f$, then $w''(e) = w(e)$, else $w''(e) = w(e)/\alpha$. Notice that doing this generates a stack containing exactly the elements in the set S'_f and exactly the same functions w_1, w_2 and g. Now by applying Lemma 7.4, we get that $w(S^*) \leqslant 2\alpha g(S'_f)$. By submodularity, we get $f(S^* \mid S'_f) \leqslant 2\alpha g(S'_f)$.

Now, we prove the lemma for $q = 1/(2\alpha + 1)$. We first define $\lambda : E \to \mathbb{R}$ for an element $e \in E$ as $\lambda(e) = f(e \mid S'_f)$. Notice that, by submodularity of f and definition of λ, we have $f(S^* \mid S'_f) \leqslant \lambda(S^*)$. Hence, it suffices to prove $\mathbb{E}[\lambda(S^*)] \leqslant 2\alpha\mathbb{E}[g(S'_f)]$. We prove this below.

Let the event that the element $e \in E$ does not give us a good enough value i.e, it satisfies $\alpha(w_1^*(e) + w_2^*(e)) \geqslant w(e)$ be R_e. We have two cases to consider now.

1. The first is when R_e is true. Then, for any fixed choice of randomness of the algorithm for which R_e is true, we argue as follows. By definition, $w_i(e) = w_i^*(e)$. Hence, $\alpha(w_1(e) + w_2(e)) \geqslant w(e)$. Also, $w(e) = f(e \mid S)$ where S is the stack when e appeared in the stream. As $S \subseteq S'_f$, by submodularity and definition of λ, we get that $w(e) \geqslant \lambda(e)$. Hence, we also get that $\alpha\mathbb{E}[w_1(e) + w_2(e)|R_e] \geqslant \mathbb{E}[\lambda(e)|R_e]$.

2. The second is when R_e is false. Then, for any fixed choice of randomness of the algorithm for which R_e is false, we argue as follows. The element e is picked with probability q given the set S at the time e appeared in the stream. If we pick e, then $w_1(e) + w_2(e) = g(e) + w_1^*(e) + g(e) + w_2^*(e) = 2w(e) - w_1^*(e) - w_2^*(e)$. If we do not pick e, then $w_1(e) + w_2(e) = w_1^*(e) + w_2^*(e)$. Hence, the expected value of $w_1(e) + w_2(e)$ satisfies,

   $$\mathbb{E}[w_1(e) + w_2(e)|\neg R_e, S] = 2qw(e) + (1 - 2q)(w_1^*(e) + w_2^*(e)) \geqslant 2qw(e).$$

   The last inequality follows as we have $q = 1/(2\alpha + 1) \leqslant 1/2$. By the choice of q and submodularity, we get that $\alpha\mathbb{E}[w_1(e) + w_2(e)|\neg R_e, S] \geqslant 2q\alpha w(e) = (1 - q)w(e) \geqslant (1 - q)\lambda(e)$. By law of total expectation and conditioned on R_e not taking place we get, $\alpha\mathbb{E}[w_1(e) + w_2(e)|\neg R_e] \geqslant \mathbb{E}[\lambda(e)|\neg R_e]$.

Finally by the law of total expectation and the points 1 and 2, we obtain that $\alpha\mathbb{E}[w_1(e) + w_2(e)] \geqslant \mathbb{E}[\lambda(e)]$ holds for any element $e \in E$. Applying this to the elements of S^*, we get that $\alpha\mathbb{E}[w_1(S^*) + w_2(S^*)] \geqslant \mathbb{E}[\lambda(S^*)]$. On the other hand, by Lemma 8.2, we have $w_i(T_i) \geqslant w_i(S^*)$ (since T_i is a max weight independent set in M_i with respect to w_i) and $w_i(T_i) = g(S'_f)$, thus $g(S'_f) \geqslant w_i(S^*)$ for $i = 1, 2$. Hence, we get that $\mathbb{E}[\lambda(S^*)] \leqslant 2\alpha\mathbb{E}[g(S'_f)]$. $\square$

Since we would like to relate the gain of elements in S'_f to the optimal solution, we now bound the value of $f(S'_f)$ in terms of the gain, similar to [49].

Lemma 8.4 *Consider the set S'_f, which is the union of S_f generated by Algorithm 4 and the elements it deletes. Then, $g(S'_f) \geqslant (1 - 1/\alpha)f(S'_f)$.*

Proof By definition, any element $e \in S'_f$, satisfies $w(e) \geqslant \alpha(w_1^*(e) + w_2^*(e))$. Hence, $g(e) \geqslant w(e) - w(e)/\alpha$. Summing over all elements in S'_f, we get $g(S'_f) \geqslant (1 - 1/\alpha)w(S'_f) \geqslant f(S'_f)$ where the last inequality (it is not an equality as S'_f also contains deleted elements) follows by definition of w and submodularity of f. $\square$

Our algorithm only has the set S_f and not S'_f which also includes the deleted elements. Hence, in our next lemma, we prove that the gain of elements in these two sets is roughly the same.

Lemma 8.5 *Consider the set S'_f which is the union of S_f generated by running Algorithm 4 with $\alpha > 1$, $y = \min(r_1, r_2)/\delta^2$ for any δ, such that $0 < \delta \leqslant \alpha - 1$ and the elements it deletes. Here, r_i is the rank of M_i for $i \in \{1, 2\}$. Then, $g(S'_f) - g(S_f) \leqslant \delta\alpha g(S_f)$. Moreover, at any point during the execution, S contains at most $r_1 + r_2 + \min(r_1, r_2)\log_\alpha(\frac{\alpha y}{\alpha - 1})$ elements.*

Proof Consider running Algorithm 3 with weight function w'. Notice that doing this generates a stack containing exactly the elements as in the set S_f, exactly the same set of deleted elements and exactly the same functions w_1, w_2 and g. Moreover, this generates the exact same stacks as the Algorithm 4 at every point of execution. Now by the proof of Lemma 7.7, we get our result. $\square$

Lastly, we prove that there exists a set T that is independent in both matroids and has a weight at least the gain of the elements in S_f.

Lemma 8.6 *Let S_f be the subset generated by Algorithm 4. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 such that $w(T) \geqslant g(S_f)$.*

Proof Consider running Algorithm 3 with weight function w'. Recall that for any element $e \in S'_f$, $w'(e) = w(e)$ or $w'(e) = 0$. Notice that doing this generates a stack containing exactly the elements as in the set S_f and exactly the same functions w_1, w_2 and g. The result follows by Theorem 7.8. $\square$

Now, we have all the lemmas to prove our main theorem which we state below.

Theorem 8.7 *The subset S_f generated by Algorithm 4 with $\alpha > 1$, $q \in \{1/(2\alpha + 1), 1\}$ and $y = \min(r_1, r_2)/\delta^2$ for any δ, such that $0 < \delta \leqslant \alpha - 1$ contains a $(4\alpha^2 - 1)/(2\alpha - 2) + O(\delta)$ approximation in expectation for the intersection of two matroids with respect to a non-monotone submodular function f. This is optimized by taking $\alpha = 1 + \sqrt{3}/2$, resulting in an approximation ratio of $4 + 2\sqrt{3} + O(\delta) \sim 7.464$. Moreover, the same algorithm run with $q = 1$ and*

$y = \min(r_1, r_2)/\delta^2$ is $(2\alpha + \alpha/(\alpha-1)) + O(\delta)$ approximate if f is monotone. This is optimized by taking $\alpha = 1 + 1/\sqrt{2}$, which yields a $3 + 2\sqrt{2} + O(\delta) \sim 5.828$ approximation.

Proof By Lemmas 8.3 and 8.4, we have that $2\alpha\mathbb{E}[g(S'_f)] \geqslant \mathbb{E}[f(S^* \mid S'_f)]$ and $g(S'_f)(\alpha/(\alpha-1)) \geqslant f(S'_f)$. Combining them, we get,

$$(2\alpha + \alpha/(\alpha-1))\mathbb{E}[g(S'_f)] \geqslant \mathbb{E}[f(S'_f) + f(S^* \mid S'_f)] = \mathbb{E}[f(S^* \cup S'_f)].$$

By Lemma 8.5, we also get that $g(S'_f) - g(S_f) \leqslant \delta\alpha g(S_f)$. This gives us that

$$(2\alpha + \alpha/(\alpha-1))(1+\delta\alpha)\mathbb{E}[g(S_f)] \geqslant \mathbb{E}[f(S^* \cup S'_f)].$$

Now, by Lemma 8.6, there exists a subset $T \subseteq S_f$ independent in M_1 and M_2 such that $w(T) \geqslant g(S_f)$. By definition of w and the submodularity of f, we get that $f(T) \geqslant w(T)$. This in turn implies, $f(T) \geqslant g(S_f)$. This gives us that

$$(2\alpha + \alpha/(\alpha-1))(1+\delta\alpha)\mathbb{E}[f(T)] \geqslant \mathbb{E}[f(S^* \cup S'_f)].$$

Notice that the above inequality also holds if $q = 1$ as all the above arguments also work if $q = 1$. Hence, if f is monotone, we get $f(S^* \cup S_f) \geqslant f(S^*)$ which gives us our desired inequality by rearranging terms. However, if f is non-monotone one has to do a little more work, as we show below.

To deal with the case when f is non-monotone, we use Lemma 8.1 and take $h(T) = f(S^* \cup T)$ for any $T \subseteq E$ within the lemma statement, to get that $\mathbb{E}[f(S^* \cup S'_f)] \geqslant (1-q)f(S^*)$ as every element of E appears in S'_f with probability at most q. Putting everything together, we get that

$$(2\alpha + \alpha/(\alpha-1))(1+\delta\alpha)\mathbb{E}[f(T)] \geqslant (1-q)f(S^*).$$

Now, substituting the value of $q = 1/(2\alpha+1)$ and rearranging terms, we get the desired inequality.

Remark 8.8 *We can exactly match the approximation ratios in [49] i.e, without the extra additive factor of $O(\delta)$ by not deleting elements. Moreover, S stores at most $O(\min(r_1, r_2)\log_\alpha |E|)$ elements at any point if we assume that values of f are polynomially bounded in $|E|$, an assumption that the authors in [49] make.*

Chapter 9

More than Two Matroids

We can easily extend Algorithm 3 to the intersection of k matroids (see Algorithm 5 for details). Most results remain true, and in particular, we can have $kg(S) \geqslant (1+\varepsilon)w(S^*)$ by carefully selecting α and y. The only part which does not work is the selection of the independent set from S. Indeed, matroid kernels are very specific to two matroids. We now prove that a similar approach fails, by proving that the logical generalization of kernels to 3 matroids where one tries to define the order given by the w_i weights is wrong and that a counter-example can arise from Algorithm 5. Thus, any attempt to find a $k+\varepsilon$ approximation using our techniques must bring some fundamentally new idea. Still, we conjecture that the generated set S contains such an approximation.

Proposition 9.1 *There exists a set S and 3 matroids $(S, I_1), (S, I_2), (S, I_3)$ such that there does not exist a set $T \subseteq S$ such that $S = D_{M_1}(T) \cup D_{M_2}(T) \cup D_{M_3}(T)$ (see Lemma 7.5 for a definition of $D_{M_i}(T)$) and T is independent in M_1, M_2 and M_3 where $<_i$ is given by w_i generated by Algorithm 5 (for α sufficiently small).*

Proof We set $S = \{a, x, y, z, b\}$. These elements are given in this order to Algorithm 5. We now define I_1, I_2, I_3 in the following way. A set of 2 elements is in I_i if and only if:

-In I_1 if it is not $\{a, x\}$

-In I_2 if it is not $\{a, y\}$

-In I_3 if it is not $\{a, z\}$

A set of 3 elements is in I_i if and only if each of its subsets of 2 elements is in I_i and:

-In I_1 if it contains z

-In I_2 if it contains x

-In I_3 if it contains y

A set of 4 elements is not in I_i.

