

ABSTRACT

Title of dissertation: APPROXIMATION ALGORITHMS FOR
RESOURCE ALLOCATION

Barna Saha, Doctor of Philosophy, 2011

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

This thesis is devoted to designing new techniques and algorithms for combinatorial optimization problems arising in various applications of resource allocation. Resource allocation refers to a class of problems where scarce resources must be distributed among competing agents maintaining certain optimization criteria. Examples include scheduling jobs on one/multiple machines maintaining system performance; assigning advertisements to bidders, or items to people maximizing profit/social fairness; allocating servers or channels satisfying networking requirements etc. Altogether they comprise a wide variety of combinatorial optimization problems. However, a majority of these problems are NP-hard in nature and therefore, the goal herein is to develop approximation algorithms that approximate the optimal solution as best as possible in polynomial time.

The thesis addresses two main directions. First, we develop several new techniques, predominantly, a new linear programming rounding methodology and a constructive aspect of a well-known probabilistic method, the Lovász Local Lemma

(LLL). Second, we employ these techniques to applications of resource allocation obtaining substantial improvements over known results. Our research also spurs new direction of study; we introduce new models for achieving energy efficiency in scheduling and a novel framework for assigning advertisements in cellular networks. Both of these lead to a variety of interesting questions.

Our linear programming rounding methodology is a significant generalization of two major rounding approaches in the theory of approximation algorithms, namely the dependent rounding and the iterative relaxation procedure. Our constructive version of LLL leads to first algorithmic results for many combinatorial problems. In addition, it settles a major open question of obtaining a constant factor approximation algorithm for the Santa Claus problem. The Santa Claus problem is a *NP*-hard resource allocation problem that received much attention in the last several years. Through out this thesis, we study a number of applications related to scheduling jobs on unrelated parallel machines, such as provisionally shutting down machines to save energy, selectively dropping outliers to improve system performance, handling machines with hard capacity bounds on the number of jobs they can process etc. Hard capacity constraints arise naturally in many other applications and often render a hitherto simple combinatorial optimization problem difficult. In this thesis, we encounter many such instances of hard capacity constraints, namely in budgeted allocation of advertisements for cellular networks, overlay network design, and in classical problems like vertex cover, set cover and *k*-median.

APPROXIMATION ALGORITHMS FOR RESOURCE ALLOCATION

by

Barna Saha

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:
Professor Samir Khuller, Chair/Advisor
Professor Amol Deshpande
Professor Mohammadtaghi Hajiaghayi
Professor Armand M. Makowski
Professor Aravind Srinivasan

© Copyright by
Barna Saha.
2011

Dedication

To my loving husband and dearest parents whose love and inspiration led to this day!

Acknowledgments

“Gratitude is the fairest blossom which springs from the soul.”

—Henry Ward Beecher

The last four years that span the duration of my doctoral study had been extra special to me. During this time, I came across a lot of people, many of whom directly or indirectly helped me to fulfill my endeavor. This acknowledgement is a little effort to say thank you to these persons. I also take this opportunity to express my gratitude to my family and friends whose love and support led to this day.

Fast and foremost, I would like to thank my PhD advisor, Professor Samir Khuller. In Samir, I found a very friendly and forgiving advisor, who gave me ample of independence in doing research and also gave me valuable guidance when I needed it. He is an excellent teacher. On my very first semester, I took his course on approximation algorithms, and right then I decided to do a Ph.D. in that subject. I have always found him available for discussions; even on weekends, he would find time to help me on conference practice talks. Meetings with him were both educative and enjoyable – we used to talk about a lot of related and unrelated things. I consider myself fortunate to have him as an advisor; without him, my Ph.D. experience could not have been the same.

Next to my advisor, I would like to thank Professor Aravind Srinivasan. My research philosophy is greatly influenced by him. We have interacted a lot during the last four years, and from every interaction, I learnt something that helped to shape my dissertation. Not only we wrote several papers together, but he also helped me to learn basic research principles, writing scholarly dispositions etc.

Apart from Samir and Aravind, Professor Amol Deshpande, Professor Mo-

hammadtaghi Hajiaghayi and Professor Armand Makowski served in my doctoral dissertation committee. I am grateful to them for their valuable time and feedbacks. I would also like to thank Prof. David Mount who was in my Ph.D. proposal committee for his time and suggestions. I have collaborated with both Amol and Mohammad and have enjoyed every bit of those experiences.

I did a number of internships during my doctoral study and got acquainted with many researchers in my field. I especially want to mention Dr. Howard Karloff, Dr. Divesh Srivastava and Dr. Ioana Roxana Stanoi. We spent significant amount of time discussing and debating about research problems. They helped me immensely during my job-search period and I cannot thank them enough for that. Apart from them, I enjoyed the company of Dr. Flip Korn, Dr. Lukasz Golab and Ravishankar Krishnaswamy. I would also like to thank Dr. Nikhil Bansal for giving me an opportunity to do internship in his group; it was a very learning experience for me. My thanks are due to all of my collaborators, Saeed Alaei, Bernhard Heaupler, Dr. Jian Li, Vahid Liaghat, Dr. Vishwanath Nagarajan, Prof. Louiqa Raschid, are only to name but a few.

I am blessed with a number of very good friends. Though miles apart, I always kept in touch with two of my school best friends, Roshni and Tapashree. They have the ability to make me laugh even in my toughest times. I would also like to mention Aravind, Jana, Lakshminath, Sumit and Vinaya here. I know I can call them any time and can share every joy and happiness with them. In Maryland, I got the company of a bunch of wonderful friends, Arijit, Biswadip, Koushik, Mohan & his family, Punarbasu, Rajibul, Tania and others. We spent many fun-filled evenings together.

I grew up in a family of teachers in a university campus environment. My father, Professor Pranab Kumar Saha, always wanted to see me in academics doing

independent research. My choice of career path and my ideology are directly influenced by him. My mother, Mrs. Asima Saha, is my friend, philosopher and guide. She sacrificed a lot for the proper upbringing of her children. Most of my school education was done under her guidance and she is the best teacher I have ever seen. Love and inspiration of my parents have been the greatest gifts in my life. I feel immensely satisfied when I see the sparkle of happiness on their faces at my achievements. My grandfather, late Professor Sailesh Bhusan Choudhuri taught me many things during my school days. My grandmothers, Late Mrs. Bhaktilata Saha and Late Mrs. Bela Choudhuri, my masi–Late Mrs. Anima Roy would have been extremely happy to see me today. I always felt their blessings and that inspired me to do better. My younger sister, Dona, who is a real doctor is the sweetest person in the world. We share a very special bond. She is an extremely hard-working girl with a very strong personality, and I know, she will achieve many great heights in her life. My little brother Dodo suffered a big tragedy at a very young age, when his parents passed away in an accident, but with his liveliness he overcame that loss. He always cheers me up and I love him a lot. He has endless opportunities in front of him and I wish him all the best.

One person for whom my life and these Maryland days became so much more beautiful is my husband Arya. He is the most caring husband one can ever ask for. I discussed each and every research work with him and he always gave me valuable suggestions. It is for him, I never had to worry about a single thing. He kept me so happy that I hardly felt the pain of staying far away from my country. He is my best friend. This thesis starts with a quotation by none but him.

I gratefully acknowledge the supports of the NSF grants CCF0728839 and CCF0937865 at different stages of my thesis research.

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Resource Allocation Problems	3
1.3	Probabilistic Methods	10
1.4	Linear Programming Relaxation and Rounding	14
1.5	Structure of Linear programs	18
1.6	Organization	19
I	Techniques	22
2	A New Linear Programming Rounding Method	23
2.1	The Dependent Randomized Rounding	24
2.1.1	Dependent Rounding on Bipartite Graphs	25
2.1.2	A Negative Correlation Property	27
2.2	Generalization to Arbitrary Linear System	31

2.3	Randomized Iterative Relaxation	33
3	Constructive Aspects of the Lovász Local Lemma	36
3.1	Preliminaries & Limitation of MT Algorithm	37
3.1.1	Review of the MT Algorithm and its Analysis	41
3.2	LLL-Distribution	42
3.3	LLL Applications with Super-Polynomially Many Bad Events . .	45
3.4	An Example: Non-repetitive Coloring	54
II	Applications	58
4	Scheduling: Handling Hard Capacities and Outliers	59
4.1	Scheduling with Hard Capacities	62
4.2	Scheduling with Outliers	73
5	Scheduling: Handling Energy Efficiency	91
5.1	Machine Activation Problems in Data Centers	92
5.2	LP Rounding for Machine Activation on Unrelated Machines . . .	97
5.3	Minimizing Machine Activation Cost and Assignment Cost	109
5.4	Extensions: Handling Release Times	112
6	Fair Allocation: Maxmin Fair Allocation Problem	114
6.1	Max-Min Fair Allocation: A Configuration LP Approach	115
6.1.1	Algorithm for Max-Min Fair Allocation	118

7	Fair Allocation: The Santa Claus Problem	131
7.1	Algorithm for the Santa Claus Problem	133
7.1.1	From a configuration LP solution to a (k, l, β) -system . . .	136
7.1.2	Construction of a γ -good solution for a (k, l, β) -system . .	138
8	Overlay Network Design	148
8.1	Designing Overlay Multicast Networks For Streaming	148
9	Matroid Median	156
9.1	The Matroid Median Problem	157
9.1.1	An LP Relaxation for MatroidMedian	164
9.1.2	Solving the LP: The Separation Oracle	165
9.2	The Rounding Algorithm for MatroidMedian	166
9.3	MatroidMedian with Penalties	183
9.4	The KnapsackMedian Problem	193
9.5	Bad Example for Local Search with Multiple Swaps	200
10	AdCell: Ad Allocation in Cellular Networks	202
10.1	AdCell: An Introduction	203
10.1.1	AdCell Model & Problem Formulation	206
10.2	Offline Setting	209
10.2.1	Rounding Algorithm	210
11	Covering with Hard Capacities	230

11.1	Capacitated Vertex Cover and Set Cover Problem	231
11.2	Vertex Cover on Multigraphs with Hard Capacities	237
11.2.1	Rounding Algorithm	239
11.3	Multi-Set Multi-cover	250
11.3.1	Rounding Algorithm	251
11.4	Set Cover with Hard Capacity Constraints	258
11.4.1	Hard-Capacitated Set Cover with Arbitrary Multiplicities .	267
12	Conclusion	270
12.1	Future Directions	271
	Index	294

Introduction

“The truth, the way we see it, is only an approximation.”

1.1 Introduction

Most combinatorial optimization problems, including those that arise naturally in many applications are **NP**-Hard. Therefore, under the widely believed complexity assumption of $\mathbf{P} \neq \mathbf{NP}$, there are no algorithms that run in time polynomial in their input size and solve these problems optimally. In order to obtain such efficient algorithms, we thus resort to *approximations*. Instead of returning an exact solution, we return in polynomial time an approximate solution that is as close to the optimal as possible.

The theory of approximation algorithms have developed in the last few decades as a systematic method for designing algorithms for **NP**-hard problems. Each discrete optimization problem has an associated objective function, which depending on the problem needs to be maximized or minimized. Given such an objective

function, we define an α -factor ($\alpha > 0$) approximation algorithm as follows.

Definition 1.1.1 (α -factor approximation algorithm). *An α -factor approximation algorithm is a polynomial time algorithm that for all instances of a problem returns a solution with its objective value within a factor of α of an optimal one.*

Approximation algorithms provide a mathematical basis to study heuristics that often return good solution in practice. Even when practical problems are too complicated to formulate and analyze, approximation algorithms are useful. Solving some simplified versions of the underlying problem may lead to designing effective heuristics and to understanding the basic hardness. It is a formidable task to enumerate all the different techniques that have been developed in this regard. But, two approaches that can claim to have been extremely fruitful are the use of mathematical programming and randomization.

A vast class of combinatorial problems can be formulated as mathematical programs, especially, as integer linear programs (ILP). In an ILP, the variables take integer values, and the constraints as well as the objective function are linear in nature. Since general ILP is **NP**-hard, we relax the requirement that the variables take integral values to allow fractional solution. This is known as the technique of *linear programming relaxation*. An optimal value of a linear programming relaxation can be computed in polynomial time. Such a fractional optimal solution is then *rounded* to a nearby integer solution as close to the original integer optimal as possible. Optimal objective value of a linear program (LP) serves as a lower (upper) bound for

the true optimal objective value for a minimization (maximization) problem. Many LP-rounding approaches have been developed in the last two decades. With the seminal work of Raghavan and Thompson [116], *randomized/probabilistic rounding* methods have been successfully applied to obtain good approximation algorithms for many **NP**-hard problems. In fact, it is difficult to overemphasize the role of randomization in algorithm design. In a lecture series called *Randomization & Religion*, Donald E. Knuth rightly said, “*If somebody would ask me, in the last 10 years, what was the most important change in the study of algorithms I would have to say that people getting really familiar with randomized algorithms had to be the winner*”.

The first part of this thesis is devoted to designing new rounding techniques and randomized methodologies that generalize or improve many previous works. In the second half, they are applied to a large class of problems that fall under the general umbrella of *resource allocation*.

1.2 Resource Allocation Problems

Resource allocation refers to a class of problems where scarce resources must be distributed among competing agents maintaining certain optimization criteria. Such problems are ubiquitous in computer science. For example, consider a job scheduling scenario in parallel machines, where jobs need to be executed on a limited number of machines in a timely manner. Each job has some data requirements

and thus can be scheduled only on a subset of machines. At any time, a machine can process only one job. Under such a constrained environment, the question is how to schedule the jobs so that certain system performance is optimized, for example, the maximum processing load on any machine is minimized. In addition, energy consumption by active machines is a critical issue, and one must attempt to optimize energy usage. In this thesis, we discuss several such scheduling problems and designing approximation algorithms for them.

Another variant of resource allocation problems deals with distributing items to people. Each person want a subset of items and the utility of a person is a function of the items received. The items may have different valuations for different persons and cannot be shared. The goal is to allocate the items in a fair way such that the minimum utility received by any person is as high as possible. Such problems arise frequently in economics and market design and are known as *max-min fair allocation* problems. A related question concerns assigning *adwords* to advertisers. Consider the internet advertisement scenario in Google Search. Each advertiser has a limited budget and bids for different keywords. There are limited number of slots per keyword to exhibit the advertisements. The question is how Google should choose and show the advertisements in the few available slots while keywords are being searched by users. Internet advertisement is a million dollar business, and adword allocation plays an important role there. In this thesis, we will encounter a variety of problems of similar flavor.

In networking, bandwidth assignment or server allocation is done through carefully designed protocols to optimize the resource usage. For instance, in a content distribution network, there are different types of servers and limited budgets allow one to open only a fixed number of servers of each type. A client is then connected to the server that can process its request most cost-effectively. Many combinatorial optimization problems are motivated from these applications. We will see a few of them in this thesis. A common feature in many of these applications is the *hard capacity* requirements. For example, the number of servers that can be opened may not exceed the allowed limit. The number of clients that can be handled by each server may also be limited. In this thesis, we will encounter many instances of hard capacity constraints.

We now give a brief description of some of the applications considered in this thesis.

Scheduling on Parallel Machines. Job scheduling problems are one of the most important classes of resource allocation problems. The scheduling literature is vast and one can propose a variety of interesting questions in this area. In this thesis, we focus our attention on perhaps one of the most widely studied machine scheduling problems, *the unrelated parallel machine scheduling to minimize makespan* [96]. Unrelated parallel machines (UPM) rightly capture different aspects of machine models in practice, especially of data centers. Data centers are massively parallel computation repository, where machines can show a significant diversity in mem-

ory, processing power, speed etc. Similarly, on UPM, *machines are unrelated*, and processing time of jobs can be machine-dependent. Related to UPM is *the generalized assignment problem* (GAP) [122] that has spurred many advances in approximation algorithm theory.

In this thesis, we consider two natural generalizations of UPM scheduling and GAP, *scheduling with capacity constraints and outliers* [118]. Often servers have limits on the number of jobs that can be scheduled on them – this leads to the capacitated scheduling problem. On the other hand, outliers consume substantial amount of resources and dropping them selectively may improve system performance by a large margin. Energy consumption is a vital issue in data centers. The data centers are provisioned to handle high work loads during peak demand periods. However, the work load on modern cloud computing platform is very cyclical with infrequent peaks and deep valleys. Thus, much energy can be saved, if machines can be shut down selectively during the low work period. Along this observation, we propose a novel model for energy savings in data centers; this introduces a new dimension of energy minimization in UPM scheduling [88].

Fair Allocation Problems. Allocating items fairly among individuals is an active area of research [30, 81, 106, 131, 141]. *Max-min fair allocation* is one such problem that has received significant attention from the theoretical computer science community in the last few years [15, 16, 20, 26, 34, 60]. In this problem n “indivisible” items need to be distributed among m children such that to maximize the

happiness of the least happy child. Each child has a valuation for each item and her happiness is directly proportional to the total values of the goods received by her. Interestingly, this problem can be viewed as a *maximization* version of makespan minimization on UPM. However, while a 2-approximation algorithm is known for makespan minimization, much remains to be shown for the max-min fair allocation problem. On one hand, the problem might have a constant, even a 2-approximation algorithm, on the other hand, the best known result for it is a n^ε -approximation [34], for any constant $\varepsilon > 0$. A particular type of linear programming relaxation that can be useful in this context is known as configuration LP relaxation [16, 20]. Our result in this thesis is to provide a stronger integrality gap¹ for such configuration LP relaxation of the max-min fair allocation problem.

A special case of max-min fair allocation problem is the *Santa Claus problem*, where value of each item is fixed, however a child may or may not want the item. Even, for this special case no constant factor approximation algorithm was known. Following, the work of Bansal and Sviridenko [20], Feige [60] and Feige, Asadpour, Saberi [15] show that the same configuration LP that has been used for the max-min fair allocation problem has a constant integrality gap for the Santa Claus problem. Surprisingly, both results were obtained using two different *nonconstructive* approaches and left the question of a constant factor approximation algorithm open. In this thesis, we propose a new constructive version of a powerful probabilistic tool, the Lovász Local Lemma (LLL) and using our algorithmic version of

¹For definition, see Section 1.4

LLL, we are able to provide the first constant factor approximation algorithm for the Santa Claus problem.

Network Server Allocation Problems. Resource allocation is particularly important in the area of networking. For example, in an overlay network, multiple commodities or streams must be routed from a source via reflectors to the sinks that are designated to serve the stream to end-users. Each source, reflector, sink have certain capacity, and we have to ensure the quality and reliability of the service in presence of network link failures. There is a maintenance cost to use links and reflectors, as well as a set-up cost. With limited budget, the goal is to build an overlay network which is as cheap as possible without compromising on the service. In this thesis, we study one such optimization problem related to creation of overlay networks, which improves upon the previous result of Andreev et al. [10].

The k -median problem is an optimization problem that is widely studied as a means for locating servers in networks [14, 38, 39, 84, 85]. The budget limitation allows to open only k servers, and for all requests to be served in a cost-effective manner, clients must be connected to their nearest open server. We can often assume that we have a metric space and the k -median problem is as follows: Given an n -vertex (client) metric space (V, d) and a bound k , the goal is to locate/open k centers $C \subseteq V$ so as to minimize the sum of distances of each vertex to its nearest open center. The k -median problem can also be viewed as a clustering problem. The currently best-known approximation guarantee for k -median is a

$(3 + \varepsilon)$ -approximation (for any constant $\varepsilon > 0$) due to Arya et al. [14].

A related problem appears in *content distribution networks* (CDN). In CDN, there are servers of t different types and for the i -th server-type ($i \in [1, t]$), only k_i of them can be opened. k -median is a special case with $t = 1$. For $t = 2$, Hajiaghayi et al. obtained a constant factor approximation algorithm [76]. In this thesis, we consider the problem for any arbitrary value of t , in fact, a generalization of it, which we call the *matroid-median* problem and give a constant factor approximation algorithm for it. In this problem, the open centers must form an independent set of a matroid². In k -median, the underlying matroid is a *uniform matroid*, while for the problem of server location in CDN, it is an instance of a *partition matroid*.

Ad-words Assignment in Mobile Advertising & Capacitated Covering Problems.

Almost all of us are familiar with the concept of advertisements being shown along side the search results on internet search websites. Bidders bid for each key-word and given a key-word, the search engines must decide which advertisements to show on the limited number of available slots to optimize their revenue. This leads to an array of interesting optimization problems [35, 125]. Nowadays, with world-wide usage of cellular phones, *mobile advertisement* is being considered as a viable alternative. However, in this scenario, typically users do not search for keywords and in absence of users' application context, it becomes difficult to provide targeted advertisements. The only information available to a wireless ser-

²For a general background on matroids, the reader is referred to the book [121].

vice providers (WSP) is the current location and time of users. Also, to ensure a *non-intrusive delivery methods*, WSP must send only a limited number of ads to each user. This introduces a “capacity” constraints on the number of advertisements that could be shown to each user, and makes the process different from usual internet advertising. In this thesis, we propose a model for mobile advertising, and derive new algorithmic results for it.

Indeed, capacitated problems arise naturally in many applications where there are resource constraints. In this thesis, we study some extremely well-known problems such as vertex cover and set cover in presence of hard-capacity limitations and show their applications in the area of resource allocation.

In the following sections of this introductory chapter, we briefly describe several concepts used throughout the dissertation. Towards the end of this chapter, a road map is given to guide the reader through different parts of the thesis.

1.3 Probabilistic Methods

The term *probabilistic methods* loosely refers to the use of identities and inequalities from probability theory to prove combinatorial statements. We construct an appropriate probability space and show that a randomly chosen element in this space has the desired property with probability greater than zero. This proves the existence of at least one object in the collection that has the property. Often a stronger claim can be made. It can be shown that an element with the desired property does

not only exist, but can be found with high probability if we make a few (polynomial in number) random choices. This then leads to a randomized Monte-Carlo algorithm.

Let us illustrate few concepts of probabilistic methods via an example that will be useful later.

Example 1. Consider k -satisfiability, in short, a k -SAT formula with m clauses and n boolean variables in conjunctive normal form. The goal is to decide whether there exists an assignment of the boolean variables that satisfy the formula.

Proposition 1.3.1. *If $m < 2^k$, then there exists an assignment of the boolean variables for any k -SAT formula with m clauses and n variables.*

Proof. Set each boolean variable to 1 with probability $1/2$ and 0 with probability $1/2$ independently. The probability that a particular clause is unsatisfiable is therefore $\frac{1}{2^k}$. Thus, the probability that there exists at least one clause which is unsatisfied by the random assignment is at most $\frac{m}{2^k}$ by *union bound*. Since $m < 2^k$, probability of a failure is strictly less than 1, thus the probability that there exists a good assignment is nonzero. Hence, a good assignment exists. \square

Given Proposition 1.3.1, a natural question arises. *What happens when $m \geq 2^k$?* The union bound is clearly not enough to provide any guarantee in this situation. Consider the case when all clauses are disjoint, that is, the random variables designating whether a clause is satisfied or not are mutually independent. In that case, irrespective of the value of m , there exists a good assignment. There is no

way to encode such *independence information* in the union bound. It is possible that each clause shares variables with only a few other clauses. Under such limited dependency scenario, a particular probabilistic method that becomes very useful is the *Lovász Local Lemma* (LLL).

Lemma 1.3.2 (The Local Lemma; Symmetric Case). *Let A_1, A_2, \dots, A_n be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of a set of all the other events A_j but at most d , and that $\Pr[A_i] \leq p$ for all $1 \leq i \leq n$. If*

$$ep(d+1) \leq 1 \tag{1.1}$$

then $\Pr[\bigcap_{i=1}^n \bar{A}_i] > 0$.

The above is the symmetric version of LLL. The more general version of local lemma is described in Chapter §3. Let us now illustrate the power of LLL via the same example of *k-SAT*.

Proposition 1.3.3. *In a k -SAT formula with m clauses and n boolean variables, if each clause shares variables with at most $\frac{2^k}{e} - 1$ other clauses, then there exists an assignment of random variables that satisfies all the clauses, or in other words, the k -SAT formula is satisfiable.*

Proof. Consider a random variable for each boolean variable and set it to 1 with probability $1/2$ and 0 with probability $1/2$ randomly and independently. Define a bad event A_i for each unsatisfied clause i , for $i = 1, 2, \dots, m$. Then $\Pr[A_i] = \frac{1}{2^k}$.

Now each clause shares variables with at most $2^k/e - 1$ other clauses. Therefore, each bad event depends only on at most $2^k/e - 1$ other bad events. Hence, we have $d = 2^k/e - 1$, and $ep(d + 1) = e \cdot \frac{1}{2^k} (\frac{2^k}{e} - 1 + 1) = 1$. Thus from the Symmetric version of LLL, we know that there exists an assignment of the boolean variables which satisfies all the clauses. \square

The condition of LLL guarantees existence of a good assignment, but it does not tell us how to find such an assignment. This led to a long series of research in developing an algorithmic version of LLL. In Chapter §3, we will study some interesting new developments in that direction.

Another way to use probabilistic method is by calculating the expected value of some random variables. It can often be claimed that the value of a random variable is concentrated around its expectation. One may use a variety of concentration inequalities to establish such claims. The Chernoff-Hoeffding bound is one such widely used concentration inequality that will be applied frequently in this thesis.

Lemma 1.3.4 (The Chernoff-Hoeffding Bound). *Let X_1, X_2, \dots, X_n be n bounded independent random variables with $X_i \in [0, 1]$, for all $i = 1, 2, \dots, n$. Then if $X = \sum_{i=1}^n X_i$ and $\mu = E[X]$, we have*

1. for any $\delta > 0$, $\Pr[|X - \mu| \geq \delta\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$,
2. for $0 < \delta \leq 1$, $\Pr[X > \mu(1 + \delta)] \leq e^{-\mu/3\delta^2}$,
3. for $0 < \delta \leq 1$, $\Pr[X < \mu(1 - \delta)] \leq e^{-\mu/2\delta^2}$.

1.4 Linear Programming Relaxation and Rounding

Linear programming is the problem of optimizing (i.e., minimizing or maximizing) a linear objective function subject to linear inequality constraints. Any solution, i.e., a setting for the variables in the linear program is said to be feasible if it satisfies all the constraints. On the other hand, a solution is optimal, if it is feasible and optimizes (i.e., maximize or minimize) the objective function. A large number of combinatorial optimization problems can be modeled as linear programs, and in most cases, the variables must satisfy integrality constraints. Such integrality constraints (integer linear program) make it **NP**-hard to obtain an optimal solution. However, if we relax the variables to take fractional values, then it is possible to optimally solve the linear program. For example, if a variable must take values either 0 or 1, we can relax it to take any fractional value in $[0, 1]$. This is known as *linear programming relaxation* (LP-relaxation). The optimal objective value of a LP-relaxation serves as a upper (lower) bound for the true optimal objective value in case of a maximization (minimization) problem. The fractional optimal solution is then rounded to an integer solution with an objective value as close to the objective value of the LP-relaxation. An important concept in this regard is that of *integrality gap*. Integrality gap of a LP-relaxation for any problem is the ratio between the true optimal and the LP optimal objective value. The best possible approximation factor achievable through a LP rounding method is bounded by its

integrality gap.

Let us illustrate the concept of LP rounding and integrality gap through an extremely well-studied combinatorial problem, namely, the *set cover* problem.

Example 2 (Set Cover). Given a universe U of n elements, a collection of subsets of U , $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$, and a cost function $c : \mathcal{S} \rightarrow \mathbb{Q}^+$, find a minimum cost subcollection of \mathcal{S} that covers all elements of U .

To formulate the set cover problem as an ILP, we assign an indicator variable x_S for each set S , such that it will be 1, if the set S is selected and 0 otherwise. For each element, we have a constraint denoting that we must pick at least one set containing the element in our solution.

$$\text{minimize } \sum_{S \in \mathcal{S}} x_S \quad (\text{SetCover}_{\text{ILP}})$$

$$\text{subject to } \sum_{a \in S} x_S \geq 1 \quad \forall a \in U \quad (1.2)$$

$$x_S \in \{0, 1\} \quad \forall S \in \mathcal{S} \quad (1.3)$$

The LP-relaxation of this ILP is obtained by letting the domain of variable x_S be $1 \geq x_S \geq 0$. Since the upper bound on x_S is redundant, we get the following LP.

$$\text{minimize } \sum_{S \in \mathcal{S}} x_S \quad (\text{SetCover}_{\text{LP}})$$

$$\text{subject to } \sum_{a \in S} x_S \geq 1 \quad \forall a \in U \quad (1.4)$$

$$x_S \geq 0 \quad \forall S \in \mathcal{S} \quad (1.5)$$

Proposition 1.4.1 ([135]). *The integrality gap of $\text{SetCover}_{\text{LP}}$ is at least $\frac{\log n}{2}$.*

Proof. Consider the following set cover instance. Let $n = 2^k - 1$, where k is a positive integer and let $U = \{e_1, e_2, \dots, e_n\}$. We view elements e_1 to e_n as n k -dimensional vectors over $GF[2]$, except the all zero vector. We construct n sets S_1, S_2, \dots, S_n such that $S_i = \{e_j | e_j \cdot e_i = 1\}$. Now it is easy to check that each element belongs to exactly $n/2$ sets. Thus if we set $x_S = \frac{2}{n}$ for all $S \in \mathcal{S}$, the constraints of $\text{SetCover}_{\text{LP}}$ are satisfied and we get a feasible LP solution of value 2. On the other hand, any integer optimal solution must pick at least k sets. This can be seen as follows. Consider any $p < k$ sets and let i_1, i_2, \dots, i_p be the indices of the p sets. Consider the $p \times k$ dimensional matrix A , where the j -th row is the k dimensional binary vector i_j . Now the null space of A is non-empty and therefore there is a vector $e_l \neq \mathbf{0}$ such that $A \cdot e_l = 0$. Clearly, the element e_l is not contained in any of the p sets. Thus at least $k = \log(n + 1)$ sets are required to cover all the elements. Hence the integrality gap is at least $\frac{\log(n+1)}{2} > \frac{\log n}{2}$. \square

Now, we will show a rounding algorithm that achieves an approximation factor of $O(\log n)$ [135].

Let \mathbf{x}^* be a fractional optimal solution of SetCover_{LP} . We pick each set S with probability x_S^* . Let \mathcal{C} be the collection of sets picked. The expected cost of \mathcal{C} is

$$\mathbb{E}[c(\mathcal{C})] = \sum_{S \in \mathcal{S}} \Pr[S \text{ is picked}] c(S) = \sum_{S \in \mathcal{S}} x_S^* c(S) = OPT_f.$$

Here OPT_f denotes the value of the objective function for the optimal solution of SetCover_{LP} .

Now the element a is covered by the sets in \mathcal{C} if at least one set containing a is picked. Thus,

$$\Pr[a \text{ is covered by } \mathcal{C}] \geq 1 - \prod_{S: a \in S} (1 - x_S^*)$$

Suppose, a belongs to k sets, then $\Pr[a \text{ is covered by } \mathcal{C}] \geq 1 - \prod_{S: a \in S} (1 - x_S^*) \geq 1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e}$. Thus the probability that a is not covered is at most $\frac{1}{e}$.

To get a complete set cover, we independently pick $2 \ln n$ such subcollections, and compute their union. Now, the probability that an element a is not covered in any of the $2 \ln n$ subcollections is at most $(\frac{1}{e})^{2 \ln n} = \frac{1}{n^2}$.

Thus, by union bound, after $2 \ln n$ rounds the probability that there exists at least one element that is not covered is at most $\frac{1}{n}$. On the other hand, the expected cost of the overall solution is at most $2 \ln n OPT_f$. By Markov's inequality, prob-

ability that the expected cost is more than $4 \ln n \text{OPT}_f$ is at most $\frac{1}{2}$.

Thus, with probability at least $\frac{1}{2} - \frac{1}{n}$, we obtain a valid set cover with cost at most $4 \ln n = O(\log n)$ times the optimal cost.

Observe that, in polynomial time, we can verify if the solution computed is a feasible set cover solution or not. If not, then we simply repeat the entire procedure. The expected number of times that we need to repeat is less than 3.

It is not possible to obtain an approximation algorithm for set cover with approximation factor $o(\log n)$ unless $\mathbf{P} = \mathbf{NP}$ [59, 99]. Thus, no polynomial time algorithm can achieve an approximation factor better than the aforementioned LP rounding algorithm by more than a constant factor. The best known approximation factor for set cover is due to [128].

1.5 Structure of Linear programs

For set cover, the best approximation ratio achievable in polynomial time is $\log n$. However, in many situations, especially those arising in geometric settings, the set systems have more structure in it [32, 36, 134]. Consequently, the $\log n$ -hardness no longer applies to such set-systems. Consider for example, *covering points by line segments in one dimension*. An element (point) is covered by a line segment, if its x -coordinate is contained in the segment. This particular instance of the set cover problem is solvable in polynomial time. If we write the constraint matrix of the linear program for this set cover instance, the matrix satisfies a property known

as *total unimodularity* [53]. This property of total unimodularity of the constrained matrix enables to obtain an integer optimal solution of the LP in polynomial time. Thus, structure of linear program can often be a crucial ingredient in obtaining better approximation algorithms. Note that for covering points by line segments, one can obtain an optimal solution using dynamic programming as well.

While we can solve linear programs to obtain optimal solution, we can also obtain extreme point solution, that is, the optimal solution that appears in one of the vertices of the linear polytope. An extreme point solution cannot be written as a convex combination of other solutions. Consider a linear program with n variables and m constraints. A feasible solution of such a LP will be an extreme point solution if and only if the number of linearly independent constraints that are satisfied tightly (i.e., with equality) is exactly equal to n . One can exploit the structure of an extreme point solution while designing rounding algorithms. In Chapter §2, our new rounding methodology relies on such structural properties of LP solutions.