Let us verify that these constraints correspond to matroids. As the problem is symmetrical, it is sufficient to verify that M_1 is a matroid. The 3 element independent sets in M_1 are exactly $\{y,z,b\},\{x,z,b\}$, $\{x,y,z\},\{a,z,b\}$ and $\{a,y,z\}$. Now we consider $X,Y \in I_1$ with $|X| < |Y|$. We should find $e \in Y \setminus X$ such that $X \cup \{e\} \in I_1$. If $X = \emptyset$, take any element from Y. If X is a singleton, then there are two cases: either it is one of $X \subseteq \{a,x\}$, or it is not. In any case, Y contains at most one element from $\{a,x\}$. As it contains at least two elements, Y has to contain an element from $\{y,z,b\}$. In the first case, we can add any of these to X to get an independent set. In the second case, $X \subseteq \{y,z,b\}$, so we can add any element to X and it will remain independent, so just pick any element from $Y \setminus X$. If X contains two elements, then Y is one of the sets from the list above. In particular, it contains z. If $z \notin X$, then we can add z to X. Otherwise, either $X \subseteq \{y,z,b\}$, in which case we can add any element, or X is $\{a,z\}$ or $\{x,z\}$. In either case, Y must contain an element from $\{y,b\}$, which we can add to X.

We now set the weights $w(a) = 1, w(x) = w(y) = w(z) = 3$ and $w(b) = 8$ and run Algorithm 5.

- Element a has weight 1, and $\{a\}$ is independent in M_1, M_2 and M_3, so we set $w_1(a) = w_2(a) = w_3(a) = g(a) = 1$ and a is added to S.
- Element x is spanned by a in M_1, and not spanned by any element in M_2 and M_3, so we get $g(x) = w(x) - w_1^*(x) - w_2^*(x) - w_3^*(x) = 3 - 1 - 0 - 0 = 2$. As $2 > 0$, we add x to S. We also set $w_1(x) = 3$ and $w_2(x) = w_3(x) = 2$.
- Element y and z are very similar to x.
- Element b is spanned in all three matroids by the elements of w_i weight at least 2. On the other hand, b is not spanned in any matroid by the elements of w_i weight strictly bigger than 2, so $w_i^*(b) = 2$ for $i = 1,2,3$, thus $g(b) = 8 - 2 - 2 - 2 = 2$ and $w_i(b) = 2 + 2 = 4$ for every i.

To recapitulate, we have $w_1(a) = 1, w_1(x) = 3, w_1(y) = w_1(z) = 2, w_1(b) = 4$ and the w_2 and w_3 weights are similar, with y respectively z being heavier.

Let us assume for contradiction that T is a solution to the problem.

T must contain b, as it is the heaviest element in every matroid.

If T contains a, then it cannot contain any of x,y,z, otherwise it would not be independent in one of the matroids, so we would have $T \subseteq \{a,b\}$. But x has to be in at least one $D_{M_i}(T)$, and the set $\{x,b\}$ is independent in every matroid, and has a bigger weight than $\{a,b\}$, so x would not be in $D_{M_i}(T)$. Thus T cannot contain a.

As the problem is symmetrical for $\{x, y, z\}$, it is sufficient to test $T = \{z, b\}, T = \{y, z, b\}$ and $T = \{x, y, z, b\}$. The last two are not in I_2, so the only remaining possibility is $T = \{z, b\}$. But then y is not in D_{M_1} or D_{M_3} because $\{z, b, y\}$ is independent in M_1 and M_3, and it is not in D_{M_2} because $w_2(y) > w_2(z) \Leftrightarrow y <_2 z$ and $\{y, b\}$ is independent in M_2. As y is not in any D_{M_i}, this concludes the proof. $\square$

Remark 9.2 *In the example of Proposition 9.1, we have $g(S) = w(a) + w(b)$, and $\{a, b\}$ is independent in all 3 matroids, so this does not contradict Conjecture 9.3.*

Conjecture 9.3 *The stack S generated by Algorithm 2 contains a k approximation for any k.*

In the case $k = 2$, this corresponds to Theorem 7.2. For any k, one can easily find examples were S does not contain more than a k approximation, but we were unable to find an example where it does not contain a k approximation.

Algorithm 5 Extension of Algorithm 3 to k matroids

```
Input: A stream of the elements and $k$ matroids (which we call $M_1, \dots, M_k$)
   on the same ground set $E$, a real number $\alpha > 1$ and a real number $y$.
Output: A set $S \subseteq E$ of "saved" elements.
   When we write an assignment of a variable with subscript $i$, it means we
   do it for $i = 1, \dots, k$.
   $S \leftarrow \emptyset$
   for element $e$ in the stream do
      calculate $w_i^*(e) = \max\Big(\{0\} \cup \{\theta : e \in \mathrm{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\Big)$.
      if $w(e) > \alpha \sum_{i=1}^k w_i^*(e)$ then
         $g(e) \leftarrow w(e) - \sum_{i=1}^k w_i^*(e)$
         $S \leftarrow S \cup \{e\}$
         $w_i(e) \leftarrow g(e) + w_i^*(e)$
         Let $T_i$ be a maximum weight independent set of $M_i$ with respect to $w_i$.

         Let $g_{max} = \max_{e \in S} g(e)$
         Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin \bigcup_{i=1}^k T_i$
         from S.
      end if
   end for
```

Part III

Submodular Santa Claus

Chapter 10

Introduction

In this part of the thesis, we study the submodular Santa Claus problem in the restricted assignment case. We note that unlike the previous parts of the thesis, we deal with the classical model of computation, i.e Random Access Machine (RAM) model, in this part of the thesis.

10.1 Literature Review

In the Santa Claus problem (sometimes referred to as Max-Min Fair Allocation) we are given a set of n players P and a set of m indivisible resources R. In its full generality, each player $i \in P$ has a utility function $f_i : 2^R \mapsto \mathbb{R}_{\geqslant 0}$, where $f_i(S)$ measures the happiness of player i if he is assigned the resource set S. The goal is to find a partition of the resources that maximizes the happiness of the least happy player. Formally, we want to find a partition $\{S_i\}_{i \in P}$ of the resources that maximizes

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

Most of the recent literature on this problem focuses on cases where f_i is a linear function for all players i. If we assume all valuation functions are linear, then the best approximation algorithm known for this problem, designed by Chakrabarty, Chuzhoy, and Khanna [14], has an approximation rate of n^ε and runs in time $n^{O(1/\varepsilon)}$ for $\varepsilon \in \Omega(\log\log(n)/\log(n))$. However, it is only known that computing a $(2-\delta)$-approximation is NP-hard [48]. Apart from this there has been significant attention paid to the so-called *restricted assignment case*. Here 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 to 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 [6] provide an $O(\log\log(m)/\log\log\log(m))$-approximation algorithm for this case. This was improved by Feige [25] to an

$O(1)$-approximation. Further progress on the constant or the running time was made since then, see e.g. [2, 18, 16, 15, 39, 3, 61].

Let us now move to the non-linear case. The problem becomes hopelessly difficult without any restrictions on the utility functions. Consider the following reduction from set packing. There are sets of resources $\{S_1, \dots, S_k\}$ 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 $S_1, \dots, S_k$ (a classical NP-hard problem) is equivalent to deciding whether the optimum of the Santa Claus problem is non-zero. In particular, obtaining any bounded approximation ratio for the Santa Claus problem in this case is NP-hard.

Two naturally arising properties of utility functions are monotonicity and submodularity, see for example the related submodular welfare problem [47, 64] where the goal is to maximize $\sum_i f_i(S_i)$. Recall that 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 part) is that the value on the empty set is zero, i.e., $f(\emptyset) = 0$. Goemans, Harvey, Iwata, and Mirrokni [33] first considered the Santa Claus problem with monotone submodular utility functions as an application of their fundamental result on submodular functions. Together with the algorithm used in [14] it implies an $O(n^{1/2+\varepsilon})$-approximation in time $O(n^{1/\varepsilon})$.

In this part of the thesis, we investigate the restricted assignment case with a monotone submodular utility function. 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.

10.2 Overview of Results and Techniques

Our main result is an approximation algorithm for the submodular Santa Claus problem in the restricted assignment case.

Theorem 10.1 *There is a randomized polynomial time $O(\log\log(n))$-approximation algorithm for the restricted assignment case with a monotone submodular utility function.*

Our way to this result is organised as follows. In Chapter 11, we first reduce our problem to a hypergraph matching problem (see next paragraph for a formal definition). We then solve this problem using Lovász Local Lemma (LLL) in Chapter 12. In [6] the authors also reduce the Santa Claus problem 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 [6]. We will elaborate later in this section why the previous techniques do not easily extend to the weighted variant.

The hypergraph matching problem. After the reduction in Chapter 11 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 $C \in \mathbb{C}$ 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 need 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 relaxation and α essentially translates to the approximation rate. This result generalizes that of Bansal and Srividenko on hypergraph matching. 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.

[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.

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

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 the 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 configurations 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 robbed of at most $|K|/2$ of its resources, and 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.

Additional implications of non-uniform hypergraph matchings to the Santa Claus problem. We believe this hypergraph matching problem is interesting in its own right. Our last contribution is to show that finding good matchings in unweighted hypergraphs with fewer assumptions than ours would have important applications for the Santa Claus problem with linear utility functions. We recall that here, each player i has its own utility function f_i that can be any linear function. In this case, the best approximation algorithm is due to Chakrabarty, Chuzhoy, and Khanna [14] who gave a $O(n^{\varepsilon})$-approximation running in time $O(n^{1/\varepsilon})$. In particular, no sub-polynomial approximation running in polynomial time is known. Consider as before $\mathcal{H} = (P \cup R, \mathcal{C})$ a non-uniform hypergraph with unit weights ($w_{j,C} = 1$ for all j, C such that $j \in C$). Finding the smallest α (or an approximation of it) such that there exists an α-relaxed perfect matching in $\mathcal{H}$ is already a very non-trivial question to solve in polynomial time.

We show, via a reduction, that a c-approximation for this problem would yield a $O((c \log^*(n))^2)$-approximation for the Santa Claus problem with arbitrary linear utility functions. In particular, any sub-polynomial approximation for this problem would significantly improve the state-of-the-art[3]. All the details of this last result can be found in Chapter 13.

A remark on local search techniques. 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 by Asadpour, Feige, and Saberi [3] who were inspired by the work of Haxell [36] and 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 can 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.

[3]We mention that our result on relaxed matchings in Chapter 12 does not imply an $O(\log\log(n))$-approximation for this problem since we make additional assumptions on the regularity of the hypergraph or the size of hyperedges.

Chapter 11

Reduction to hypergraph matching problem

In this chapter 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 $C \in \mathbb{C}(i,T)$. The configurations $\mathbb{C}(i,T)$ 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.

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

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 [6], 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 11.1 *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 C.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

$$f(\{j\}) \geqslant \frac{T^*}{100\alpha}.$$

Resources that are not fat are called *thin*. We call a configuration $C \in \mathbb{C}(i,T)$ thin, if it contains only thin resources and denote by $\mathbb{C}_t(i,T) \subseteq \mathbb{C}(i,T)$ 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 $C \in \mathbb{C}_t(i, T^*/O(\alpha))$. For our next step towards the hypergraph problem we use a technique borrowed from Bansal and Srividenko [6]. 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 11.2 *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 $K_1 \cup \dots \cup K_k \cup Q = P$ and multisets of configurations $\mathbb{C}_h \subseteq \bigcup_{i \in K_h} \mathbb{C}_T(i, T^*/5)$, $h = 1, \dots, k$, such that*

1. *$|\mathbb{C}_h| = \ell$ for all $h = 1, \dots, k$ and*
2. *Each small resource appears in at most ℓ configurations of $\bigcup_h \mathbb{C}_h$.*
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 $j \in \Gamma_i$.*

The role of the players Q in the lemma above is that each one of them gets a fat resource for certain. The proof closely follows that in [6]. For completeness we include it in Appendix C.1.

We are now ready to define the hypergraph matching instance. The vertices of our hypergraph are the clusters $K_1, \dots, K_k$ and the thin resources. Let $\mathbb{C}_1, \dots, \mathbb{C}_k$ be the multisets of configurations as in Lemma 11.2. For each K_h and $C \in \mathbb{C}_h$ there is a hyperedge containing K_h and all resources in C. Let $\{j_1, \dots, j_\ell\} = C$ ordered arbitrarily, but consistently. Then we define the weights as normalized marginal gains of resources if they are taken in this order, that is,

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

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

Lemma 11.3 *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 the restricted submodular Santa Claus problem.*

Proof The α-relaxed perfect matching implies that cluster K_h gets some small resources C' where $C' \subseteq C$ for some $C \in \mathbb{C}_h$ 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 C.1 all others using fat resources. $\square$

The proof above is the most critical place in our work 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 w.l.o.g. since we can just rescale the weights inside each configuration. This does not hurt the property that all weights are small enough.

11.1 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 $C \in \mathbb{C}$. If this is not the case for some $C \in \mathbb{C}$, 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

$$n \cdot \frac{1}{2n} = \frac{1}{2}.$$

(Recall that we rescaled the weights so that $\sum_{j\in C} w_{j,C} = 1$).

Hence an α-relaxed perfect matching in the new hypergraph after these two operations, 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 each correspond 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 $\mathcal{S}_C = \{C_1, \dots, C_s, \dots, C_{\log(n)}\}$ 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 $C_1, \dots, C_s, \dots, C_{\log(n)}$ 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.

Lemma 11.4 *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 $i_1, \dots, i_s, \dots, i_{\log(n)}$ in $\mathcal{H}'$ corresponding to a player i in $\mathcal{H}$. These players are assigned a consistent set of configurations $C_1, \dots, C_s, \dots, C_{\log(n)}$ 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 $\mathcal{H}$. □

Chapter 12

Matchings in regular hypergraphs

In this chapter we solve the hypergraph matching problem we arrived to in the previous chapter. 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 chapters, 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

$$\left\lfloor \frac{\mathcal{S}_A^{(i)}}{\alpha} \right\rfloor$$

resources from $\mathcal{S}_A^{(i)}$.

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

Theorem 12.1 *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 12.1 together with the reduction from the previous section will prove our main result (Theorem 10.1) stated in the introduction.

12.1 Overview and notations

To prove Theorem 12.1, 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 11.2 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 11), the proof of Lemma 11.2 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 $\mathbb{C}_i$ the set of configurations that contain i. We now group the configurations in $\mathbb{C}_i$ by size: We denote by $\mathbb{C}_i^{(0)}$ the configurations of size within $[0, \ell^4)$ and for $k \geqslant 1$ we write $\mathbb{C}_i^{(k)}$ for the configurations of size within $[\ell^{k+3}, \ell^{k+4})$. Moreover, define $\mathbb{C}^{(k)} = \bigcup_i \mathbb{C}_i^{(k)}$ and $\mathbb{C}^{(\geqslant k)} = \bigcup_{h \geqslant k} \mathbb{C}^{(h)}$. Let d be the smallest number such that $\mathbb{C}^{(\geqslant d)}$ is empty. Note that $d \leqslant \log(n)/\log(\ell)$.

Now consider the following random process.

Random Experiment 12.2 *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 $C \in \mathbb{C}^{(k)}$ we have that $\mathbb{E}[|R_k \cap C|] = \mathrm{poly}(\ell)$.

The proof of Theorem 12.1 is organized as follows. In Section 12.2, we give some properties of the resource sets constructed by Random Experiment 12.2 that hold with high probability. Then in Section 12.3, 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 the Lovász Local Lemma. Finally in Section 12.4, we demonstrate how these configurations allows us to reconstruct a consistent α-relaxed perfect matching for an appropriate assignment of resources to configurations.

12.2 Properties of resource sets

In this section, we give a precise statement of the key properties that we need from Random Experiment 12.2. The first two lemmas have a straightforward proof. The last one is a generalization of an argument used by Bansal and Srividenko [6]. Since the proof is technical and tedious, we defer it to Appendix C.2 along with the proofs of the first two lemmas.

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 when applying LLL later.

Lemma 12.3 *Consider Random Experiment 12.2 with $\ell \geqslant 300.000 \log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathbb{C}^{(\geqslant k)}$ 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 atmost 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 $C' \in \mathbb{C}^{(k)}$ is restricted to the resource set R_k.

Lemma 12.4 *Consider Random Experiment 12.2 with $\ell \geqslant 300.000 \log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathbb{C}^{(\geqslant k)}$ we have*

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

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 12.5 *Consider Random Experiment 12.2 with $\ell \geqslant 300.000 \log^3(n)$. Fix $k \geqslant 0$. Conditioned on the event that the bounds in Lemma 12.3 hold for k, then with probability at least $1 - 1/n^{10}$ the following holds for all $\mathcal{F} \subseteq \mathbb{C}^{(\geqslant k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$, 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.

12.3 Selection of configurations

In this section, 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 by $\mathcal{S}_A$ the selected consistent set for group A and for ease of notation we will denote by $K_i = \mathcal{S}_A^{(i)}$ 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 12.6 *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 $h = 0, \ldots, k$. Then there exists a selection of one consistent set $\mathcal{S}_A$ for each group A such for all $k = 0, \ldots, d$, $C \in \mathcal{C}^{(k)}$ and $j = 0, \ldots, k$ 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 $R_1, \ldots, R_d$ constructed as in Random Experiment 12.2. 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

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

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 prove the lemma by performing the following random experiment and by the Lovász Local Lemma show that there is a positive probability of success.

Random Experiment 12.7 *For each group A, select one consistent set $\mathcal{S}_A$ uniformly at random. Then for each player $i \in A$ set $K_i = \mathcal{S}_A^{(i)}$.*

For all $h = 0, \ldots, d$ and $i \in P$ we define the random variable

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

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 define a set of bad events. As we will show later, if none of them occur, the properties from the premise hold. For each k, $C \in \mathcal{C}^{(k)}$, and $h \leqslant k$ let $B_C^{(h)}$ be the event that

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

There is an intuitive reason as to why we define these two different bad events. 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 12.6 since we sum this bigger deviation only a constant number of times. With this intuition in mind, we claim the following.

Claim 12.8 *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 Proposition 2.3) 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

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

We now restate Lovász Local Lemma for the convenience of the reader and use it in our setting.

Proposition ((2.4 restated) Lovász Local Lemma (LLL)) *Let $B_1, \dots, B_t$ be bad events, and let $G = (\{B_1, \dots, B_t\}, E)$ be a dependency graph for them, in which for every i, event B_i is independent of all events B_j for which $(B_i, B_j) \notin E$. Let x_i for $1 \leqslant i \leqslant t$ be such that $0 < x(B_i) < 1$ and $\mathbb{P}[B_i] \leqslant x(B_i) \prod_{(B_i,B_j)\in E}(1 - x(B_j))$. Then with positive probability no event B_i holds.*

Let $k \in \{0, \dots, d\}$, $C \in \mathcal{C}^{(k)}$ 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 $C' \in \mathbb{C}^{(h')}$ 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 Proposition 2.4 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 $0 \leqslant j \leqslant k$,

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

This proves Lemma 12.6.

Remark 12.9 *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 [55] can be applied to find a selection of configurations such that no bad events occur in randomized polynomial time.*

12.4 Assignment of resources to configurations

In this section, 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 section. We will denote as in the previous section $\mathcal{K}_i^{(k)} = \{K_i\}$ if $K_i \in \mathbb{C}_i^{(k)}$ 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 12.6 implies the following bound. For sake of brevity, the proof is deferred to Appendix C.3.

Claim 12.10 *For any* $k \geqslant 0$, *any* $0 \leqslant j \leqslant k$, *and any* $C \in \mathcal{K}^{(k)}$

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

The main technical part of this section is the following lemma, which is proved by induction.

Lemma 12.11 *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 proceeding to the proof, we first give 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 12.3). 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 12.10. Hence by choosing γ to be $\Theta(\log\log(n))$ we get that the bound in Lemma 12.11 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.

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

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

resources from R_j. We show that this implies a solution at level $j-1$ in the following way. First by Lemma 12.5, 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

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

resources and no resource of R_{j-1} is taken more than γ times. Note that we can apply Lemma 12.5 since we have by Claim 12.10 and Lemma 12.3 that

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

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. However, for better clarity we defer its proof to appendix C.3.

Claim 12.12 *For any $C\in\mathcal{K}^{(\geqslant j)}$,*

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

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 12.3. 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

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

resources which concludes the proof. $\square$

Corollary 12.13 *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 12.11 for $k = 0$ and Claim 12.10 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. $\square$

Chapter 13

Further connections between hypergraph matching and Santa Claus

In Chapter 12, we essentially prove that every regular (non-uniform) hypergraph has an α-relaxed perfect matching for some $\alpha = O(\log\log(n))$, assuming that all hyperedges contain at least α resources. This means that we give a sufficient condition for a hypergraph to have a good relaxed matching. A natural optimization problem that arises from this is the following: Given any unweighted hypergraph, which is not necessarily regular and whose hyperedges do not necessarily contain all resources, what is the minimum α such that there exists an α-relaxed perfect matching in this hypergraph?

In this chapter, we investigate the relationship between this problem and the Santa Claus problem with linear utility functions. Formally, the two problems considered are precisely the following.

Matching in general hypergraphs. Consider a (non-uniform) hypergraph $\mathcal{H} = (P \cup R, \mathcal{C})$ with unit weights, that is, $w_{j,C} = 1$ for all j, C such that $j \in C$. The problem is to find the minimum α such that $\mathcal{H}$ has an α-relaxed perfect matching (and output such a matching).

The Santa Claus problem with linear utility functions. In this case, each player i has an arbitrary linear utility function f_i. We note that there is no relationship assumed between the utility functions of different players. The goal is to assign resources to players to maximize the minimum utility among players. As mentioned in the introduction, the best approximation algorithm for this problem is an $O(n^\varepsilon)$-approximation running in time $O(n^{1/\varepsilon})$.

We show by a straightforward reduction that a c-approximation for the Santa Claus problem immediately implies a c-approximation for the matching problem.

13.1 From Matchings to Santa Claus

The idea in this reduction is to replace each player by a set of players, one for each of the t configuration containing him. These players will share together $t-1$ large new resources, but to satisfy all, one of them has to get other resources, which are the original resources in the corresponding configuration.

Players. For every vertex $v \in P$, and every hyperedge $C \in \mathbb{C}$ that v belongs to, we create a player $p_{v,C}$ in the Santa Claus instance.