1.6 Organization

Apart from this introductory chapter, the dissertation is divided into two main parts. The first part is devoted to developing techniques, which are used in different applications in the second half. In Chapter §2, we develop our new LP rounding technique. Chapter §3 describes our contribution towards algorithmic aspects of

the Lovász Local Lemma. Starting from Chapter §4, in the second half of the thesis, a large variety of different applications are described. While in most of them, the main techniques used are those developed in Chapter §2 and §3, several problem specific innovations are required as well.

Chapter §4 and §5 are devoted towards applications of scheduling in parallel machines. In Chapter §5, we consider two generalizations of UPM scheduling to minimize makespan. The first generalization is to consider hard capacity constraints on machines, and the second generalization is to consider outliers. In Chapter §5, we develop new model for energy minimization in scheduling, and describe our algorithmic contribution.

Chapter §6 and §7 describe the *max-min* fair allocation problem and the Santa Claus problem respectively. In Chapter §6, we show how using our rounding technique, one can obtain better integrality gap for a particular configuration LP for the max-min fair allocation problem. In Chapter §7, we develop the first constant factor approximation algorithm for the Santa Claus problem.

Chapter §8 and §9 are devoted towards two applications of networking, an overlay network design problem and a problem of locating servers in content distribution networks. In Chapter §9, we will see how structural properties of linear program can help us to obtain a good approximation algorithm.

In Chapter §10, we develop a model of advertisement allocation in cellular networks, and show how to abstract the problem using a linear program and obtain

an approximation algorithm for it. In Chapter §11, we consider the vertex cover and set cover problem with hard capacity bounds on vertices and sets respectively.

The new linear programming rounding method of Chapter §2 and its related applications in Chapter §4, §6 and §8 appeared in [118]. The constructive version of the Lovász Local Lemma (Chapter §3) and the result on the Santa Claus problem appeared in [75]. The scheduling work to minimize energy on data centers was published in [88]. The materials of Chapter §9 and Chapter §10 appeared respectively in [91] and [1].

Part I

Techniques

A New Linear Programming Rounding Method

A large class of optimization problems can be modeled as integer linear programs (ILP). However solving an ILP optimally is NP-hard and in contrast its linear-programming (LP) relaxation is solvable in polynomial time. This has resulted in the much-used paradigm of “relax-and-round” wherein an ILP is relaxed to get an LP and an efficiently-computed optimal solution of the LP is rounded to an integer solution that *approximates* the true ILP-optimum. This second “rounding” step is often a crucial ingredient, and many general techniques have been developed for it.

In Chapter 1, we have seen a simple example of LP rounding method for set cover problem. In that problem, random variables are rounded independently. The method, known as *independent randomized rounding*, was first proposed by Raghavan and Thompson [116]. In many applications, however such independent rounding procedure fails to guarantee any good approximation. As a result, a new generic technique of *dependent rounding* has been developed in the last decade [2, 65, 92, 127]. Our proposed rounding methodology makes progress in

that direction. We start with a description of dependent rounding and then elaborate on our contribution.

2.1 The Dependent Randomized Rounding

Dependent rounding is a generic rounding methodology which has been proven quite effective in variety of applications of scheduling, packet-routing and in general in combinatorial optimization [2, 65, 92, 127]. During any rounding procedure, we map an LP-optimal fractional solution x to a nearby integer solution X . When the rounding is done *independently*, we typically choose a value α that is problem-specific, and, for each i , define X_i to be 1 with probability αx_i , and to be 0 with the complementary probability of $1 - \alpha x_i$. Independence can, however, lead to noticeable deviations from the mean for random variables that are *required* to be very close to (or even be equal to) their mean. A fruitful idea developed in [65, 92, 127] is to carefully introduce *dependencies* into the rounding process: in particular, some sums of random variables are held fixed with probability one, while still retaining randomness in the individual variables. This is the premise of dependent randomized rounding. See [2] for a related deterministic approach that precedes these works.

In [65], dependent randomized rounding procedure was developed for a simple setting of bipartite graphs, which we describe next.

2.1.1 Dependent Rounding on Bipartite Graphs

Suppose we are given a bipartite graph $G = (A, B, E)$ with the bipartition (A, B) and the edges E . There is a variable $x_{i,j} \in [0, 1]$ associated with each edge $(i, j) \in E$. Dependent rounding on the bipartite graph is a randomized polynomial-time scheme that rounds each $x_{i,j} \in [0, 1]$ to $X_{i,j} \in \{0, 1\}$, that satisfies the following three properties:

(P1) **Marginal distribution.** For every edge (i, j) , $\Pr[X_{i,j} = 1] = x_{i,j}$. This property can be ensured through simple independent rounding of [116] as well. As a corollary of it,

$$\forall (i, j) \in E, \mathbf{E}[X_{i,j}] = x_{i,j}. \quad (2.1)$$

(P2) **Degree-preservation.** For every vertex $i \in A \cup B$, if the fractional degree $\sum_{j \in A \cup B} x_{i,j} = d$, then with probability 1, the integral degree $\sum_{j \in A \cup B} X_{i,j} \in \{\lceil d \rceil, \lfloor d \rfloor\}$.

This is a property, very specific to dependent rounding and often plays a vital role when certain cardinality constraints need to be maintained.

(P3) **Negative correlation.** For any vertex i and any subset of edges S ,

$$\forall b \in \{0, 1\}, \Pr[\prod_{(i,j) \in S} (X_{i,j} = b)] \leq \prod_{(i,j) \in S} \Pr[(X_{i,j} = b)] \quad (2.2)$$

This property enables us to use “Chernoff” type bounds on each vertex. The Chernoff-Hoeffding bound can be applied for sum of random variables that are negatively correlated.

The rounding starts by selecting all the edges that already have their x values in $\{0, 1\}$ and setting the corresponding $X_{i,j}$ values equal to $x_{i,j}$. All these integral-valued edges are removed from the bipartite graph. Thus the edge (i, j) is in E if and only if $x_{i,j} \in (0, 1)$. Next the algorithm chooses an even cycle \mathcal{C} (note that in a bipartite graph every cycle has even length) or a maximal path \mathcal{P} in G , and partitions the edges in \mathcal{C} or \mathcal{P} into two matchings \mathcal{M}_1 and \mathcal{M}_2 . Then, two positive scalars α and β are defined as follows:

$$\alpha = \min\{\eta > 0 : ((\exists(i, j) \in \mathcal{M}_1 : x_{i,j} + \eta = 1) \bigcup (\exists(i, j) \in \mathcal{M}_2 : x_{i,j} - \eta = 0))\};$$

$$\beta = \min\{\eta > 0 : ((\exists(i, j) \in \mathcal{M}_1 : x_{i,j} - \eta = 0) \bigcup (\exists(i, j) \in \mathcal{M}_2 : x_{i,j} + \eta = 1))\};$$

Now with probability $\frac{\beta}{\alpha + \beta}$, set

$$Y_{i,j} = x_{i,j} + \alpha \text{ for all } (i, j) \in \mathcal{M}_1$$

and $Y_{i,j} = x_{i,j} - \alpha \text{ for all } (i, j) \in \mathcal{M}_2;$

with complementary probability of $\frac{\alpha}{\alpha+\beta}$, set

$$Y_{i,j} = x_{i,j} - \beta \text{ for all } (i,j) \in \mathcal{M}_1$$

and $Y_{i,j} = x_{i,j} + \beta \text{ for all } (i,j) \in \mathcal{M}_2;$

Thus at least one $x_{i,j}$ is rounded to 0 or 1. Hence the rounding algorithm always make progress and in $O(|E|)$ iterations all the edges are rounded to integral value.

The above rounding scheme satisfies the three properties: marginal distribution, degree-preservation and the negative correlation specified above.

2.1.2 A Negative Correlation Property

We now show a useful negative correlation property for dependent rounding on bipartite graphs for the case of *matching*.

Definition 2.1.1 (Negative Correlation for Indicator Random Variables). *A collection of indicator random variables $\{z_i\}, i \in [1, n]$ are said to be negatively correlated, if for any subset of t variables, $t \in [1, n]$, and any $b \in \{0, 1\}$,*

$$\Pr \left[\bigwedge_{j=1}^t z_{i_j} = b \right] \leq \prod_{j=1}^t \Pr [z_{i_j} = b].$$

Theorem 2.1.2. *Define an indicator random variable z_j for each vertex $j \in B$ (similarly for vertices in A), such that $z_j = 1$ if item j is matched. Then, the indicator random variables $\{z_j\}$ are negatively correlated.*

Proof. Consider any collection of vertices j_1, j_2, \dots, j_t belonging to B . Let $b = 1$

(the proof for the case $b = 0$ is identical). Let $Y_{i,j,k}$ denote the value of $Y_{i,j}$ at the beginning of the k -th iteration of bipartite dependent rounding. Define, $z_{j,k} = \sum_{i,(i,j) \in E} Y_{i,j,k}$. Clearly, $z_j = \sum_{i,(i,j) \in E} Y_{i,j,|E|+1}$. We will show that

$$\forall k, \mathbb{E} \left[\prod_{i=1}^t z_{j_i,k} \right] \leq \mathbb{E} \left[\prod_{i=1}^t z_{j_i,k-1} \right] \quad (2.3)$$

Thus, we will have

$$\begin{aligned} \Pr \left[\bigwedge_{i=1}^t z_{j_i} = 1 \right] &= \mathbb{E} \left[\prod_{i=1}^t z_{j_i,|M|+1} \right] \\ &\leq \mathbb{E} \left[\prod_{i=1}^t z_{j_i,1} \right] \\ &= \prod_{i=1}^t \sum_v Y_{v,j_i,1} = \prod_{i=1}^t \Pr [z_{j_i} = 1] \end{aligned}$$

We now prove (2.3) for a fixed k . Note that any vertex that is not the end point of the maximal path or the cycle on which dependent rounding is applied on the $(k-1)$ -th round retains their previous z value. There are three cases to consider.

Case 1: *Two vertices among j_1, j_2, \dots, j_t have their values modified.* Let these vertices be say j_1 and j_2 . Therefore, these two vertices must be the end points of the maximal path on which dependent rounding is applied on the $(k-1)$ -th round. The path length must be even. Let $B(j_1, j_2, \alpha, \beta)$ denote the event that the jobs

$\{j_1, j_2\}$ have their values modified in the following probabilistic way:

$$(z_{j_1, k}, z_{j_2, k}) = \begin{cases} (z_{j_1, k-1} + \alpha, z_{j_2, k-1} - \alpha) & \text{with probability } \frac{\beta}{\alpha + \beta} \\ (z_{j_1, k-1} - \beta, z_{j_2, k-1} + \beta) & \text{with probability } \frac{\alpha}{\alpha + \beta} \end{cases}$$

Thus

$$\begin{aligned} & \mathbb{E} \left[\prod_{i=1}^t z_{j_i, k} \mid \forall i \in [1, t], z_{j_i, k-1} = a_{j_i} \wedge B(j_1, j_2, \alpha, \beta) \right] \\ &= \mathbb{E} [z_{j_1, k} z_{j_2, k} \mid \forall i \in [1, t], z_{j_i, k-1} = a_{j_i} \wedge B(j_1, j_2, \alpha, \beta)] \prod_{i=3}^t a_{j_i} \end{aligned}$$

The above expectation can be written as $(\psi + \phi) \prod_{i=3}^t a_{j_i}$, where

$$\psi = (\beta / (\alpha + \beta))(a_{j_1} + \alpha)(a_{j_2} - \alpha), \text{ and}$$

$$\phi = (\alpha / (\alpha + \beta))(a_{j_1} - \beta)(a_{j_2} + \beta).$$

Now, it can be easily seen that $\psi + \phi \leq a_{j_1} a_{j_2}$. Thus for any fixed j_1, j_2 and

for any fixed (α, β) , and for fixed values a_f the following holds:

$$\mathbb{E} \left[\prod_{i=1}^t z_{j_i, k} \mid \forall i \in [1, t], z_{j_i, k-1} = a_{j_i} \wedge B(j_1, j_2, \alpha, \beta) \right] \leq \prod_{i=1}^t a_{j_i}.$$

Hence, $\mathbb{E} [\prod_{i=1}^t z_{j_i, k} \mid \text{Case 1}] \leq \mathbb{E} [\prod_{i=1}^t z_{j_i, k-1} \mid \text{Case 1}]$.

Case 2: *One vertex among j_1, j_2, \dots, j_t has its value modified.* Let the vertex be j_1 say. Therefore, this vertex must be the end point of the maximal path on which dependent rounding is applied on the $(k-1)$ -th round. The path length must be odd. Let $B(j_1, \alpha, \beta)$ denote the event that the job j_1 has its value modified in the following probabilistic way:

$$z_{j_1, k} = \begin{cases} z_{j_1, k-1} + \alpha & \text{with probability } \frac{\beta}{\alpha + \beta} \\ z_{j_1, k-1} - \beta & \text{with probability } \frac{\alpha}{\alpha + \beta} \end{cases}$$

Thus,

$$\mathbb{E}[z_{j_1, k} \mid \forall i \in [1, t], z_{j_i, k-1} = a_{j_i} \wedge B(j_1, \alpha, \beta)] = a_{j_1}.$$

Since the values of $z_{j_i}, i \in [2, t]$ remains unchanged and the above equation holds for any j_1, α, β , we have $\mathbb{E}[\prod_{i=1}^t z_{j_i, k} \mid \text{Case 2}] \leq \mathbb{E}[\prod_{i=1}^t z_{j_i, k-1} \mid \text{Case 2}]$.

Case 3: *None among j_1, j_2, \dots, j_t has its value modified.*

In this case, the value of $z_{j_i, k}$'s, $i \in [1, t]$, do not change. Hence, $\mathbb{E}[\prod_{i=1}^t z_{j_i, k} \mid \text{Case 3}] \leq \mathbb{E}[\prod_{i=1}^t z_{j_i, k-1}]$.

This establishes the claim. \square

We can extend the bipartite dependent rounding method to weighted depen-

dent rounding such that the three properties of dependent rounding [(PI), (PII) and (PIII)] hold. As a result, we can also obtain negative correlation bound like Theorem 2.1.2 for weighted bipartite matching. We leave these as exercises to the readers.

2.2 Generalization of Dependent Rounding for Arbitrary Linear Systems

The above bipartite dependent rounding scheme can be extended to any arbitrary system of linear constraints. Suppose we are given a system of linear equations, $Ax \leq b, x \in [0, 1]^n$, where A is an $m \times n$ matrix and b is an m -dimensional vector. Also for notational simplicity, assume x is some given solution to this system of linear equations and thus it satisfies $Ax = b'$, for some $b' \leq b$. Once again, our goal is to obtain a feasible solution $X \in \{0, 1\}^n$ satisfying $AX = b'$.

First, as in the bipartite-dependent-rounding, if any $x_j, j \in [n]$, already has a value in $\{0, 1\}$, we set $X_j = x_j$ permanently. Next we project x to only those coordinates that are in $(0, 1)$ and thus consider a reduced system of equations $A'x' = b'$. Suppose the linear system, $A'x' = b'$, is underdetermined, i.e., the number of linearly independent constraints are less than the number of variables. Then since the null-space of A' is non-trivial, we can efficiently find a non-zero vector r , such that $A'r = 0$. Since $x \in (0, 1)^n$, we can also find strictly-positive scalars α and β such that: 1) all entries of $x + \alpha r$ and $x - \beta r$ lie in $[0, 1]$ and, 2) at least one entry of

$x + \alpha r$ and $x - \beta r$ becomes either 0 or 1. With probability $\frac{\beta}{\alpha + \beta}$, we set $Y = x + \alpha r$ and with complementary probability $\frac{\alpha}{\alpha + \beta}$, we set $Y = x - \beta r$. Since at least one new variable is set to an integer, the rounding terminates after at most n rounds. Let X' denote the final-rounded solution corresponding to x' , then the rounding satisfies the following two properties:

(P'1) **Marginal distribution.** For $j \in [n]$, $\Pr[X_j = 1] = x_j$. Thus, $\forall j \in [n]$, $\mathbb{E}[X_j] = x_j$.

(P'2) **Linear-constraint-preservation.** $\Pr[A'X' = b'] = 1$

It is important to ensure that the linear-system is underdetermined in order to guarantee the existence of a non-trivial vector in the null-space of A . When the system becomes determined, depending on the application, we may select suitable constraints to drop such that the system again becomes underdetermined.

Bipartite dependent rounding can be viewed as a special case of this approach. When we select a maximal path in bipartite dependent rounding, for each intermediate vertex we maintain the sum of the fractional values of the edges incident on them. Thus if the maximal path has s vertices, then we have $(s - 1)$ variables and $(s - 2)$ constraints. On a cycle, there are s variables and s constraints, but one of the constraint is linearly dependent on the rest and thus can be ignored.

2.3 Randomized Iterative Relaxation

We further generalize the above approach. This leads to a rounding methodology that not only falls under the umbrella of dependent rounding, but can be best categorized as *randomized iterative relaxation*. This establishes the connection between two, probably the most effective, rounding methods: the dependent rounding and the iterative relaxation procedure.

The general paradigm of iterative rounding was first developed by Jain for approximating survivable network design problems [83]. The basic idea in iterative rounding method is to consider a basic solution of a linear programming relaxation and utilizing the property of a basic solution to show that a variable exists with fractional values at least say $1/2$. These almost integral variables are then rounded up to the nearest integer, and fixing these variables the remaining linear program is solved once again. The procedure continues iteratively unless all the variables are rounded to integers. Since the basic iterative rounding technique loses a constant factor in the approximation, the technique is extended by interleaving *relaxation* steps, dropping constraints without compromising too much in the feasibility, and rounding iteratively [67, 94, 123].

We generalize the methods of [94, 123] as well as that of [2, 65, 87, 92, 127], via a type of random walk toward a vertex of the underlying polytope that we outline next.

Suppose we are given a polytope \mathcal{P} in n dimensions, and a *non-vertex* point x belonging to \mathcal{P} . An appropriate basic-feasible solution will of course lead us to a vertex of \mathcal{P} , but we approach (not necessarily reach) a vertex of \mathcal{P} by a random walk as follows. Let \mathcal{C} denote the set of constraints defining \mathcal{P} which are satisfied *tightly* (i.e., with equality) by x . Then, note that there is a non-empty linear subspace S of \mathbb{R}^n such that for any nonzero $r \in S$, we can travel up to some strictly-positive distance $f(r)$ along r starting from x , while staying in \mathcal{P} and *continuing* to satisfy all constraints in \mathcal{C} tightly. Our broad approach to conduct a random move $Y := x + R$ by choosing an appropriately random R from S , such that the property “ $\mathbb{E} [Y_j] = x_j$ ” of the previous paragraph still holds. In particular, let **RandMove** (x, \mathcal{P}) – or simply **RandMove** (x) if \mathcal{P} is understood – be as follows. Choose a nonzero $r \in S$ arbitrarily, and set $Y := x + f(r)r$ with probability $f(-r)/(f(r) + f(-r))$, and $Y := x - f(-r)r$ with the complementary probability of $f(r)/(f(r) + f(-r))$. Note that if we repeat **RandMove**, we obtain a random walk that finally leads us to a vertex of \mathcal{P} ; the high-level idea is to intersperse this walk with the idea of “judiciously dropping some constraints” as well as combining certain constraints together into one.

To contrast with the dependent rounding procedure for linear systems, note that, here *tightly* satisfied constraints play a vital role. As long as a constraint is not tightly satisfied, we can simply ignore it while determining whether the system is underdetermined. We take a **RandMove** and make progress until one of the

non-tight constraint becomes tight or some variable gets rounded.

Interspersing the random walk towards a vertex matches the *relaxation* step of the iterative relaxation method. However in iterative relaxation, a linear program is solved repetitively in order to get a basic solution at every step—here instead we make a random choice of vector r and pick a direction to proceed along r probabilistically. This procedure thus satisfies the property “ $E[Y_j] = x_j$ ” and depending on the polytope may lead to *negative correlation* type results as in basic bipartite dependent rounding technique.

As discussed later in the thesis, our recipe appears fruitful in a number of directions in scheduling, fair allocation, budgeted adword allocation, and as a new rounding technique in general.

Constructive Aspects of the Lovász Local Lemma

The well-known Lovász Local Lemma (LLL) [55] is a powerful probabilistic approach to prove the existence of certain combinatorial structures. Its diverse range of applications include breakthroughs in packet-routing [95], a variety of theorems in graph-coloring including list coloring, frugal coloring, total coloring, and coloring graphs with lower-bounded girth [109], as well as a host of other applications where probability appears at first sight to have no role [9]. Furthermore, almost all known applications of the LLL have no alternative proofs known. While the original LLL was non-constructive – it was unclear how the existence proofs could be turned into polynomial-time algorithms – a series of works [5, 27, 50, 108–112, 129] beginning with Beck [27] and culminating with the breakthrough of Moser & Tardos (MT) [111] have led to efficient algorithmic versions for most such proofs. However, there are several LLL applications to which these approaches inherently

cannot apply. In this chapter, we make progress toward bridging this gap, by uncovering and exploiting new properties of [111].

3.1 Preliminaries & Limitation of MT Algorithm

In Chapter 1, we have seen the symmetric version of LLL (Theorem 1.3.2). Let \mathcal{P} be a collection of n mutually independent random variables $\{P_1, P_2, \dots, P_n\}$, and let $\mathcal{A} = \{A_1, A_2, \dots, A_m\}$ be a collection of m (“bad”) events, each determined by some subset of \mathcal{P} . For any event B that is determined by a subset of \mathcal{P} we denote the smallest such subset by $\text{vbl}(B)$. For any event B that is determined by the variables in \mathcal{P} , we furthermore write $\Gamma(B) = \Gamma_{\mathcal{A}}(B)$ for the set of all events $A \neq B$ in \mathcal{A} with $\text{vbl}(A) \cap \text{vbl}(B) \neq \emptyset$. This neighborhood relation induces the following standard *dependency graph* or *variable-sharing graph* on \mathcal{A} : For the vertex set \mathcal{A} let $G = G_{\mathcal{A}}$ be the undirected graph with an edge between events $A, B \in \mathcal{A}$ iff $A \in \Gamma(B)$. We often refer to events in \mathcal{A} as *bad events* and want to find a point in the probability space, or equivalently an assignment to the variables \mathcal{P} , wherein none of the bad events happen. We call such an assignment a **good assignment**.

With these definitions the general (“asymmetric”) version of the LLL simply states:

Theorem 3.1.1 (Asymmetric Lovász Local Lemma). *With \mathcal{A}, \mathcal{P} and Γ defined as*

above, if there exists an assignment of reals $x : \mathcal{A} \rightarrow (0, 1)$ such that

$$\forall A \in \mathcal{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B)); \quad (3.1)$$

then the probability of avoiding all bad events is at least $\prod_{A \in \mathcal{A}} (1 - x(A)) > 0$ and thus there exists a good assignment to the variables in \mathcal{P} .

The LLL (Theorem 3.1.1) shows sufficient conditions under which, with positive probability, none of the events A_i holds: i.e., that there is a choice of values for the variables in \mathcal{P} (corresponding to a discrete structure such a suitable coloring of a given graph) that avoids all the A_i . Under these same sufficient conditions, MT shows the following very simple algorithm to make such a choice: (i) initially choose the P_i independently from their given distributions; (ii) *while* the current assignment to \mathcal{P} does not avoid all the A_i , *repeat*: arbitrarily choose a currently-true A_i , and resample, from their product distribution, the variables in \mathcal{P} on which A_i depends. The amazing aspect of MT is that the expected number of resamplings is small [111]: at most $\text{poly}(n, m)$ in all known cases of interest. However, there are two problems with implementing MT, that come up in some applications of the LLL:

- (a) the number of events m can be superpolynomial in the number of variables n ; this can result in a superpolynomial running time in the “natural” parameter n ¹; and, even more seriously,

¹ n is the parameter of interest since the output we seek is one value for each of P_1, P_2, \dots, P_n .

(b) given an assignment to \mathcal{P} , it can be computationally hard (e.g., NP-hard or yet-unknown to be in polynomial time) to either certify that no A_i holds, or to output an index i such that A_i holds.

Since detection and resampling of a currently-bad event is the seemingly unavoidable basic step in the MT algorithm, these applications seemed far out of reach. We deal with a variety of applications wherein (a) and/or (b) hold, and develop Monte Carlo (and in many cases, *RNC*) algorithms whose running time is polynomial in n : some of these applications involve a small loss in the quality of the solution. (We loosely let “*RNC* algorithms” denote randomized parallel algorithms that use $\text{poly}(n)$ processors and run in $\text{polylog}(n)$ time, to output a correct solution with high probability.) First we show that the MT algorithm needs only $O(n^2 \log n)$ many resampling steps in all applications that are known (and in most cases $O(n \cdot \text{polylog}(n))$), even when m is superpolynomial in n . This makes those applications constructive that allow an *efficient* implicit representation of the bad events (in very rough analogy with the usage of the ellipsoid algorithm for convex programs with exponentially many constraints but with good separation oracles). Still, most of our applications have problem (b). For these cases, we introduce a new proof-concept based on the (*conditional*) *LLL-distribution* – the distribution D on \mathcal{P} that one obtains when conditioning on no A_i happening. Some very useful properties are known for D [9]: informally, if B depends “not too heavily” on the events in \mathcal{A} , then the probability placed on B by D is “not much more than” the

unconditional probability $\Pr [B]$: at most $f_{\mathcal{A}}(B) \cdot \Pr [B]$ (see (3.2)). Such bounds in combination with further probabilistic analysis can be used to give interesting (non-constructive) results. Our next main contribution is that the MT algorithm has an output distribution (say D') that “approximates” the LLL-distribution D : in that for every B , the *same* upper bound $f_{\mathcal{A}}(B) \cdot \Pr [B]$ as above, holds in D' as well. This can be used to make probabilistic proofs that use the LLL-condition constructive.

Problem (b), in all cases known to us, comes from problem (a): it is easy to test if any *given* A_i holds currently (e.g., if a given subset of vertices in a graph is a clique), with the superpolynomiality of m being the apparent bottleneck. To circumvent this, we develop our third main contribution: the very general Theorem 3.3.3 that is simple and directly applicable in all LLL instances that allow a small slack in the LLL’s sufficient conditions. This theorem proves that a small $\text{poly}(n)$ -sized core-subset of the events in \mathcal{A} can be selected and avoided efficiently using the MT algorithm. Using the LLL-distribution and a simple union bound over the non-core events, we get efficient (Monte Carlo and/or *RNC*) algorithms for these problems.

Our method is applied to several applications including the Santa Claus problem (see Chapter 7), non-repetitive coloring of graphs (see Section 3.4, general Ramsey type graphs (see [75]), acyclic edge coloring (see [75]). All of these have problem (a), and all but the acyclic-coloring application have problem (b). The recent break-through result of Andrews on approximating edge-disjoint paths

problem in undirected graphs is another example, where LLL is applied to avoid super-polynomially many bad events [11].

The following subsection 3.1.1 reviews the MT algorithm and its analysis, which will be helpful to understand some of our proofs and technical contributions; the reader familiar with the MT algorithm may skip it.

3.1.1 Review of the MT Algorithm and its Analysis

Recall the resampling-based MT algorithm; let us now review some of the technical elements in the analysis of this algorithm, that will help in understanding our technical contributions better.

A witness tree $\tau = (T, \sigma_T)$ is a finite rooted tree T together with a labeling $\sigma_T : V(T) \rightarrow \mathcal{A}$ of its vertices to events, such that the children of a vertex $u \in V(T)$ receive labels from $\Gamma(\sigma_T(u)) \cup \sigma_T(u)$. In a proper witness tree distinct children of the same vertex always receive distinct labels. The “log” C of an execution of MT lists the events as they have been selected for resampling in each step. Given C , we can associate a witness tree $\tau_C(t)$ with each resampling step t that can serve as a justification for the necessity of that correction step. $\tau_C(t)$ will be rooted at $C(t)$. A witness tree is said to occur in C , if there exists $t \in N$, such that $\tau_C(t) = \tau$. It has been shown in [111] that if τ appears in C , then it is proper and it appears in C with probability at most $\prod_{v \in V(\tau)} \Pr[\sigma_T(v)]$.

To bound the running time of the MT algorithm, one needs to bound the number of times an event $A \in \mathcal{A}$ is resampled. If N_A denotes the random variable for the

number of resampling steps of A and C is the execution log; then N_A is the number of occurrences of A in this log and also the number of distinct proper witness trees occurring in C that have their root labeled A . As a result one can bound the expected value of N_A simply by summing the probabilities of appearances of distinct witness trees rooted at A . These probabilities can be related to a Galton-Watson branching process to obtain the desired bound on the running time.

A Galton-Watson branching process can be used to generate a proper witness tree as follows. In the first round the root of the witness tree is produced, say it corresponds to event A . Then in each subsequent round, for each vertex v independently and again independently, for each event $B \in \Gamma_{\sigma_T(v)} \cup \sigma_T(v)$, B is selected as a child of v with probability $x(B)$ and is skipped with probability $(1 - x_B)$. We will use the concept of a proper witness trees and Galton-Watson process in several of our proofs.

3.2 LLL-Distribution

When trying to turn the non-constructive Lovász Local Lemma into an algorithm that finds a good assignment the following straightforward approach comes to mind: draw a random sample for the variables in \mathcal{P} until one is found that avoids all bad events. If the LLL-conditions are met this rejection-sampling algorithm certainly always terminates but because the probability of obtaining a good assignment is typically exponentially small it takes an expected exponential number of resam-

plings and is therefore non-efficient. While the celebrated algorithm of Moser (and Tardos) is much more efficient, the above rejection-sampling method has a major advantage: it does not just produce an arbitrary assignment but provides a randomly chosen assignment from the distribution that is obtained when one conditions on no bad event happening. In the following, we call this distribution *LLL-distribution* or *conditional LLL-distribution*.

The following is a well-known bound on the probability $\Pr_D [B]$ that the LLL-distribution D places on *any* event B that is determined by variables in \mathcal{P} (its proof is an easy extension of the standard non-constructive LLL-proof [9]):

Theorem 3.2.1. *If the LLL-conditions from Theorem 3.1.1 are met, then the LLL-distribution D is well-defined. For any event B that is determined by \mathcal{P} , the probability $\Pr_D [B]$ of B under D satisfies:*

$$\Pr_D [B] := \Pr \left[B \mid \bigwedge_{A \in \mathcal{A}} \overline{A} \right] \leq \Pr [B] \cdot \prod_{C \in \Gamma(B)} (1 - x_C)^{-1}; \quad (3.2)$$

here, $\Pr [B]$ is the probability of B holding under a random choice of P_1, P_2, \dots, P_n .

The fact that the probability of an event B does not increase much in the conditional LLL-distribution when B does not depend on “too many” $C \in \mathcal{A}$, is used critically in the rest of the paper.

More importantly, the following theorem states that the output distribution D'

of the MT-algorithm approximates the LLL-distribution D and has the very nice property that it essentially also satisfies (3.2):

Theorem 3.2.2. *Suppose there is an assignment of reals $x : \mathcal{A} \rightarrow (0, 1)$ such that (3.1) holds. Let B be any event that is determined by \mathcal{P} . Then, the probability that B was true at least once during the execution of the MT algorithm on the events in \mathcal{A} , is at most $\Pr[B] \cdot (\prod_{C \in \Gamma(B)} (1 - x_C))^{-1}$. In particular the probability of B being true in the output distribution of MT obeys this upper-bound.*

Proof. The bound on the probability of B ever happening is a simple extension of the MT proof [111]. Note that we want to prove the theorem irrespective of whether B is in \mathcal{A} or not. In either case we are interested in the probability that the event was true at least once during the execution, i.e., if B is in \mathcal{A} whether it could have been resampled at least once. The witness trees that certify the *first time* B becomes true are the ones that have B as a root and all non-root nodes from $\mathcal{A} \setminus \{B\}$. Similarly as in [111], we calculate the expected number of these witness trees via a union bound. Let τ be a fixed proper witness tree with its root vertex labeled B . Following the proof of Lemma 3.1 and using the fact that B cannot be a child of itself, it can be shown that the probability p_τ with which the Galton-Watson process that starts with B yields exactly the tree τ is $p_\tau = \prod_{A \in \Gamma(B)} (1 - x(A)) \cdot \prod_{v \in V(\tau)} x'(\sigma_v)$. Here $V(\tau)$ are the non-root vertices of τ and $x'(\sigma_v) = x(\sigma_v) \prod_{C \in \Gamma(\sigma_v)} (1 - x(C))$. Plugging this in the arguments following the proof of Lemma 3.1 of [111] it is easy to see that the union bound over all these trees

and therefore also the desired probability is at most $\Pr[B] \cdot (\prod_{C \in \Gamma(B)} (1 - x_C))^{-1}$ where the term “ $\Pr[B]$ ” accounts for the fact that the root-event B has to be true as well. □

Using this theorem we can view the MT algorithm as an *efficient* way to obtain a sample that comes approximately from the conditional LLL-distribution. This efficient sampling procedure makes it possible to make proofs using the conditional LLL-distribution constructive and directly convert them into algorithms.

3.3 LLL Applications with Super-Polynomially Many Bad Events

In several applications of the LLL, the number of bad events is super-polynomially larger than the underlying variables. In these cases we aim for an algorithm that still runs in time polynomial in the number of variables, and it is not efficient to have an explicit representation of all bad events. Surprisingly, Theorem 3.3.1 shows that the number of resamplings done by the MT algorithm remains quadratic and in most cases even near-linear in the number of variables n .

We introduce a key parameter:

$$\delta := \min_{A \in \mathcal{A}} x(A) \prod_{B \in \Gamma(A)} (1 - x(B)). \tag{3.3}$$

Note that without loss of generality $\delta \leq \frac{1}{4}$ because otherwise all $A \in \mathcal{A}$ are inde-

pendent, i.e., defined on disjoint sets of variables. Indeed if $\delta > \frac{1}{4}$ and there is an edge in G between $A \in \mathcal{A}$ and $B \in \mathcal{A}$ then we have $\frac{1}{4} > x(A)(1 - x(B))$ and $\frac{1}{4} > x(B)(1 - x(A))$, i.e., $\frac{1}{4} \cdot \frac{1}{4} > x(A)(1 - x(A)) \cdot x(B)(1 - x(B))$ which is a contradiction because $x(1 - x) \leq \frac{1}{4}$ for all x (the maximum is attained at $x = \frac{1}{2}$).

We allow our algorithms to have a running-time that is polynomial in $\log(1/\delta)$; in all applications known to us, $\delta \geq \exp(-O(n \log n))$, and hence, $\log(1/\delta) = O(n \log n)$. In fact because δ is an upper bound for $\min_{A \in \mathcal{A}} P(A)$ in any typical encodings of the domains and the probabilities of the variables, $\log(1/\delta)$ will be at most linear in the size of the input or the output.

Theorem 3.3.1. *Suppose there is an $\varepsilon \in [0, 1)$ and an assignment of reals $x : \mathcal{A} \rightarrow (0, 1)$ such that:*

$$\forall A \in \mathcal{A} : \Pr[A] \leq (1 - \varepsilon)x(A) \prod_{B \in \Gamma(A)} (1 - x(B)).$$

With δ denoting $\min_{A \in \mathcal{A}} x(A) \prod_{B \in \Gamma(A)} (1 - x(B))$, we have

$$T := \sum_{A \in \mathcal{A}} x_A \leq n \log(1/\delta). \quad (3.4)$$

Furthermore:

1. if $\varepsilon = 0$, then the expected number of resamplings done by the MT algorithm is at most $v_1 = T \max_{A \in \mathcal{A}} \frac{1}{1 - x(A)}$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_1 resamplings with probability at least $1 -$

$1/\lambda$.

2. if $\varepsilon > 0$, then the expected number of resamplings done by the MT algorithm is at most $v_2 = O(\frac{n}{\varepsilon} \log \frac{T}{\varepsilon})$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_2 resamplings with probability $1 - \exp(-\lambda)$.

Proof. The main idea of relating the quantity T to n and δ is to use: (i) the fact that the variable-sharing graph G is very dense, and (ii) the nature of the LLL-conditions which force highly connected events to have small probabilities and x -values. To see that G is dense, consider for any variable $P \in \mathcal{P}$ the set of events

$$\mathcal{A}_P = \{A \in \mathcal{A} \mid P \in \text{vbl}(A)\},$$

and note that these events form a clique in G . Indeed, the m vertices of G can be partitioned into n such cliques with potentially further edges between them, and therefore has at least $n \cdot \binom{m/n}{2} = m^2/(2n) - m/2$ edges, which is high density for $m \gg n$.

Let us first prove the bound on T . To do so, we fix any $P \in \mathcal{P}$ and show that $\sum_{B \in \mathcal{A}_P} x_B \leq \log(1/\delta)$, which will clearly suffice. Recall from the discussion following (3.3) that we can assume w.l.o.g. that $\delta \leq \frac{1}{4}$. If $|\mathcal{A}_P| = 1$, then of course $\sum_{B \in \mathcal{A}_P} x_B \leq 1 \leq \log(1/\delta)$. If $|\mathcal{A}_P| > 1$, let $A \in \mathcal{A}_P$ have the smallest x_A value.

Note that by definition

$$\delta \leq x_A \prod_{B \in \mathcal{A}_P \setminus A} (1 - x_B) = \frac{x_A}{1 - x_A} \prod_{B \in \mathcal{A}_P} (1 - x_B).$$

If $x_A \leq 1/2$, then $\delta \leq \prod_{B \in \mathcal{A}_P} (1 - x_B) \leq e^{-\sum_{B \in \mathcal{A}_P} x_B}$, and we get $\sum_{B \in \mathcal{A}_P} x_B \leq \ln(1/\delta) < \log(1/\delta)$ as required. Otherwise, if $x_A > 1/2$, let $B_1 \in \mathcal{A}_P \setminus A$. Then,

$$\begin{aligned} \delta &\leq x_A \cdot \prod_{B \in \mathcal{A}_P \setminus A} (1 - x_B) = x_A(1 - x_{B_1}) \prod_{B \in \mathcal{A}_P \setminus (A \cup B_1)} (1 - x_B) \\ &\leq x_A(1 - x_{B_1}) e^{-\sum_{B \in \mathcal{A}_P \setminus (A \cup B_1)} x_B}. \end{aligned} \quad (3.5)$$

Let us now show that for $1/2 \leq x_A \leq x_{B_1} \leq 1$,

$$x_A(1 - x_{B_1}) \leq e^{-(x_A + x_{B_1})}. \quad (3.6)$$

Fix x_A . We thus need to show $e^{x_{B_1}}(1 - x_{B_1}) \leq \frac{1}{x_A e^{x_A}}$. The derivative of $e^{x_{B_1}}(1 - x_{B_1})$ is negative for $x_{B_1} \geq 0$, showing that it is a decreasing function in the range $x_{B_1} \in [x_A, 1]$. Therefore the maximum value of $e^{x_{B_1}}(1 - x_{B_1})$ is obtained at $x_{B_1} = x_A$ and for (3.6) to hold, it is enough to show that, $x_A(1 - x_A) \leq e^{-2x_A}$ holds. The second derivative of $e^{-2x_A} - x_A(1 - x_A)$ is positive. Differentiating $e^{-2x_A} - x_A(1 - x_A)$ and equating the derivative to 0, returns the minimum in $[1/2, 1]$ at $x_A = 0.7315$. The minimum value is $0.0351 > 0$. Thus we have (3.6)

and so we get

$$x_A(1 - x_{B_1})e^{-\sum_{B \in \mathcal{A}_P \setminus (A \cup B_1)} x_B} \leq e^{-\sum_{B \in \mathcal{A}_P} x_B};$$

using this with (3.5), we obtain $\sum_{B \in \mathcal{A}_P} x_B \leq \ln(1/\delta) < \log(1/\delta)$ as desired.

Given the bound on T , part (1) follows directly from the main theorem of [111] and by a simple application of Markov's inequality.

Part (2) now also follows from [111]. In section 5 of [111] it is shown that saving an $1 - \varepsilon$ factor in the probability of every resampling step implies that with high probability, no witness tree of size $\Omega(\frac{1}{\varepsilon} \log \sum_{A \in \mathcal{A}} \frac{x_A}{1-x_A})$ occurs. This easily implies that none of the n variables can be resampled more often. It is furthermore shown that without loss of generality all x -values can be assumed to be bounded away from 1 by at least $O(\varepsilon)$. This simplifies the upper bound on the expected running time to $n \cdot O(\frac{1}{\varepsilon} \log \frac{T}{\varepsilon})$. \square

As mentioned following the introduction of δ in (3.3), $\log(1/\delta) \leq O(n \log n)$ in all applications known to us, and is often even smaller.

While Theorem 3.3.1 gives very good bounds on the running time of MT even for applications with $\Omega(n) \leq m \leq \text{poly}(n)$ many events, it unfortunately often fails to be directly applicable when m becomes super-polynomial in n . The reason is that maintaining bad events implicitly and running the resampling process requires an efficient way to find violated events. In many examples with super-

polynomially many events, finding violated events or even just verifying a good assignment is not known to be in polynomial time (often even provably NP-hard). To capture the sets of events for which we can run the MT algorithm efficiently we use the following definition:

Definition 3.3.2. (*Efficient verifiability*) *A set \mathcal{A} of events that are determined by variables in \mathcal{P} is efficiently verifiable if, given an arbitrary assignment to \mathcal{P} , we can efficiently find an event $A \in \mathcal{A}$ that holds or detect that there is no such event.*

Because many large \mathcal{A} of interest are not efficiently verifiable, a direct application of the MT-algorithm is not efficient. Nevertheless we show in the rest of this section that using the randomness in the output distribution of the MT-algorithm characterized by Theorem 3.2.2, it is still practically always possible to obtain efficient Monte Carlo algorithms that produce a good assignment with high probability.

The main idea is to judiciously select an efficiently verifiable *core subset* $\mathcal{A}' \subseteq \mathcal{A}$ of bad events and apply the MT-algorithm to it. Essentially instead of looking for violated events in \mathcal{A} we only resample events from \mathcal{A}' and terminate when we cannot find one such violated event. The non-core events will have small probabilities and will be sparsely connected to core events and as such their probabilities in the LLL-distribution and therefore also the output distribution of the algorithm does not blow up by much. There is thus hope that the non-core events remain unlikely to happen even though they were not explicitly fixed by the algorithm.

While the concept of an efficiently verifiable core is easy to understand, it is not clear how often and how such a core can be found. Furthermore having such a core is only useful if the probability of the non-core events is small enough to make the failure probability, which is based on the union bound over those probabilities, meaningful. The following main theorem shows that in all applications that can tolerate a small “exponential” ε -slack as introduced by [37], finding such a good core is straightforward:

Theorem 3.3.3. *Suppose $\log 1/\delta \leq \text{poly}(n)$. Suppose further that there is a fixed constant $\varepsilon \in (0, 1)$ and an assignment of reals $x : \mathcal{A} \rightarrow (0, 1 - \varepsilon)$ such that:*

$$\forall A \in \mathcal{A} : \Pr[A]^{1-\varepsilon} \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B)).$$

Then for every $p \geq \frac{1}{\text{poly}(n)}$ the set $\{A_i \in \mathcal{A} : \Pr[A_i] \geq p\}$ has size at most $\text{poly}(n)$, and is thus essentially always an efficiently verifiable core subset of \mathcal{A} . If this is the case, then there is a Monte Carlo algorithm that terminates after $O(\frac{n}{\varepsilon^2} \log \frac{n}{\varepsilon^2})$ resamplings and returns a good assignment with probability at least $1 - n^{-c}$, where $c > 0$ is any desired constant.

Proof. Note that the set \mathcal{A}' on which the actual MT-algorithm is run fulfills the LLL-conditions. This makes Theorem 3.3.1 applicable. To argue about the success probability of the modified algorithm, note that $x(A) \geq P(A) \prod_{B \in \Gamma'(A)} (1 - x(B))$ where $\Gamma'(A)$ are the neighbors of A in the variable sharing graph defined on \mathcal{A}' .

Using Theorem 3.2.2 we get that the probability that a non-core bad event $A \in \mathcal{A} \setminus \mathcal{A}'$ holds in the assignment produced by the modified algorithm is at most x_A . Since core-events are avoided completely by the MT-algorithm a simple union bound over all conditional non-core event probabilities results in a failure probability of at most $\sum_{A \in \mathcal{A} \setminus \mathcal{A}'} x_A$.

For a probability $p = 1/\text{poly}(n)$ to be fixed later we define \mathcal{A}' as the set of events with probability at least p . Recall from Theorem 3.3.1 that $\sum_{A \in \mathcal{A}} x_A \leq O(n \log(1/\delta))$. Since $x_A \geq p$ for $A \in \mathcal{A}'$, we get that $|\mathcal{A}'| \leq O(n \log(1/\delta)/p) = \text{poly}(n)$. By assumption \mathcal{A}' is efficiently verifiable and we can run the modified resampling algorithm with it.

For every event we have $\Pr[A] \leq x_A < 1 - \varepsilon$ and thus get an $(1 - \varepsilon)^\varepsilon = (1 - \Theta(\varepsilon^2))$ -slack; therefore Theorem 3.3.1 applies and guarantees that the algorithm terminates with high probability after $O(\frac{n}{\varepsilon^2} \log \frac{n}{\varepsilon^2})$ resamplings.

To prove the failure probability note that for every non-core event $A \in \mathcal{A} \setminus \mathcal{A}'$, the LLL-conditions with the “exponential ε -slack” provide an extra multiplicative $p^{-\varepsilon}$ factor over the LLL-conditions in Theorem 3.3.1. We have $x(A)\Pr[A]^\varepsilon \geq \Pr[A] \prod_{B \in \Gamma'(A)} (1 - x(B))$ where $\Gamma'(A)$ are the neighbors of A in the variable sharing graph defined on \mathcal{A}' . Using Theorem 3.2.2 and setting $p = n^{-\Theta(1/\varepsilon)}$, we get that the probability that a non-core bad event $A \in \mathcal{A} \setminus \mathcal{A}'$ holds in the assignment produced by the modified algorithm is at most $x_A \Pr[A]^\varepsilon \leq x_A n^{-\Theta(1)}$. Since core-events are avoided completely by the MT-algorithm, a simple union bound

over all conditional non-core event probabilities results in a failure probability of at most $\frac{1}{n^{\Theta(1)}} \sum_{A \in \mathcal{A} \setminus \mathcal{A}'} x_A$. Now since, $\sum_{A \in \mathcal{A} \setminus \mathcal{A}'} x_A \leq \sum_{A \in \mathcal{A}'} x_A = T = \text{poly}(n)$ holds, we get that we fail with probability at most n^{-c} on non-core events while safely avoiding the core. This completes the proof of the theorem. \square

The last theorem shows that in practically all applications of the general LLL it is possible to obtain a fast Monte Carlo algorithm with arbitrarily high success probability. The conditions of Theorem 3.3.3 are very easy to check and are usually directly fulfilled. That is, in all LLL-based proofs (with a large number of events A_i) known to us, the set of high-probability events forms a polynomial-sized core that is trivially efficiently verifiable, e.g., by exhaustive enumeration. Theorem 3.3.3 makes these proofs constructive without further complicated analysis. Only in cases where the LLL-condition is used are adjustments in the bounds needed, to respect the ε -slack.

Remarks

- Note that the failure probability can be made an arbitrarily small inverse polynomial. This is important since for problems with non-efficiently verifiable solutions the success probability of Monte Carlo algorithms cannot be boosted using standard probability amplification techniques.
- In all applications known to us, the core above has further nice structure: usually the probability of an event A_i is exponentially small in the number

of variables it depends on. Thus, each event in the core only depends on $O(\log n)$ many A_i , and hence is usually trivial to enumerate. This makes the core efficiently verifiable, even when finding a general violated event in \mathcal{A} is NP-hard.

- The fact that the core consists of polynomially many events with usually logarithmically many variables each, makes it often even possible to enumerate the core in parallel and to evaluate each event in parallel. If this is the case one can get an RNC algorithm by first building the dependency graph on the core and then computing an MIS of violated events in each round (using MIS algorithms such as [6, 100]). Using the proof of Theorem 3.3.1 which is based on some ideas from the parallel LLL algorithm of MT, it is easy to see that only logarithmically many rounds of resampling these events are needed.

3.4 An Example: Non-repetitive Coloring

In this section, we give an efficient Monte-Carlo construction for non-repetitive coloring of graphs. Call a word (string) w “squarefree” or “non-repetitive” if there does not exist any strings $u, v, x, v \neq \emptyset$, such that w can be written as $w = uvvx$. Let us refer to graphs using the symbol H instead of G , to not confuse with our dependency graphs G . A k -coloring of the edges of H (not necessarily a proper coloring as in standard graph-coloring terminology) is called *non-repetitive* if the sequence of colors along any path in H is squarefree: i.e., we want a coloring in

which no path has a color-sequence of the form xx . (All paths here refer to simple paths.) The smallest k such that H has a non-repetitive coloring using k colors is called the *Thue number* of H and is denoted by $\pi(H)$. The Thue number was first defined by Alon, Grytczuk, Hauszczak and Riordan in [8]: it is named after Thue who proved in 1906 that if H is a simple path, then $\pi(H) = 3$ [132]. While the method of Thue is constructive, no efficient construction is known for general graphs. Alon et al. showed through application of the asymmetric LLL that $\pi(H) \leq c\Delta(H)^2$ for some absolute constant c . Their proof was nonconstructive. The number of bad events is exponential and even checking whether a given coloring is non-repetitive is coNP-Hard, [105]. Thus checking if some “bad event” holds in a given coloring is coNP-Hard. Since the work of Alon et al., the non-repetitive coloring of graphs has received a good deal of attention in the last few years [7, 31, 49, 69, 93, 119]. Yet no efficient construction is known till date, except for some special classes of graphs such as complete graphs, cycles and trees.

Randomized Algorithm for Obtaining a Non-repetitive Coloring Suppose we are given a graph H with maximum degree Δ . We first give the proof of Alon et al. which shows that $\pi(H) \leq c\Delta^2$, and then show how to convert this proof directly into a constructive algorithm (with the loss of a Δ^ϵ factor in the number of colors used):

Theorem 3.4.1 (Theorem 1 of [8]). *There exists an absolute constant c such that $\pi(H) \leq c\Delta^2$ for all graphs H with maximum degree at most Δ .*

Proof. Let $C = (2e^{16} + 1)\Delta^2$. Randomly color each edge of H with colors from C . Consider the following types of bad events B_i , for $i \geq 1$: “there exists a path P of length $2i$, such that the second half of P is colored identically to its first half”.

We have for a path P of length $2i$, $i \geq 1$, $\Pr[P \text{ has coloring of the form } xx] = \frac{1}{C^i}$. Also, a path of length $2i$ intersects at most $4ij\Delta^{2j}$ paths of length $2j$. Thus, for any bad event A of type i , we have $\Pr[A] = \frac{1}{C^i}$ and that each bad event of type i share variables with at most $4ij\Delta^{2j}$ bad events of type B_j . Set $x_i = \frac{1}{2^i\Delta^{2i}}$. We have $(1 - x_j) \geq e^{-2x_j}$; this, along with the fact that $\sum_{j \geq 1} j/2^j = 2$, shows that

$$x_i \prod_j (1 - x_j)^{4ij\Delta^{2j}} \geq x_i e^{-8i \sum_j x_j j \Delta^{2j}} \geq \frac{1}{2^i \Delta^{2i}} e^{-8i \sum_j \frac{j}{2^j}} = (2e^{16} \Delta^2)^{-i}.$$

Since $C = (2e^{16} + 1)\Delta^2$, the condition of the LLL is satisfied and we are guaranteed the existence of such a non-repetitive coloring. \square

Now we see that using just a slightly higher number of colors suffices to make Theorem 3.3.3 apply.

Theorem 3.4.2. *There exists an absolute constant c such that for every constant $\varepsilon > 0$ there exists a Monte Carlo algorithm that given a graph H with maximum degree Δ , produces a non-repetitive coloring using at most $c\Delta^{2+\varepsilon}$ colors. The failure probability of the algorithm is an arbitrarily small inverse polynomial in the size of H .*

Proof. We apply the LLL using the same random experiments and bad events as

in Theorem 3.4.1 but with $C' = C^{\frac{1}{1-\varepsilon'}}$ colors such that $C' < c\Delta^{2+\varepsilon}$. Using the same settings for x_A gives an exponential ε' slack in the LLL-conditions since the probability of a bad event of type i is now at most $\frac{1}{C'^i} = \left(\frac{1}{C^i}\right)^{\frac{1}{1-\varepsilon'}}$. Recall Theorem 3.3.3. Clearly, $\log 1/\delta = O(n^2)$ and so the last thing to check to apply Theorem 3.3.3 is that for any inverse polynomial p , the bad events with probability at least p are efficiently verifiable. Here these events consist of paths smaller than a certain length (of the form $O((1/\varepsilon) \log n / \log \Delta)$, where n is the number of vertices), and Theorem 3.3.3 guarantees that there are only polynomially many of these. Using breadth-first-search to go through these paths and checking each of them for non-repetitiveness is efficient and thus Theorem 3.3.3 directly applies. \square

Part II

Applications

Scheduling: Handling Hard Capacities and Outliers

Scheduling jobs on unrelated parallel machines (UPM) is a fundamental scheduling model which has spurred many advances and applications in combinatorial optimization, including linear-, quadratic- & convex-programming relaxations and new rounding approaches [17, 20, 26, 34, 54, 73, 92, 96, 122, 124]. Herein, we are given a set J of n jobs, a set M of m machines, and non-negative values $p_{i,j}$ ($i \in M, j \in J$): each job j has to be assigned to some machine, and assigning it to machine i will impose a processing time of $p_{i,j}$ on machine i . (The word “unrelated” arises from the fact that there may be no pattern among the given numbers $p_{i,j}$.) Variants such as the type of objective function(s) to be optimized in such an assignment, whether there is an additional “cost-function”, whether a few jobs can be dropped, and situations where there are release dates for, and precedence constraints among, the jobs, lead to a rich spectrum of problems and techniques. Two

such highly-impactful results are [96, 122]. The primary UPM objective in these works is to minimize the *makespan* – the maximum total load on any machine. It is shown in [96] that this problem can be approximated to within a factor of 2; furthermore, even some natural special cases cannot be approximated better than 1.5 unless $\mathbf{P} = \mathbf{NP}$ [96]. Despite much effort, these bounds have not been improved. The work of [122] builds on the upper-bound of [96] to consider the *generalized assignment problem* (GAP) where we incur a cost $c_{i,j}$ if we schedule job j on machine i ; a simultaneous $(2, 1)$ -approximation for the (makespan, total cost)-pair is developed in [122], leading to numerous applications (see, e.g., [10, 40]).

In this chapter, we consider two significant generalizations of scheduling on UPM and show the effectiveness of the rounding approach developed in Chapter 2. The first generalization involves introducing capacities on machines that dictate how many jobs can be scheduled on each machine, and the second one considers outliers with the possibility of improving the system performance by dropping a few of them. We also study these two problems in the setting of GAP.

Capacity constraints on machines. Handling “hard capacities” – those that cannot be violated – is generally tricky in various settings, including facility-location and other covering problems [46, 63, 113]. Motivated by problems in crew-scheduling [56, 117] and by the fact that servers have a limit on how many jobs can be assigned to them, the natural question of scheduling with a hard capacity-constraint of “at most b_i jobs to be scheduled on each machine i ” has been studied

in [41,133,137,139,144]. Most recently, the work of [41] has shown that this problem can be approximated to within a factor of 3 in the special case where the machines are *identical* (job j has processing time p_j on any machine). We generalize this to the setting of GAP and obtain the GAP bounds of [122] – i.e., approximation ratios of 2 and 1 for the makespan and cost respectively, while satisfying the capacity constraints: the improvements are in the more-general scheduling model, handling the cost constraint, and in the approximation ratio. We anticipate that such a capacity-sensitive generalization of [122] would lead to improved approximation algorithms for several applications of GAP.

Scheduling with outliers: makespan and fairness. Another direction for generalization of GAP result [122] is to consider “outliers” in scheduling [73]. For instance, suppose in the “processing times $p_{i,j}$ and costs $c_{i,j}$ ” setting of GAP, we also have a profit π_j for choosing to schedule each job j . Given a “hard” target profit Π , target makespan T and total cost C , the LP-rounding method of [73] either proves that these targets are not simultaneously achievable, or constructs a schedule with values $(\Pi, 3T, C(1 + \varepsilon))$ for any constant $\varepsilon > 0$. We improve this to $(\Pi, (2 + \varepsilon)T, C(1 + \varepsilon))$. (The factors of ε in the cost are required due to the hardness of knapsack [73].) Also, fairness is a fundamental issue in dealing with outliers: e.g., in repeated runs of such algorithms, we may not desire long starvation of individual job(s) in sacrifice to a global objective function. We can accommodate fairness in the form of scheduling-probabilities for the jobs that can be part of

the input.

4.1 Scheduling with Hard Capacities

Scheduling on UPM with hard capacities is a special case of random bipartite b -matchings with target degree bounds and sharp tail bounds for given linear functions; see [57] for applications to models for complex networks. Recall that a (b) -matching is a subgraph in which every vertex v has degree at most $b(v)$. Given a *fractional* b -matching x in a bipartite graph $G = (J, M, E)$ of N vertices and a collection of k linear functions $\{f_i\}$ of x , many works have considered the problem of constructing (b) -matchings X such that $f_i(X)$ is “close” to $f_i(x)$ simultaneously for each i [12, 65, 68, 115]. The works [68, 115] focus on the case of constant k ; those of [12, 65] consider general k , and require the usual “discrepancy” term of $\Omega(\sqrt{f_i(x) \log N})$ in $|f_i(X) - f_i(x)|$ for most/all i ; in a few cases, $o(N)$ vertices will have to remain unmatched also.

Theorem 4.1.1 shows that if there is one structured objective function f_i with bounded coefficients associated with each $i \in M$, then in fact all the $|f_i(X) - f_i(x)|$ can be bounded independent of N .

Theorem 4.1.1. *Let $G = (J, M, E)$ be a bipartite graph with “jobs” J and “machines” M . Let \mathcal{F} be the collection of edge-indexed vectors y (with $y_{i,j}$ denoting y_e where $e = (i, j) \in E$). Suppose we are given: (i) an integer requirement r_j for each $j \in J$ and an integer capacity b_i for each $i \in M$; (ii) for each $i \in M$,*

a linear objective function $f_i : \mathcal{F} \rightarrow \mathfrak{R}$ given by $f_i(y) = \sum_{j: (i,j) \in E} p_{i,j} y_{i,j}$ such that $0 \leq p_{i,j} \leq \ell_i$ for each j , (iii) a global cost constraint $\sum_{i,j} c_{i,j} y_{i,j} \leq C$, and (iv) a vector $x \in \mathcal{F}$ with $x_e \in [0, 1]$ for each e . Then, we can efficiently construct a random subgraph of G given by a binary vector $X \in \mathcal{F}$, such that: (a) with probability one, each $j \in J$ has degree at least r_j , each $i \in M$ has degree at most b_i , and $|f_i(X) - f_i(x)| < \ell_i \forall i$; and (b) for all $e \in E$, $\mathbb{E}[X_e] = x_e$ which implies $\mathbb{E}\left[\sum_{i,j} c_{i,j} X_e\right] = \sum_e c_e x_e = C$

We first prove the result of GAP with individual capacity constraints on each machine (Theorem 4.1.2). The full proof of Theorem 4.1.1 follows after Theorem 4.1.2.

The capacity constraint specifies the maximum number of jobs that can be scheduled on any machine, and is a hard constraint. Formally the problem is as follows, where $x_{i,j}$ is the indicator variable for job j being scheduled on machine i . Given m machines and n jobs, where job j requires a processing time of $p_{i,j}$ in machine i and incurs a cost of $c_{i,j}$ if assigned to i , the goal is to minimize the makespan $T = \max_i \sum_j x_{i,j} p_{i,j}$, subject to the constraint that the total cost $\sum_{i,j} x_{i,j} c_{i,j}$ is at most C and for each machine i , $\sum_j x_{i,j} \leq b_i$. C is the given upper bound on total cost and b_i is the capacity of machine i , that must be obeyed.

Our main contribution here is an efficient algorithm **Sched-Cap** that has the following guarantee, generalizing the GAP bounds of [122]:

Theorem 4.1.2. *There is an efficient algorithm **Sched-Cap** that returns a sched-*

ule maintaining all the capacity constraints, of cost at most C and makespan at most $2T$, where T is the optimal makespan with cost C that satisfies the capacity constraints.

Algorithm Sched-Cap Algorithm **Sched-Cap** proceeds as follows. First we guess the optimum makespan T by binary search as in [96]. If $p_{i,j} > T$, $x_{i,j}$ is set to 0. The solution to the following integer program gives the optimum schedule:

$$\sum_{i,j} c_{i,j} x_{i,j} \leq C \quad (\text{Cost})$$

$$\sum_{i,j} x_{i,j} = 1 \quad \forall j \quad (\text{Assign})$$

$$\sum_j p_{i,j} x_{i,j} \leq T \quad \forall i \quad (\text{Load})$$

$$\sum_j x_{i,j} \leq b_i \quad \forall i \quad (\text{Capacity})$$

$$x_{i,j} \in \{0, 1\} \quad \forall i, j$$

$$x_{i,j} = 0 \quad \text{if } p_{i,j} > T$$

We relax the constraint “ $x_{i,j} \in \{0, 1\} \forall (i, j)$ ” to “ $x_{i,j} \in [0, 1] \forall (i, j)$ ” to obtain an LP relaxation **LP-Cap**. We solve the LP to obtain an optimal LP solution x^* ; we next show how **Sched-Cap** rounds x^* to obtain an integral solution within the approximation guarantee.

Note that $x_{i,j}^* \in [0, 1]$ denotes the “fraction” of job j assigned to machine i . Initialize $X = x^*$. The algorithm is composed of several iterations. The random

value of the assignment-vector X at the end of iteration h of the overall algorithm is denoted by X^h . Each iteration h conducts a randomized update using the **RandMove** on the polytope of a linear system constructed from a *subset* of the constraints of **LP-Cap**. Therefore, by induction on h , we will have for all (i, j, h) that $\mathbb{E} [X_{i,j}^h] = x_{i,j}^*$; we use this property and drop the cost constraint since on expectation it is maintained.

Let J and M denote the set of jobs and machines, respectively. Suppose we are at the beginning of some iteration $(h + 1)$ of the overall algorithm: we are currently looking at the values $X_{i,j}^h$. We will maintain four invariants.

Invariants across iterations:

- (I1) Once a variable $x_{i,j}$ gets assigned to 0 or 1, it is never changed;
- (I2) The constraints (Assign) always hold; and
- (I3) Once a constraint in (Capacity) becomes tight, it remains tight, and
- (I4) Once a constraint is dropped in some iteration, it is never reinstated.

Iteration $(h + 1)$ of **Sched-Cap** consists of three main steps:

1. We first remove all $X_{i,j}^h \in \{0, 1\}$; i.e., we project X^h to those co-ordinates (i, j) for which $X_{i,j}^h \in (0, 1)$, to obtain the current vector Y of “floating” (to-be-rounded) variables; let $\mathcal{S} \equiv (A_h Y = u_h)$ denote the current linear system that represents **LP-Cap**. (A_h is some matrix and u_h is a vector; we avoid using “ \mathcal{S}_h ” to simplify notation.) In particular, the “capacity” of machine i in \mathcal{S} is its residual

capacity b'_i , i.e., b_i minus the number of jobs that have been permanently assigned to i thus far. Note that the cost constraint is not included in the constraint matrix $A_h Y = u_h$, which we continue to maintain exactly. Nevertheless since all the variables maintains its initial assignment on expectation, the expected cost remains unaltered. The entire process as we demonstrate at the end can be derandomized and hence the cost upper bound of C is obeyed.

2. Let $Y \in \mathfrak{R}^v$ for some v ; note that $Y \in (0, 1)^v$. Let M_k denote the set of all machines i for which exactly k of the values $Y_{i,j}$ are positive. We will now drop some of the constraints in \mathcal{S} :

(D1) for each $i \in M_1$, we drop its load and capacity constraints from \mathcal{S} ;

(D2) for each $i \in M_2$, we drop its load constraint and rewrite its capacity constraint as $x_{i,j_1} + x_{i,j_2} \leq \lceil X_{i,j_1}^h + X_{i,j_2}^h \rceil$, where j_1, j_2 are the two jobs fractionally assigned to i .

(D3) for each $i \in M_3$ for which *both* its load and capacity constraints are tight in \mathcal{S} , we drop its load constraint from \mathcal{S} .

3. Let \mathcal{P} denote the polytope defined by this reduced system of constraints. A key claim that is proven in Lemma 4.1.3 below is that Y is *not* a vertex of \mathcal{P} . We now invoke **RandMove**(Y, \mathcal{P}); this is allowable if Y is indeed not a vertex of \mathcal{P} .

The above three steps complete iteration $(h + 1)$.

Analysis. It is not hard to verify that the invariants **(I1)**-**(I4)** hold true (though the fact that we drop the all-important capacity constraint for machines $i \in M_1$ may look bothersome, a moment's reflection shows that such a machine cannot have a tight capacity-constraint since its sole relevant job j has value $Y_{i,j} \in (0, 1)$). Since we make at least one further constraint tight via **RandMove** in each iteration, invariant **(I4)** shows that we terminate, and that the number of iterations is at most the initial number of constraints. Let us next present Lemma 4.1.3, a key lemma:

Lemma 4.1.3. *In no iteration is Y a vertex of the current polytope \mathcal{P} .*

Proof. Suppose that in a particular iteration, Y is a vertex of \mathcal{P} . Fix the notation v , M_k etc. w.r.t. this iteration; let $m_k = |M_k|$, and let n' denote the remaining number of jobs that are yet to be assigned permanently to a machine. Let us lower- and upper-bound the number of variables v . On the one hand, we have $v = \sum_{k \geq 1} k \cdot m_k$, by definition of the sets M_k ; since each remaining job j contributes at least two variables (co-ordinates for Y), we also have $v \geq 2n'$. Thus we get

$$v \geq n' + \sum_{k \geq 1} (k/2) \cdot m_k. \quad (4.1)$$

On the other hand, since Y has been assumed to be a vertex of \mathcal{P} , the number t of constraints in \mathcal{P} that are satisfied *tightly* by Y , must be at least v . How large can t be? Each current job contributes one (Assign) constraint to t ; by our “dropping constraints” steps **(D1)**, **(D2)** and **(D3)** above, the number of tight constraints (“load”

and/or “capacity”) contributed by the machines is at most $m_2 + m_3 + \sum_{k \geq 4} 2m_k$.