Resources. For every vertex $u \in R$, create a resource r_u in the Santa Claus instance. For any vertex $v \in P$ such that it belongs to t edges in $\mathbb{C}$, create $t-1$ resources $r_{v,1}, r_{v,2}, \dots, r_{v,t-1}$.

Values. For any resource r_u for some $u \in R$ and any player $p_{v,C}$ for some $C \in \mathbb{C}$, the resource has a value $\frac{1}{|C|-1}$ if $u \in C$, otherwise it has value 0. Any resource $r_{v,i}$ for some $v \in P$ and $i \in \mathbb{N}$, has value 1 for any player $p_{v,C}$ for some $C \in \mathbb{C}$ and 0 to all other players.

It is easy to see that given an α-relaxed matching in the original instance, one can construct an α-approximate solution for the Santa Claus instance.

For the other direction, notice that for each $v \in P$, there exists a player $p_{v,C}$ for some $C \in \mathbb{C}$, such that it gets resources only of the type r_u. One can simply assign the resource $u \in R$ to the player v for any resource r_u assigned to $p_{v,C}$.

Interestingly, there is also a close connection in the opposite direction.

13.2 From Santa Claus to Matchings

Theorem 13.1 *A c-approximation algorithm to the hypergraph matching problem in general hypergraphs yields an $O((c \log^*(n))^2)$-approximation algorithm to the Santa Claus problem.*

Before giving the proof, we mention some remarks regarding the statement above.

Remark 13.2 *Notice that Theorem 13.1 implies that any sub-polynomial approximation to the matching problem would be a significant improvement of the state-of-the-art for the Santa Claus problem with arbitrary linear utility functions.*

Remark 13.3 *Since hypergraphs considered here might be non-regular and some hyperedges might contain very few resources, our result in Chapter 12 does not imply any approximation for the optimization problem considered here. Our reduction in this section makes a crucial use of small hyperedges containing only one resource.*

This shows that handling the small hyperedges is one of the core difficulties in this case.

We now give the proof of Theorem 13.1.

Proof We write $(\log)^k(n) = \underbrace{\log \cdots \log}_{\times k}(n)$ and $(\log)^0(n) = n$.

Construction. We describe how to construct a hypergraph matching instance from a Santa Claus instance in four steps by reducing to the following more and more special cases.

(1) Geometric grouping. In this step, given arbitrary v_{ij}, we reduce it to an instance such that OPT $= 1$ and for each i, j we have $v_{ij} = 2^{-k}$ for some integer k and $1/(2n) < v_{ij} \leqslant 1$. This step follows easily from guessing OPT, rounding down the sizes, and omitting all small elements in a solution.

(2) Reduction to O(log*(n)) size ranges. Next, we reduce to an instance in which there is some $k \leqslant \log^*(2n)$ for each player i such that for each resource j, $v_{ij} \in \{0, 1\}$ or $1/(\log)^k(2n) < v_{ij} \leqslant 1/(\log)^{k+1}(2n)$. We explain this step below.

Each player and resource is copied to the new instance. However, we will also add auxiliary players and resources. Let i be a player. In the optimal solution there is some $0 \leqslant k \leqslant \log^*(2n)$ such that the values of all resources j with $1/(\log)^k(2n) < v_{ij} \leqslant 1/(\log)^{k+1}(2n)$ assigned to player i sum up to at least $1/\log^*(2n)$. Hence, we create $\log^*(2n)$ auxiliary players which correspond to each k and each of which share an resource with the original player that has value 1 for both. The original player needs to get one of these resources, which means one of the auxiliary players needs to get a significant value from the resources with $1/(\log)^k(2n) < v_{ij} \leqslant 1/(\log)^{k+1}(2n)$. This reduction loses a factor of at most $\log^*(2n)$. Hence, OPT $\geqslant 1/\log^*(2n)$.

(3) Reduction to 3 sizes. We further reduce to an instance in which for each player i there is some value v_i such that for each resource j, $v_{ij} \in \{0, v_i, 1\}$.

Let i be some player who has only resources of value $v_{ij} \in \{0,1\}$ or $1/(\log)^k(2n) < v_{ij} \leqslant 1/(\log)^{k+1}(2n)$ for some integer k. There are at most $\log((\log)^k(2n)) \leqslant (\log)^{k+1}(2n)$ distinct values of the latter kind. The idea is to assign bundles of resources of value $0.5/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ to player i.

Fix a resource value s such that $1/(\log)^k(2n) < s \leqslant 1/(\log)^{k+1}(2n)$. We denote by R_s the set of resources j such that $v_{ij} = s$.

We define the integer

$$b = \left\lceil \frac{0.5}{s \log^*(2n)(\log)^{k+1}(2n)} \right\rceil$$

which is the number of resources of value s that are needed to make a bundle of total value at least $0.5/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$. We remark that if $s > 0.5/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ we have $b = 1$. However, since $s \leqslant 1/(\log)^{k+1}(2n)$, the value of a bundle never exceeds $1/(\log)^{k+1}(2n)$ in the instance of step (2).

Then we create

$$\lfloor |R_s|/b \rfloor$$

auxiliary players $i_1, i_2, \ldots$ and auxiliary resources $j_1, j_2, \ldots$ (note that we create 0 players and resources if $|R_s| < b$).

Each auxiliary player i_ℓ shares resource j_ℓ with player i. This resource has value $2/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ for player i and value 1 for player i_ℓ. Then for all resources $j \in R_s$, we set $v_{ij} = 0$ and

$$v_{i_\ell j} = \frac{1}{(\log^*(2n))^2 b}$$

for any auxiliary player i_ℓ that was created.

We see that we are now in the case where for each player i, there exists some v_i such that $v_{ij} \in \{0, v_i, 1\}$ for all resources j. We claim the following.

Claim 13.4 *In the instance created at step (3), we have that* $\mathrm{OPT} \geqslant 1/(\log^*(2n))^2$.

Proof To see this, take an assignment of resources to player that gives $1/\log^*(2n)$ value to every player in the instance obtained at the end of step (2). Define R_i to be the set of resources assigned to player i in this solution. Either R_i contains a resource of value 1 or only resources that are in a range $(1/(\log)^k(2n), 1/(\log)^{k+1}(2n)]$ for some integer k. In the first case, nothing needs to be done as the resource j of value 1 assigned to i still satisfies $v_{ij} = 1$ in the new instance. Hence we assign j to i and all auxiliary players created for player i get their auxiliary resource of value 1.

In the second case, fix a resource value s. Let $R_{i,s}$ be the set of resources assigned to i for which $v_{ij} = s$ and b defined as before. We select $\lfloor |R_{i,s}|/b \rfloor$ auxiliary players to receive b resources from $R_{i,s}$ and player i takes the corresponding auxiliary resources. The remaining auxiliary players of the corresponding value take their auxiliary resource.

Doing this, we ensure that all auxiliary players receive either a value of 1 (by taking the auxiliary resource) or $1/(\log^*(2n))^2$ by taking resources assigned to i in the instance of step (2). Moreover, we claim that i receives a total value of at least $1/(\log^*(2n))^2$. To see this, we have 3 cases depending on the value of b and $\lfloor |R_{i,s}|/b \rfloor$.

- If $b = 1$, then $\lfloor |R_{i,s}|/b \rfloor = |R_{i,s}|$. We note that the value of a bundle of b resources of size s never exceeds $1/(\log)^{k+1}(2n)$ in instance (2). Since each auxiliary resource represents a value of $2/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ to player i in instance (3), it must be that player i receives in instance (3) at least a $2/\log^*(2n)$ fraction of the value he would receive in instance (2).
- If $b > 1$ and $\lfloor |R_{i,s}|/b \rfloor > 0$. Then we have that $\lfloor |R_{i,s}|/b \rfloor \geqslant |R_{i,s}|/(2b)$. Since in this case we have $s < 0.5/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ it must be that each bundle of b resources of size s represents a total value of at most $1/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$. Since the value of auxiliary resources is twice this value and because $\lfloor |R_{i,s}|/b \rfloor \geqslant |R_{i,s}|/(2b)$ it must be that in this case player i receives in instance (3) at least the same value he would receive in instance (2).
- If $\lfloor |R_{i,s}|/b \rfloor = 0$, then player i receives 0 value from resources of this value. However, when we combine all the values s for which $\lfloor |R_{i,s}|/b \rfloor = 0$, it represents to player i in instance (2) a total value of at most

  $$0.5/\left(\log^*(2n)(\log)^{k+1}(2n)\right) \cdot (\log)^{k+1}(2n) = 0.5/\log^*(2n)$$

  since there are at most $(\log)^{k+1}(2n)$ different resource values.

Putting everything together, we see that in the first two cases, player i receives at least a $2/\log^*(2n)$ fraction of the value he would receive in instance (2) and that he looses at total value of at most $0.5/\log^*(2n)$ in the third case. Since in instance (2) we have that $\text{OPT} \geqslant 1/\log^*(2n)$ we see that in instance (3) player i receives a value at least

$$(2/\log^*(2n)) \cdot (1/\log^*(2n) - 0.5/\log^*(2n)) \geqslant 1/(\log^*(2n))^2. \qquad \square$$

Finally, we also claim that it is easy to reconstruct an approximate solution to the instance obtained at step (1) from an approximate solution to the instance at step (3).

Claim 13.5 *A c-approximate solution to the instance obtained at step (3) induces a $O((c\log^*(2n))^2)$-approximate solution to the instance obtained at step (1).*

Proof To see this, note that a c-approximate solution must give at least $1/(c(\log^*(2n))^2)$ value to every player since $\text{OPT} \geqslant 1/(\log^*(2n))^2$ (by Claim 13.4). This means that each player i either takes a resource of value 1 which also has a value 1 for him in the instance at step (1) or he must take at total value of $1/(c(\log^*(2n))^2)$ in auxiliary resources and the corresponding auxiliary players must take bundles of resources that represent a value of at least

$$0.5/\left(c\log^*(2n)(\log)^{k+1}(2n)\right)$$

for player i in the instance at step (1). We simply assign all the resources appearing in these bundles to the player i in the instance of step (1). Since the value of an auxiliary resource for player i is $2/\left(\log^*(2n)(\log)^{k+1}(2n)\right)$ it must be that player i takes at least

$$\frac{1/(c(\log^*(2n))^2)}{2/\left(\log^*(2n)(\log)^{k+1}(2n)\right)} = \frac{(\log)^{k+1}(2n)}{2c\log^*(2n)}$$

auxiliary resources. Since each auxiliary resource brings a value of

$$0.5/\left(c\log^*(2n)(\log)^{k+1}(2n)\right)$$

to player i (in the instance at step (1)) then player i receives in total a value of at least

$$\frac{1}{(2c\log^*(2n))^2}$$

in the instance of step (1). □

Before the last step, we rescale the instance appropriately to get OPT $= 1$ (we keep the property that each player i has 3 distinct sizes 0,1 and v_i).

(4) Reduction to hypergraph matching. For each player create a vertex in P and for each resource create a vertex in R. For each player add one hyperedge for each resource he values at 1 (containing i and this resource). Moreover, for every player i, add $1/v_i$ *new* vertices to P and the same number of *new* resources to R. Pair these $1/v_i$ new vertices in P and R together (one from R and one from P) and for each pair add a hyperedge containing these two vertices in the pair. Add another hyperedge for i containing i and all corresponding $1/v_i$ new vertices in R. Finally, for each new vertex in P and each resource that i values at v_i, add a hyperedge containing them. See Figure 13.1 for an illustration: New resources and players are marked as squares and hyperedges containing only 2 vertices are marked as simple edges.