Thus we have

$$v \leq t \leq n' + m_2 + m_3 + \sum_{k \geq 4} 2m_k. \quad (4.2)$$

Comparison of (4.1) and (4.2) and a moment’s reflection shows that such a situation is possible only if: (i) $m_1 = m_3 = 0$ and $m_5 = m_6 = \dots = 0$; (ii) the capacity constraints are tight for all machines in $M_2 \cup M_4$ – i.e., for all machines; and (iii) $t = v$. However, in such a situation, the t constraints in \mathcal{P} constitute the *tight* assignment constraints for the jobs and the *tight* capacity constraints for the machines, and are hence linearly dependent (since the total assignment “emanating from” the jobs must equal the total assignment “arriving into” the machines). Thus we reach a contradiction, and hence Y is not a vertex of \mathcal{P} . \square

We next show that the final makespan is at most $2T$ with probability one:

Lemma 4.1.4. *Let X denote the final rounded vector. Algorithm **Sched-Cap** returns a schedule, where with probability one: (i) all capacity-constraints on the machines are satisfied, and (ii) for all i , $\sum_{j \in J} X_{i,j} p_{i,j} < \sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in (0,1)} p_{i,j}$.*

Proof. Part (i) essentially follows from the fact that we never drop any capacity constraint; the only care to be taken is for machines i that end up in M_1 and hence have their capacity-constraint dropped. However, as argued soon after the description of the three steps of an iteration, note that such a machine cannot have a tight

capacity-constraint when such a constraint was dropped; hence, even if the remaining job j got assigned finally to i , its capacity constraint cannot be violated.

Let us now prove (ii). Fix a machine i . If at all its load-constraint was dropped, it must be when i ended up in M_1, M_2 or M_3 . The case of M_1 is argued as in the previous paragraph. So suppose $i \in M_\ell$ for some $\ell \in \{2, 3\}$ when its load constraint got dropped. Let us first consider the case $\ell = 2$. Let the two jobs fractionally assigned on i at that point have processing times (p_1, p_2) and fractional assignments (y_1, y_2) on i , where $0 \leq p_1, p_2 \leq T$, and $0 < y_1, y_2 < 1$. If $y_1 + y_2 \leq 1$, we know that at the end, the assignment vector X will have at most one of X_1 and X_2 being one. Simple algebra now shows that $p_1X_1 + p_2X_2 < p_1y_1 + p_2y_2 + \max\{p_1, p_2\}$ as required. If $1 < y_1 + y_2 \leq 2$, then both X_1 and X_2 can be assigned and again, $p_1X_1 + p_2X_2 < p_1y_1 + p_2y_2 + \max\{p_1, p_2\}$. For the case $\ell = 3$, we know from **(I3)** and **(D3)** that its capacity-constraint must be *tight* at some integral value u at that point, and that this capacity-constraint was preserved until the end. We must have $c = 1$ or 2 here. Let us just consider the case $c = 2$; the case of $c = 1$ is similar to the case of $\ell = 2$ with $y_1 + y_2 \leq 1$. Here again, simple algebra yields that if $0 \leq p_1, p_2, p_3 \leq T$ and $0 < y_1, y_2, y_3 < 1$ with $y_1 + y_2 + y_3 = c = 2$, then for any binary vector (X_1, X_2, X_3) of Hamming weight $c = 2$, $p_1X_1 + p_2X_2 + p_3X_3 < p_1y_1 + p_2y_2 + p_3y_3 + \max\{p_1, p_2, p_3\}$. \square

Finally we have the following lemma.

Lemma 4.1.5. *Algorithm **Sched-Cap** can be derandomized to create a schedule of*

cost at most C .

Proof. Let $X_{i,j}^h$ denote the value of $x_{i,j}$ at iteration h . We know for all i, j, h , $E[X_{i,j}^h] = x_{i,j}^*$, where $x_{i,j}^*$ is solution of **LP-Cap**. Therefore, at the end, we have that the total expected cost incurred is C . The procedure can be derandomized directly by the method of conditional expectation, giving an 1-approximation to the cost. \square

Lemmas 4.1.4 and 4.1.5 yield Theorem 4.1.2.

Proof of Theorem 4.1.1. We now consider the full proof of Theorem 4.1.1. The following integer program gives an optimal matching:

$$\begin{aligned}
 \sum_{i,j} c_{i,j} x_{i,j} &\leq C && \text{(Cost)} \\
 \sum_{i,j} x_{i,j} &\geq r_j \quad \forall j && \text{(Assign)} \\
 \sum_j p_{i,j} x_{i,j} &= f_i \quad \forall i && \text{(Load)} \\
 \sum_j x_{i,j} &\leq b_i \quad \forall i && \text{(Capacity)} \\
 x_{i,j} &\in \{0, 1\} \quad \forall i, j \\
 x_{i,j} &= 0 \quad \text{if } p_{i,j} > l_i
 \end{aligned}$$

The proof of Theorem 4.1.1 is quite similar to Theorem 4.1.2. We elaborate upon the necessary modifications. First, while removing $X_{i,j}^h \in \{0, 1\}$, we up-

date the assignment requirements of the jobs as well as the capacity constraints of the machines accordingly. The dropping rules **(D1)** and **(D3)** remain the same. However, **(D2)** is modified as follows:

(Modified D2) For each $i \in M_2$, we drop its load constraint and rewrite its capacity constraint. Let j_1, j_2 be the two jobs assigned to machine i with fractional assignment x_{i,j_1} and x_{i,j_2} . Then if $x_{i,j_1} + x_{i,j_2} \leq 1$, set the capacity constraint to $x_{i,j_1} + x_{i,j_2} \leq 1$. Else if $1 < x_{i,j_1} + x_{i,j_2} < 2$, set the capacity constraint to $x_{i,j_1} + x_{i,j_2} \geq 1$.

Lemma 4.1.3, Lemma 4.1.5 remain unchanged. We have a new Lemma 4.1.6 corresponding to Lemma 4.1.4, which we prove next.

Lemma 4.1.6. Let X denote the final rounded vector. Then X satisfies with probability one: (i) all capacity-constraints on the machines are satisfied, and (ii) for all i , $\sum_j x_{i,j}^* p_{i,j} - \max_{j \in J: x_{i,j}^* \in (0,1)} p_{i,j} < \sum_{j \in J} X_{i,j} p_{i,j} < \sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in (0,1)} p_{i,j}$.

Proof. Part (i) is similar to Part (i) of Lemma 4.1.5 and follows from the facts that the capacity constraints are never violated and machines in M_1 cannot have tight capacity constraints.

Let us now prove (ii). Note that in **(Modified D2)** the upper bound on capacity constraint is maintained as in **(D2)**. Hence from Lemma 4.1.4, we get $\sum_{j \in J} X_{i,j} p_{i,j} < \sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in (0,1)} p_{i,j}$. So we only need to show the lower bound on the load. Fix a machine i . If at all its load-constraint was

dropped, it must be when i ended up in $M_1 \cup M_2 \cup M_3$. In the case of M_1 , at most one job fractionally assigned to it may not be assigned in the final rounded vector. So suppose $i \in M_l$ for some $l \in \{2, 3\}$ when i has its load constraint dropped. Let us first consider the case of $\ell = 2$. Let the two jobs fractionally assigned to i at that point have processing times (p_1, p_2) and fractional assignments (y_1, y_2) on i , where $0 \leq p_1, p_2 \leq T$, and $0 < y_1, y_2 < 1$. If $y_1 + y_2 \leq 1$, then at the end, none of the jobs may get assigned. Simple algebra now shows that $0 > p_1 y_1 + p_2 y_2 - \max\{p_1, p_2\}$ as required. If $1 < y_1 + y_2 \leq 2$, then at least one of the two jobs X_1 and X_2 get assigned to i and again, $p_1 X_1 + p_2 X_2 > p_1 y_1 + p_2 y_2 - \max\{p_1, p_2\}$. For the case $\ell = 3$, we know from **(I3)** and **(D3)** that i 's capacity-constraint must be *tight* at some integral value u at that point, and that this capacity-constraint was preserved until the end. We must have $c = 1$ or 2 in this case. Let us just consider the case $c = 2$; the case of $c = 1$ is similar to the case of $\ell = 2$ with $y_1 + y_2 \leq 1$. Here again, simple algebra yields that if $0 \leq p_1, p_2, p_3 \leq T$ and $0 < y_1, y_2, y_3 < 1$ with $y_1 + y_2 + y_3 = c = 2$, then for any binary vector (X_1, X_2, X_3) of Hamming weight $c = 2$, $p_1 X_1 + p_2 X_2 + p_3 X_3 > p_1 y_1 + p_2 y_2 + p_3 y_3 - \max\{p_1, p_2, p_3\}$. \square

Lemmas 4.1.6 and 4.1.5 yield Theorem 4.1.1.

This completes the description of this section. We have shown through our technique of rounding how a random subgraph of a bipartite graph with hard degree-constraints can be obtained that near-optimally satisfies a collection of linear constraints and respects a given cost-budget. As a special case of this, we obtained

a 2 approximation algorithm for the generalized assignment problem with hard capacity-constraints on the machines.

4.2 Scheduling with Outliers

In this section, we consider GAP with outliers and with a hard profit constraint [73]. Formally, the problem is as follows. Let $x_{i,j}$ be the indicator variable for job j to be scheduled on machine i . Given m machines and n jobs, where job j requires processing time of $p_{i,j}$ in machine i , incurs a cost of $c_{i,j}$ if assigned to i and provides a profit of π_j if scheduled, the goal is to minimize the makespan, $T = \max_i \sum_j x_{i,j} p_{i,j}$, subject to the constraint that the total cost $\sum_{i,j} x_{i,j} c_{i,j}$ is at most C and total profit $\sum_j \pi_j \sum_i x_{i,j}$ is at least Π . The problem is motivated from improving the scheduling performance by dropping a few outliers that may be costly to schedule.

Our main contribution here is the following:

Theorem 4.2.1. *For any constant $\varepsilon > 0$, there is an efficient algorithm **Sched-Outlier** that returns a schedule of profit at least Π , cost at most $C(1 + \varepsilon)$ and makespan at most $(2 + \varepsilon)T$, where T is the optimal makespan among all schedules that simultaneously have cost C and profit Π .*

This is an improvement over the work of Gupta, Krishnaswamy, Kumar and Segev [73], where they constructed a schedule with makespan $3T$, profit Π and cost $C(1 + \varepsilon)$. In addition, our approach also accommodates *fairness*, a basic

requirement in dealing with outliers, especially when problems have to be run repeatedly. We formulate fairness via stochastic programs that specify for each job j , a lower-bound r_j on the probability that it gets scheduled. We adapt our approach to honor such requirements:

Theorem 4.2.2. *There is an efficient randomized algorithm that returns a schedule of profit at least Π , expected cost at most $2C$ and makespan at most $3T$ and guarantees that for each job j , it is scheduled with probability r_j , where T is the optimal expected makespan with expected cost C and expected profit Π . If the fairness guarantee on any one job can be relaxed, then for every fixed $\varepsilon > 0$, there is an efficient algorithm to construct a schedule that has profit at least Π , expected cost at most $C(1 + 1/\varepsilon)$ and makespan at most $(2 + \varepsilon)T$.*

We start with Theorem 4.2.1 and describe the algorithm **Sched-Outlier** first. Next, we prove Theorem 4.2.2.

Algorithm Sched-Outlier. The algorithm starts by guessing the optimal makespan T by binary search as in [96]. If $p_{i,j} > T$, then $x_{i,j}$ is set to 0. Next pick any constant $\varepsilon > 0$. The running time of the algorithm depends on ε and is $O(n^{\frac{1}{\varepsilon^c}})$, where c is some constant. We guess all assignments (i, j) where $c_{i,j} > \varepsilon' C$, with $\varepsilon' = \varepsilon^2$. Any valid schedule can have at most $1/\varepsilon'$ pairs with assignment costs higher than $\varepsilon' C$; since ε' is a constant, this guessing can be done in time $O((mn)^{\frac{1}{\varepsilon'}}) = O(n^{\frac{1}{\varepsilon^2}})$. For all (i, j) with $c_{i,j} > \varepsilon' C$, let $\mathcal{G}_{i,j} \in \{0, 1\}$ be a correct guessed assignment. By enumeration, we know the optimal $\mathcal{G}_{i,j}$. For any (i, j) with

$c_{i,j} > \varepsilon' C$ and $c_{i,j} \notin \mathcal{G}_{i,j}$, we set $x_{i,j} = 0$. Similarly, if $c_{i,j} > \varepsilon' C$ and $c_{i,j} \in \mathcal{G}_{i,j}$, then we set $x_{i,j} = 1$.

The solution to the following integer linear program then gives an optimal solution:

$$\sum_{i,j} c_{i,j} x_{i,j} \leq C \quad (\text{Cost})$$

$$\sum_i x_{i,j} = y_j, \forall j \quad (\text{Assign})$$

$$\sum_j p_{i,j} x_{i,j} \leq T, \forall i \quad (\text{Load})$$

$$\sum_j \pi_j y_j \geq \Pi \quad (\text{Profit})$$

$$x_{i,j} \in \{0, 1\}, y_j \in \{0, 1\}, \forall i, j$$

$$x_{i,j} = 0 \quad \text{if } p_{i,j} > T$$

$$x_{i,j} = \mathcal{G}_{i,j} \quad \text{if } c_{i,j} > \varepsilon' C$$

We relax the constraint “ $x_{i,j} \in \{0, 1\}$ and $y_j \in \{0, 1\}$ ” to “ $x_{i,j} \in [0, 1]$ and $y_j \in [0, 1]$ ” to obtain the LP relaxation **LP-Out**. We solve the LP to obtain an optimal LP solution x^*, y^* ; we next show how **Sched-Outlier** rounds x^*, y^* to obtain the claimed approximation. The rounding proceeds in stages as in Section 4.1, and as before, each variable maintains its initial assignment in x^* on expectation over the course of rounding. Hence, there is no need to explicitly consider the cost

constraint. The cost constraint is dropped, yet the cost is maintained on expectation. The entire process can be derandomized efficiently. Therefore, as long as we apply our general recipe of rounding, **RandMove**, the cost is maintained exactly. Also note that if we maintain all the assign-constraints, then the profit-constraint can be dropped and is not violated. Therefore, we consider the profit constraint if and only if there are one or more assign constraints that are dropped. Also, we only need to maintain the total profit obtained from the jobs for which the assign constraints have been dropped. We now proceed to describe the rounding on each stage formally.

Note that $x_{i,j}^* \in [0, 1]$ denotes the fraction of job j assigned to machine i in x^* . Initially, $\sum_i x_{i,j}^* = y_j^*$. Initialize $X = x^*$. The algorithm is composed of several iterations; the random values at the end of iteration h of the overall algorithm are denoted by X^h . (Since y_j is given by the equality $\sum_i x_{i,j}$, X^h is effectively the set of variables.) Each iteration h (except perhaps the last one) conducts a randomized update using **RandMove** on a suitable polytope constructed from a *subset* of the constraints of **LP-Out**. Therefore, for all h except perhaps the last, we have $\mathbb{E} [X_{i,j}^h] = x_{i,j}^*$. A variable $X_{i,j}^h$ is said to be *floating* if it lies in $(0, 1)$, and a job is *floating* if it is not yet finally assigned. The subgraph of (J, M, E) composed of the floating edges (i, j) , naturally suggests the following notation at any point of time: machines of “degree” k in an iteration are those with exactly k floating jobs assigned fractionally, and jobs of “degree” k are those assigned fractionally to

exactly k machines in iteration h . Note that since we allow $y_j < 1$, there can exist singleton (i.e., degree-1) jobs which are floating.

Suppose we are at the beginning of some iteration $(h + 1)$ of the overall algorithm; so we are currently looking at the values $X_{i,j}^h$. We will maintain the following invariants:

Invariants across iterations:

- (I1')** Once a variable $x_{i,j}$ gets assigned to 0 or 1, it is never changed;
- (I2')** If j is not a singleton, then $\sum_i x_{i,j}$ remains at its initial value;
- (I3')** The constraint (Profit) always holds;
- (I4')** Once a constraint is dropped, it is never reinstated.

Algorithm **Sched-Outlier** starts by initializing with **LP-Out**. Iteration $(h + 1)$ consists of four major steps.

1. We remove all $X_{i,j}^h \in \{0, 1\}$ as in Section 4.1, i.e., we project X^h to those coordinates (i, j) for which $X_{i,j}^h \in (0, 1)$, to obtain the current vector Z of “floating” variables; let $\mathcal{S} \equiv (A_h Z = u_h)$ denote the current linear system that represents **LP-Out**. (A_h is some matrix and u_h is a vector.)
2. Let $Z \in \Re^v$ for some v ; note that $Z \in (0, 1)^v$. Let M_k and N_k denote the set of degree- k machines and degree- k jobs respectively, with $m_k = |M_k|$ and $n_k = |N_k|$. We will now drop/replace some of the constraints in \mathcal{S} :

(D1') for each $i \in M_1$, we drop its load constraint from \mathcal{S} ;

(D2') for each $i \in N_1$, we drop its assignment constraint from \mathcal{S} ; we add one profit constraint (if already exists, we replace the old one) ,

$$\sum_{j \in N_1} Z_{i,j} \pi_j = \sum_{j \in N_1} X_{i,j}^h \pi_j.$$

(Note that at this point, the values $X_{i,j}^h$ are some known values.)

Thus, the assignment constraints of the singleton jobs are replaced by *one* profit constraint. As we noted earlier, it is not required to maintain the contribution to profit by the non-singleton jobs for which the assignment constraints are maintained explicitly.

3. If Z is a vertex of \mathcal{S} then define the fractional assignment of a machine i by $h_i = \sum_{j \in J} Z_{i,j}$. Define a job j to be tight if $\sum_{i \in M} Z_{i,j} = 1$. Drop all the assignment constraints of the non-tight jobs (denoted by J_N) and maintain a single profit constraint,

$$\sum_{j \in N_1 \cup J_N} Z_{i,j} \pi_j = \sum_{j \in N_1 \cup J_N} X_{i,j}^h \pi_j.$$

While there exists a machine i' whose degree d satisfies $h_{i'} \geq (d - 1 - \varepsilon)$, drop the load constraint on machine i' .

4. Let \mathcal{P} denote the polytope defined by this reduced system of constraints. If Z is not a vertex of \mathcal{P} , invoke **RandMove**(Z, \mathcal{P}). Else we proceed differently depending on the configuration of machines and jobs in the system. If none of the following

configurations is achieved (which we will show never happens at a vertex), then we report error and exit. There are five possible configurations.

- **Config-1:** *Machine and job nodes form disjoint cycles.*

Orient the edges in the bipartite graph to assign the remaining jobs in a way, so that each machine gets at most one extra job. Note that such an orientation is easy in disjoint cycles since they have even length.

- **Config-2:** *Machine and job nodes form disjoint cycles and has exactly one path with both end-points being job nodes. Thus there are two singleton jobs.*

Discard one among the two singleton jobs that has less profit. Again orient the edges in the remaining bipartite graph to assign the remaining jobs in a way, so that each machine gets at most one extra job. Such an orientation is easy in disjoint cycles (they have even length) and paths with equal number of machines and nodes.

- **Config-3:** *There is exactly one job of degree-3 and one singleton job. Rest of the jobs have degree 2 and all the machines have degree-2.*

Assign the singleton job to the degree-2 machine it is fractionally attached to and remove the other edge (but not the job) associated with that machine. We are left with disjoint cycles. Orient the edges in the cycles of the bipartite graph to assign the remaining jobs in a way, so that each machine gets at most one extra job.

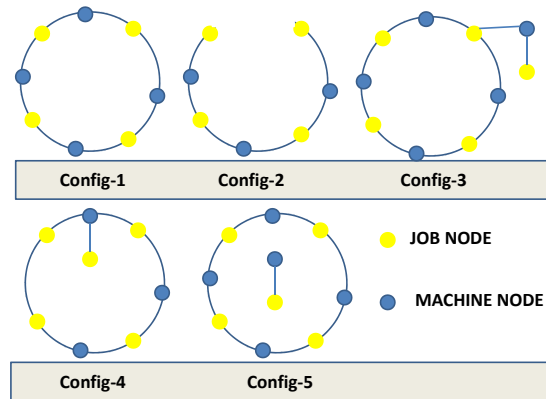


Figure 4.1: Different configurations of machine-job bipartite graph at step 4 of Sched-Outlier

- **Config-4:** *There is only one degree-3 machine with one singleton job. Rest of the machines have exactly two non-singleton jobs attached to it fractionally. Each non-singleton job is attached fractionally to exactly two machines.*

Assign the singleton job and the cheaper (less processing time) of the two non-singleton jobs to the degree-3 machines. Rest of the jobs and the machines form disjoint cycles in the machine-job bipartite graph or form disjoint paths each with equal number of machines and jobs in it. Orient the edges in this remaining bipartite graph in a way such that each machine gets one among the two jobs fractionally attached to it.

- **Config-5:** *Machine and job nodes form disjoint cycles. There is one extra edge with one singleton job and one singleton machine.*

Assign the singleton job to the singleton machine. Orient the edges in the cycles of the bipartite graph to assign the remaining jobs in a way, so that each machine

gets at most one extra job.

The different configurations are shown pictorially in Figure 4.1.

Analysis. Analysis follows the following structure. First, we prove a key lemma, Lemma 4.2.3, which shows that if Z is a vertex and the algorithm reaches step 4, then one of the five configurations as described above happens and also the number of machines is less than $\frac{1}{\varepsilon}$. Lemma 4.2.3 is followed by Lemma 4.2.4. Lemma 4.2.4 establishes that the dropping and the modification of constraints in step 2 and 3, along with the assignment of jobs in step 4 do not violate the load constraint by more than a factor of $(2 + \varepsilon)$ and maintain the profit constraint. Lemma 4.2.5 bounds the cost.

Recall that in the bipartite graph $G = (J, M, E)$, we have in iteration $(h + 1)$ that $(i, j) \in E$ iff $X_{i,j}^h \in (0, 1)$. Any job or machine having degree 0 is thus not part of G . We prove Lemma 4.2.3 next.

Lemma 4.2.3. *Let m denote the number of machine-nodes in G . If $m \geq \frac{1}{\varepsilon}$, then Z is not a vertex of the polytope at the beginning of step 4.*

Proof. Let us consider the different possible configurations of G , when Z becomes a vertex of the polytope \mathcal{P} at the beginning of step 3. There are several cases to consider depending on the number of singleton floating jobs in G in that iteration.

Case 1: There is no singleton job: We have $n_1 = 0$. Then, the number of

constraints in \mathcal{S} is

$$EQ = \sum_{k \geq 2} m_k + \sum_{k \geq 2} n_k.$$

Remember, since there is no singleton job, we do not consider the profit constraint explicitly. Also the number of floating variables is $v = \sum_{k \geq 2} kn_k$. Alternatively, $v = \sum_{k \geq 1} km_k$. Therefore,

$$v = \sum_{k \geq 2} \frac{k}{2}(m_k + n_k) + \frac{m_1}{2}.$$

Z being a vertex of \mathcal{P} , $v \leq EQ$. Thus, we must have, $n_k, m_k = 0, \forall k \geq 3$ and $m_1 = 0$. Hence, every floating machine has exactly two floating jobs assigned to it and every floating job is assigned exactly to two floating machines. This is handled by Config-1.

Case 2: There are at least 3 singleton jobs: We have $n_1 \geq 3$. Then the number of linear constraints is $EQ = \sum_{k \geq 2} m_k + \sum_{k \geq 2} n_k + 1$. The last “1” comes from considering one profit constraint for the singleton jobs. The number of floating variables v again by the averaging argument as above is

$$v = \frac{n_1}{2} + \sum_{k \geq 2} \frac{k}{2}(m_k + n_k) + \frac{m_1}{2} \geq \frac{3}{2} + \sum_{k \geq 2} \frac{k}{2}(m_k + n_k) + \frac{m_1}{2}.$$

Hence, the system is always underdetermined and Z cannot be a vertex of \mathcal{P} .

Case 3: There are exactly 2 singleton jobs: We have $n_1 = 2$. Then the number

of linear constraints is

$$EQ = \sum_{k \geq 2} m_k + \sum_{k \geq 2} n_k + 1.$$

Again the last “1” comes from considering one profit constraint for the singleton jobs. The number of floating variables v by the averaging argument is

$$v = \frac{n_1}{2} + \sum_{k \geq 2} \frac{k}{2} (m_k + n_k) + \frac{m_1}{2} \geq 1 + \sum_{k \geq 2} \frac{k}{2} (m_k + n_k) + \frac{m_1}{2}.$$

Thus, we must have, $n_k = 0, m_k = 0, \forall k \geq 3$ and $m_1 = 0$. Hence every floating machine has exactly two floating jobs assigned to it and each job except two is assigned to exactly two machines fractionally This is handled by Config-2.

Case 4: There is exactly 1 singleton job: We have $n_1 = 1$. Then the number of linear constraints is

$$EQ = \sum_{k \geq 2} m_k + \sum_{k \geq 2} n_k + 1.$$

The number of floating variables is,

$$v \geq \frac{1}{2} + n_2 + \frac{3}{2}n_3 + \frac{m_1}{2} + m_2 + \frac{3}{2}m_3 + \sum_{k \geq 4} \frac{k}{2} (m_k + n_k).$$

If Z is a vertex of \mathcal{P} , then $v \leq EQ$. There are only three possible configurations that might arise in this case.

- (i) Only one job of degree 3 and one job of degree 1. All the other jobs have

degree 2 and all the machines have degree 2. This is handled by Config-3.

(ii) Only one machine of degree 3 and one job of degree 1. The rest of the jobs and machines have degree 2. This is handled by Config-4.

(iii) Only one machine of degree 1 and one job of degree 1. The rest of the jobs and machines have degree 2. This is handled by Config-5.

Each configuration can have an arbitrary number of vertex disjoint cycles. In all these configurations, it is easy to check that for Z to be a vertex, $v = EQ$. Thus if just one constraint can be dropped, then the system becomes underdetermined.

Since we have reached a vertex at the beginning of step 3, we drop,

- All the assignment constraints for the non-tight jobs.
- Any machine i' that has degree d (where $d > 0$ is a positive integer) and the total fractional assignment from all the jobs fractionally assigned to it is at least $d - 1 - \varepsilon$ loses its load-constraint.
- If the profit constraint is not already considered and some non-tight job loses its assignment constraint; we add the profit constraint.

Now we have $v = EQ$ at the beginning of step 3 and at the beginning of step 4 as well. Hence it implies either *we have not been able to drop any constraint*, or *we have dropped one assignment constraint for a non-tight job and have added one profit constraint*. We will now show that when $m \geq \frac{1}{\varepsilon}$, we always drop more constraints than we add. This will give a contradiction.

In any configuration, if there is a cycle with all tight jobs, then there always exists a machine with total fractional assignment 1 and hence its load constraint can always be dropped to make the system underdetermined. So we assume there is no such cycle in any configurations. Now suppose the algorithm reaches Config-1. If there are two non-tight jobs, then we drop two assignment constraints and only add one profit constraint. Thus the system becomes underdetermined. Therefore, there can be at most one non-tight job and only one cycle (say C) with that non-tight job. Let C have m machines and thus m jobs. Therefore, $\sum_{i,j \in C} x_{i,j} \geq m - 1$. Thus there exists a machine, such that the total fractional assignment of jobs on that machine is $\geq \frac{m-1}{m} = 1 - 1/m$. If $m \geq \frac{1}{\varepsilon}$, then there exists a machine with degree 2 and with total fractional assignment $\geq (1 - \varepsilon)$. Thus the load-constraint on that machine gets dropped making the system underdetermined.

If the algorithm reaches Config-2, then all the non-singleton jobs must be tight for Z to be a vertex. If there are m machines, then the number of non-singleton jobs is $m-1$. Let the two singleton jobs be j_1 and j_2 . Let the two machines to which jobs j_1 and j_2 are fractionally attached with be i_1 and i_2 respectively. If $x_{i_1,j_1} + x_{i_2,j_2} \geq 1$, then the total fractional assignment from all the jobs in the system is m . Thus the machine with maximum fractional assignment must have an assignment at least 1. Since the same machine has degree 2, its load constraint gets dropped. Otherwise, the total fractional assignment from all the jobs in the system is at least $m - 1$. Thus there exists a machine, such that the total fractional assignment of jobs on

that machine is $\geq \frac{m-1}{m} = 1 - 1/m$. If $m \geq \frac{1}{\varepsilon}$, then there exists a machine with degree 2 and with total fractional assignment $\geq (1 - \varepsilon)$. Thus the load- constraint on that machine gets dropped making the system underdetermined.

For Config-3 and 5, if Z is a vertex of \mathcal{P} , then all the jobs must be tight and using essentially the same argument, there exists a machine with fractional assignment at least $(1 - \varepsilon)$ if the algorithm reaches Config-3 and there exists a machine with fractional assignment 1, if the algorithm reaches Config-5.

If the algorithm reaches Config-4, then again all the jobs must be tight. If the degree-3 machine has fractional assignment at least $2 - \varepsilon$, then its load constraint can be dropped to make the system underdetermined. Otherwise, the total assignment to the degree-2 machines from all the jobs in the cycle is at least $m - 2 + \varepsilon$. Therefore, there exists at least one degree-2 machine with fractional assignment at least $\frac{m-2+\varepsilon}{m-1} = 1 - \frac{1-\varepsilon}{m-1} \geq 1 - \varepsilon$, if $m \geq \frac{1}{\varepsilon}$. The load-constraint on that machine can be dropped making the system underdetermined. This completes the proof of Lemma 4.2.3. □

We next show that the final profit is at least Π and the final makespan is at most $(2 + \varepsilon)T$:

Lemma 4.2.4. *Let X denote the final rounded vector. Algorithm **Sched-Outlier** returns a schedule, where with probability one, (i) the profit is at least Π , (ii) for all i , $\sum_{j \in J} X_{i,j} p_{i,j} < \sum_j x_{i,j}^* p_{i,j} + (1 + \varepsilon) \max_{j \in J: x_{i,j}^* \in \{0,1\}} p_{i,j}$.*

Proof. (i) This essentially follows from the fact that whenever assignment con-

straint on any job is dropped, its profit constraint is included in the global profit constraint of the system. At step 4 except for one configuration (Config-2), all the jobs are always assigned. Thus the profit can not decrease in those configurations. In Config-2, since we are at a vertex the total fractional assignment from the two singleton jobs is less than 1. Otherwise the system remains underdetermined from Lemma 4.2.3. Thus a singleton job (say j_1) is dropped, only when G has two singleton jobs j_1, j_2 fractionally assigned to i_1 and i_2 respectively, with total assignment $x_{i_1, j_1} + x_{i_2, j_2} < 1$. Since the job with the higher profit is retained, $\pi_{j_1} x_{i_1, j_1} + \pi_{j_2} x_{i_2, j_2} \leq \max\{\pi_{j_1}, \pi_{j_2}\}$.

(ii) From Lemma 4.2.3 and **(D1')**, load constraints are dropped from machines $i \in M_1$ and might be dropped from machine $i \in M_2 \cup M_3$. For $i \in M_1$, only the remaining job j with $X_{i,j}^h > 0$, can get fully assigned to it. Hence for $i \in M_1$, its total load is bounded by $\sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in \{0,1\}} p_{i,j}$. For any machine $i \in M_2 \cup M_3$, if their degree d (2 or 3) is such that, its fractional assignment is at least $d - 1 - \varepsilon$, then by simple algebra, it can be shown that for any such machine i , its total load is at most $\sum_j x_{i,j}^* p_{i,j} + (1 + \varepsilon) \max_{j \in J: x_{i,j}^* \in \{0,1\}} p_{i,j}$ at the end of the algorithm. For the remaining machines consider what happens at step 4. Since this is the last iteration, if we can show that the load does not increase by too much in this last iteration, we are done. Except when Config-4 is reached, any remaining machine i gets at most one extra job, and thus its total load is bounded by $\sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in \{0,1\}} p_{i,j}$. When Config-4 is reached at step 4, if the

degree-3 machine has a fractional assignment at most 1 from the two jobs in the cycle, then for any value of m , there will exist a degree-2 machine whose fractional assignment is 1, giving a contradiction. Hence, let j_1, j_2, j_3 be the three jobs assigned fractionally to the degree-3 machine i and let j_3 be the singleton job, and $x_{i,j_1} + x_{i,j_2} > 1$. If $p_{i,j_1} \leq p_{i,j_2}$, then the degree-3 machine gets j_1, j_3 . Else the degree-3 machine gets j_2, j_3 . The degree-3 machine gets 2 jobs, but its fractional assignment from j_1 and j_2 is already at least 1. Since the job with less processing time among j_1 and j_2 is assigned to i , its increase in load can be at most $\sum_j x_{i,j}^* p_{i,j} + \max_{j \in J: x_{i,j}^* \in \{0,1\}} p_{i,j}$. This completes the proof of Lemma 4.2.4 \square

Finally we have the following lemma.

Lemma 4.2.5. *Algorithm **Sched-Outlier** can be derandomized to output a schedule of cost at most $C(1 + \varepsilon)$.*

Proof. In all iterations h , except the last one, for all i, j , $E[X_{i,j}^h] = x_{i,j}^*$, where $x_{i,j}^*$ is solution of **LP-Out**. Therefore, before the last iteration, we have that the total expected cost incurred is C . The procedure can be derandomized directly by the method of conditional expectation, giving an 1-approximation to cost, just before the last iteration. Now at the last iteration, since at most $\frac{1}{\varepsilon}$ jobs are assigned and each assignment requires at most $\varepsilon' C = \varepsilon^2 C$ in cost, the total increase in cost is at most εC , giving the required approximation. \square

Lemmas 4.2.4 and 4.2.5 yield Theorem 4.2.1.

We now consider Theorem 4.2.2 that maintains fairness in the allocation of jobs while handling outliers.

Proof of Theorem 4.2.2

Proof. In order to maintain the scheduling probabilities of the jobs, we do not guess the assignment of jobs with high cost (cost higher than $\varepsilon' C$ as in Theorem 4.2.1). For Part (i), we consider the first two steps of Algorithm **Sched-Outlier**. If \mathcal{P} denote the polytope defined by the reduced system of constraints and the current vector Z is not a vertex of \mathcal{P} , then we invoke **RandMove**(Z, P) and proceed. Else from Lemma 4.2.3, Z is a vertex of \mathcal{P} only if one of the configurations, Config-1 to Config-5, as described in step 4 of Algorithm **Sched-Outlier** is achieved and $m < \frac{1}{\varepsilon}$. For any singleton job, we assign the singleton job to the corresponding machine with probability equal to its fractional assignment. Thus Theorem 4.2.2 remains valid for these singleton jobs. For each non-singleton job, we consider the machines to which it is fractionally assigned and allocate it to the machine which has cheaper assignment cost for it. If the algorithm reached Config-1, 2, 3 or 5, each machine can get at most two extra jobs and the expected cost is maintained. However if the algorithm reached Config-4 and the three jobs associated with the degree-3 machine were all assigned to it, then we remove one non-singleton job from the degree-3 machine. This job is assigned to the degree-2 machine in the cycle on which it had non-zero fractional assignment. This may increase the expected cost by a factor of 2 but ensures that each machine gets at most 2 additional jobs.

For Part (ii), note that the cost is maintained until the last iteration. In the last iteration, since at most $\frac{1}{\epsilon}$ jobs are assigned and each assignment requires at most C cost, we get the desired result. \square

This completes the description of this chapter. We have shown two natural generalizations of GAP: (i) handling hard capacities and (ii) handling outliers. In both these applications, we have applied the rounding algorithm developed in Chapter 2. The choice of linear systems to apply **RandMove**, the constraints to drop or combine are problem specific. In Chapter 10 and 8, we will see another two applications of this rounding method, and we believe this technique will find many more applications in future.

Scheduling: Handling Energy Efficiency

Scheduling it turns out, comes down to deciding how to spend money.

— SIGACT News article [29].

In this chapter, we consider another generalization of GAP. The application is motivated from saving energy while running machines to schedule jobs. In a recent SIGACT News article [29], Birman et al. describe the importance of scheduling to save energy and thus to reduce cost in modern data centers. In the same article, Hamilton, a researcher from Amazon, argues that a ten fold reduction in the power needs of the data centers may be possible if we can explore ways to simply do less during surge load periods, and to migrate work in time taking advantage of the infrequent peaks and deep valleys of work load in modern cloud platforms. Following his observation, in this chapter, we define a collection of new problems referred to as “machine activation” problems. The central framework we introduce

considers a collection of m machines (unrelated or related) with each machine i having an *activation cost* of a_i . There is also a collection of n jobs that need to be performed, and $p_{i,j}$ is the processing time of job j on machine i . Standard scheduling models assume that the set of machines is fixed and all machines are available. However, in our setting, we assume that there is an activation cost budget of A – we would like to *select* a subset S of the machines to activate with total cost $a(S) \leq A$ and *find* a schedule for the n jobs on the machines in S minimizing the makespan (or any other metric).

5.1 Machine Activation Problems in Data Centers

Large scale data centers have emerged as an extremely popular way to store and manage a large volume of data. Most large corporations, such as Google, HP and Amazon have dozens of data centers. These data centers are typically composed of thousands of machines, and have extremely high energy requirements. Data centers are now being used by companies such as Amazon Web Services, to run large scale computation tasks for other companies who do not have the resources to create their own data centers. This is in addition to their own computing requirements.

These data centers are designed to be able to handle extremely high work loads in periods of peak demand. However, since the workload on these data centers fluctuates over time, we could selectively shut down part of the system to save energy when the demand on the system is low. Energy savings results not just from

putting machines in a sleep state, but also from savings in cooling costs.

Hamilton (see the recent SIGACT News article [29]) argues that a ten fold reduction in the power needs of the data center may be possible if we can simply build systems that are optimized with power management as their primary goal. Suggested examples (summarizing from the original text) are:

1. Explore ways to simply do less during surge load periods.
2. Explore ways to migrate work in time. The work load on modern cloud platforms is very cyclical, with infrequent peaks and deep valleys. Even valley time is made more expensive by the need to own a power supply to be able to handle the peaks, a number of nodes adequate to handle surge loads, a network provisioned for worst case demand.

This leads to the issue of *which machines can we shut down*, since all machines in a data center are not necessarily identical. Each machine stores some data, and is thus not capable of performing every single job efficiently unless some data is first migrated to the machine. We will formalize this question very shortly.

To quote from the recent article by Birman et al. (SIGACT News [29]) “Scheduling mechanisms that assign tasks to machines, but more broadly, play the role of provisioning the data center as a whole. As we’ll see below, this aspect of cloud computing is of growing importance because of its organic connection to power consumption: both to spin disks, and run machines, but also because active machines produce heat and demand cooling. Scheduling, it turns out, comes down

to deciding how to spend money.”

Data is replicated on storage systems for both load balancing during peak demand periods, as well as for fault tolerance. Typically many jobs have to be scheduled on the machines in the data center. In many cases profile information for a set of jobs is available in advance, as well as estimates of cyclical workloads. Jobs may be I/O intensive or CPU intensive, in either case, an estimate of its processing time on each type of machine is available. Jobs that need to access specific data can be assigned to any one of the *subset* of machines that store the needed data. Our goal is to first *select* a subset of machines to activate, and then schedule the jobs on the active machines. From this aspect our problems differ from standard scheduling problems with multiple machines, where the set of active machines is the set of all machines. Here we have to decide *which machines to activate* and then schedule all jobs on the active machines.

The scheduling literature is vast, and one can formulate a variety of interesting questions in this model. We initiate this work by focusing our attention on perhaps one of the most widely studied machine scheduling problems since it matches the requirements of the application. We have a collection of jobs and unrelated machines, and need to decide which subset of machines to activate. The jobs can only be scheduled on active machines. This provides an additional dimension for scheduling problems that was not previously considered. This situation also makes sense when we have a certain set of computational tasks to process, a cost budget,

and can purchase access to a set of machines.

Given a set J of n jobs and a set M of m machines, our goal is to activate a subset S of machines and then map each job to an active machine in S , minimizing the overall makespan. Each machine has an activation cost of a_i . The activation cost of the subset S is $a(S) = \sum_{i \in S} a_i$. We show that if there is a schedule with activation cost A and makespan T , then we can find a schedule with activation cost $2(1 + \frac{1}{\varepsilon})(\ln \frac{n}{OPT} + 1)A$ and makespan $(2 + \varepsilon)T$ for any $\varepsilon > 0$ (we call this is a $((2 + \varepsilon), 2(1 + \frac{1}{\varepsilon})(\ln \frac{n}{OPT} + 1))$ -approximation). Actually, the $\ln n$ term in the activation cost with this general formulation is unavoidable, since this problem is at least as hard to approximate as the set cover problem, for which a $(1 - \varepsilon) \ln n$ approximation algorithm will imply that $NP \subseteq DTIME(n^{O(\log \log n)})$ [59].

We further consider the case where allocating a job to a machine requires some cost. For example, a job j with processing time $p_{i,j}$ on machine i might require a assignment cost of $c_{i,j} = \alpha p_{i,j}$, where α is some constant. The total processing cost is a simple sum of the assignment cost of the individual jobs. The assignment cost nicely captures the energy consumed to process a job in active machines. We show that if there exists a schedule with total activation and assignment cost A and makespan T , then we can find a schedule with total activation and assignment cost $O(\frac{1}{\varepsilon} \log n + m)A$ and makespan $(2 + \varepsilon)T$ for any $\varepsilon > 0$.

Related work on Speed Scaling A well studied problem is that of processor speed scaling. In this model, a processor can be run at speed s , that can be adjusted

based on workload. Jobs are arriving in an online manner with deadlines d_j and processing requirement p_j . The goal is to complete all jobs between their arrival time and their deadline in a way that minimizes total energy consumed. Much work was devoted to the single processor case [19, 140], and more recently to the multiprocessor case [4]. This work is somewhat orthogonal to our problem – in the single processor model, the main issue is at what speed to run the processor when there is a set of waiting jobs in the queue – in the absence of more jobs, its better to run the processor as slowly as possible, while still completing all the jobs by their deadlines. If suddenly a lot of new jobs arrive, then in trying to complete partially processed jobs and the new jobs, we may have to run at a significantly higher speed (using a lot of power) than necessary had we finished the initial set of jobs earlier.

The paper by Albers et al. [4] deals with multiple processors and unit length jobs (each job has a release time and deadline). The main focus of the paper is to show how to exploit techniques for the single processor case to attack the multi processor case, and these are shown to be effective in certain situations. In contrast, our problem is an offline problem where we have a large collection of jobs, and we have to decide which machines can go into a sleep state and which machines will remain active.

5.2 LP Rounding for Machine Activation on Unrelated Machines

In this section, we first provide a simple rounding scheme with an approximation ratio of $(O(\log n), O(\log n))$. Then we improve it to a $(2+\varepsilon, 2(1+\frac{1}{\varepsilon})(\ln \frac{n}{OPT}+1))$ -approximation by utilizing the rounding scheme developed in Chapter 2.

We can formulate the scheduling activation problem as an integer program. We define a variable y_i for each machine i , which is 1 if the machine is open and 0, if it is closed. For every machine-job pair, we have a variable $x_{i,j}$, which is 1, if job j is assigned to machine i and is 0, otherwise. In the corresponding linear programming relaxation, we relax the y_i and $x_{i,j}$ variables to be in $[0, 1]$. The first set of constraints require that each job is assigned to some machine. The second set of constraints restrict the jobs to be assigned to only active machines, and the third set of constraints limit the workload on a machine. We require that $1 \geq x_{i,j}, y_j \geq 0$ and if $p_{i,j} > T$ then $x_{i,j} = 0$. The formulation is as shown below:

$$\begin{aligned}
 & \min \sum_{i=1}^m a_i y_i & (5.1) \\
 \text{s.t.} & \sum_{i \in M} x_{i,j} = 1 \quad \forall j \in J \\
 & x_{i,j} \leq y_i \quad \forall i \in M, j \in J \\
 & \sum_j p_{i,j} x_{i,j} \leq T y_i \quad \forall i
 \end{aligned}$$

Suppose an integral solution with activation cost A and makespan T exist. The LP relaxation will have cost at most A with the correct choice of T . All the bounds we show are with respect to these terms. In Section 2.2 we show that unless we relax the makespan constraint, there is a large integrality gap for this formulation.

Simple Rounding We first start with a simple rounding scheme. Let us denote the optimum LP solution by \bar{y}, \bar{x} . The rounding consists of the following four steps:

1. Round each y_i to 1, with probability \bar{y}_i and 0 with probability $1 - \bar{y}_i$. If y_i is rounded to 1, open machine i .
2. For each open machine i , consider the set of jobs j , that have fractional assignment > 0 on machine i . For each such job, set $X_{i,j} = \frac{\bar{x}_{i,j}}{\bar{y}_i}$. If $\sum_j p_{i,j} X_{i,j} < T$, (it is always $\leq T$) then uniformly increase $X_{i,j}$. Stop increasing any $X_{i,j}$ that reaches 1. Stop the process, when either the total fractional makespan is T or all $X_{i,j}$'s are 1. If $X_{i,j} = 1$, assign job j to machine i . If machine i has no job fractionally assigned to it, drop machine i from further consideration. For each job j that has fractional assignment $X_{i,j}$, assign it to machine i with probability $X_{i,j}$.
3. Discard all assigned jobs. If there are some unassigned jobs, repeat the procedure.
4. If some job is assigned to multiple machines, choose any one of them arbitrarily.

In the above rounding scheme, we use \bar{y}_i 's as probabilities for opening machines and for each opened machine, we assign jobs following the probability distribution given by $X_{i,j}$'s. It is obvious that the expected activation cost of machines in each iteration is exactly the cost of the fractional solution given by the LP. The following lemmas bound the number of iterations and the final load on each machine.

Lemma 5.2.1. *The number of iterations required by the rounding algorithm is $O(\log n)$.*

Proof. Consider a job j . In a single iteration, $\Pr[\text{job } j \text{ is not assigned to machine } i] \leq (1 - \bar{y}_i) + \bar{y}_i(1 - \frac{\bar{x}_{i,j}}{\bar{y}_i}) = 1 - \bar{x}_{i,j}$. Hence,

$$\begin{aligned} \Pr[\text{job } j \text{ is not assigned in an iteration}] \\ \leq \prod_i (1 - \bar{x}_{i,j}) \leq (1 - \frac{1}{m})^m \leq \frac{1}{e} \end{aligned}$$

The second inequality holds since $\sum_i \bar{x}_{i,j} = 1$ and the quantity is maximized when all $\bar{x}_{i,j}$'s are equal. Then, it is easy to see the probability that job j is not assigned after $2 \ln n$ iterations is at most $\frac{1}{n^2}$. Therefore, by union bound, with probability at least $1 - \frac{1}{n}$, all jobs can be assigned in $2 \ln n$ iterations. \square

Lemma 5.2.2. *The load on any machine is $O(T \log n)$ with high probability.*

Proof. Consider any iteration h . Denote the value of $X_{i,j}$ at iteration h , by $X_{i,j}^h$.

For each open machine i and each job j , define a random variable

$$Z_{i,j,h} = \begin{cases} \frac{p_{i,j}}{T}, & \text{if job } j \text{ is assigned to machine } i \\ 0, & \text{otherwise} \end{cases} \quad (5.2)$$

Clearly, $0 \leq Z_{i,j,h} \leq 1$. Define, $Z_i = \sum_{j,h} Z_{i,j,h}$. Clearly,

$$\mathbb{E}[Z_i] = \frac{\sum_h \sum_j p_{i,j} X_{i,j}^h}{T} \leq \sum_h 1 \leq \Theta(\log n)$$

Denote by M_i the load on machine i . Therefore, $M_i = TZ_i$, thus $\mathbb{E}[M_i] \leq \Theta(T \log n)$. Now by the standard Chernoff-Hoeffding bound [80, 120], we get the result. \square

Integrality Gap of the Natural LP for Strict Makespan Let there be m jobs and m machines. Call these machines A_1, A_2, \dots, A_{m-1} , and B . Processing time for all jobs on machines A_1, A_2, \dots, A_{m-1} is T and on B it is $\frac{T}{m}$. Activation costs of opening machines A_1, A_2, \dots, A_{m-1} is 1, and for B it is very high compared to m , say $R (R \gg m)$. An integral optimum solution has to open machine B with total cost at least R .

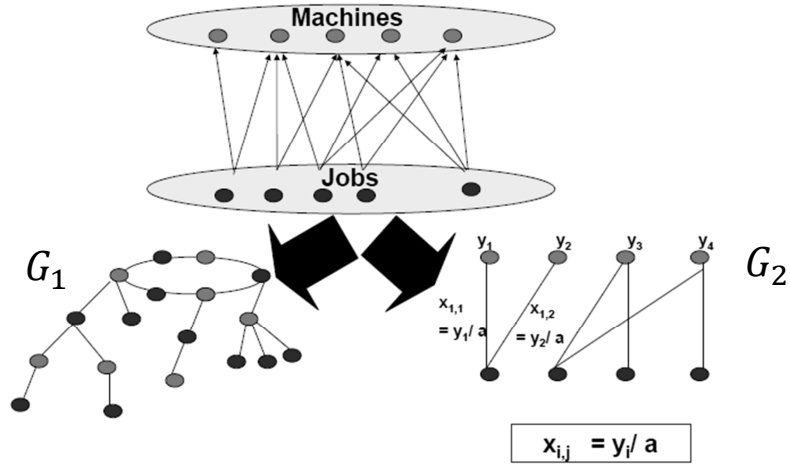
Now consider a fractional solution, where all machines A_1, A_2, \dots, A_{m-1} are fully open, but machine B is open only to the extent of $1/m$. All jobs are assigned to the extent of $1/m$ on each machine A_1, A_2, \dots, A_{m-1} . So the total processing time on any machine A_i is $m \frac{T}{m} = T$. The remaining $\frac{1}{m}$ part of each job is assigned

to B . So total processing time on B is $\frac{T}{m \cdot m} \cdot m = \frac{T}{m}$. It is easy to see the optimal fractional cost is at most $m + \frac{R}{m}$ (by setting $y_B = \frac{1}{m}$). Therefore, the integrality gap is at least $\approx m$.

Main Rounding Algorithm The algorithm begins by solving LP 5.1. As before \bar{x}, \bar{y} denote the optimal fractional solution of the LP. Let M denote the set of machines and J denote the set of jobs. Let $|M| = m$ and $|J| = n$. We define a bipartite graph $G = (M \cup J, E)$ as follows: $M \cup J$ are the vertices of G and $e = (i, j) \in E$, if $\bar{x}_{i,j} > 0$. The weight on edge (i, j) is $\bar{x}_{i,j}$ and the weight on machine node i is \bar{y}_i . Rounding consists of several iterations. Initialize $X = \bar{x}$ and $Y = \bar{y}$. The algorithm iteratively modifies X and Y , such that at the end X and Y become integral. Random variables at the end of iteration h are denoted by $X_{i,j}^h$ and Y_i^h .

The two main steps of rounding are as follow:

1. *Transforming the Solution:* It consists of creating two graphs G_1 and G_2 from G , where G_1 has an almost forest structure and in G_2 the weight of an edge and the weight of the incident machine node is very close (see figure in the next page). In this step, only $X_{i,j}$'s are modified, while Y_i 's remain fixed at \bar{y}_i 's.
2. Exploiting the properties of G_1 and G_2 , and *rounding on G_1 and G_2 separately.*



Decomposition of machine-job bipartite graph into G_1 and G_2

We now describe each of these steps in detail.

1. Transforming the Solution

We decompose G into two graphs G_1 and G_2 through several rounds. Initially, $V(G_1) = V(G) = M \cup J$, $E(G_1) = E(G)$, $V(G_2) = M$ and $E(G_2) = \emptyset$. In each round, we either move one job node and/or one edge from G_1 to G_2 or delete an edge from G_1 . Thus we always make progress. An edge moved to G_2 retains its weight through the rest of the iterations, while the weights of the edges in G_1 keep on changing.

We maintain the following invariants,

(I1) $\forall (i, j) \in E(G_1)$, and $\forall h$, $X_{i,j}^h \in (0, y_i/\gamma)$, $p_{i,j} > 0$.

(I2) $\forall i \in M$ and $\forall h$, $\sum_j X_{i,j}^h p_{i,j} \leq T y_i$.

(I3) $\forall (i, j) \in E(G_2)$ and $\forall h, 1 \geq X_{i,j}^h \geq y_i/\gamma$.

(I4) Once a variable is rounded to 0 or 1, it is never changed.

Consider round one. Remove any machine node that has $Y_i^1 = 0$ from both G_1 and G_2 . Activate any machine that has $Y_i^1 = 1$. Similarly, discard any edge (i, j) with $X_{i,j}^1 = 0$, and if $X_{i,j}^1 = 1$, assign job j to machine i and remove j . If $X_{i,j}^1 \geq \bar{y}_i/\gamma$, then remove the edge (i, j) from G_1 and add the job j (if not added yet) and the edge (i, j) with weight $x_{i,j}(\geq \bar{y}_i/\gamma)$ to G_2 . Note that, if for some $(i, j) \in G$, $p_{i,j} = 0$, then we can simply take $\bar{x}_{i,j} = \bar{y}_i$ and move the edge to G_2 . Thus we can always assume for every edge $(i, j) \in G_1$, $p_{i,j} > 0$. It is easy to see that, after iteration one, all the invariants **(I1-I4)** are maintained.

Let us consider iteration $(h + 1)$ and let J', M' denote the set of jobs and machine nodes in G_1 with degree at least 1 at the beginning of the iteration. Note that $Y_i^h = Y_i^1 = \bar{y}_i$ for all h . Let $|M'| = m'$ and $|J'| = n'$. As in iteration one, any edge with $X_{i,j}^h = 0$ in G_1 is discarded and any edge with $X_{i,j}^h \geq \bar{y}_i/\gamma$ is moved to G_2 (if node j does not belong to G_2 , add it to G_2 also). We denote by $w_{i,j}$ the weight of an edge $(i, j) \in G_2$. Any edge and its weight moved to G_2 will not be changed further. Since $w_{i,j}$ is fixed when (i, j) is inserted to G_2 , we can treat it as a constant thereafter. Consider the linear system $(\mathbf{Ax} = \mathbf{b})$ as in Figure 5.1.

We call the fractional solution \mathbf{x} *canonical*, if $x_{i,j} \in (0, y_i/\gamma)$, for all (i, j) . Clearly $\{X_{i,j}^h\}$, for $(i, j) \in E(G_1)$ is a canonical feasible solution for the linear system in Figure 5.1. We now apply **RandMove** to the linear system defined by

$$\forall j \in J', \quad \sum_{\substack{i \in M', \\ (i,j) \in E(G_1)}} x_{i,j} = 1 - \sum_{\substack{i \in M', \\ (i,j) \in E(G_2)}} w_{i,j} \quad (5.3)$$

$$\forall i \in M', \quad \sum_{\substack{j \in J', \\ (i,j) \in E(G_1)}} p_{i,j} x_{i,j} = \sum_{j \in J'} p_{i,j} X_{i,j}^h - \sum_{\substack{j \in J', \\ (i,j) \in E(G_2)}} p_{i,j} w_{i,j} \quad (5.4)$$

Figure 5.1: Linear System at the beginning of iteration $(h + 1)$

A. To recall, if a linear system is under-determined, we can efficiently find a non-zero vector \mathbf{r} , with $\mathbf{A}\mathbf{r} = \mathbf{0}$. Since \mathbf{x} is canonical, we can also efficiently identify strictly positive reals, α and β , such that for all (i, j) , $x_{i,j} + \alpha r_{i,j}$ and $x_{i,j} - \beta r_{i,j}$ lie in $[0, y_i/\gamma]$ and there exists at least one (i, j) , such that one of the two entries, $x_{i,j} + \alpha r_{i,j}$ and $x_{i,j} - \beta r_{i,j}$, is in $\{0, y_i/\gamma\}$. Therefore, we can apply the basic randomized rounding step, **RandMove**($\mathbf{A}, \mathbf{x}, \mathbf{b}$) : with probability $\frac{\beta}{\alpha+\beta}$, return the vector $\mathbf{x} + \alpha\mathbf{r}$ and with complementary probability of $\frac{\alpha}{\alpha+\beta}$, return the vector $\mathbf{x} - \beta\mathbf{r}$.

If $\mathbf{X} = \mathbf{RandMove}(\mathbf{A}, \mathbf{x}, \mathbf{b})$, then the returned solution has the following properties:

$$\Pr[\mathbf{A}\mathbf{X} = \mathbf{b}] = 1 \quad (5.5)$$

$$\mathbb{E}[X_{i,j}] = x_{i,j} \quad (5.6)$$

If the linear system in Figure 5.1 is under-determined, then we apply **RandMove** to obtain the updated vector \mathbf{X}^{h+1} . If for some (i, j) , $X_{i,j}^{h+1} = 0$, then we remove that edge (variable) from G_1 . If $X_{i,j}^{h+1} = \bar{y}_i/\gamma$, then we remove the edge from G_1 and add it with weight \bar{y}_i/γ to G_2 . Thus the invariants (**I1**, **I3** and

I4) are maintained. Since the weight of any edge in G_2 is never changed and load constraints on all machine nodes belong to the linear system, we get from [92],

Lemma 5.2.3. *For all i, j, h, u , $\mathbb{E} \left[X_{i,j}^{h+1} \mid X_{i,j}^h = u \right] = u$. In particular, $\mathbb{E} \left[X_{i,j}^{h+1} \right] = \bar{x}_{i,j}$. Also for each machine i and iteration h , $\sum_j X_{i,j}^h p_{i,j} = \sum_j \bar{x}_{i,j} p_{i,j}$ with probability 1.*

Thus the invariant **(I2)** is maintained as well.

If the linear system (Figure 5.1) becomes determined, then this step ends. Let M' and N' be the machine and job nodes respectively in G_1 at that point. If $|M'| = m'$ and $|N'| = n'$, then the number of edges in G_1 is $|E(G_1)| \leq m' + n'$. Otherwise, the linear system (Figure 5.1) remains underdetermined. In fact, in each connected component of G_1 , the number of edges is at most the number of vertices. Therefore, each component of G_1 can contain at most one cycle.

2. Rounding on G_1 and G_2

The previous step ensures, that G_1 is almost a forest with each component containing at most one cycle and in G_2 , $X_{i,j} \geq \bar{y}_i/\gamma$, for all $(i, j) \in E(G_2)$. We remove any isolated nodes from G_1 and G_2 , and round them separately.

Further Relaxing the Solution Let us denote the job and the machine nodes in G_1 (G_2) by $J(G_1)$ (or $J(G_2)$) and $M(G_1)$ (or $M(G_2)$) respectively. Consider a job node $j \in J(G_2)$. If $\sum_{i:(i,j) \in E(G_2)} X_{i,j} < 1/\delta$ (we choose δ later), we simply remove all the edges (i, j) from G_2 and the following must hold:

$\sum_{i:(i,j) \in E(G_1)} X_{i,j} \geq 1 - 1/\delta$. Otherwise, if $\sum_{i:(i,j) \in E(G_2)} X_{i,j} \geq 1/\delta$, we remove all edges $(i, j) \in E(G_1)$ from G_1 . Therefore at the end of this modification, a job node can belong to either $J(G_1)$ or $J(G_2)$, but not both. If $j \in J(G_1)$, we have $\sum_{i \in M} X_{i,j} \geq 1 - 1/\delta$. Else, if $j \in J(G_2)$, $\sum_{i \in M} X_{i,j} \geq 1/\delta$.

For the makespan analysis it will be easier to partition the edges incident on a machine node i into two parts – the job nodes incident to it in G_1 and in G_2 . The fractional processing time due to jobs in $J(G_1)$ (or $J(G_2)$) will be denoted by $T'\bar{y}_i$ (or $T''\bar{y}_i$), i.e., $T'\bar{y}_i = \sum_{j \in J(G_1)} p_{i,j} X_{i,j}$ (or $T''\bar{y}_i = \sum_{j \in J(G_2)} p_{i,j} X_{i,j}$).

Rounding on G_2 : In G_2 , for any machine node i , recall $\sum_{j \in J(G_2)} X_{i,j} p_{i,j} = T''y_i$. Since we have for all $i \in M(G_2), j \in J(G_2)$, $X_{i,j} \geq y_i/\gamma$, we have $\sum_{j \in J(G_2)} p_{i,j} \leq T''\gamma$. Therefore, if we decide to open a machine node $i \in M(G_2)$, then we can assign all the nodes $j \in J(G_2)$, that have an edge $(i, j) \in E(G_2)$, by paying at most $T''\gamma$ in the makespan.

Hence, we only concentrate on opening a machine in G_2 , and then if the machine is opened, we assign it all the jobs incident to it in G_2 . For each machine $i \in M(G_2)$, we define $Y_i = \min\{1, \bar{y}_i\delta\}$. Since, for all job nodes $j \in J(G_2)$, we know $\sum_{i \in M(G_2)} X_{i,j} \geq 1/\delta$, after scaling we have for all $j \in J(G_2)$, $\sum_{(i,j) \in E(G_2)} Y_i \geq 1$. Therefore, this exactly forms a fractional set-cover instance, which can be rounded using the randomized rounding method developed in [128] to get activation cost within a factor of $\delta(\log \frac{n}{OPT} + 1)$. The instance in G_2 thus nicely captures the hard part of the problem, which comes from the hardness of

approximating set cover. Thus we have the following lemma.

Lemma 5.2.4. *Considering only the job nodes in G_2 , the final load on any machine $i \in M(G_2)$ is at most $T''\gamma$ and the total activation cost is at most $\delta(\log \frac{n}{OPT} + 1)OPT$, where T'' is the fractional load on machine $i \in M(G_2)$ before rounding on G_2 and OPT is the optimum activation cost.*

Rounding on G_1 : For rounding in G_1 , we root each component of G_1 at the only cycle it has (in the case that the component is a tree, we just root it arbitrarily at any node) and traverse bottom up. We consider all the job nodes except for those in the cycle. If there is a job node j , that is a child of a machine node i , then if $X_{i,j} < 1/\eta$ (η to be fixed later), we remove the edge (i, j) from G_1 . Since initially $j \in J(G_1)$, $\sum_{i \in M} X_{i,j} \geq 1 - 1/\delta$, even after these edges are removed, we have for $j \in J(G_1)$, $\sum_{i \in M(G_1)} X_{i,j} \geq 1 - 1/\delta - 1/\eta$. However if $X_{i,j} \geq 1/\eta$, simply open machine i , if it is not already open and add job j to machine i . Initially $\bar{y}_i \geq 1/\eta$, since $\bar{y}_i \geq X_{i,j}$. The initial contribution to cost by machine i was $\geq \frac{1}{\eta}a_i$. Now it becomes a_i . If $\sum_j \frac{X_{i,j}}{y_i} p_{i,j} = T'$, with $X_{i,j} \geq 1/\eta$, now it can become at most $\eta T'$.

After the above modification, we have one cycle and the yet to be assigned jobs in $J(G_1)$ outside the cycle form disjoint stars, with the job nodes at their centers.

In the cycle, we assign a job to a machine, if its fractional value is at least $\frac{1-\frac{1}{\delta}}{2}$. Consider any job j in the cycle attached to two machines m_1 and m_2 with fractional value $X_{1,j} + X_{2,j} \geq 1 - \frac{1}{\delta}$. Therefore, the job j is assigned at least to an extent of $\frac{1-\frac{1}{\delta}}{2}$ to one of the two machines m_1 and m_2 . If a machine i gets at most one job

from the cycle, then its makespan is at most $T'\eta + \max_{i,j} p_{i,j}$. Otherwise, if it gets two jobs, then its make span is at most

$$\eta \left(T' - \frac{1 - \frac{1}{\delta}}{2} (p_{i,1} + p_{i,2}) \right) + p_{i,1} + p_{i,2},$$

where $p_{i,1}, p_{i,2}$ is the processing time of the two jobs in the cycle that got assigned to machine i . Since $(1 - \frac{1}{\delta}) > \frac{1}{\eta}$, the above expression is at most $\eta(T' - \frac{1}{\eta}(p_{i,1} + p_{i,2})) + p_{i,1} + p_{i,2} < \eta T' + \max_{i,j} p_{i,j}$. The increase in cost is at most $\frac{2}{(1-\frac{1}{\delta})}$ factor of the total optimal cost of the machines in that cycle.

Consider each star, S_j with job node j at its center. Let $i_1, i_2, \dots, i_{\ell_j}$ be all the machine nodes in S_j , then we have, $\sum_{k=1}^{\ell_j} X_{i_k, j} \geq 1 - 1/\delta - 1/\eta$. Therefore $\sum_{k=1}^{\ell_j} \bar{y}_{i_k} \geq 1 - 1/\delta - 1/\eta$. If there is already some opened machine, i_l , assign j to i_l by increasing the makespan at most by an additive T . Otherwise, open machine i_l with the cheapest a_{i_l} . Since the total contribution of these machines to the cost is $\sum_{k=1}^{\ell_j} \bar{y}_{i_k} a_{i_k} \geq \sum_{k=1}^{\ell_j} \bar{y}_{i_k} a_{i_l} \geq (1 - 1/\delta - 1/\eta)a_{i_l}$, we are within a factor $\frac{1}{1-1/\delta-1/\eta}$ of the total cost contributed from G_1 .

Hence, we have the following lemma,

Lemma 5.2.5. *Considering only the job nodes in G_1 , the final load on any machine $i \in M(G_1)$ is at most $T'\eta + \max_{i,j} p_{i,j}$ and the total activation cost is at most $\max \left(\frac{2}{(1-\frac{1}{\delta})}, \frac{1}{\eta}, \frac{1}{(1-1/\delta-1/\eta)} \right) OPT$, where T' is the fractional load on machine $i \in M(G_1)$ before rounding on G_1 and OPT is the optimum activation cost.*

Now combining, Lemma 5.2.4 and 5.2.5, and by optimizing the values of δ, η and γ , we get the following theorem.

Theorem 5.2.6. *A schedule can be constructed efficiently with machine activation cost $2(1 + \frac{1}{\varepsilon})(\ln \frac{n}{OPT} + 1)OPT$ and makespan $(2 + \varepsilon)T$, where T is the optimum makespan possible for any schedule with activation cost OPT .*

Proof. From Lemma 5.2.4 and 5.2.5, we have,

- Machine opening cost is at most $\left(\max \left(\frac{2}{(1-\frac{1}{\delta})}, \frac{1}{\eta}, \frac{1}{(1-1/\delta-1/\eta)} \right) + \delta (\ln \frac{n}{OPT} + 1) \right) OPT$
- Makespan is at most $T (\max(\gamma, \eta)) + \max_{i,j} p_{i,j}$

Now $\eta \geq \gamma$, since otherwise any edge with $X_{i,j} \geq 1/\eta$ will be moved to G_2 and $1-1/\delta \geq 1/\eta$. Now set, $\gamma = \eta$, $\delta = 1 + \zeta$, for some $\zeta > 0$. So $1-1/\delta = \zeta/(1+\zeta)$. Set $1/\eta = \zeta/(1+\zeta) - 1/((1+\zeta)(\ln \frac{n}{OPT} + 1))$. Thus, we have an activation cost at most $2(1 + \zeta)(\ln \frac{n}{OPT} + 1)OPT$ and makespan $\leq T(1 + \frac{\ln n + 1}{\zeta \ln n - 1}) + \max_{i,j} p_{i,j}$. Therefore, if we set $\zeta = 1 + 2/\ln n$, we get an activation cost bound of $4(\ln \frac{n}{OPT} + 1)OPT$ and makespan $\leq 2T + \max_{i,j} p_{i,j}$. In general, by setting $\varepsilon = \frac{1}{\zeta}$, we get an activation cost at most $2(1 + \frac{1}{\varepsilon})(\ln \frac{n}{OPT} + 1)OPT$ and makespan $\leq (2 + \varepsilon)T$. \square

5.3 Minimizing Machine Activation Cost and Assignment

Cost

We now consider the scheduling problem with assignment costs and machine activation costs. As before, each job can be scheduled only on one machine, and

processing job j on machine i requires $p_{i,j}$ time and incurs a cost of $c_{i,j}$. Each machine is available for T time units and the objective is to minimize the total incurred cost. In this version of the machine activation model, we wish to minimize the sum of the machine activation and job assignment costs. Our objective now is

$$\min \sum_{i \in M} a_i y_i + \sum_{(i,j)} c_{i,j} x_{i,j}$$

subject to the same constraints as the LP defined in Eq(5.1). Our algorithm for simultaneous minimization of machine activation and assignment cost follows the same paradigm as in the previous section with a few differences. We specify those here.