We claim that there exists a 1-relaxed perfect matching in this instance. Since OPT $= 1$ there is an assignment of resources to players such that every player gets a value of 1. If player i takes one resource of value 1, give to player i the corresponding hyperedge and the resource in it in the hypergraph. All the new players get the new resource they are paired to. If player i takes $1/v_i$ resources of value v_i, give to player i in the hypergraph all the $1/v_i$ new resources contained in the new hyperedge. Then we give to each new player

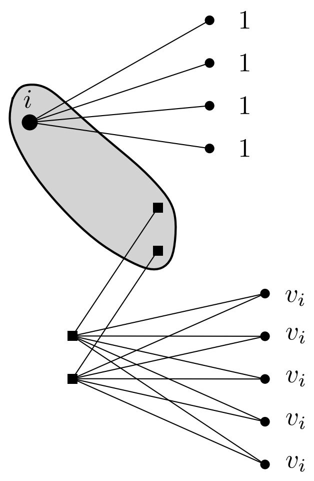

Figure 13.1: An example of the reduction to hypergraph matching for player i with $v_i = 1/2$.

the hyperedge (and the resource in it) corresponding to a resource that is assigned to i in instance from step (3). This is indeed a 1-relaxed perfect matching.

Correctness. In the reduction we arrive at in step (3), we prove that a c-approximate solution can be used to easily reconstruct a $O((c \log^*(2n))^2)$-approximate solution to the original instance (in Claim 13.5). It remains to show that a c-relaxed perfect matching in the instance (4) induces a c-approximate solution to step (3). To see this, note that a c-relaxed perfect matching in the instance (4) either gives to player i the resource in one hyperedge corresponding to a resource of value 1 to player i in instance (3). In that case we assign this resource to player i in instance (3). Or it gives at least $1/(cv_i)$ new resources to player i. In this case, it must be that each new player paired to one of these resources takes one resource of value v_i in instance (3). We give these resources to i in instance (3). In this case i receives a total value of $v_i/(cv_i) = 1/c$ which ends the proof.

We finish by remarking that the size of our construction is indeed polynomial in the size of the original instance. This is clear for step (1). In step (2), only $O(\log^*(n))$ new players and items are created for each player in the original instance. In step (3), for each player i and each resource size v_{ij}, at most a polynomial number of resources and players are created. As for the last step, $O(1/v_i)$ new resources and players are created for each player i which is also polynomial since $v_i = \Omega(1/n)$. The number of hyperedges in the hypergraph is also clearly polynomial in the number of vertices in our construction. □

Chapter 14

Conclusions

Submodular functions are an important class of functions that have been influential in theoretical computer science. In this thesis, we have studied fundamental combinatorial optimization problems with submodular objective functions. In the first two parts of the thesis we focussed mainly on the streaming model of computation whereas in the third part we focussed on approximation algorithms in the classical model of computation.

In Part I of our thesis, we developed robust streaming algorithms for the problems of cardinality constrainted submodular maximization and unweighted maximum matching. We introduced a semi-random model called adversarial-injections with the motivation of eliminating algorithms that overfit to random-order streams while still being easier than adversarial-order streams. We studied two classical problems in combinatorial optimization in this model.

For unweighted matching, we could beat $1/2$ in the streaming setting whereas we observed from [31] that we could not beat $1/2$ in the online setting. This also makes our model non-trivial as there is a separation between the online and streaming setting.

For monotone submodular maximization with cardinality constraint k, we obtained a 0.55-approximate streaming algorithm with a memory footprint that is only a function of k. The obvious open question is whether one can design a $(1 - 1/e)$-approximation algorithm which stores number of elements that is independent of n. Does our algorithm have an approximation ratio of $1 - 1/e$? We observed that the algorithm in [59] is a $1/2 + \varepsilon$ approximation for a very small $\varepsilon > 0$. The algorithm stores $\text{poly}(k)$ elements. Can one design an algorithm that stores only $\text{poly}(k)$ elements and beats $1/2$ by a significant constant or, even better, gets $1 - 1/e$?

In Part II, we studied the matroid intersection problem in the streaming setting. Our main result was a $(2 + \varepsilon)$-approximation semi-streaming algorithm for the weighted matroid intersection problem. Even though our algorithm is based on the local-ratio technique, the analysis is a departure from previous works as we use the concept of kernels from [29] for the analysis of our algorithm. Moreover, we were able to extend our results to submodular functions using ideas from [49]. Lastly, we showed that our algorithm could be generalized in a natural way to work for the intersection of k matroids. However, we proved that the logical generalization of matroid kernels to 3 matroids is wrong and specifically gave a counter-example. Hence, we need new techniques for the analysis. We, however conjecture that the natural generalization of the algorithm contains a $(k+\varepsilon)$-approximation.

Finally, in Part III, we investigated the submodular Santa Claus problem in the restricted assignment case and gave a $O(\log\log(n))$-approximation for this problem. This represents a significant generalization of the results for the linear case. The submodularity of the utility function introduced new obstacles compared to the linear case. These difficulties are captured by the fact that we need to solve a new matching problem in non-uniform hypergraphs that generalizes the case of uniform hypergraphs which has been already studied in the context of the resttricted Santa Claus problem with a linear utility function. Under the assumption that the hypergraph is regular and all edges are sufficiently large, we proved that there is always a α-relaxed perfect matching for $\alpha = O(\log\log(n))$. This result generalizes the work of Bansal and Srividenko [6]. It remains an intriguing question whether one can get $\alpha = O(1)$ as it is possible in the uniform case. One idea (similar to Feige's proof in the uniform case [25]) would be to view our proof as a sparsification theorem and to apply it several times. Given a set of hyperedges such that every player has ℓ hyperedges and every resource appears in no more than ℓ hyperedges, one would like to select polylog(ℓ) hyperedges for each player such that all resources appear in no more than polylog(ℓ) of the selected hyperedges. It is not difficult to see than our proof actually achieves this when $\ell = \text{polylog}(n)$. However, repeating this after the first step seems to require new ideas since our bound on the number of times each resource is taken is $\Omega\left(\frac{d+\ell}{\ell}\log(\ell)\right)$ where ℓ is the current sparsity and d the number of configuration sizes. For the first step, we conveniently have that $d = O(\log(n)) = O(\ell)$ but after the first sparsification, it may not be true.

We also provided a reduction from the Santa Claus problem with arbitrary linear utility functions to the hypergraph matching problem in general hypergraphs. This shows that finding the smallest α such that a hypergraph has an α-relaxed perfect matching (or approximating it) is a very non-trivial problem (even within a sub-polynomial factor). Another interesting question is to improve the $O(\log^*(n))^2$ factor in the reduction to a constant.

Appendix A

Deferred Proofs from Part I

A.1 Removing the assumption that $|M^*|$ is known

We start by briefly recalling the algorithm Match. We refer the reader to Section 4.1 for complete details.

Match runs two algorithms in parallel and outputs the better solution. The first algorithm runs greedy whereas the second algorithm runs greedy up to a certain point (Phase 1), and then starts collecting 3-augmenting paths.

Now we recall some definitions introduced in Chapter 4 that we will use here. Let M^* denote the maximum matching and M_1 the variable that is updated by the first algorithm.

Our algorithm Match assumed that it knew $|M^*|$. This is because the second algorithm that is run in Match needs to know $|M^*|$ as it starts collecting 3-augmenting paths when it has collected at least $|M^*|(1/2 - \varepsilon)$ edges. Until this point in the stream, it has collected exactly $|M_1|$ edges. By definition, $|M_1|$ is also a lower bound on $|M^*|$. Hence at any point, we will run multiple copies of the second algorithm initialized with a guess for $|M^*|$ based on what the value of $|M_1|$ is at that point. Thus, for any fixed $\delta > 0$, we can guess the value of $|M^*|$ up to a factor of $(1+\delta)$ by running the algorithm in parallel for all powers i of $1+\delta$, that satisfy $|M_1|/(1+\delta) \leqslant (1+\delta)^i \leqslant 4|M_1|/(1-2\varepsilon)$. Notice that some copies of our algorithm may stop after some time as their $|M^*|$ estimate is no longer valid, others will continue and new ones with $|M^*|$ estimates that are not already running will start initialized with the matching M_1 at that point in the stream. This increases algorithm's space only by a factor of $O(\log \frac{4(1+\delta)}{1-2\varepsilon})$ and deteriorates the solution value by at most $O(\delta|M^*|)$.

A.2 Omitted proofs for submodular function maximization

We start by briefly recalling the algorithm. We refer the reader to Section 5.2 for the complete details.

The algorithm produces a tree of height at most k (the cardinality constraint) and each root to node (at height i) path corresponds to a solution (of i elements). Each node has at most $|I|$ (the set of all possible increments) children. At the beginning of the stream, the root of the tree stores the empty set. For each element e in the stream, we add it as a child of any node T at height less than k, if for no existing child c of T it holds that $f(c \mid S)$=$f(e \mid S)$ where S denotes the solution corresponding to the path from root to T. At the end of the stream, the algorithm produces the best solution among all leaves.

We will choose I to be a set of size $O(k)$. For this however, we first need to know OPT. We show below how to remove this assumption with a small increase in space.

A.2.1 Assumption that OPT is known

The algorithm presented in Section 5.2 uses the assumption that it knows the value OPT. We will describe in the following how to remove this assumption. We use the same trick as described in [5]. Let $\delta > 0$ be a small constant. It is easy to see that when using some value $g \leqslant \mathrm{OPT}$ instead of OPT, the algorithm produces a solution of value at least $0.5506g$. We will run the algorithm in parallel for multiple guesses of g. If $g \leqslant \mathrm{OPT} \leqslant (1+\delta)g$ for some guess, we would only loose a factor of $(1+\delta)$ in the approximation ratio. However, the range in which OPT lies cannot be bounded. Hence, we must adapt our guesses as we read the stream. To that end, we keep track of the maximum element in the stream at any point of time. Let m_i denote the maximum element after observing the first i elements of the stream σ, i.e., $m_i = \max_{j \leqslant i} f(\{\sigma_j\})$. It is easy to see that any subset of $\sigma_1, \ldots, \sigma_i$ with cardinality at most k has a value between m_i and $k \cdot m_i$. Let

$$G_i = \left\{ (1+\delta)^j : j \in \mathbb{N}, \frac{1}{1+\delta} m_i \leqslant (1+\delta)^j \leqslant \frac{k}{\delta} m_i \right\}.$$

At any point i in the stream, we will run our algorithm in parallel for all guesses in G_i. When a new maximum element arrives, this may remove previously existing guesses, in which case we stop the execution for this guess and dismiss its result. On the other hand, new guesses may be added and we start a new execution pretending the stream begins at σ_i.

Let us consider g, the correct guess for OPT, i.e., $g \leqslant \mathrm{OPT} \leqslant (1+\delta)g$. Once g is added to G_i, it remains in the set of guesses until the end of the execution:

If it was removed, this would mean there exists an element of value greater than $(1+\delta)g \geqslant \mathrm{OPT}$. However, no set smaller than k can have a value larger than OPT. It remains to check that the error induced by starting the algorithm late is not significant. Let O denote the elements from the optimal solution and $O' \subseteq O$ those that arrived before execution for g was started. All elements in O' were smaller than $g\delta/k$. Hence, $f(O') \leqslant k \cdot g\delta/k = g\delta$. This implies

$$f(O \setminus O') \geqslant f(O) - f(O') \geqslant \mathrm{OPT} - \delta g \geqslant (1-\delta)\mathrm{OPT}.$$

Hence, the approximation ratio decreases by a factor of at most $(1-\delta)/(1+\delta)$ and the space increases by a factor of $\log_{1+\delta}(k(1+\delta)/\delta) = O(1/\delta \log(k/\delta))$.

A.2.2 Bounding the number of increases

Recall, the tree algorithm stores roughly $|I|^k$ elements. Here $I = \{f(e \mid S) \mid S \subseteq E, |S| < k, e \in E\}$ is the set of all possible increases of f. In order to achieve a reasonable memory bound, we need to make sure that $|I|$ is small. In the following we describe a way that bounds $|I|$ by $O(k)$ and only decreases the approximation ratio marginally.

We assume that OPT is known (see previous section). Let $\delta > 0$ be a small constant. We divide the range 0 to OPT into k/δ buckets each of size $\delta \cdot \mathrm{OPT}/k$. The idea is that I now represents the set of possible range of increases of f where each bucket corresponds to a range. We now argue that this discretization does not affect the approximation ratio much. Recall that the recursion $R(k,h)$ was defined as follows:

$$R(k,h) = \min\left(\frac{t}{k} + \left(1-\frac{t}{k}\right)R(k,h-1), \frac{1}{k} + \left(1-\frac{1+t}{k}\right)R(k-1,h-1), \frac{1}{1+t}\right).$$

In Section 5.2.2, $R(k,k) \cdot \mathrm{OPT}$ was proven to be a lower bound on the expected value of the solution returned by the algorithm. Due to bucketing, the element that is picked now might differ in value by at most $(\mathrm{OPT} \cdot \delta/k)$ from the value promised by the analysis. As the recursion $R(k,k)$ has depth k (the solution S_k consists of k elements), it is not hard to see that one can place a lower bound on the expected value of the returned solution after bucketing by $R(k,k) \cdot \mathrm{OPT} - k \cdot O(\delta/k)\mathrm{OPT}$. Thus the loss incurred by discretization can be made arbitrarily small by appropriately selecting $\delta > 0$.

A.2.3 Analysis of recursion function

In order to prove that our algorithm is 0.55 competitive, we will lower bound the value of the recursion $R(k,h)$ where $R(k,h)$ was defined as the lower bound on the approximation ratio of the solution (we defined in Section 5.2.2) at height i as compared to OPT over all submodular functions, streams and opt elements. The heart of the proof lies in the fact that for a certain threshold

t and large k, the complicated recursion which involves taking the minimum of three terms simplifies to a recursion (that one can solve easily) involving just the first term. We prove this in Lemma A.4. For complete details on the recursion definition and the solution we analyze, we refer the reader to Section 5.2.2.

We first state some technical claims which will be helpful in proving Lemma A.4.

Claim A.1 *The following inequality is satisfied for all $x \in [-0.1, 0]$:*

$$e^x - \frac{x^2}{2} \leqslant 1 + x \leqslant e^x - \frac{x^2}{2} - \frac{x^3}{6}.$$

Proof We first write down the taylor expansion of e^x:

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}.$$

For $x \in [-0.1, 0]$,

$$\sum_{i=3}^{\infty} \frac{x^i}{i!} \leqslant -\frac{|x|^3}{6} + \frac{|x|^3}{24} \cdot \sum_{i=1}^{\infty} |x|^i = -\frac{|x|^3}{6} + \frac{|x|^3}{24} \cdot \frac{1}{1-|x|} \leqslant 0.$$

Hence, $e^x \leqslant 1 + x + \frac{x^2}{2}$.
Similarly,

$$\sum_{i=4}^{\infty} \frac{x^i}{i!} \geqslant \frac{|x|^4}{24} - \frac{|x|^4}{120} \cdot \sum_{i=1}^{\infty} |x|^i = \frac{|x|^4}{24} - \frac{x^4}{120} \cdot \frac{1}{1-|x|} \geqslant 0.$$

Hence, $e^x \geqslant 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$.

Claim A.2 *For $k \geqslant 999$ and $t \leqslant 1$, the following inequality is satisfied:*

$$\frac{1}{2} \sum_{i=2}^{\infty} \left(\frac{t^2 \cdot e^{\frac{t}{k}}}{1.9 \cdot k} \right)^i \leqslant \frac{t^4}{3 \cdot k^2}.$$

Proof By using the formula for the sum of an infinite geometric series:

$$\frac{1}{2} \sum_{i=2}^{\infty} \left(\frac{t^2 \cdot e^{\frac{t}{k}}}{1.9 \cdot k} \right)^i = \frac{t^4 \cdot e^{\frac{2 \cdot t}{k}}}{1.9^2 \cdot k^2} \cdot \frac{1}{1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{1.9 \cdot k}} \leqslant \frac{t^4}{3 \cdot k^2}. \qquad \square$$

Now we prove a closed form expression for $R(k, h)$ when only the first rule is applied.

Claim A.3 *For any integer $k \geqslant 1000$ and any non-negative integer $h \leqslant k$ such that $R(k, h') = t/k + (1 - t/k)R(k, h' - 1)$ for all $1 \leqslant h' \leqslant h$ it holds that*

$$R(k, h) = 1 - \left(1 - \frac{t}{k}\right)^h .$$

In fact, Lemma A.4 will show that for large values of k (and any h) the condition and thereby this closed form holds.

Proof We will prove the equality by induction on h for any $k \geqslant 1000$.
For $h = 0$, the equality holds as $R(k, h) = 0$. Then by induction hypothesis

$$\begin{aligned}R(k, h) &= \frac{t}{k} + \left(1 - (1 - \frac{t}{k})^{h-1}\right) \cdot (1 - \frac{t}{k}) \\ &= \frac{t}{k} + 1 - \frac{t}{k} - \left(1 - \frac{t}{k}\right)^h \\ &= 1 - \left(1 - \frac{t}{k}\right)^h .\end{aligned}$$

□

Lemma A.4 *With $t = 0.8$ for every $k \geqslant 1000$ and $h \leqslant k$ it holds that*

$$R(k, h) = \frac{t}{k} + \left(1 - \frac{t}{k}\right) R(k, h - 1).$$

Proof We will prove by induction on k and h.
For $h = 1$, $R(k, 1) = \frac{t}{k}$. For $k = 1000$, we have verified that the induction hypothesis holds by computer assisted calculation. Hence the base case holds.
Recall that $R(k, h)$ is defined as:

$$R(k, h) = \min\left(\frac{t}{k} + R(k, h - 1) \cdot \left(1 - \frac{t}{k}\right), \frac{1}{k} + R(k - 1, h - 1) \cdot \left(1 - \frac{1 + t}{k}\right), \frac{1}{1 + t}\right).$$

By induction hypothesis and Claim A.3, we know that $R(k, h - 1) = 1 - (1 - t/k)^{h-1}$. Hence for $k \geqslant 1000$ and $h \leqslant k$, we get:

$$\begin{aligned}\frac{t}{k} + R(k, h - 1) \cdot \left(1 - \frac{t}{k}\right) &= \frac{t}{k} + (1 - (1 - \frac{t}{k})^{h-1}) \cdot \left(1 - \frac{t}{k}\right) \\ &= 1 - (1 - \frac{t}{k})^h \\ &\leqslant 1 - (1 - \frac{t}{k})^k.\end{aligned}$$

As $(1-\frac{t}{k})^k$ is monotonically decreasing in k and $t = 0.8$, we get:

$$\begin{aligned}\frac{t}{k} + R(k,h-1)\cdot\left(1-\frac{t}{k}\right) &\leqslant 1-(1-\frac{t}{1000})^{1000}\\ &\leqslant 0.5509\\ &\leqslant \frac{1}{1+t}.\end{aligned}$$

Hence it suffices to prove the below for $k \geqslant 1000$ and $h \leqslant k$:

$$\frac{t}{k} + R(k,h-1)\cdot\left(1-\frac{t}{k}\right) \leqslant \frac{1}{k} + R(k-1,h-1)\cdot\left(1-\frac{1+t}{k}\right).$$

Or, equivalently:

$$t \leqslant 1 + R(k-1,h-1)\cdot(k-1-t) - R(k,h-1)\cdot(k-t)\,.$$

By induction hypothesis and Claim A.3, we know that $R(k,h-1) = 1-(1-\frac{t}{k})^{h-1}$ and $R(k-1,h-1) = 1-(1-\frac{t}{k-1})^{h-1}$.

$$\begin{aligned}t &\leqslant 1 + (1-(1-\frac{t}{k-1})^{h-1})\cdot(k-1-t) - (1-(1-\frac{t}{k})^{h-1})\cdot(k-t)\\ &\leqslant 1 + \left(k-1-t-\frac{(k-1-t)^h}{(k-1)^{h-1}}\right) - \left(k-t-\frac{(k-t)^h}{(k)^{h-1}}\right)\\ &\leqslant \frac{(k-t)^h}{(k)^{h-1}} - \frac{(k-1-t)^h}{(k-1)^{h-1}}\\ &\leqslant \underbrace{k\cdot(1-\frac{t}{k})^h - (k-1)\cdot(1-\frac{t}{k-1})^h}_{E_1}.\end{aligned} \tag{A.1}$$

E_1 is monotonically decreasing in h as shown below:

$$\begin{aligned}&\left(k\cdot(1-\frac{t}{k})^h - (k-1)\cdot(1-\frac{t}{k-1})^h\right)\\ &\qquad - \left(k\cdot(1-\frac{t}{k})^{h+1} - (k-1)\cdot(1-\frac{t}{k-1})^{h+1}\right)\\ &= k\cdot(1-\frac{t}{k})^h\cdot(1-(1-\frac{t}{k})) - (k-1)\cdot(1-\frac{t}{k-1})^h\cdot(1-(1-\frac{t}{k-1}))\\ &= t\cdot(1-\frac{t}{k})^h - t\cdot(1-\frac{t}{k-1})^h \geqslant 0.\end{aligned}$$

Hence we can lower bound E_1 by assuming $h = k$. We lower bound E_1 below:

$$E_1 \geqslant k \cdot (1 - \frac{t}{k})^k - (k-1) \cdot (1 - \frac{t}{k-1})^k.$$

By using Claim A.1, $k \geqslant 1000$ and $t \leqslant 1$, we get:

$$\begin{aligned} E_1 &\geqslant k \cdot (e^{\frac{-t}{k}} - \frac{t^2}{2 \cdot k^2})^k - (k-1) \cdot \left(\underbrace{e^{\frac{-t}{k-1}} - \frac{t^2}{2 \cdot (k-1)^2} + \frac{t^3}{6 \cdot (k-1)^3}}_{T_1} \right)^k \\ &\geqslant k \cdot e^{-t} \cdot \underbrace{\left(1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k^2}\right)^k}_{T_2} - (k-1) \cdot T_1 \cdot e^{-t} \cdot \underbrace{\left(1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)^2} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^3}\right)^{k-1}}_{T_3}. \end{aligned} \tag{A.2}$$

We lower bound the term T_2.

$$\begin{aligned} T_2 &= \left(1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k^2}\right)^k \\ &\geqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k} - \sum_{i=2}^{\infty} \frac{k^i}{2} \cdot \left(\frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k^2}\right)^i \text{ (By Binomial Expansion.)} \\ &= 1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k} - \frac{1}{2} \sum_{i=2}^{\infty} \left(\frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k}\right)^i \\ &\geqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k} - \frac{t^4}{3 \cdot k^2}. \text{ (By Claim A.2, } t \leqslant 1 \text{ and } k \geqslant 1000.) \end{aligned}$$