1. Transforming the Solution

After solving the LP, we obtain, $C = \sum_{i,j} c_{i,j} x_{i,j}$. Though, we have an additional constraint $C = \sum_{i,j} c_{i,j} x_{i,j}$ to care about, we **do not** include it in the linear system and proceed exactly as in Subsection 5.2. At the end, we have G_1 where each component contains at most one cycle and in G_2 , for all $(i, j) \in E(G_2)$, we have $X_{i,j} \geq \bar{y}_i / \gamma$. By Property 5.6, $\forall i, j, h, E[X_{i,j}^h] = \bar{x}_{i,j}$ and hence, we have that the expected cost is $\sum_{i,j} c_{i,j} \bar{x}_{i,j}$. The procedure can be directly derandomized by the method of conditional expectation giving an 1-approximation to assignment cost.

2. Rounding on G_1, G_2

The first part involves further relaxing the solution, that is identical to the one

described in previous section. Therefore, we now concentrate on rounding G_1 and G_2 separately.

Rounding on G_2 In G_2 , since we have for all $(i, j) \in E(G_2)$, $X_{i,j} = \bar{y}_i/\gamma$, if we decide to open machine i , all the jobs $j \in J(G_2)$ can be assigned to i , by losing only a factor of γ in the makespan. Therefore, we just need to concentrate on minimizing the cost of opening machines and the total assignment cost, subject to the constraint that all the jobs in $J(G_2)$ must have an open machine to get assigned. This is exactly the case of *non-metric uncapacitated facility location* and we can employ the rounding approach developed in [126] to obtain an approximation factor of $O(\log \frac{n+m}{OPT}) + O(1)$ on the machine activation and assignment costs.

Rounding on G_1 Rounding on G_1 is similar to the case when there is no assignment costs with a few modifications. We proceed in the same manner, we assign jobs (not in the cycle) to their parent if the fractional contribution is at least $\frac{1}{\eta}$, we then assign the jobs in the cycle if the fractional contribution is at least $\frac{(1-\frac{1}{\delta})}{2}$ and finally, obtain the stars with job nodes at the centers. Now for each star S_j , with j at its center, we consider all the machine nodes in S_j . If some machine $i \in S_j$ is already open, we make its opening cost 0. Now we open the machine, $\ell \in S_j$, for which $c_j + a_{\ell,j}$ is minimum. Again using the same reasoning as in Subsection 5.2, the total cost does not exceed by more than a factor of $\frac{1}{1-1/\delta-1/\eta}$. We thus again have the following two bounds.

- Machine opening and activation cost is at most

$$\left(\max \left(\frac{2}{(1-\frac{1}{\delta})}, \frac{1}{\eta}, \frac{1}{(1-1/\delta-1/\eta)} \right) + \delta \left(\ln \frac{n+m}{OPT} + 1 \right) \right) OPT$$

- Makespan is at most $T(\max(\gamma, \eta)) + \max_{i,j} p_{i,j}$

Now optimizing α, β, γ , we get the following theorem,

Theorem 5.3.1. *If there is a schedule with total machine activation and assignment cost as OPT and makespan T , then a schedule can be constructed efficiently in polynomial time, with total cost $O(\frac{1}{\varepsilon} (\log \frac{n+m}{OPT} + 1))OPT$ and makespan $\leq (2 + \varepsilon)T$.*

5.4 Extensions: Handling Release Times

Suppose each job j has a machine related release time r_{ij} , i.e, job j can only be processed on machine i after time r_{ij} . We can modify the algorithm in Section 2 to handle release times as follows.

For any “guess” of the makespan T , we let $x_{i,j} = 0$ if $r_{ij} + p_{i,j} > T$ in the LP formulation. Then, we run the $((2 + \varepsilon), 2(1 + \frac{1}{\varepsilon})(\ln \frac{n}{OPT} + 1))$ -approximation regardless of the release times and obtain a subset of active machines and an assignment of jobs to these machines. Suppose the subset J_i of jobs is assigned to machine i . We can now schedule the jobs in J_i on machine i in order by release time. It is not hard to see the makespan of machine i is at most $T + \sum_{j \in J_i} p_{i,j}$ since every job can be scheduled on machine i after time T . Therefore, we get a

$(3 + \varepsilon, 2(1 + \frac{1}{\varepsilon})(\log \frac{n}{OPT} + O(1)))$ approximation. Similar extensions can be done for the case with activation and assignment costs.

This concludes the description of this chapter. In summary, herein, we develop a new model for saving energy on data centers via scheduling mechanism. Data centers are modeled by unrelated parallel machines, and energy consumption is modeled by the cost of machine activation, in addition to the power usage to process jobs on these machines. We describe an approximation algorithm that optimizes makespan given a bound on the energy cost. The algorithm utilizes the rounding technique developed in Chapter 2.

Fair Allocation: Maxmin Fair Allocation Problem

In the next two chapters, we study fair allocation problems which form an important class of resource allocation problems. The fair allocation problems deal with distribution of *indivisible* items, i.e., items that cannot be broken and assigned fractionally among persons. Each person has individual utilities for the items and the items must be allocated to them satisfying certain criteria. Many issues of fair allocations have spanned the literature of economists [30, 106, 141], as well as the literature of the operations research community [81, 131]. Questions such as the structural properties of markets for the existence/non-existence of fair allocation with certain properties, role of money, cases with a single item, or with general utility functions have resulted in several research challenges in the last decade [90, 130]. In computer science, one such fair allocation problem that has received a good deal of attention is the *max-min fair allocation problem* [15, 16, 20, 26, 34, 60, 118].

In this chapter, we study the properties of a configuration LP for max-min fair allocation and provide tighter results for its integrality gap. In the next chapter, we

will see a special case of max-min fair allocation which goes by the name of the Santa Claus problem.

6.1 Max-Min Fair Allocation: A Configuration LP Approach

In this problem there are m goods that need to be distributed indivisibly among k persons. Each person i has a non-negative integer valuation $u_{i,j}$ for good j . The valuation functions are linear, i.e., $u_{i,C} = \sum_{j \in C} u_{i,j}$ for any set of C goods. The goal is to allocate each good to a person such that the “least happy person is as happy as possible”: i.e., $\min_i u_{i,C}$ is maximized. Our main contribution in this regard is to near-optimally pin-point the integrality gap of a *configuration LP* previously proposed and analyzed in [16, 20]. Our algorithm uses bipartite dependent rounding [64] and its generalization to weighted graphs. Bipartite dependent rounding can be viewed as a specific type of **RandMove** on bipartite graphs. A crucial ingredient of our analysis is to show certain random variables satisfy the property of *negative correlation* and hence the Chernoff type concentration bounds can be applied to guarantee small deviation from the expected value of their sum.

Configuration LP for Max-Min Fair Allocation The configuration LP formulation for the max-min fair allocation problem was first considered in [20]. A *configuration* is a subset of items and in the LP there is a variable for each valid config-

uration. Using binary search, first the optimal solution value T is guessed and then we define valid configurations based on the approximation factor λ sought for. We call a configuration C *valid* for person i if either of the following two conditions hold:

- $u_{i,C} \geq T$ and all the items in C have value at most $\frac{T}{\lambda}$. These are called *small* items.
- C contains only one item j and $u_{i,j} \geq \frac{T}{\lambda}$. We call such an item j to be a *big* item for person i .

We define a variable $x_{i,C}$ for assigning a valid configuration C to person i . Let $C(i, T)$ denote the set of all valid configurations corresponding to person i with respect to T . The configuration LP relaxation of the problem is as follows:

$$\begin{aligned}
 \forall j : \sum_{C \ni j} \sum_i x_{i,C} &\leq 1 & (6.1) \\
 \forall i : \sum_{C \in C(i,T)} x_{i,C} &= 1 \\
 \forall i, C : x_{i,C} &\geq 0
 \end{aligned}$$

The above LP formulation may have exponential number of variables, yet if the LP is feasible, then a fractional allocation where each person receives either a big item or at least a utility of $T(1 - \varepsilon)$ can be computed in polynomial time [20]. Here ε is any constant greater than zero. In the subsequent discussion and analysis, we

ignore the multiplicative $1 + \varepsilon$ factor; it is hidden in the Θ notation of the ultimate approximation ratio.

The integrality gap of the above configuration LP is $\Omega(\frac{1}{\sqrt{k}})$ and again follows from [20]. In [16], Asadpour and Saberi gave a rounding procedure for the configuration LP that achieved an approximation factor of $O\left(\frac{1}{\sqrt{k}(\ln k)^3}\right)$. Here we further lower the gap and prove the following theorem.

Theorem 6.1.1. *Given any feasible solution to the configuration LP, it can be rounded to a feasible integer solution such that every person gets at least $\Theta\left(\frac{1}{\sqrt{k \ln k}}\right)$ fraction of the optimal utility with probability at least $1 - \Theta(\frac{1}{k})$ in polynomial time.*

Our proof is also significantly simpler than the one in [16].

Note that the recent work of Chakrabarty, Chuzhoy and Khanna [44] has an improved approximation factor of m^ε (also note that $m \geq k$) but that does not use the configuration LP.

In the context of fair allocation, an additional important criterion can be an *equitable partitioning* of goods: we may impose an upper bound on the number of items a person might receive. For example, we may want each person to receive at most $\lceil \frac{m}{k} \rceil$ goods. Theorem 4.1.1 from Chapter 4 leads to the following:

Theorem 6.1.2. *Suppose, in max-min allocation, we are given upper bounds c_i on the number of items that each person i can receive, in addition to the utility values $u_{i,j}$. Let T be the optimal max-min allocation value that satisfies c_i for all i . Then,*

we can efficiently construct an allocation in which for each person i the bound c_i holds and he receives a total utility of at least $T - \max_j u_{i,j}$.

This generalizes the result of [28], which yields the “ $T - \max_j u_{i,j}$ ” value when no bounds such as the c_i are given. To our knowledge, the results of [15, 16, 20, 34] do not carry over to the setting of such “fairness bounds” c_i .

6.1.1 Algorithm for Max-Min Fair Allocation

We now describe the algorithm and the proof of Theorem 6.1.1.

Algorithm We define a weighted bipartite graph G with the vertex set $A \cup B$ corresponding to the persons and the items respectively. There is an edge between a vertex corresponding to person $i \in A$ and item $j \in B$, if a configuration C containing j is fractionally assigned to i . Define

$$w_{i,j} = \sum_{C \ni j} x_{i,C},$$

i.e., $w_{i,j}$ is the fraction of item j that is allocated to person i by the fractional solution of the LP. An edge (i, j) is called a matching edge, if the item j is big for person i . Otherwise it is called a flow edge.

Let M and F represent the set of matching and flow edges respectively. For each vertex $v \in A \cup B$, let m_v be the total fraction of the matching edges incident to it. Also define $f_v = 1 - m_v$. The main steps of the algorithm are as follows,

1. Guess the value of the optimal solution T by doing a binary search. Solve LP (7.1). Obtain the set M and m_v, f_v for each vertex v in G constructed from the LP (7.1) solution.

2 **Allocating Big Items** : Select a random matching from edges in M using *bipartite dependent rounding* (Chapter 2, Section) such that for every $v \in A \cup B$, the probability that v is saturated by the matching is $m_v = 1 - f_v$.

3 **Allocating Small Items** :

(a) Discard any item j , with $m_j \geq (1 - \varepsilon_1)$, $\varepsilon_1 = \sqrt{\frac{\ln k}{k}}$, and also discard all the persons and the items saturated by the matching.

(b) (Scaling) In the remaining graph containing only flow edges for unsaturated persons and items, set for each person i , $w'_{i,j} = \frac{w_{i,j}}{f_i}$, $\forall j$.

(c) Further discard any item j with $\sum_i w'_{i,j} \geq \frac{(3+o(1)) \ln k}{\ln \ln k}$.

(d) (weighted bipartite dependent rounding) Scale down the weights on all the remaining edges by a factor of $\frac{(3+o(1)) \ln k}{\ln \ln k}$ and do a *weighted bipartite dependent rounding* to assign items to persons.

We now analyze each step. The main proof idea is in showing that there remains enough left-over utility in the flow graph for each person not saturated by the matching. This is obtained through proving a new negative correlation property among the random variables defined on a collection of vertices. Previously,

the negative correlation property due to bipartite dependent rounding has only been proven for variables defined on edges incident on any particular vertex. Such “local” negative correlation property is not sufficient for our case.

Allocating Big Items

Consider the edges in M in the person-item bipartite graph. Remove all the edges (i, j) that have already been rounded to 0 or 1. Additionally, if an edge is rounded to 1, remove both its endpoints i and j . We initialize for each $(i, j) \in M$, $y_{i,j} = w_{i,j}$, and modify the $y_{i,j}$ values probabilistically in rounds using bipartite dependent rounding.

Recall from Chapter 2, the bipartite dependent rounding being a special case of our new rounding scheme also satisfies the following two properties. Let $Y_{i,j}$ denote the rounded variables then

$$\forall i, j, \mathbb{E}[Y_{i,j}] = y_{i,j} \tag{6.2}$$

$$\exists i, j, Y_{i,j} \in \{0, 1\} \tag{6.3}$$

Thus, Property (6.2) guarantees for every edge (i, j) , $\mathbb{E}[Y_{i,j}] = w_{i,j}$. This gives the following corollary.

Corollary 6.1.3. *The probability that a vertex $v \in A \cup B$ is saturated in the matching generated by the algorithm is m_v .*

Proof. Let there be $l \geq 0$ edges $e_1, e_2, \dots, e_l \in M$ that are incident on v . Then,

$$\begin{aligned} \Pr[v \text{ is saturated}] &= \Pr[\exists e_i, i \in [1, l] \text{ s.t. } v \text{ is matched with } e_i] \\ &= \sum_{i=1}^l \Pr[v \text{ is matched with } e_i] = \sum_{i=1}^l w_i = m_v \end{aligned}$$

Here the second equality follows by replacing the union bound by sum since the events are mutually exclusive. \square

Now we prove two additional properties of this rounding, which will be used crucially for the analysis of the next step.

Definition 6.1.4 (Negative Correlation for Indicator Random Variables). *A collection of indicator random variables $\{z_i\}, i \in [1, n]$ are said to be negatively correlated, if for any subset of t variables, $t \in [1, n]$, and any $b \in \{0, 1\}$,*

$$\Pr\left[\bigwedge_{j=1}^t z_{i_j} = b\right] \leq \prod_{j=1}^t \Pr[z_{i_j} = b].$$

Theorem 6.1.5. *Define an indicator random variable z_j for each item $j \in B$ with $m_j < 1$, such that $z_j = 1$ if item j is saturated by the matching. Then, the indicator random variables $\{z_j\}$ are negatively correlated.*

Proof. We have already shown this proof in Chapter 2, Theorem 2.1.2. \square

As a corollary of the above theorem, we get the following claim.

Corollary 6.1.6. *Define an indicator random variable z_i for each person $i \in A$, such that $z_i = 1$ if person i is saturated by the matching. Then, the indicator*

random variables $\{z_i\}$ are negatively correlated.

Proof. Do the same analysis as in Theorem 2.1.2 with items replaced by persons.

□

Allocating small items

We now prove in Lemma 11.3.1 that after the matching phase, each unsaturated person has available items with utility at least $\sqrt{\frac{\ln k}{k} \frac{T}{5}}$ in the flow graph. Additionally we prove in Lemma 6.1.8 that each item is not claimed more than $3 \ln k / \ln \ln k$. Both the results are probabilistic and hold with high probability. We use the following form of the well-known Chernoff-Hoeffding Bound.

The Chernoff-Hoeffding Bound [114]: *Suppose $X = \sum_i X_i$ where X_i are independent/negatively correlated random variables taking values in $[0, 1]$ with $E[X] = \mu$, then*

1. for $1 > \delta > 0$, we have $\Pr[X \geq \mu(1 + \delta)] \leq e^{-\mu\delta^2/3}$,
2. for $1 > \delta > 0$, we have $\Pr[X \leq \mu(1 - \delta)] \leq e^{-\mu\delta^2/2}$,
3. for $\delta \geq 1$, we have $\Pr[X \geq \mu(1 + \delta)] \leq e^{-\mu(\delta+1) \ln(\delta+1) [1 - \frac{\delta}{(1+\delta) \ln(1+\delta)}]}$.

Lemma 6.1.7. *After Allocation of Big Items by bipartite dependent rounding, each unsatisfied person has a total utility of at least $\sqrt{\frac{\ln k}{k} \frac{T}{5}}$ from the unsaturated items with probability at least $1 - \frac{1}{k}$.*

Proof. Consider a person v who is not saturated by the matching. In step (a) of **Allocation of Small Items**, all items j with m_j at least $(1 - \varepsilon_1)$ are discarded. We will set $\varepsilon_1 = \sqrt{\frac{\ln k}{k}}$ later. Since the total sum of m_j can be at most k (the number of persons), there can be at most $\frac{k}{1 - \varepsilon_1}$ items with m_j at least $1 - \varepsilon_1$. Therefore, for the remaining items, we have $f_j \geq \varepsilon_1$. Each person is connected only to small items in the flow graph. After removing the items with m_j at least $1 - \varepsilon_1$, the remaining utility in the flow graph for person v is at least

$$\left(T - \sum_{j: f_j \leq \varepsilon_1 \text{ and } j \text{ is unsaturated}} u_{v,j} f_j \right) \geq \left(T - \frac{\varepsilon_1 k}{1 - \varepsilon_1} \frac{T}{\lambda} \right). \quad (6.4)$$

Define $w'_{v,j} = \frac{w_{v,j}}{f_v}$ and select a $\lambda_1 \leq \lambda$. Now consider random variables $Y_{v,j}$ for each of these unsaturated items:

$$Y_{v,j} = \begin{cases} \frac{w'_{v,j} u_{v,j}}{T/\lambda_1} & : \text{if item } j \text{ is not saturated} \\ 0 & : \text{otherwise} \end{cases} \quad (6.5)$$

Since each $u_{v,j} \leq T/\lambda \leq T/\lambda_1$ and $w_{v,j} \leq f_v$, $Y_{v,j}$ are random variables bounded in $[0, 1]$. Person v is not saturated by the matching with probability $1 - m_v = f_v$. Each such person v gets a fractional utility of $w'_{v,j} u_{v,j}$ from the small (with respect to the person) item j in the flow graph, if item j is not saturated by the matching. The later happens with probability f_j .

Define $G_v = \sum_j Y_{v,j}$. Then $\frac{T}{\lambda_1} G_v$ is the total fractional utility after step (b). It

follows from Equation 9.1.1

$$\mathbb{E}[G_v] = \sum_j \frac{w'_{v,j} u_{v,j} f_j}{T/\lambda_1} \geq \varepsilon_1 \lambda_1 \left(1 - \frac{\varepsilon_1 k}{(1 - \varepsilon_1) \lambda_1}\right)$$

Set $\varepsilon_1 = \sqrt{\frac{\ln k}{k}}$, $\lambda_1 = 25\sqrt{k \ln k}$ and we have $\lambda \geq \lambda_1$.

Thus for sufficiently large k ,

$$\begin{aligned} \mathbb{E}[G_v] &\geq \varepsilon_1 \lambda_1 \left(1 - \frac{\varepsilon_1 k}{(1 - \varepsilon_1) \lambda_1}\right) \\ &\geq 24 \ln k \end{aligned}$$

That $Y_{v,j}$'s are negatively correlated follows from Theorem 2.1.2. Therefore, applying the standard Chernoff-Hoeffding bound for the negatively-correlated random variables, we get for any $\delta \in (0, 1)$

$$\begin{aligned} \Pr[G_v \leq (1 - \delta)\mathbb{E}[G_v]] &\leq e^{-E[G_v]\delta^2/3} \\ &\leq e^{-24 \ln k/12} \text{ for } \delta \geq \frac{1}{2} \\ &= \frac{1}{k^2}. \end{aligned}$$

Thus we get

$$\Pr\left[\frac{T}{\lambda_1} G_v \leq \frac{1}{2} \frac{T}{\lambda_1} \mathbb{E}[G_v]\right] \leq \frac{1}{k^2}$$

Hence,

$$\exists v, \Pr \left[\frac{T}{\lambda_1} G_v \leq \frac{1}{2} \frac{T}{\lambda_1} \mathbb{E}[G_v] \right] \leq \frac{1}{k}$$

Therefore the net fractional utility that remains for each person in the flow graph after scaling is at least $\frac{1}{2} \frac{T}{\lambda_1} \mathbb{E}[G_v] = \frac{1}{2} \frac{T}{25\sqrt{k \ln k}} 12 \ln k \geq \frac{T}{5} \sqrt{\frac{\ln k}{k}}$, with probability at least $1 - \frac{1}{k}$. \square

Lemma 6.1.8. *After the matching and the scaling (step (b)), each unsaturated item has a total fractional incident edge-weight to be at most $\frac{3 \ln k}{\ln \ln k}$ from the unsaturated persons with probability at least $1 - \frac{1}{k^3}$.*

Proof. Note that for any person v and for any job j that is small for v , $w_{v,j} \leq f_v$, hence $w'_{v,j} = \frac{w_{v,j}}{f_v} \leq 1$. Fix an item j , and define a random variable $Z_{v,j}$ for each person such that

$$Z_{v,j} = \begin{cases} w'_{v,j} & : \text{if person } v \text{ is not saturated} \\ 0 & : \text{otherwise} \end{cases} \quad (6.6)$$

Let $X_j = \sum_v Z_{v,j}$. Then X_j is the total weight of all the edges incident on item j in the flow graph after scaling and removal of all saturated persons. We have $\mathbb{E}[X_j] = \sum_v w'_{v,j} f_v = \sum_v w_{v,j} \leq 1$. Now that the variables $Z_{v,j}$ are negatively correlated follows from Corollary 6.1.6, and thus applying the Chernoff-Hoeffding

bound for the negatively correlated variables we get

$$\Pr \left[X_j \geq \frac{(3 + o(1)) \ln k}{\ln \ln k} \right] \leq \frac{1}{k^3}$$

This completes the proof. \square

Recall the third step, step (c), of **Allocating Small Bundles**. Any job in the remaining flow graph with total weight of incident edges more than $\frac{(3+o(1)) \ln k}{\ln \ln k}$ is discarded in this step. We now calculate the utility that remains for each person in the flow graph after step (c).

Lemma 6.1.9. *After removing all the items that have total degree more than $\frac{(3+o(1)) \ln k}{\ln \ln k}$ in the flow graph, that is after step (c) of **Allocating Small Items**, the remaining utility of each unsaturated person in the flow graph is at least $\sqrt{\frac{\ln k}{k}} \frac{T}{2^{*(3+o(1))}}$ with probability at least $1 - \frac{2}{k}$.*

Proof. Fix a person v and consider the utility that v obtains from the fractional assignments in the flow graph before step (c). It is at least $\sqrt{\frac{1}{k \ln k}} \frac{T}{4}$ from Lemma 11.3.1. Define a random variable for each item that v claims with nonzero value in the flow graph at step (b):

$$Z'_{v,j} = \begin{cases} u_{v,j} & : \text{if item } j \text{ has total weighted degree at least } \frac{(3+o(1)) \ln k}{\ln \ln k} \\ 0 & : \text{otherwise} \end{cases} \quad (6.7)$$

We have $\Pr \left[Z'_{v,j} = u_{v,j} \right] \leq \frac{1}{k^3}$ from Lemma 6.1.8. Therefore, the expected utility from all the items in the flow graph that have total incident weight more than $\frac{(3+o(1)) \ln k}{\ln \ln k}$ is at most $\frac{T}{k^3}$. Hence by Markov's inequality, the utility from the discarded items is more than $\frac{T}{k}$ is at most $\frac{1}{k^2}$. Now, by union bound, the utility from the discarded items is more than $\frac{T}{k}$ for at least one unsaturated person is at most $\frac{1}{k}$. The initial utility before step (c) was at least $\sqrt{\frac{\ln k}{k}} \frac{T}{5}$ with probability $1 - \frac{1}{k}$. Thus after step (c), the remaining utility is at least $\sqrt{\frac{\ln k}{k}} \frac{T}{5} - \frac{T}{k}$ with probability at least $1 - \frac{2}{k}$. Hence, the result follows. \square

The next and the final step (d) of allocations is to do a weighted dependent rounding on a *scaled down* flow graph. The weight on the remaining edges is scaled down by a factor of $\frac{(3+o(1)) \ln k}{\ln \ln k}$ and hence for every item node that survives step (c), the total edge-weight incident on it is at most *one*. Let us denote by $W_{i,j}$ the fractional weight on the edge (i, j) in this graph. Hence after scaling down the utility of any person v in the flow graph is $\sum_j u_{v,j} W_{v,j} \geq \frac{\ln \ln k}{\ln k} \sqrt{\frac{\ln k}{k}} \frac{T}{2*(3+o(1))} = \frac{\ln \ln k}{\sqrt{k} \ln k} \frac{T}{2*(3+o(1))}$.

Weighted Dependent Rounding We remove all (i, j) that have already been rounded to 0 or 1. Let F' be the current graph consisting of those $W_{i,j}$ that lie in $(0, 1)$. Choose any maximal path $P = (v_0, v_1, \dots, v_s)$ or a cycle $C = (v_0, v_1, \dots, v_s = v_0)$. The current W value of an edge $e_t = (v_{t-1}, v_t)$ is denoted by y_t , that is $y_t = W_{t-1,t}$.

We next choose the values z_1, z_2, \dots, z_s such that any unsaturated person retains the utility it started with after scaling down, as long as there are at least two edges incident to it. We update the W value of each edge $e_t = (v_{t-1}, v_t)$ to $y_t + z_t$.

Suppose we have initialized some value for z_1 and that we have chosen the increments z_1, z_2, \dots, z_t for some $t \geq 1$. Then the value z_{t+1} corresponding to the edge $e_{t+1} = (v_t, v_{t+1})$ is chosen as follows:

(PI) v_t is an item, then $v_{t+1} = -v_t$. (Each item is not assigned more than once.)

(PII) v_t is a person. Then choose z_{t+1} so that the utility of w_t remains unchanged.

$$\text{Set } z_{t+1} = z_t \frac{-u_{v_t, v_{t-1}}}{u_{v_t, v_{t+1}}}.$$

The vector $z = (z_1, z_2, \dots, z_s)$ is completely determined by z_1 . We denote this by $f(z)$.

Now let μ be the smallest positive value such that if we set $z_1 = \mu$, then all the W values (after incrementing by the vector z as specified above) stay in $[0, 1]$, and at least one of them becomes 0 or 1. Similarly, let γ be the smallest value such that if we set $z_1 = -\gamma$, then this rounding progress property holds.

When considering a cycle, assume v_0 is a person. The assignment of z_i values ensure all the objects in the cycle are assigned exactly once and utility of all the persons except v_0 remains unaffected. Now the change in the value of z_s is

$$-z_1 \frac{u_{v_2, v_1} u_{v_4, v_3} \dots u_{v_{s-1}, v_{s-2}}}{u_{v_2, v_3} u_{v_4, v_5} \dots u_{v_{s-1}, v_s}}.$$

If

$$\frac{u_{v_2, v_1} u_{v_4, v_3} \cdots u_{v_{s-1}, v_{s-2}}}{u_{v_2, v_3} u_{v_4, v_5} \cdots u_{v_{s-1}, v_s}} > 1,$$

we set $z_1 = -\gamma$, else we set $z_1 = \mu$. Therefore the utility of the person v_0 can only increase.

When we are considering a maximal path we choose the vector z as either $f(\mu)$ or $f(-\gamma)$ arbitrarily.

Lemma 6.1.10. *Each person unsaturated by the matching receives a utility of at least $\Theta(\frac{\ln \ln k}{\sqrt{k \ln k}} T)$ after step (d).*

Proof. While the weighted dependent rounding scheme is applied on a cycle, all persons in the remaining graph maintains their utility. Only when the rounding is applied on a maximal path, the two persons at two ends might lose one item.

Hence, the net utility received by any person after step (d) is at most $\frac{T}{\lambda}$ less than what it was just before starting the weighted dependent rounding step. Thus each person receives a utility of $\frac{\ln \ln k}{\sqrt{k \ln k}} \frac{T}{2^{*(3+o(1))}} - \frac{T}{\lambda}$. From Lemma 11.3.1, $\lambda \geq 25\sqrt{k \ln k}$. Substituting $\lambda = 25\sqrt{k \ln k}$, we get the desired result. \square

We have thus established Theorem 6.1.1. The approximation ratio is $\Theta(\frac{1}{\sqrt{k \ln k}})$. This provides an upper bound of $\sqrt{k \ln k}$ on the integrality gap of the configuration LP for the max-min fair allocation problem. In contrast, the lower bound is $\Omega(\sqrt{k})$ [20]. Theorem 6.1.2 that incorporates fairness in allocation by providing a limit on the cardinality of items each person can receive is a direct corollary of Theorem

4.1.1. Such fairness results in the context of the max-min fair allocation problem was not known earlier.

Fair Allocation: The Santa Claus Problem

The *Santa Claus* problem is the restricted assignment version of the max-min allocation problem of indivisible goods. In this chapter we present the first efficient randomized constant-factor approximation algorithm for this problem.

In the max-min allocation problem, as we saw in the previous chapter, there is a set \mathcal{C} of n items, and m children. The value (utility) of item j to child i is $p_{i,j} \geq 0$. An item can be assigned to only one child. If a child i receives a subset of the items $S_i \subseteq \mathcal{C}$, then the total valuation of the items received by i is $\sum_{j \in S_i} p(i, j)$. The goal is to maximize the minimum total valuation of the items received by any child, that is, to maximize $\min_i \sum_{j \in S_i} p(i, j)$. (The “minmax” version of this “maxmin” problem is the classical problem of makespan minimization in unrelated parallel machine scheduling [96].) This problem has received much attention recently [15, 16, 20, 26, 34, 60, 118].

A restricted version of max-min allocation is where each item has an intrinsic value, and where for every child i , $p_{i,j}$ is either p_j or 0. This is known as the

Santa Claus problem. The Santa Claus problem is NP-hard and no efficient approximation algorithm better than $1/2$ can be obtained unless $P = NP$ [28]. Bansal and Sviridenko [20] considered a linear-programming (LP) relaxation of the problem known as the configuration LP, and showed how to round this LP to obtain an $O(\log \log \log m / \log \log m)$ -approximation algorithm for the Santa Claus problem. They also showed a reduction to a crisp combinatorial problem, a feasible solution to which implies a constant-factor integrality gap for the configuration LP.

Subsequently, Feige [60] showed that the configuration LP has a constant integrality gap. Normally such a proof immediately gives a constant-factor approximation algorithm that rounds an LP solution along the line of the integrality-gap proof. In this case Feige's proof could not be made constructive because it was heavily based on repeated reductions that apply the asymmetric version of the LLL to exponentially many events. Due to this unsatisfactory situation, the Santa Claus problem was the first on a list of problems reported in the survey "Estimation Algorithms versus Approximation Algorithms" [61] for which a constructive proof would be desirable. Using a completely different approach, Asadpour, Feige and Saberi [15] could show that the configuration LP has an integrality gap of at most $\frac{1}{5}$. Their proof uses local-search and hypergraph matching theorems of Haxell [77]. Haxell's theorems are again highly non-constructive and the stated local-search problem is not known to be efficiently solvable. Thus this second non-constructive proof still left the question of a constant-factor approximation algorithm open.

In this chapter we show how our Theorem 3.3.3 can be used to easily and directly constructivize the LLL-based proof of Feige [60], giving the first constant-factor approximation algorithm for the Santa Claus problem.

It is to be noted that the more general max-min fair allocation problem appears significantly harder. It is known that for general max-min fair allocation, the configuration LP has a gap of $O(\sqrt{m})$. Asadpour and Saberi [16] gave an $O(\sqrt{n} \log^3(n))$ approximation factor for this problem using the configuration LP. In Chapter 6, we showed an improved approximation factor of $O(\sqrt{n \log n / \log \log n})$. So far the best approximation ratio known for this problem due to Chakraborty, Chuzhoy and Khanna is $O(n^\varepsilon)$ [34], for any constant $\varepsilon > 0$; their algorithm runs in $O(n^{1/\varepsilon})$ time.

7.1 Algorithm for the Santa Claus Problem

We focus on the Santa Claus problem here. We start by describing the configuration LP and the reduction of it to a combinatorial problem over a set system, albeit with a constant factor loss in approximation. Next we give a constructive solution for the set system problem, thus providing a constant-factor approximation algorithm for the Santa Claus problem.

We guess the optimal solution value T using binary search. An item j is said to be small, if $p_j < \alpha T$, otherwise it is said to be big. Here $\alpha < 1$ is the approximation ratio, which will get fixed later. A configuration is a subset of items. The value of

a configuration C to child i is denoted by $p_{i,C} = \sum_{j \in C} p_{i,j}$. A configuration C is called valid for child i if:

- $p_{i,C} \geq T$ and all the items are small; or
- C contains only one item j and $p_{i,j} = p_j \geq \alpha T$, that is, j is a big item for child i .

Let $C(i, T)$ denote the set of all valid configurations corresponding to child i with respect to T . We define an indicator variable $y_{i,C}$ for each child i and all valid configurations $C \in C(i, T)$ such that it is 1 if child i receives configuration C and 0 otherwise. These variables are relaxed to take any fractional value in $[0, 1]$ to obtain the configuration LP relaxation.

$$\begin{aligned}
 \forall j : \sum_{C \ni j} \sum_i y_{i,C} &\leq 1 & (7.1) \\
 \forall i : \sum_{C \in C(i,T)} y_{i,C} &= 1 \\
 \forall i, C : y_{i,C} &\geq 0
 \end{aligned}$$

Bansal and Sviridenko showed that if the above LP is feasible, then it is possible to find a fractional allocation that assigns a configuration with value at least $(1-\varepsilon)T$ to each child in polynomial time.

The algorithm of Bansal and Sviridenko starts by solving the configuration LP (7.1). Then by various steps of simplification, they reduce the problem to the fol-

lowing instance:

There are p groups, each group containing l children. Each child is associated with a collection of k items with a total valuation of $\frac{T}{c}$, for some constant $c > 0$. Each item appears in at most βl sets for some $\beta \leq 3$. Such an instance is referred to as (k, l, β) -system.

The goal is to efficiently select one child from each group and assign at least $\lfloor \gamma k \rfloor$ items to each of the chosen children, such that each item is assigned only once. If such an assignment exists, then the corresponding (k, l, β) -system is said to be γ -good (k, l, β) -system.

Feige showed that indeed the (k, l, β) -system that results from the configuration LP is γ -good, where $\gamma = O\left(\frac{1}{\max(1, \beta)}\right)$ [60]. This established a *constant* factor integrality gap for the configuration LP. However, the proof being non-constructive, no algorithm was known to efficiently find such an assignment. In the remaining of this section, we make Feige's argument constructive, thus giving a constant-factor approximation algorithm for the Santa Claus problem. But before that, for the sake of completeness, we briefly describe the procedure that obtains a (k, l, β) -system from an optimal solution of the configuration LP [20].

7.1.1 From a configuration LP solution to a (k, l, β) -system

The algorithm starts by simplifying the assignment of *big* items in an optimal solution (say) y^* of the configuration LP. Let J_B denote the set of big items. Consider a bipartite graph G with children M on the right side and big items J_B on the left side. An edge $(i, j), i \in M, j \in J_B$ of weight $w_{i,j} = \sum_{j \in C(i,T)} y_{i,C}^*$ is inserted in G if $w_{i,j} > 0$. These $w_{i,j}$ values are then modified such that after the modification the edges of G with weight in $(0, 1)$ form a forest.

Lemma 5 [20]. *The solution y^* can be transformed into another feasible solution of the configuration LP where the graph G is a forest.*

The transformation is performed using the simple cycle-breaking trick. Each cycle is broken into two matchings; weights on the edges of one matching are increased gradually while the weights on the other are decreased until some weights hit 0 or 1. If a $w_{i,j}$ becomes 0 in this procedure, the edge (i, j) is removed from G . Else if it becomes 1, then item j is permanently assigned to child i and the edge (i, j) is removed.

Suppose G' is the forest obtained after this transformation. The forest structure is then further exploited to form groups of children and big items.

Lemma 6 [20]. *The solution y^* can be transformed into another solution y' such that children M and big items J_B can be clustered into p groups M_1, M_2, \dots, M_p and $J_{B,1}, J_{B,2}, \dots, J_{B,p}$ respectively with the following properties.*

1. For each $i = 1, 2, \dots, p$, the number of jobs $J_{B,i}$ in group M_i is exactly $|M_i| - 1$. The group $J_{B,i}$ could possibly be empty.
2. Within each group the assignment of big job is entirely flexible in the sense that they can be placed feasibly on any of the $|M_i| - 1$ children out of the $|M_i|$ children.
3. For each group M_i , the solution y' assigns exactly one unit of small configurations to children in M_i and all the $|M_i| - 1$ units of configurations correspond to big jobs in $J_{B,i}$. Also, for each small job j , $\sum_{C \ni j} \sum_i y'_{i,C} \leq 2$.

Lemma 6 implies that the assignment of big items to children in a group is completely flexible and can be ignored. We only need to choose one child from each group who will be satisfied by a configuration of small items. Let y' assigns a small configuration C to an extent of $y'_{m,C}$ to some child $c \in M_i, i \in [1, p]$, then we say that M_i contains the small configuration C for child $c \in M_i$. Without loss of generality, it can be assumed that each child in the groups is fractionally assigned to exactly one small configuration. Bansal and Sviridenko further showed that y' can again be simplified such that each small configuration is assigned to at least to an extent of $\frac{1}{l} = \frac{1}{n+m}$ to each child and for each small job j , $\sum_{C \ni j} \sum_i y'_{i,C} \leq 3$. This implies, if we consider all the small configurations across p groups, then each small job appears in at most βl configurations, where $\beta = 3$.

Finally, the following lemma shows that by losing a constant factor in the approximation, one can assume that all the small jobs have same size.

Lemma 8 [20]. *Given the algorithmic framework above, by losing a constant factor in the approximation, each small job can be assumed to have size $\frac{\epsilon T}{n}$.*

As a consequence of the above lemma, we now have the following scenario.

There are p groups M_1, M_2, \dots, M_p , each containing at most l children. Each child is associated with a set that contains $k = \Theta(\frac{n}{\epsilon})$ items. Each item belongs to at most βl sets. The goal is to pick one child from each group and assign at least a constant fraction of the items in its set such that each item is assigned exactly once.

Therefore, we arrive at what is referred as a (k, l, β) -system.

7.1.2 Construction of a γ -good solution for a (k, l, β) -system

We now point out the main steps in Feige's algorithm, and in detail, describe the modifications required to make Feige's algorithm constructive.

Feige's Nonconstructive Proof for γ -good (k, l, β) -system: Feige's approach is based on a systematic reduction of k and l in iterations, finally arriving to a system where k or l are constants. For constant k or l the following lemma asserts a constant γ .

Lemma 7.1.1 (Lemma 2.1 and 2.2 of [60]). *For every (k, l, β) -system a γ -good solution with γ satisfying, $\gamma = \frac{1}{k}$ or $\gamma k = \lfloor \frac{k}{\beta l} \rfloor$ can be found efficiently.*

The reduction of (k, l, β) -system to constant k and l involves two main lemmas, which we refer to as *Reduce- l* lemma and *Reduce- k* lemma respectively.

Lemma 7.1.2 (Lemma 2.3 of [60], Reduce-1). *For $l > c$ (c is a sufficiently large constant), every γ -good (k, l, β) -system with $k \leq l$ can be transformed into a γ -good (k, l', β') -system with $l' \leq \log^5 l$ and $\beta' \leq \beta(1 + \frac{1}{\log l})$.*

Lemma 7.1.3 (Lemma 2.4 of [60], Reduce-k). *Every (k, l, β) -system with $k \geq l \geq c$ can be transformed into a (k', l, β) -system with $k' \leq \frac{k}{2}$ and with the following additional property: if the original system is not γ -good, then the new system is not γ' -good for $\gamma' = \gamma(1 + \frac{3 \log k}{\sqrt{\gamma k}})$. Conversely, if the new system is γ' -good, then the original system was γ -good.*

If β is not a constant to start with, then by applying the following lemma repeatedly, β can be reduced below 1.

Lemma 7.1.4 (Lemma 2.5 of [60]). *For $l > c$, every γ -good (k, l, β) -system can be transformed into a γ -good (k', l, β') -system with $k' = \lfloor \frac{k}{2} \rfloor$ and $\beta' \leq \frac{\beta}{2} \left(1 + \frac{\log \beta l}{\sqrt{\beta l}}\right)$.*

However in our context, $\beta \leq 3$, thus we ignore Lemma 2.5 of [60] from further discussions.

Starting from the original system, as long as $l > c$, Lemma Reduce-1 is applied when $l > k$ and Lemma Reduce-k is applied when $k \geq l$. In this process β grows at most by a factor of 2. Thus at the end, l is a constant and so is β . Thus by applying Lemma 7.1.1, the constant integrality gap for the configuration LP is established.

Randomized Algorithm for γ -good (k, l, β) -system: There are two main steps in the algorithm.

1. Show a constructive procedure to obtain the reduced system through Lemma Reduce-l and Lemma Reduce-k.
2. Map the solution of the final reduced system back to the original system.

We now elaborate upon each of these.

Making Lemma Reduce-l Constructive

This follows quite directly from [111]. The algorithm picks $\lfloor \log^5 l \rfloor$ sets uniformly at random and independently from each group. Thus while the value of k remains fixed, l is reduced to $l' = \lfloor \log^5 l \rfloor$. Now in expectation the value of β does not change and the probability that $\beta' > \beta(1 + \frac{1}{\log l})$, and hence $\beta'l' > \beta l(1 + \frac{1}{\log l})$, is at most $e^{-\beta'l'/3 \log^2 l} \leq e^{-\log^3 l} = l^{-\log^2 l}$. We define a bad event corresponding to each element:

- A_j : Element j has more than $\beta'l'$ copies.

Now noting that the dependency graph has degree at most $kl\beta l \leq 6l^3$, the uniform (symmetric) version of the LLL applies. Now it is easy to check if there exists a violated event: we simply count the number of times an element appears in all the sets. Thus we directly follow [111]; setting $x_{A_j} = \frac{1}{e^{l \log^2 l}}$, we get the expected running time to avoid all the bad events to be $O(plk/l^{\log^2 l}) = O(p) = O(m)$.

Making Lemma Reduce-k Constructive

This is the main challenging part. The random experiment involves selecting each item independently at random with probability $\frac{1}{2}$. To characterize the bad events, we need a structural lemma from [60]. Construct a graph on the sets, where there is an edge between two sets if they share an element. A collection of sets is said to be connected if and only if the subgraph induced by this collection is connected.

We consider two types of bad events:

1. B_1 : some set has less than $k' = \left(1 - \frac{\log k}{\sqrt{k}}\right) \frac{k}{2}$ items surviving, and
2. B_i for $i \geq 2$: there is a connected collection of i sets from distinct groups whose union originally contained at most $i\gamma k$ items, of which more than $i\delta' \frac{k}{2}$ items survive, where $\delta' = \gamma \left(1 + \frac{\log k}{\sqrt{\gamma k}}\right)$.

If none of the above bad events happen, then we can consider the first k' items from each set and yet the second type of bad events do not happen. These events are chosen such that γ' -goodness ($\gamma' = \delta' \frac{k}{2} \frac{1}{k'} \leq \gamma \left(1 + \frac{3 \log k}{\sqrt{\gamma k}}\right)$) of the new system certifies that the original system was γ -good. That this is indeed the case follows directly from Hall's theorem as proven by Feige:

Lemma 7.1.5 (Lemma 2.7 of [60]). *Consider a collection of n sets and a positive integer q .*

1. *If for some $1 \leq i \leq n$, there is a connected subcollection of i sets whose union contains less than iq items, then there is no choice of q items per set*

such that all items are distinct.

2. *If for every i , $1 \leq i \leq n$, the union of every connected subcollection of i sets contains at least iq (distinct) items, then there is a choice of q items per set such that all items are distinct.*

Feige showed in [60] that for bad events of type $B_i, i \geq 1$, taking $x_i = 2^{-10i \log k}$ is sufficient to satisfy the condition (3.1) of the asymmetric LLL. More precisely, suppose we define, for any bad event $B \in \bigcup_{i \geq 1} B_i$, $\Gamma(B)$ to be as in Chapter 3: i.e., $\Gamma(B)$ is the set of all bad events $A \neq B$ such that A and B both depend on at least one common random variable in our “randomly and independently selecting items” experiment. Then, it is shown in [60] that with the choice $x_i = 2^{-10i \log k}$ for all events in B_i , we have

$$\forall (i \geq 1) \forall (B \in B_i), \Pr[B] \leq 2^{-20i \log k} \leq x_i \prod_{j \geq 1} \prod_{A \in (B_j \cap \Gamma(B))} (1 - x_j). \quad (7.2)$$

Thus by the LLL, there exists an assignment that avoids all the bad events. However, no efficient construction was known here, and as Feige points out, “the main source of difficulty in this respect is Lemma 2.4, because there the number of bad events is exponential in the problem size, and moreover, there are bad events that involve a constant fraction of the random variables.” Our Theorem 3.3.3 again directly makes this proof constructive and gives an efficient Monte Carlo algorithm for producing a reduce- k system with high probability.

Lemma 7.1.6. *There is a Monte Carlo algorithm that produces a valid reduce- k system with probability at least $1 - 1/m^2$.*

Proof. Note from (7.2) that we can take $\delta = 2^{-20m \log k}$. So, we get that $\log 1/\delta = O(m \log k) = O(n \log n)$ where n is the number of items and $m \leq n$ is the number of children. We furthermore get that all events with probability larger than a fixed inverse-polynomial involve only connected subsets of size $O(\frac{\log m}{\log k})$ and Theorem 3.3.3 implies that there are only polynomially many such “high” probability events. (This can also be seen directly since the degree of a subset is bounded by $k\beta l \leq 6k^2$ and the number of connected subcollections is therefore at most $(6k^2)^{O(\frac{\log m}{\log k})} = m^{O(1)} = n^{O(1)}$.) The connected collections of subsets are easy to enumerate using, e.g., breadth-first search and are therefore efficiently verifiable (in fact, even in parallel). Theorem 3.3.3 thus applies and directly proves the lemma. \square

Mapping the solution of the final reduced system back to the original system

By repeatedly applying algorithms to produce Reduce-1 or Reduce- k system, we can completely reduce down the original system to a system with a constant number of children per group, where β can increase from 3 to at most 6 due to Lemma Reduce-1. This involves at most $\log m$ Reduce-1 reductions and at most $\log n$ Reduce- k reductions. We can furthermore assume that $n < 2^m$ since otherwise simply all combinations of one child per group could be tried in time polynomial in n . Since, each Reduce-1 or Reduce- k operation produces a desired solu-

tion with probability at least $1 - \frac{1}{m^2}$, by union bound, with probability at least $1 - O(\log n \log m/m^2) = 1 - O(\log m/m)$ a final (k, l, β) -system is produced that is γ -good for some constant γ by Lemma 7.1.1. Using Lemma 7.1.1, we can also find a γ -good selection of children. Now, once one child from each group is selected, we can construct a standard network flow instance to assign items to these chosen children (Lemma 7.1.8). This finishes the process of mapping back a solution of the reduced system to the original (k, l, β) -system. While checking whether an individual reduction failed seems to be a NP-hard task, it is easy to see in the end whether a good enough assignment is produced. This enables us to rerun the algorithm in the unlikely event of a failure. Thus, the Monte Carlo algorithm can be strengthened to an algorithm that always produces a good solution and has an expected polynomial running-time.

The details of the above are given in two lemmas, Lemma 7.1.7 and Lemma 7.1.8. Theorem 7.1.9 follows from the two lemmas.

Suppose we start with a (k_1, l_1, β_1) -system and after repeated application of either Lemma Reduce-1 or Lemma Reduce-k reach at a (k_s, l_s, β_s) -system, where $l_s < c$, a constant. We then employ Lemma 7.1.1 to obtain a γ_s -good (k_s, l_s, β_s) -system, where γ_s satisfies $\gamma_s k_s = \lfloor \frac{k_s}{\beta_s l_s} \rfloor$. Since l_s is a constant and $\beta_s \leq 6$, γ_s is also a constant. Lemma 7.1.1 also gives a choice of a child from each group, denoted by a function $f : \{1, \dots, p\} \rightarrow \{1, \dots, l_s\}$ that serves as a witness for γ_s -goodness of (k_s, l_s, β_s) -system. We use this same mapping for the original system.

The following lemma establishes the goodness of the (k_1, l_1, β_1) -system.

Lemma 7.1.7. *Given a sequence of reductions of k , $(k_1, l_1, \beta_1) \rightarrow \dots \rightarrow (k_s, l_s, \beta_s)$, interleaved with reductions of l , let for all $s \geq 2$, $\gamma_s = \gamma_{s-1}(1 + \frac{3 \log k_{s-1}}{\sqrt{\gamma_{s-1} k_{s-1}}})$. Then if the final reduced system is γ_s -good and the function $f : \{1, \dots, p\} \rightarrow \{1, \dots, l_s\}$ serves as a witness for its γ_s -goodness, then f also serves as a witness of γ -goodness of (k_1, l_1, β_1) system with high probability. In other words, we can simply use the assignment given by f to select one child from each group and that assignment serves as a witness of γ -goodness of the original system with high probability.*

Proof. Suppose there exists a function f that serves as a witness for γ_s -goodness of the (k_s, l_s, β_s) -system, but does not serve as a witness that $(k_{s-1}, l_{s-1}, \beta_{s-1})$ -system is γ_{s-1} -good. Then there must exist a connected collection of $i, i > 0$ sets chosen from p groups according to f , such that their union contains less than $\gamma_{s-1} k_{s-1} i$ items. However in the reduced system, their union has $\gamma_s k_{s-1} i$ elements. Call such a function f bad. Thus every bad function is characterized by a violation of event of type $B_i, i \geq 1$, described in Section 7.1.2. However, by Lemma 7.1.6 we have $\Pr[\exists \text{ a bad function } f] \leq \Pr[\text{an event of type } B_i, i \geq 1 \text{ happens}] \leq \frac{1}{m^2}$.

Now the maximum number of times the Reduce- k step is applied is at most $\log k_1 \leq \log n$. Thus if the Reduce-1 step is not applied at all, then by a union bound, function f is γ -good for the (k_1, l_1, β_1) -system with probability at least

$1 - \frac{\log m \log n}{m^2}$. We can assume without loss of generality that $n \leq 2^m$. (Otherwise in polynomial time we can guess the children who receive small items and thus know f . Once f is known, an assignment of small items to the children chosen by f can be done in polynomial time through Lemma 7.1.8.) Since $n \leq 2^m$, function f is γ -good for the (k_1, l_1, β_1) -system with probability at least $1 - \log m/m$. Now since the Reduce-1 step only reduces l and keeps k intact, it does not affect the goodness of the set system. \square

Once we know the function f , using Lemma 7.1.8, we can get a valid assignment of $\lfloor k\gamma \rfloor$ items to each chosen child:

Lemma 7.1.8. *Given a function $f : \{1, \dots, p\} \rightarrow \{1, \dots, l\}$, and parameter γ , there is a polynomial time algorithm to determine, whether f is γ -good and we can determine the subset of $\lfloor k\gamma \rfloor$ items received by each child $f(i), i \in [1, p]$.*

Proof. We construct a bipartite graph with a set of vertices $U = \{1, \dots, p\}$ corresponding to each chosen child from the p groups, a set of vertices V corresponding to the small items in the sets of the chosen children, a source s and a sink t . Next we add a directed edge of capacity $\lfloor \gamma k \rfloor$ from source s to each vertex in U . We also add directed edges $(u, v), u \in U, v \in V$, if the item u belongs to the set of v . These edges have capacity 1. Finally we add a directed edge from each vertex in V to the sink t with capacity 1. We claim that this flow network has a maximum flow of $\lfloor k\gamma \rfloor p$ iff f is γ -good:

For the one direction let f be γ -good. Thus there exists a set of $\lfloor \gamma k \rfloor$ elements

that can be assigned to each child $u \in U$. Send one unit of flow from each child to these items that it receives. The outgoing flow from each $u \in U$ is exactly $\lfloor \gamma k \rfloor$. Since each item is assigned to at most one child, flow on each edge $(v, t), v \in V$ is at most 1. Thus all the capacity constraints are maintained and the flow value is $\lfloor \gamma k \rfloor p$.

For the other direction consider an integral maximum flow of $\lfloor k\gamma \rfloor p$. Since the total capacity of all the edges emanating from the source is $\lfloor k\gamma \rfloor p$, they must all be saturated by the maxflow. Since the flow is integral, for each child u there are exactly $\lfloor \gamma k \rfloor$ edges with flow 1 corresponding to the items that it receives. Also since no edge capacity is violated, each item is assigned to exactly one child. Therefore f is γ -good.

To check a function f for γ -goodness and obtain the good assignment we construct the flow graph and run a max flow algorithm that outputs in an integral flow. As proven above a max flow value of $\lfloor k\gamma \rfloor p$ indicates γ -goodness and for a γ -good function f , the assignment can be directly constructed from the flow by considering only the flow carrying edges. \square

Theorem 7.1.9. *There exists a constant $\alpha > 0$ and a randomized algorithm for the Santa Claus problem that runs in expected polynomial time and always assigns items of total valuation at least $\alpha \cdot \text{OPT}$ to each child.*

Overlay Network Design

In this chapter, we consider an application of resource allocation in networking. The goal here is to design multicast overlay networks that deliver streams to end-users in an effective and timely manner. The main technical tool is the LP rounding scheme developed in Chapter 2. This work improves the results of [10]. In this chapter, we refer to lemmas from [10] without proofs. The interested readers may find it helpful to look at the original paper [10] for full details.

8.1 Designing Overlay Multicast Networks For Streaming

An overlay network can be represented as a tripartite digraph $N = (V, E)$. The nodes V are partitioned into sets of entry points called sources (S), reflectors (R), and edge-servers or sinks (D). There are multiple commodities or streams, that must be routed from sources, via reflectors, to the sinks that are designated to serve that stream to end-users. Without loss of generality, we can assume that each source holds a single stream. Now given a set of streams and their respective edge-server

$$\begin{aligned}
\min \quad & \sum_{i \in R} r_i z_i + \sum_{i \in R} \sum_{k \in S} c_{k,i,k} y_{i,k} + \sum_{i \in R} \sum_{k \in S} \sum_{j \in D} c_{i,j,k} x_{i,j,k} \\
s.t. \quad & \\
& y_{k,i} \leq z_i \quad \forall i \in R, \quad \forall k \in S & (8.1) \\
& x_{i,j,k} \leq y_{i,k} \quad \forall i \in R, \quad \forall j \in D, \quad \forall k \in S & (8.2) \\
& \sum_{k \in S} \sum_{j \in D} x_{i,j,k} \leq F_i z_i \quad \forall i \in R & (8.3) \\
& \sum_{i \in R} x_{i,j,k} w_{i,j,k} \geq W_{j,k} \quad \forall j \in D, \quad \forall k \in S & (8.4) \\
& x_{i,j,k} \in \{0, 1\}, y_{i,k} \in \{0, 1\}, z_i \in \{0, 1\} & (8.5) \\
& & (8.6)
\end{aligned}$$

Table 8.1: Integer Program for Overlay Multicast Network Design

destinations, a cheapest possible overlay network must be constructed subject to certain *capacity*, *quality*, and *reliability* requirements. There is a cost associated with usage of every link and reflector. There are capacity constraints, especially on the reflectors, that dictate the maximum total bandwidth (in bits/sec) that the reflector is allowed to send. The quality of a stream is directly related to whether or not an edge-server is able to reconstruct the stream without significant loss of accuracy. Therefore even though there is some loss threshold associated with each stream, at each edge-server only a maximum possible reconstruction-loss is allowed. To ensure reliability, multiple copies of each stream may be sent to the designated edge-servers.

All these requirements can be captured by an integer program. Let us use indicator variable z_i for building reflector i , $y_{i,k}$ for delivery of k -th stream to the i -th reflector and $x_{i,j,k}$ for delivering k -th stream to the j -th sink through the i -th reflec-

tor. F_i denotes the fanout constraint for each reflector $i \in R$. Let $p_{x,y}$ denote the failure probability on any edge (source-reflector or reflector-sink). We transform the probabilities into weights: $w_{i,j,k} = -\log(p_{k,i} + p_{i,j} - p_{k,i}p_{i,j})$. Therefore, $w_{i,j,k}$ is the negative log of the probability of a commodity k failing to reach sink j via reflector i . On the other hand, if $\phi_{j,k}$ is the minimum required success probability for commodity k to reach sink j , we instead use $W_{j,k} = -\log(1 - \phi_{j,k})$. Thus $W_{j,k}$ denotes the negative log of maximum allowed failure. r_i is the cost for opening the reflector i and $c_{x,y,k}$ is the cost for using the link (x, y) to send commodity k . Thus we have the IP (see Table 8.1).

Constraints (8.2) and (8.3) are natural consistency requirements; constraint (8.4) encodes the fanout restriction. Constraint (8.5), the *weight* constraint, ensures quality and reliability. Constraint (8.6) is the standard integrality-constraint that will be relaxed to construct the LP relaxation.

There is an important stability requirement that is referred as *color constraint* in [10]. Reflectors are grouped into m color classes, $R = R_1 \cup R_2 \cup \dots \cup R_m$. We want each group of reflectors to deliver not more than one copy of a stream into a sink. This constraint translates to

$$\sum_{i \in R_l} x_{i,j,k} \leq 1 \quad \forall j \in D, \forall k \in S, \forall l \in [m] \quad (8.7)$$

Each group of reflectors can be thought to belong to the same ISP. Thus we want to make sure that a client is served only with one – the best – stream possible

from a certain ISP. This diversifies the stream distribution over different ISPs and provides stability. If an ISP goes down, still most of the sinks will be served. We refer the LP-relaxation of integer program (Table 8.1) with the color constraint (8.7) as **LP-Color**.

All of the above is from [10].

The work of [10] uses a two-step rounding procedure and obtains the following guarantee.

First stage rounding: Rounds z_i and $y_{i,k}$ for all i and k to decide which reflector should be open and which streams should be sent to a reflector. The results from rounding stage 1 can be summarized in the following lemma:

Lemma 8.1.1. (*[10]*) *The first-stage rounding algorithm incurs a cost at most a factor of $64 \log |D|$ higher than the optimum cost, and with high probability violates the weight constraints by at most a factor of $\frac{1}{4}$ and the fanout constraints by at most a factor of 2. Color constraints are all satisfied.*

By incurring a factor of $\Theta(\log n)$ in the cost, the constant factors loss in the weights and fanouts can be improved as shown in [10].

Second stage rounding: Rounds $x_{i,j,k}$'s using the open reflectors and streams that are sent to different reflectors in the first stage. The results in this stage can be summarized as follows:

Lemma 8.1.2. (*[10]*) *The second-stage rounding incurs a cost at most a factor of 14 higher than the optimum cost and violates each of fanout, color and weight*

constraint by at most a factor of 7.

Our main contribution is an improvement of the second-stage rounding through the use of repeated **RandMove** and by judicious choices of constraints to drop. Let us call the linear program that remains just at the end of first stage **LP-Color2**:

$$\begin{aligned}
& \min \sum_{i \in R} \sum_{k \in S} \sum_{j \in D} c_{i,j,k} x_{i,j,k} \\
& \text{s.t.} \\
& \sum_{k \in S} \sum_{j \in D} x_{i,j,k} \leq F_i \quad \forall i \in R \text{ (Fanout)} \\
& \sum_{i \in R} x_{i,j,k} w_{i,j,k} \geq W_{j,k} \quad \forall j \in D, \forall k \in S \text{ (Weight)} \\
& \sum_{i \in R_l} x_{i,j,k} \leq 1 \quad \forall j \in D, \forall k \in S, \forall l \in [m] \text{ (Color)} \\
& x_{i,j,k} \in \{0, 1\} \quad \forall i \in R, \forall j \in D, \forall k \in S
\end{aligned}$$

We show:

Lemma 8.1.3. *LP-Color2 can be efficiently rounded such that cost and weight constraints are satisfied exactly, fanout constraints are violated at most by additive 1 and color constraints are violated at most by additive 3.*

Proof. Let $x_{i,j,k}^* \in [0, 1]$ denote the fraction of stream generated from source $k \in S$ reaching destination $j \in D$ routed through reflector $i \in R$ after the first stage of

rounding. Initialize $X = x^*$. The algorithm consists of several iterations. the random value at the end of iteration h is denoted by X^h . Each iteration h conducts a randomized update using **RandMove** on the polytope of a linear system constructed from a subset of constraints of **LP-Color2**. Therefore by induction on h , we will have for all (i, j, h) that $E[X_{i,j}^h] = x_{i,j}^*$. Thus the cost constraint is maintained exactly on expectation. The entire procedure can be derandomized giving the required bounds on the cost.

Let R and SD denote the set of reflectors and source, destination pairs respectively. Suppose we are at the beginning of some iteration $(h + 1)$ of the overall algorithm and currently looking at the values $X_{i,j,k}^h$. We will maintain two invariants:

(I1'') Once a variable $x_{i,j,k}$ gets assigned to 0 or 1, it is never changed;

(I2'') Once a constraint is dropped in some iteration, it is never reinstated.

Iteration $(h + 1)$ of rounding consists of three main steps:

1. Since we aim to maintain **(I1)**, let us remove all $X_{i,j,k}^h \in \{0, 1\}$; i.e., we project X^h to those coordinates (i, j, k) for which $X_{i,j,k}^h \in (0, 1)$, to obtain the current vector Y of floating (yet to be rounded) variables; let $\mathcal{S} \equiv (A_h Y = u_h)$ denote the current linear system that represents **LP-Color2**. In particular, the fanout constraint for a reflector in \mathcal{S} is its residual fanout F'_i ; i.e., F_i minus the number of streams that are routed through it.

2. Let v denote the number of floating variables, i.e., $Y \in (0, 1)^v$. We now drop the following constraint:

(D1'') Drop fanout constraint for degree 1 reflector denoted R_1 , i.e, reflectors with only one floating variable associated with it. For any degree 2 reflectors denoted R_2 , if it has a tight fanout of 1 drop its fanout constraint.

(D2'') Drop color constraint for a group of reflectors R_l , if they have atmost 4 floating variable associated with them.

Let \mathcal{P} denote the polytope defined by this reduced system of constraints. A key claim is that Y is not a vertex of \mathcal{P} and thus we can apply **Rand-Move** and make progress either by rounding a new variable or by dropping a new constraint. We count the number of variables v and the number of tight constraints t separately.

We have,

$$t = \sum_{i \in R \setminus R_1} 1 + \sum_{k \in S} \sum_{j \in D} (l_{k,j} + 1),$$

where $l_{j,k}$ is the number of color constraints for the stream generated at source k and to be delivered to the destination j . We have, $v \geq \sum_{i \in R} F_i + 1$. Also the number of variables $v \geq \sum_{k \in S, j \in D, l_{k,j} > 0} 4l_{k,j} + \sum_{k \in S, j \in D, l_{k,j} = 0} 2$. Thus by an averaging argument, the number of variables

$$v \geq \frac{\sum_{i \in R} F_i + 1}{2} + \sum_{k \in S, j \in D, l_{k,j} > 0} 2l_{k,j} + \sum_{k \in S, j \in D, l_{k,j} = 0} 1.$$

A moment's reflection shows that the system can become underdetermined only if there is no color constraint associated with a stream (j, k) , each reflector i has two floating variables associated with it with total contribution 1 towards fanout and each stream (j, k) is routed fractionally through two reflectors. But in this situation all the fanout constraints are dropped violating fanout at most by an additive one and making the system underdetermined once again. Coloring constraints are dropped only when there are less than 4 floating variable associated with that group of reflectors. Hence the coloring constraint can be violated at most by an additive factor of 3. The fanout constraint is dropped only for singleton reflectors or degree-2 reflectors with fanout equalling 1. Hence fanout is violated only by an additive 1. Weight constraint is never dropped and maintained exactly. □

Matroid Median

In this chapter, we consider another application of resource allocation in the domain of networking. In content distribution networks, there are servers of different types, and resource limitation restricts the number of servers of each type that can possibly be opened to serve requests from clients. The cost to connect a client to a server depends on the server location; the strategic question here is to decide which servers to open such that this connection cost is minimized. This problem can be modeled by a generalization of classical k -median problem, which we refer as *Matroid Median*. In this chapter, we study an approximation algorithm for the Matroid Median problem. The algorithm is based on LP rounding and crucially uses the structure of the constraint matrix. We simplify the structure of the fractional solution by doing a partial rounding and then argue that the constraint matrix of the simplified LP is totally unimodular. While, we have extensively used the property of extreme point solution in developing our rounding technique in Chapter 2, this is the first time, we exploit more structural information about the considered LP

relaxation.

9.1 The Matroid Median Problem

The k -median problem is an extensively studied location problem. Given an n -vertex metric space (V, d) and a bound k , the goal is to locate/open k centers $C \subseteq V$ so as to minimize the sum of distances of each vertex to its nearest open center. (The distance of a vertex to its closest open center is called its connection cost.) The first constant-factor approximation algorithm for k -median on general metrics was by Charikar et al. [39]. The approximation ratio was later improved in a sequence of papers [38, 84, 85] to the currently best-known guarantee of $3 + \varepsilon$ (for any constant $\varepsilon > 0$) due to Arya et al. [14]. A number of techniques have been successfully applied to this problem, such as LP rounding, primal-dual and local search algorithms.

Motivated by applications in Content Distribution Networks, Hajiaghayi et al. [76] introduced a generalization of k -median where there are *two types* of vertices (red and blue), and the goal is to locate at most k_r red centers and k_b blue centers so as to minimize the sum of connection costs. For this *red-blue median* problem, [76] gave a constant factor approximation algorithm. Here, we consider a substantially more general setting where there are an arbitrary number T of vertex-types with bounds $\{k_i\}_{i=1}^T$, and the goal is to locate at most k_i centers of each type- i so as to minimize the sum of connection costs. These vertex-types denote

different types of servers in the Content Distribution Networks applications; the result in [76] only holds for $T = 2$.