We now simplify and lower bound $k \cdot e^{-t} \cdot T_2$:

$$\begin{aligned} k \cdot e^{-t} \cdot T_2 &\geqslant k \cdot e^{-t} \cdot (1 - \frac{t^2 \cdot e^{\frac{t}{k}}}{2 \cdot k} - \frac{t^4}{3 \cdot k^2}) \\ &= k \cdot e^{-t} - \frac{t^2 \cdot e^{\frac{t}{k}} \cdot e^{-t}}{2} - \frac{t^4 \cdot e^{-t}}{3 \cdot k}. \end{aligned} \tag{A.3}$$

We now upper bound T_3.

$$T_3 = \left(1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)^2} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^3}\right)^{k-1}.$$

By using Binomial Expansion, we get:

$$\begin{aligned}
T_3 &\leqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^3} + \sum_{i=2}^{\infty} \frac{(k-1)^i}{2} \cdot \left(\frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)^2} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^3} \right)^i \\
&= 1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^3} + \frac{1}{2} \cdot \sum_{i=2}^{\infty} \left(\frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)^2} \right)^i .
\end{aligned}$$

By using that $t \leqslant 1$ and $k \geqslant 1000$, we get:

$$\begin{aligned}
T_3 &\leqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{t^3}{3 \cdot (k-1)^2} + \frac{1}{2} \cdot \sum_{i=2}^{\infty} \left(\frac{t^2 \cdot e^{\frac{t}{k-1}}}{1.9 \cdot (k-1)} \right)^i \\
&\leqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{t^3}{3 \cdot (k-1)^2} + \frac{t^4}{3 \cdot (k-1)^2} \quad \text{(By Claim A.2, } t \leqslant 1, k \geqslant 1000.) \\
&\leqslant 1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{2 \cdot t^3}{3 \cdot (k-1)^2}.
\end{aligned}$$

We now upper bound T_1. By using Claim A.1, $k \geqslant 1000$ and $t \leqslant 1$, we get:

$$T_1 \leqslant 1 - \frac{t}{k-1} + \frac{t^3}{6 \cdot (k-1)^3}.$$

We now simplify and upper bound $(k-1) \cdot T_1 \cdot e^{-t} \cdot T_3$:

$$\begin{aligned}
&(k-1) \cdot T_1 \cdot e^{-t} \cdot T_3 \\
&\leqslant (k-1) \cdot e^{-t} \cdot T_1 \cdot (1 - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} + \frac{2 \cdot t^3}{3 \cdot (k-1)^2}) \\
&= e^{-t} \cdot T_1 \cdot ((k-1) - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} + \frac{2 \cdot t^3}{3 \cdot (k-1)}) \\
&\leqslant e^{-t} \cdot (1 - \frac{t}{k-1} + \frac{t^3}{6 \cdot (k-1)^3}) \cdot ((k-1) - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} + \frac{2 \cdot t^3}{3 \cdot (k-1)}) \\
&\leqslant e^{-t} \cdot \left(\left((k-1) - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} + \frac{2 \cdot t^3}{3 \cdot (k-1)} \right) + \left(-t + \frac{t^3 \cdot e^{\frac{t}{k-1}}}{2 \cdot (k-1)} \right) \right) \\
&\qquad\qquad + e^{-t} \cdot \left(\left(\frac{t^2 \cdot e^{\frac{t}{-1}}}{3 \cdot (k-1)^2} \right) \right) \\
&\leqslant e^{-t} \cdot \left(k - 1 - t - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} + \frac{7 \cdot t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)} + \frac{t^2 \cdot e^{\frac{t}{k-1}}}{3 \cdot (k-1)^2} \right) . \qquad \text{(A.4)}
\end{aligned}$$

By replacing (A.3) and (A.4) in (A.2), we get:

$$\begin{aligned}
E_1 &\geqslant k \cdot e^{-t} - \frac{t^2 \cdot e^{\frac{t}{k}} \cdot e^{-t}}{2} - \frac{t^4 \cdot e^{-t}}{3 \cdot k} \\
&\quad - e^{-t} \cdot \Big(k - 1 - t - \frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} + \frac{7 \cdot t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)} + \frac{t^2 \cdot e^{\frac{t}{k-1}}}{3 \cdot (k-1)^2}\Big) \\
&= e^{-t} \cdot (1 + t) + e^{-t} \cdot \Big(\frac{t^2 \cdot e^{\frac{t}{k-1}}}{2} - \frac{t^2 \cdot e^{\frac{t}{k}}}{2}\Big) - e^{-t} \cdot \Big(\frac{t^4}{3 \cdot k} + \frac{7 \cdot t^3 \cdot e^{\frac{t}{k-1}}}{6 \cdot (k-1)}\Big) \\
&\quad - e^{-t} \cdot \frac{t^2 \cdot e^{\frac{t}{k-1}}}{3 \cdot (k-1)^2} \\
&\geqslant e^{-t} \cdot (1 + t) - e^{-t} \cdot \frac{2 \cdot t^3 \cdot e^{\frac{t}{k-1}}}{(k-1)} - e^{-t} \cdot \frac{t^2 \cdot e^{\frac{t}{k-1}}}{3 \cdot (k-1)^2}. \qquad \text{(A.5)}
\end{aligned}$$

By using (A.1) and (A.5), we get that it suffices to prove the following:

$$t \leqslant e^{-t} \cdot (1 + t) - e^{-t} \cdot \frac{2 \cdot t^3 \cdot e^{\frac{t}{k-1}}}{(k-1)} - e^{-t} \cdot \frac{t^2 \cdot e^{\frac{t}{k-1}}}{3 \cdot (k-1)^2}.$$

Here, $t = 0.8$ satisfies the final inequality. □

Lemma (5.2 restated) *For all positive integers k, $R(k,k) \geqslant 0.5506$.*

Proof For $k \leqslant 1000$, $R(k,k) \geqslant 0.5506$ can easily be verified with computer assistance, since it involves only a dynamic program of size 1000×1000. For $k > 1000$, by Lemma A.4 and Claim A.3, $R(k,k) \geqslant 1 - (1 - \frac{0.8}{k})^k \geqslant 1 - e^{-0.8} \geqslant 0.5506$. □

Appendix B

Deferred Proofs from Part II

B.1 Extending Algorithm 3 when matroid ranks are unknown

In this section, we extend Algorithm 3 to the case where the matroid ranks, i.e r_i, are unknown. Since r_i is not known, we can not set $y = \frac{\min(r_1,r_2)}{\varepsilon^2}$. The idea is to guess the rank of one of the matroids, say M_1 and adapt the y value that would be assigned to newly added elements as we update our guess. The y values are set in a way that the errors arising from deleting elements specific to a single y value form a geometrically decreasing sequence.

More concretely, we set a y value specific to each element $e \in S$ using the notion of stacks introduced in the proof of Theorem 7.7. To recap, on the arrival of element e that is added to our set S initially (may be deleted later), the following two things might happen to the maximum weight independent set with respect to w_1 i.e, T_1. Either it replaces an element e' in T_1, or it is added to T_1. In the former case, we say that e is added to the stack that contains e'. In the latter, we say that we create a new stack and add e to it. In Algorithm 6, we use s to denote the number of stacks at any point during the execution. The y value of an element is decided based on the stack it belongs to. Specifically, the y value of the elements in the first two stacks is set to $\frac{4}{\varepsilon^2}$, the next four stacks to $\frac{16}{\varepsilon^2}$ and in general a value of $\frac{4^i}{\varepsilon^2}$ for the next 2^i stacks. This is done in Algorithm 6 by the function $z : \mathbb{N} \mapsto \mathbb{R}$ where $z(1) = z(2) = \frac{4}{\varepsilon^2}$, $z(3) = z(4) = z(5) = z(6) = \frac{16}{\varepsilon^2}$ and in general a value of $\frac{4^i}{\varepsilon^2}$ for the next 2^i numbers. The values are set in such a way so that the error introduced in each bundle of stacks forms a geometrically decreasing sequence. Using arguments used in the proof of Lemma 7.7, the error introduced in the i^{th} bundle of stacks that contains 2^i stacks is at most $\frac{2^i g_{max}\varepsilon^2(1+\varepsilon)}{4^i\varepsilon} = \frac{g_{max}\varepsilon(1+\varepsilon)}{2^i}$. Summing this over all i gives us that the error is at most $g_{max}\varepsilon(1+\varepsilon)$ giving us the same error we obtained in the proof of Lemma

7.7. By an analysis similar to the one in the proof of Lemma 7.7, we can prove that the size of S never exceeds $r_1 + r_2 + \max(r_1, r_2)\log_\alpha(\frac{\alpha y_f}{\varepsilon})$ where y_f is the value in the last bundle of stacks formed. By simple calculation, y_f is at most $\frac{\max(r_1,r_2)^2}{\varepsilon^2}$.

Algorithm 6 Extension of Algorithm 3 to unknown r_i

```
Input: A stream of the elements and 2 matroids (which we call $M_1, M_2$) on
  the same ground set $E$, a real number $\alpha > 1$ and a function $z : \mathbb{N} \mapsto \mathbb{R}$.
Output: A set $X \subseteq E$ that is independent in both matroids.
  Whenever we write an assignment of a variable with subscript $i$, it means
  we do it for $i = 1, 2$.
  $S \leftarrow \emptyset$, $T_i \leftarrow \emptyset$
  $s = 0$                                   {//initialize no. of stacks to zero.}
  for element $e$ in the stream do
    calculate $w_i^*(e) = \max\Big(\{0\} \cup \{\theta : e \in \mathrm{span}_{M_i}(\{f \in S \mid w_i(f) \geqslant \theta\})\}\Big)$.
    if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ then
      $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$
      $S \leftarrow S \cup \{e\}$
      $w_i(e) \leftarrow g(e) + w_i^*(e)$
      Let $H_i$ be a maximum weight independent set of $M_i$ with respect to
      $w_i$.
      if $\{e\} \cup T_1 = H_1$ then
        $s = s + 1$
        $y(e) = z(s)$                       {//$H_1$ is formed by adding $e$ to $T_1$}
      else
        $\{e'\} = T_1 \setminus (H_1 \setminus \{e\})$      {//$H_1$ is formed by replacing $e'$ with $e$ in
        $T_1$}
        $y(e) = y(e')$
      end if
      $T_i = H_i$
      Let $g_{max} = \max_{e \in S} g(e)$
      Remove all elements $e' \in S$, such that $y(e') \cdot g(e') < g_{max}$ and $e' \notin$
      $T_1 \cup T_2$ from S.
    end if
  end for
  return a maximum weight set $T \subseteq S$ that is independent in $M_1$ and $M_2$
```

Appendix C

Deferred Proofs from Part III

C.1 Omitted proofs from Chapter 11

C.1.1 Solving the configuration LP

The goal of this section is to prove Theorem 11.1. We consider the dual of the configuration LP (after adding an artificial minimization direction $\min 0^T x$).

$$\begin{aligned}
&\max \sum_{i \in P} y_i - \sum_{j \in R} z_j \\
&\sum_{j \in C} z_j \geqslant y_i \quad \text{for all } i \in P, C \in \mathbb{C}(i, T) \\
&y_j, z_i \geqslant 0
\end{aligned}$$

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

$$\sum_{i \in P} y_i - \sum_{j \in R} z_j \geqslant 1. \qquad (*)$$

We approximate the dual in the variant with a 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 [63]. 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 $(1 - 1/e)\mathrm{OPT}$, including 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 $T > \mathrm{OPT}$. 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 $C \in \mathbb{C}(i, T)$. 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 $C \in \mathbb{C}(i, T)$ it holds that $\sum_{j \in C} z_j \geqslant y_i$, i.e., z, y is feasible for objective value T and hence $\mathrm{OPT} < T$. If the ellipsoid method terminates without concluding that $\mathrm{OPT} < T$, 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 11.1.

C.1.2 Clusters

This section is devoted to proving Lemma 11.2. The arguments are similar to those used in [6].

Lemma C.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 $i \in P$, $C \in \mathbb{C}_t(i, T^*)$ 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,*

$$\sum_{i \in P} \sum_{C \in \mathbb{C}_t(i,T^*) : j \in C} x'_{i,C} \leqslant 1$$

We say that the congestion on item j is at most 1.

2. *every cluster K_j gets at least $1/2$ thin configurations in x', that is,*

$$\sum_{i \in K_j} \sum_{C \in \mathbb{C}_t(i,T^*)} x'_{i,C} \geqslant 1/2;$$

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 player added 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 a degree of 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 $i_1, i_2, \dots$ (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 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. $\square$

Lemma C.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 $\mathfrak{C}_t(i, T^*/5)$, but satisfies (2) in Lemma C.1 with 2 instead of 1/2. This can be achieved by splitting each configuration $C \in \mathfrak{C}_t(i, T^*)$ in 4 disjoint parts $C_1, C_2, C_3, C_4 \in \mathfrak{C}_t(i, T^*/5)$. 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 $C \in \mathfrak{C}(i, T^*)$ increasing x''_{i,C_1}, x''_{i,C_2}, x''_{i,C_3}, and x''_{i,C_4} by $x'_{i,C}$. Here $C_1, C_2, C_3, C_4 \in \mathfrak{C}(i, T^*/5)$ 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 11.2 which we restate for convenience.

Lemma (11.2 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 $K_1 \cup \dots \cup K_k \cup Q = P$ and multisets of configurations $\mathfrak{C}_h \subseteq \bigcup_{i \in K_h} \mathfrak{C}_T(i, T^*/5)$, $h = 1, \dots, k$, such that*

1. *$|\mathfrak{C}_h| = \ell$ for all $h = 1, \dots, k$ and*
2. *Each small resource appears in at most ℓ configurations of $\bigcup_h \mathfrak{C}_h$.*
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 C.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 $\{x''_{i,C}/2\}_{i\in K_h, C\in \mathbb{C}_t(i,T^*/5)}$. 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

$$\sum_{i\in P}\sum_{C\in \mathbb{C}(i,T^*/5):j\in C} x''_{i,C}/2 \leqslant 1/2$$

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 2.2), we have that with probability at most

$$\exp\left(-\ell/6\right) \leqslant 1/n^2$$

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. □

C.2 Omitted proofs from Section 12.2

Theorem (12.3 restated) *Consider Random Experiment 12.2 with* $\ell \geqslant 300.000\log^3(n)$. *For any* $k \geqslant 0$ *and any* $C \in \mathbb{C}^{(\geqslant k)}$ *we have*

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

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

Proof The lemma trivially holds for $k = 0$. For $k > 0$, by assumption $C \in \mathbb{C}^{(\geqslant k)}$ 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 2.3), 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 (12.4 restated) *Consider Random Experiment 12.2 with $\ell \geqslant 300.000 \log^3(n)$. For any $k \geqslant 0$ and any $C \in \mathbb{C}^{(\geqslant k)}$*

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

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

Proof The expected value of the random variable $X = \sum_{C' \in \mathbb{C}^{(k)}} |C' \cap C \cap R_k|$ is

$$\mathbb{E}(X) = \frac{1}{\ell^k} \sum_{C' \in \mathbb{C}^{(k)}} |C' \cap C|.$$

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 Proposition 2.3), 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 the main body of the Part III of our thesis, this statement is a generalization of some ideas that already appeared in [6]. However, in [6], 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 (12.5 restated) *Consider Random Experiment 12.2 with $\ell \geqslant 300.000 \log^3(n)$. Fix $k \geqslant 0$. Conditioned on the event that the bounds in Lemma 12.3 hold for k, then with probability at least $1 - 1/n^{10}$ the following holds for all $\mathcal{F} \subseteq \mathbb{C}^{(\geqslant k+1)}$, $\alpha : \mathcal{F} \to \mathbb{N}$, 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*

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

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

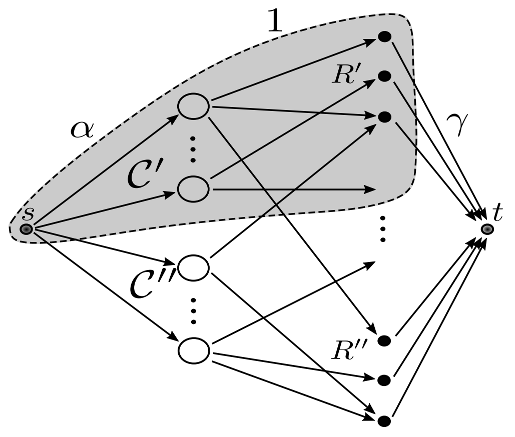

Figure C.1: The directed network and an s-t cut

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 $\mathcal{F} \subseteq \mathbb{C}^{(\geqslant k+1)}$, 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 C.1 for an illustration.

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 C.3 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 C.2 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 C.2 *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. $\square$

Lemma C.3 *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*

$$\text{maxflow}\,(\mathcal{N}) \geqslant \frac{\ell}{1+0.5/\log(n)}\text{maxflow}\,(\tilde{\mathcal{N}})$$

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 C.4 *For every $C \in \mathcal{F}$, the outdegree of the vertex corresponding to C in $\mathcal{N}$ is at least $\ell^4/2$.*

Since $C \in \mathbb{C}^{(\geqslant k+1)}$ and by Lemma 12.3, we clearly have that $|C \cap R_k| \geqslant \ell^4/2$.

Claim C.5 *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 C.4) 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 C.5. 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 Proposition 2.3) 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 C.5 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

$$c(\tilde{S}) = \sum_{C \in \mathcal{C}''} \alpha(C) + e(\mathcal{C}', \tilde{R}'') + \gamma|\tilde{R}'| \leqslant \frac{1}{\ell}c(S) + \frac{0.5}{\ell \log(n)} c(S). \qquad \square$$

We are now ready to prove Lemma 12.5. Note that Lemma C.3 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 C.3 ($(m\ell)^s \leqslant (n\ell)^s$ 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 C.3 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 C.2 the 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 C.3, this implies that 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 C.2, this implies a (α', γ)-good assignment from R_k to $\mathcal{F}$, where

$$\begin{aligned} \alpha'(C) = \lfloor \ell/(1 + 0.5/\log(n)) \rfloor \alpha(C) &\geqslant \ell/(1 + 1/\log(n))\alpha(C) \\ &\geqslant \ell(1 - 1/\log(n))\alpha(C). \end{aligned}$$

C.3 Omitted proofs from Section 12.4

Claim (12.10 restated) *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 12.6 we have that

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

Furthermore, by Lemma 12.4, we get

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

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

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

Putting everything together we conclude

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

Claim (12.12 restated) *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. □

Bibliography

[1] Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 1:1–1:19, 2019.

[2] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Trans. Algorithms*, 13(3):37:1–37:28, 2017.

[3] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Trans. Algorithms*, 8(3), July 2012.

[4] Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. *arXiv preprint arXiv:2102.07011*, 2021.

[5] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680, 2014.

[6] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 31–40, New York, NY, USA, 2006. Association for Computing Machinery.

[7] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 27 – 45. North-Holland, 1985.

[8] Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. in memoriam:

Shimon even 1935-2004. *ACM Computing Surveys (CSUR)*, 36:422–463, 12 2004.

[9] Aaron Bernstein. Improved bound for matching in random-order streams. *arXiv preprint arXiv:2005.00417*, 2020.

[10] Domagoj Bradac, Anupam Gupta, Sahil Singla, and Goran Zuzic. Robust algorithms for the secretary problem. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 32:1–32:26, 2020.

[11] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1433–1452. SIAM, 2014.

[12] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019.

[13] Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

[14] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116. IEEE Computer Society, 2009.

[15] Siu-Wing Cheng and Yuchen Mao. Restricted max-min fair allocation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 37:1–37:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[16] Siu-Wing Cheng and Yuchen Mao. Restricted max-min allocation: Approximation and integrality gap. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 38:1–38:13, 2019.

[17] M. Crouch and D.M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. *Leibniz International Proceedings in Informatics, LIPIcs*, 28:96–104, 09 2014.

[18] Sami Davies, Thomas Rothvoss, and Yihao Zhang. A tale of santa claus, hypergraphs and matroids. In *Proceedings of the 2020 ACM-SIAM*

Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, pages 2748–2757, 2020.

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

[20] Jack Edmonds. Matroid intersection. In *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 39 – 49. Elsevier, 1979.

[21] Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for randomized preemptive online matching. *Inf. Comput.*, 259(1):31–40, 2018.

[22] Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 169–186, 2015.

[23] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785, 2020.

[24] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[25] Uriel Feige. On allocations that maximize fairness. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 287–293, USA, 2008. Society for Industrial and Applied Mathematics.

[26] Uriel Feige. Tighter bounds for online bipartite matching. *CoRR*, abs/1812.11774, 2018.

[27] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[28] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the Fifty-Second Annual ACM on Symposium on Theory of Computing, STOC (to appear)*, 2020.

[29] Tamás Fleiner. A matroid generalization of the stable matching polytope. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 105–114. Springer, 2001.

[30] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019.

[31] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37, 2019.

[32] Mohsen Ghaffari and David Wajc. Simplified and Space-Optimal Semi-Streaming (2+epsilon)-Approximate Matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:8, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[33] Michel X Goemans, Nicholas JA Harvey, Satoru Iwata, and Vahab Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 535–544. SIAM, 2009.

[34] Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 273–279, 2006.

[35] Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 241–253, 2017.

[36] Penny E Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995.

[37] Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of monotone submodular function maximization under cardinality and matroid constraints in the streaming model. *CoRR*, abs/2002.05477, 2020.

[38] Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2875–2886, 2019.

[39] Klaus Jansen and Lars Rohwedder. A note on the integrality gap of the configuration lp for restricted santa claus. *Information Processing Letters*, 164:106025, 2020.

[40] Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013.

[41] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[42] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990.

[43] Thomas Kesselheim, Robert D. Kleinberg, and Rad Niazadeh. Secretary problems with non-uniform arrival order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 879–888, 2015.

[44] Thomas Kesselheim and Andreas Tönnis. Submodular secretary problems: Cardinality, matching, and linear constraints. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 16:1–16:22, 2017.

[45] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242. 2012.

[46] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[47] Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games Econ. Behav.*, 55(2):270–296, 2006.

[48] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 217–224, 1987.

[49] Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1914–1933. SIAM, 2021.

[50] Paul Liu, Aviad Rubinstein, Jan Vondrak, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. *Advances in Neural Information Processing Systems*, 34, 2021.

[51] László Lovász. On determinants, matchings, and random algorithms. In *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574, 1979.

[52] Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[53] Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019.

[54] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[55] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15, 2010.

[56] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[57] George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

[58] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[59] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3826–3835, 2018.

[60] Ami Paz and Gregory Schwartzman. A (2+ ε)-approximation for maximum weight matching in the semi-streaming model. *ACM Transactions on Algorithms (TALG)*, 15(2):1–15, 2018.

[61] Lukas Polacek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 726–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[62] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

[63] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[64] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74. ACM, 2008.