In fact, we study an even more general problem where the set of open centers have to form an independent set in a given matroid, with the objective of minimizing sum of connection costs. This formulation captures several intricate constraints on the open centers, and contains as special cases: the classic k -median (uniform matroid of rank k), and the CDN applications above (partition matroid with T parts). Our main result is a constant-factor approximation algorithm for this *Matroid Median* problem.

Our Results and Techniques In this chapter we introduce the Matroid Median problem, which is a natural generalization of k -median, and obtain a 16-approximation algorithm for it. Thus it also gives the first constant approximation for the k -median problem with multiple (more than two) vertex-types, which was introduced in [76].

For the standard k -median problem (and also red-blue median), it is easy to obtain an $O(\log n)$ -approximation algorithm using probabilistic tree-embeddings [58], and exactly solving the problem on a tree (via a dynamic program). However, even this type of guarantee is not obvious for the Matroid Median problem, since the problem on a tree-metric does not look particularly easier.

Our algorithm is based on the natural LP-relaxation and is surprisingly simple. Essentially, the main insight is in establishing a connection to matroid intersection.

The algorithm computes an optimal LP solution and rounds it in two phases, the key points of which are described below:

- The first phase sparsifies the LP solution while increasing the objective value by a constant factor. This is somewhat similar to the LP-rounding algorithm for k -median in Charikar et al. [39]. However we cannot consolidate fractionally open centers as in [39]; this is because the open centers must additionally satisfy the matroid rank constraints. In spite of this, we show that the vertices and the centers can be clustered into disjoint ‘star-like’ structures.
- This structure ensured by the first phase of rounding allows us to write (in the second phase) another linear program for which the sparsified LP solution is feasible, and has objective value at most $O(1)$ times the original LP optimum. Then we show that the second phase LP is in fact integral, via a relation to the matroid-intersection polytope. Finally we re-solve the second phase LP to obtain an extreme point solution, which is guaranteed to be integral. This corresponds to a feasible solution to Matroid Median of objective value $O(1)$ times the LP optimum.

We next consider the *Penalty Matroid Median* (a.k.a. prize-collecting matroid median problem), where a vertex could either connect to a center incurring the connection cost, or choose to pay a penalty in the objective function. The prize-collecting version of several well-known optimization problems like TSP, Steiner Tree etc., including k -median and red-blue median have been studied in prior work

(See [76] and the references therein). Extending the idea of the Matroid Median algorithm, we also obtain an $O(1)$ approximation algorithm for the Penalty version of the problem.

Finally, we look at the *Knapsack Median* problem (a.k.a. weighted W -median [76]), where the centers have weights and the open centers must satisfy a knapsack constraint; the objective is, like before, to minimize the total connection cost of all the vertices. For this problem we obtain a 16-approximation algorithm that violates the knapsack constraint by an *additive* f_{max} term (where f_{max} is the maximum opening cost of any center). This algorithm is again based on the natural LP relaxation, and follows the same approach as for Matroid Median. However, the second phase LP here is not integral (it contains the knapsack problem as a special case). Instead we obtain the claimed bicriteria approximation by using the iterative rounding framework [62, 83, 94, 123]. It is easy to see that our LP-relaxation for the Knapsack Median problem has unbounded integrality gap, if we do not allow any violation in the knapsack constraint (see eg. [38]). Moreover, we show that the integrality gap remains unbounded even after the addition of *knapsack-cover inequalities* [33] to the basic LP relaxation. We leave open the question of obtaining an $O(1)$ -approximation for Knapsack Median without violating the knapsack constraint.

Related Work

The first approximation algorithm for the metric k -median problem was due to Lin and Vitter [98] who gave an algorithm that for any $\varepsilon > 0$, produced a solution of objective at most $2(1 + \frac{1}{\varepsilon})$ while opening at most $(1 + \varepsilon)k$ centers; this was based on the filtering technique for rounding the natural LP relaxation. The first approximation algorithm that opened only k centers was due to Bartal [24], via randomized tree embedding (mentioned earlier). Charikar et al. [39] obtained the first $O(1)$ -approximation algorithm for k -median, by rounding the LP relaxation; they obtained an approximation ratio of $6\frac{2}{3}$. The approximation ratio was improved to 6 by Jain and Vazirani [85], using the primal dual technique. Charikar and Guha [38] further improved the primal-dual approach to obtain a 4-approximation. Later Arya et al. [14] analyzed a natural local search algorithm that exchanges up to p centers in each local move, and proved a $3 + \frac{2}{p}$ approximation ratio (for any constant $p \geq 1$). Recently, Gupta and Tangwongsan [74] gave a considerably simplified proof of the Arya et al. [14] result. It is known that the k -median problem on general metrics is hard to approximate to a factor better than $1 + \frac{2}{e}$. On Euclidean metrics, the k -median problem has been shown to admit a PTAS by Arora et al. [13].

Very recently, Hajiaghayi et al. [76] introduced the red-blue median problem — where the vertices are divided into two categories and there are different bounds on the number of open centers of each type — and obtained a constant factor approximation algorithm. Their algorithm uses a local search using single-swaps for each

vertex type. The motivation in [76] came from locating servers in Content Distribution Networks, where there are T server-types and strict bounds on the number of servers of each type. The red-blue median problem captured the case $T = 2$. It is unclear whether their approach can be extended to multiple server types, since the local search with single swap for each server-type has neighborhood size $n^{\Omega(T)}$. Furthermore, even a $(T - 1)$ -exchange local search has large locality-gap— see Section 9.5. Hence it is not clear how to apply local search to MatroidMedian, even in the case of a partition matroid. [76] also discusses the difficulty in applying the Lagrangian relaxation approach (see [85]) to the red-blue median problem; this is further compounded in the MatroidMedian problem since there are exponentially many constraints on the centers.

The most relevant paper to ours with regard to the rounding technique is Charikar et al. [39]: our algorithm builds on many ideas used in their work to obtain our approximation algorithm.

For the penalty k -median problem, the best known bound is a 3-approximation due to Hajiaghayi et al. [76] that improves upon a previous 4-approximation due to Charikar and Guha [38]. Hajiaghayi et al. also consider the penalty version of the red-blue median problem and give a constant factor approximation algorithm.

The knapsack median problem admits a bicriteria approximation ratio via the filtering technique [98]. The currently best known tradeoff [38] implies for any $\varepsilon > 0$, a $(1 + \frac{2}{\varepsilon})$ -approximation in the connection costs while violating the knapsack

constraint by a *multiplicative* $(1 + \varepsilon)$ factor. Charikar and Guha [38] also shows that for each $\varepsilon > 0$, it is not possible to obtain a trade-off better than $(1 + \frac{1}{\varepsilon}, 1 + \varepsilon)$ relative to the natural LP. On the other hand, our result implies a $(16, 1 + \varepsilon)$ -tradeoff in $n^{O(1/\varepsilon)}$ time for each $\varepsilon > 0$; this algorithm uses enumeration combined with the natural LP-relaxation. As mentioned in [76], an $O(\log n)$ -approximation is achievable for knapsack median (without violation of the knapsack constraint) via a reduction to tree-metrics, since the problem on trees admits a PTAS.

Preliminaries

The input to the MatroidMedian problem consists of a finite set of vertices V and a distance function $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$ which is symmetric and satisfies the triangle inequality, i.e. $d(u, v) + d(v, w) \geq d(u, w)$ for all $u, v, w \in V$. Such a tuple (V, d) is called a finite metric space. We are also given a matroid \mathcal{M} , with ground set V and set of independent sets $\mathcal{J}(\mathcal{M}) \subseteq 2^V$. The goal is to open an independent set $S \in \mathcal{J}(\mathcal{M})$ of centers such that the sum $\sum_{u \in V} d(u, S)$ is minimized; here $d(u, S) = \min_{v \in S} d(u, v)$ is the connection cost of vertex u . We assume some familiarity with matroids, for more details see eg. [121].

In the penalty version, additionally a penalty function $p : V \rightarrow \mathbb{R}_{\geq 0}$ is provided and the objective is now modified to minimize $\sum_{u \in V} d(u, S)(1 - h(u)) + p(u)h(u)$. Here $h : V \rightarrow \{0, 1\}$ is an indicator function that is 1 if the corresponding vertex is not assigned to a center and therefore pays a penalty and 0 otherwise.

The KnapsackMedian problem (aka weighted W -median [76]) is similarly de-

fined. We are given a finite metric space (V, d) , non-negative weights $\{f_i\}_{i \in V}$ (representing facility costs) and a bound F . The goal is to open centers $S \subseteq V$ such that $\sum_{j \in S} f_j \leq F$ and the objective $\sum_{u \in V} d(u, S)$ is minimized (Section 9.4).

9.1.1 An LP Relaxation for MatroidMedian

In the following linear program, y_v is the indicator variable for whether vertex $v \in V$ is opened as a center, and x_{uv} is the indicator variable for whether vertex u is served by center v . Then, the following LP is a valid relaxation for the MatroidMedian problem.

$$\text{minimize } \sum_{u \in V} \sum_{v \in V} d(u, v) x_{uv} \quad (\text{LP}_1)$$

$$\text{subject to } \sum_{v \in V} x_{uv} = 1 \quad \forall u \in V \quad (9.1.1)$$

$$x_{uv} \leq y_v \quad \forall u \in V, v \in V \quad (9.1.2)$$

$$\sum_{v \in S} y_v \leq r_{\mathcal{M}}(S) \quad \forall S \subseteq V \quad (9.1.3)$$

$$x_{uv}, y_v \geq 0 \quad \forall u, v \in V \quad (9.1.4)$$

If x_{uv} and y_v are restricted to only take values 0 or 1, then this is easily seen to be an exact formulation for MatroidMedian. The first constraint models the requirement that each vertex u must be connected to some center v , and the second one requires that it can do so only if the center v is opened, i.e. $x_{uv} = 1$ only if y_v is

also set to 1. The constraints (9.1.3) are the matroid rank-constraint on the centers: they model the fact that the open centers form an independent set with respect to the matroid \mathcal{M} . Here $r_{\mathcal{M}} : 2^V \rightarrow \mathbb{Z}_{\geq 0}$ is the *rank-function* of the matroid, which is monotone and submodular. The objective function exactly measures the sum of the connection costs of each vertex. (It is clear that given integrally open centers $y \in \{0, 1\}^V$, each vertex $u \in V$ sets $x_{uv} = 1$ for its closest center v with $y_v = 1$.) Let Opt denote an optimal solution of the given MatroidMedian instance, and let LPOpt denote the LP optimum value. From the above discussion, we have that,

Lemma 9.1.1. *The LP cost LPOpt is at most the cost of an optimal solution Opt .*

9.1.2 Solving the LP: The Separation Oracle

Even though the LP relaxation has an exponential number of constraints, it can be solved in polynomial time (using the Ellipsoid method) assuming we can, in polynomial time, verify if a candidate solution (x, y) satisfies all the constraints. Indeed, consider any fractional solution (x, y) . Constraints (9.1.1), and (9.1.2) can easily be verified in $O(n^2)$ time, one by one.

Constraint (9.1.3) corresponds to checking if the fractional solution $\{y_v : v \in V\}$ lies in the matroid polytope for \mathcal{M} . Checking (9.1.3) is equivalent to seeing whether:

$$\min_{S \subseteq V} \left(r_{\mathcal{M}}(S) - \sum_{v \in S} y_v \right) \geq 0.$$

Since the rank-function $r_{\mathcal{M}}$ is submodular, so is the function $f(S) := r_{\mathcal{M}}(S) -$

$\sum_{v \in S} y_v$. So the above condition (and hence (9.1.3)) can be checked using sub-modular function minimization, eg. [82,121]. There are also more efficient methods for separating over the matroid polytope – refer to [121] for more details on efficiently testing membership in matroid polyhedra. Thus we can obtain an optimal LP solution in polynomial time.

9.2 The Rounding Algorithm for MatroidMedian

Let (x^*, y^*) denote the optimal LP solution. Our rounding algorithm consists of two stages. In the first stage, we only alter the x_{uv}^* variables such that the modified solution, while still being feasible to the LP, is also very sparse in its structure. In the second stage, we write another LP which exploits the sparse structure, for which the modified fractional solution is feasible, and the objective function has not increased by more than a constant factor. We then proceed to show that the new LP in fact corresponds to an integral polytope. Thus we can obtain an integral solution where the open centers form an independent set of \mathcal{M} , and the cost is $O(1)\text{LPOpt}$.

Stage I: Sparsifying the LP Solution

In the first stage, we follow the outline of the algorithm of Charikar et al. [39], but we can not directly employ their procedure because we can't alter/consolidate the y_v^* variables in an arbitrary fashion (since they need to satisfy the matroid polytope constraints). Specifically, step (i) below is identical to the first step (consolidating

locations) in [39]. The subsequent steps in [39] do not apply since they consolidate centers; however using some ideas from [39] and with some additional work, we obtain the desired sparsification in steps (ii)-(iii) *without altering the y^* -variables*.

Step (i): Consolidating Clients. We begin with some notation, which will be useful throughout the paper. For each vertex u , let $LP_u = \sum_{v \in V} d(u, v)x_{uv}^*$ denote the contribution to the objective function $LPOpt$ of vertex u . Also, let $\mathcal{B}(u, r) = \{v \in V \mid d(u, v) \leq r\}$ denote the ball of radius r centered at vertex u . For any vertex u , we say that $\mathcal{B}(u, 2LP_u)$ is the *local ball* centered at u .

Initialize $w_u \leftarrow 1$ for all vertices. Order the vertices according to non-decreasing LP_u values, and let the ordering be u_1, u_2, \dots, u_n . Now consider the vertices in the order u_1, u_2, \dots, u_n . For vertex u_i , if there exists another vertex u_j with $j < i$ such that $d(u_i, u_j) \leq 4LP_{u_i}$, then set $w_{u_j} \leftarrow w_{u_j} + 1$, and $w_{u_i} \leftarrow 0$. Essentially we can think of moving u_i to u_j for the rest of the algorithm (which is why we are increasing the weight of u_j and setting the weight of u_i to be zero).

After the above process, let V' denote the set of locations with positive weight, i.e. $V' = \{v \mid w_v > 0\}$. For the rest of the paper, we will refer to vertices in V' as *clients*. By the way we defined this set, it is clear that the following two observations holds.

Observation 9.2.1. For $u, v \in V'$, we have $d(u, v) > 4 \max(LP_u, LP_v)$.

This is true because, otherwise, if (without loss of generality) $LP_v \geq LP_u$ and $d(u, v) \leq 4LP_v$, then we would have moved v to u when we were considering v .

Observation 9.2.2.

$$\sum_{u \in V'} w_u \sum_{v \in V} d(u, v) x_{uv}^* \leq \sum_{u \in V} \sum_{v \in V} d(u, v) x_{uv}^*$$

This is because, when we move vertex u_i to u_j , we replace the term corresponding to LP_{u_i} (in the LHS above) with an additional copy of that corresponding to LP_{u_j} , and we know by the vertex ordering that $\text{LP}_{u_i} \geq \text{LP}_{u_j}$.

Also, the following lemma is a direct consequence of Markov's inequality.

Lemma 9.2.3. *For any client $u \in V'$, $\sum_{v \in \mathcal{B}(u, 2\text{LP}_u)} x_{uv} \geq 1/2$. In words, each client is fractionally connected to centers in its local ball to at least an extent of $1/2$.*

Finally, we observe that if we obtain a solution to the new (weighted) instance and incur a cost of C , the cost of the same set of centers with respect to the original instance is then at most $C + 4\text{LP}_{\text{Opt}}$ (the additional distance being incurred in moving back each vertex to its original location).

We now assume that we have the weighted instance (with clients V'), and are interested in finding a set $S \subseteq V$ of centers to minimize $\sum_{u \in V'} w_u d(u, S)$. Note that centers may be chosen from the entire vertex-set V , and are not restricted to V' . Consider an LP-solution (x^1, y^*) to this weighted instance, where $x_{uv}^1 = x_{uv}^*$ for all $u \in V'$, $v \in V$. Note that (x^1, y^*) satisfies constraints (9.1.1)-(9.1.2) with u ranging over V' , and also constraint (9.1.3); so it is indeed a feasible fractional

solution to the weighted instance. Also, by Observation 9.2.2, the objective value of (x^1, y^*) is $\sum_{u \in V'} w_u \sum_{v \in V} d(u, v) x_{uv}^1 \leq \text{LPOpt}$, i.e. at most the original LP optimum.

After this step, even though we have made sure that the clients are well-separated, a client $u \in V'$ may be fractionally dependent on several partially open centers, as governed by the x_{uv} variables. More specifically, it may be served by centers which are contained in the ball $\mathcal{B}(u, 2\text{LP}_u)$, or by centers which are contained in another ball $\mathcal{B}(u', 2\text{LP}_{u'})$, or some centers which do not lie in any of the balls around the clients. The subsequent steps further simplify the structure of these connections.

Remark: To illustrate the high-level intuition behind our algorithm, suppose it is the case that for all $u \in V'$, client u is completely served by centers inside $\mathcal{B}(u, 2\text{LP}_u)$. Then, we can infer that it is sufficient to open a center inside each of these balls, while respecting the matroid polytope constraints. Since we are guaranteed that for $u, v \in V'$, $\mathcal{B}(u, 2\text{LP}_u) \cap \mathcal{B}(v, 2\text{LP}_v) = \emptyset$ (from Observation 9.2.1), this problem reduces to that of finding an independent set in the intersection of *matroid* \mathcal{M} and the *partition matroid* defined by the balls $\{\mathcal{B}(u, 2\text{LP}_u) \mid u \in V'\}$! Furthermore, the fractional solution (x^*, y^*) is feasible for the natural LP-relaxation of the matroid intersection problem. Now, because the matroid intersection polytope is integral, we can obtain an integer solution of low cost (relative to LPOpt).

However, the vertices may not in general be fully served by centers inside their

corresponding local balls, as mentioned earlier. Nevertheless, we establish some additional structure (in the next three steps) which enables us to reduce to a problem (in Stage II) of intersecting matroid \mathcal{M} with some *laminar constraints* (instead of just partition constraints as in the above example).

Step (ii): Making the objective function uniform & centers private. We now simplify connections that any vertex participates outside its local ball. We start with the LP-solution (x^1, y^*) and modify it to another solution (x^2, y^*) . Initially set $x^2 \leftarrow x^1$.

(A). For any client u that depends on a center v which is contained in another client u' 's local ball, we change the coefficient of x_{uv} in the objective function from $d(u, v)$ to $d(u, u')$. Because the clients are well-separated, this changes the total cost only by a small factor. Formally,

$$\begin{aligned} d(u, v) &\geq d(u, u') - 2\text{LP}_{u'} && \text{Since } v \in \mathcal{B}(u', 2\text{LP}_{u'}) \\ &\geq d(u, u') - d(u, u')/2 && \text{From Obs 9.2.1} \\ &\geq (1/2)d(u, u') \end{aligned}$$

Thus we can write:

$$\begin{aligned} &\sum_{u \in V'} w_u \left[\sum_{u' \in V' \setminus u} d(u, u') \sum_{v \in \mathcal{B}(u', 2\text{LP}_{u'})} x_{uv}^2 \right] && (9.2.5) \\ &\leq 2 \sum_{u \in V'} w_u \sum_{u' \in V' \setminus u} \sum_{v \in \mathcal{B}(u', 2\text{LP}_{u'})} d(u, v) x_{uv}^1 \end{aligned}$$

(B). We now simplify centers that are not contained in any local ball, and ensure that each such center has only one client dependent on it. Consider any vertex $v \in V$ which does not lie in any local ball, and has at least two clients dependent on it. Let these clients be u_0, u_1, \dots, u_k ordered such that $d(u_0, v) \leq d(u_1, v) \leq \dots \leq d(u_k, v)$. The following claim will be useful for re-assignment.

Claim 9.2.4. For all $i \in \{1, \dots, k\}$, $d(u_i, u_0) \leq 2d(u_i, v)$. Furthermore, for any vertex $v' \in \mathcal{B}(u_0, 2LP_{u_0})$, $d(u_i, v') \leq 3d(u_i, v)$.

Proof. From the way we have ordered the clients, we know that $d(u_i, v) \geq d(u_0, v)$; so $d(u_i, u_0) \leq d(u_i, v) + d(u_0, v) \leq 2d(u_i, v)$ for all $i \in \{1, \dots, k\}$. Also, if v' is some center in $\mathcal{B}(u_0, 2LP_{u_0})$, then we have $d(u_i, v') \leq d(u_i, u_0) + 2LP_{u_0} \leq (3/2)d(u_i, u_0)$, where in the final inequality we have used Observation 9.2.1. Therefore, we have $d(u_i, v') \leq 3d(u_i, v)$ for any $v' \in \mathcal{B}(u_0, 2LP_{u_0})$, which proves the claim. ■

Now, for each $1 \leq i \leq k$, we remove the connection (u_i, v) (ie. $x_{u_i v}^2 \leftarrow 0$) and arbitrarily increase connections (for a total extent $x_{u_i v}^1$) to edges (u_i, v') for $v' \in \mathcal{B}(u_0, 2LP_{u_0})$ while maintaining feasibility (ie. $x_{u_i v'}^2 \leq y_{v'}^*$). But we are ensured that a feasible re-assignment exists because for every client u_i , the extent to which it is connected outside its ball is at most $1/2$, and we are guaranteed that the total extent to which centers are opened in $\mathcal{B}(u_0, 2LP_{u_0})$ is at least $1/2$ (Lemma 9.2.3). Therefore, we can completely remove any connection u_i might have to v and re-assign it to centers in $\mathcal{B}(u_0, 2LP_{u_0})$ and for each of these reassign-

ments, we use $d(u_i, u_0)$ as the distance coefficient. From Claim 9.2.4 and observing that the approximation on cost is performed on *disjoint* set of edges in **(A)** and **(B)**, we obtain that:

$$\sum_{u \in V'} w_u \sum_{v \in V} d(u, v) x_{uv}^2 \leq 2 \cdot \sum_{u \in V'} w_u \sum_{v \in V} d(u, v) x_{uv}^1. \quad (9.2.6)$$

After this step, we have that for each center v not contained in any ball around the clients, there is only one client, say u , which depends on it. In this case, we say that v is a *private center* to client u . Let $\mathcal{P}(u)$ denote the set of all vertices that are either contained in $\mathcal{B}(u, 2LP_u)$, or are private to client u . Notice that $\mathcal{P}(u) \cap \mathcal{P}(u') = \emptyset$ for any two clients $u, u' \in V'$. Also denote $\mathcal{P}^c(u) := V \setminus \mathcal{P}(u)$ for any $u \in V'$.

We further change the LP-solution from (x^2, y^*) to (x^3, y^*) as follows. In x^3 we ensure that any client which depends on centers in other clients' local balls, will in fact depend *only* on centers in the local ball of its nearest other client. For any client u , we reassign all connections (in x^2) to $\mathcal{P}^c(u)$ to centers of $\mathcal{B}(u', 2LP_{u'})$ (in x^3) where u' is the closest other client to u . This is possible because the total reassignment for each client is at most half and every local-ball has at least half unit of centers. Clearly the value of (x^3, y^*) under the new objective is at most that of (x^2, y^*) , by the way we have altered the objective function.

Now, for each $u \in V'$, if we let $\eta(u) \in V' \setminus \{u\}$ denote the closest other client to u , then u depends only on centers in $\mathcal{P}(u)$ and $\mathcal{B}(\eta(u), 2LP_{\eta(u)})$. Thus, the new

objective value of (x^3, y^*) is exactly:

$$\sum_{u \in V'} w_u \left(\sum_{v \in \mathcal{P}(u)} d(u, v) x_{uv}^3 + d(u, \eta(u)) \left(1 - \sum_{v \in \mathcal{P}(u)} x_{uv}^3 \right) \right) \leq 2 \cdot \text{LPOpt} \quad (9.2.7)$$

Observe that we retained for each $u \in V'$ only the x_{uv} -variables with $v \in \mathcal{P}(u)$; this suffices because all other x_{uw} -variables (with $w \in \mathcal{P}^c(u)$) pay the same coefficient $d(u, \eta(u))$ in the objective (due to the changes made in **(A)** and **(B)**). Since the cost of the altered solution is at most that of (x^2, y^*) , we get the same bound of 2LPOpt .

Furthermore, for any client u which depends on a private center $v \in \mathcal{P}(u) \setminus \mathcal{B}(u, 2\text{LP}_u)$, it must be that $d(u, v) \leq d(u, \eta(u))$; otherwise, we can re-assign this (uv) connection to a center $v' \in \mathcal{B}(\eta(u), 2\text{LP}_{\eta(u)})$ and improve in the (altered) objective function; again we use the fact that u might depend on $\mathcal{P}(u) \setminus \mathcal{B}(u, 2\text{LP}_u)$ to total extent at most half and $\mathcal{B}(\eta(u), 2\text{LP}_{\eta(u)})$ has at least half unit of open centers.

To summarize, the above modifications ensure that fractional solution (x^3, y^*) satisfies the following:

- (i) For any two clients $u, u' \in V'$, we have $d(u, u') > 4 \max(\text{LP}_u, \text{LP}_{u'})$. In words, this means that all clients are well-separated.
- (ii) For each center v that does not belong to any ball $\mathcal{B}(u, 2\text{LP}_u)$, we have only

one client that depends on it.

- (iii) Each client u depends only on centers in its ball, its private centers, and centers in the ball of its nearest client. The extent to which it depends on centers of the latter two kinds is at most $1/2$.
- (iv) If client u depends on a private center v , $d(u, v) \leq d(u, u')$ for any other client $u' \in V'$.
- (v) The total cost under the modified objective is at most $2 \cdot \text{LPOpt}$.

Step (iii): Building Small Stars. Let us modify mapping η slightly: for each $u \in V'$, if it only depends (under LP solution x^3) on centers in $\mathcal{P}(u)$ (ie. centers in its local ball or its private centers) then reset $\eta(u) \leftarrow u$. Consider a directed dependency graph on just the clients V' , having arc-set $\{(u, \eta(u)) \mid u \in V', \eta(u) \neq u\}$. Each component will almost be a tree, except for the possible existence of one 2-cycle¹ (see Figure 9.2.1). We will call such 2-cycles *pseudo-roots*. If there is a vertex with no out-arc, that is also called a pseudo-root. Observe that every pseudo-root contains at least a unit of open centers.

The procedure we describe here is similar to the reduction to “3-level trees” in [39]. We break the trees up into a collection of stars, by traversing the trees in a bottom-up fashion, going from the leaves to the root. For any arc (u, u') , we say that u' is the *parent* of u , and u is a *child* of u' . Any client $u \in V'$ with no in-arc

¹In general, each component might have one cycle of any length; but since all edges in a cycle will have the same length, we may assume without loss of generality that there are only 2-cycles.

is called a *leaf*. Consider any non-leaf vertex u which is not part of a pseudo-root, such that all its children are leaves. Let u^{out} denote the parent of u .

1. Suppose there exists a child u_0 of u such that $d(u_0, u) \leq 2d(u, u^{\text{out}})$, then we make the following modification: let u_1 denote the child of u that is closest to u ; we replace the directed arc (u, u^{out}) with (u, u_1) , and make the collection $\{u, u_1\}$ (which is now a 2-cycle), a pseudo-root. Observe that $d(u_0, u) \geq d(u, u^{\text{out}})$ because u chose to direct its arc towards u^{out} instead of u_0 .
2. If there is no such child u_0 of u , then for every child u^{in} of u , replace arc (u^{in}, u) with a new arc $(u^{\text{in}}, u^{\text{out}})$. In this process, u has its in-degree changed to zero thereby becoming a leaf.

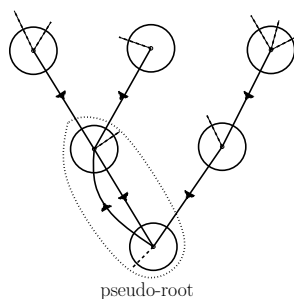


Figure 9.2.1: The Dependency Tree: Dashed edges represent private centers, circles represent the local balls

Notice that we have maintained the invariant that there are no out-arcs from any pseudo-root, and every node has at most one out-arc. Define mapping $\sigma : V' \rightarrow V'$ as follows: for each $u \in V'$, set $\sigma(u)$ to u 's parent in the final dependency graph (if

it exists); otherwise (if u is itself a pseudo-root) set $\sigma(u) = u$. Note that the final dependency graph is a collection of stars with centers as pseudo-roots.

Claim 9.2.5. For each $w \in V'$, we have $d(w, \sigma(w)) \leq 2 \cdot d(w, \eta(w))$.

Proof. Suppose that when w is considered as vertex u in the above procedure, step 1 applies. Then it follows that the out-arc of w is never changed after this, and by definition of step 1, $d(w, \sigma(w)) \leq 2 \cdot d(w, \eta(w))$. The remaining case is that when w is considered as vertex u , step 2 applies.

Then from the definition of steps 1 and 2, we obtain that there is a directed path $\langle w = w_0, w_1, \dots, w_t \rangle$ in the initial dependency graph such that $\eta(w) = w_1$ and $\sigma(w) = w_t$. Let $d(w, \eta(w)) = d(w_0, w_1) = a$.

We claim by induction on $i \in \{1, \dots, t\}$ that $d(w_i, w_{i-1}) \leq a/2^{i-1}$. The base case of $i = 1$ is obvious. For any $i < t$, assuming $d(w_i, w_{i-1}) \leq a/2^{i-1}$, we will show that $d(w_{i+1}, w_i) \leq a/2^i$. Consider the point when w 's out-arc is changed from (w, w_i) to (w, w_{i+1}) ; this must be so, since w 's out-arc changes from (w, w_1) to (w, w_t) through the procedure. At this point, step 2 must have occurred at node w_i , and w_{i-1} must have been a child of w_i ; hence $d(w_{i+1}, w_i) \leq \frac{1}{2} \cdot d(w_i, w_{i-1}) \leq a/2^i$.

Thus we have $d(w, \sigma(w)) \leq \sum_{i=1}^t d(w_i, w_{i-1}) \leq a \sum_{i=1}^t \frac{1}{2^{i-1}} < 2a = 2 \cdot d(w, \eta(w))$. ■

At this point, we have a fractional solution (x^3, y^*) that satisfies con-

straints (9.1.1)-(9.1.4) and:

$$\sum_{u \in V'} w_u \left[\sum_{v \in \mathcal{P}(u)} d(u, v) x_{uv}^3 + d(u, \sigma(u)) \left(1 - \sum_{v \in \mathcal{P}(u)} x_{uv}^3\right) \right] \leq 4 \cdot \text{LPOpt} \quad (9.2.8)$$

The inequality follows from (9.2.7) and Claim 9.2.5.

Stage II: Reformulating the LP

Based on the star-like structure derived in the previous subsection, we propose another linear program for which the fractional solution (x^3, y^*) is shown to be feasible with objective value as in (9.2.8). Crucially, we will show that this new LP is integral. Hence we can obtain an integral solution to it of cost at most $4 \cdot \text{LPOpt}$. Finally we show that any integral solution to our reformulated LP also corresponds to an integral solution to the original MatroidMedian instance, at the loss of another constant factor.

Consider the LP described in Figure 9.2.2.

The reason we have added the constraint 9.2.10 is the following: In the objective function, each client incurs only a cost of $d(u, \sigma(u))$ to the extent to which a private facility from $\mathcal{P}(u)$ is not assigned to it. This means that in our integral solution, we definitely want a facility to be chosen from the pseudo-root to which u is connected if we do not open a private facility from $\mathcal{P}(u)$; this fact becomes clearer later. Also, this constraint does not increase the optimal value of the LP, as

$$\begin{aligned}
& \text{minimize } \sum_{u \in V'} w_u && \left[\sum_{v \in \mathcal{P}(u)} d(u, v) z_v + d(u, \sigma(u)) \left(1 - \sum_{v \in \mathcal{P}(u)} z_v \right) \right] \\
& && \text{(LP}_2\text{)} \\
& \text{subject to } \sum_{v \in \mathcal{P}(u)} z_v \leq 1 && \forall u \in V' && (9.2.9) \\
& \sum_{v \in \mathcal{P}(u_1)} z_v + \sum_{v \in \mathcal{P}(u_2)} z_v \geq 1 && \forall \text{pseudo-roots } \{u_1, u_2\} && (9.2.10) \\
& \sum_{v \in S} z_v \leq r_{\mathcal{M}(S)} && \forall S \subseteq V && (9.2.11) \\
& z_v \geq 0 && \forall v \in V && (9.2.12)
\end{aligned}$$

Figure 9.2.2: Stage II LP Relaxation

shown below.

Claim 9.2.6. The linear program LP_2 has optimal value at most $4 \cdot \text{LPOpt}$.

Proof. Consider the solution z defined as: $z_v = \min\{y_v^*, x_{uv}^3\} = x_{uv}^3$ for all $v \in \mathcal{P}(u)$ and $u \in V'$; all other vertices have z -value zero. It is easy to see that constraints (9.2.9) and (9.2.11) are satisfied.

Constraint (9.2.10) is also trivially true for pseudo-roots consisting of only one client. Else, let $\{u_1, u_2\}$ be any pseudo-root consisting of two clients. Recall that each $u \in \{u_1, u_2\}$ is connected to centers in ball $\mathcal{B}(u, 2\text{LP}_u) \subseteq \mathcal{P}(u)$ to extent at least half; hence the total z -value inside $\mathcal{P}(u_1) \cup \mathcal{P}(u_2)$ is at least one. Thus z is feasible for LP_2 , and by (9.2.8) its objective value is at most $4 \cdot \text{LPOpt}$. ■

We show next that LP_2 is in fact, an integral polytope.

Lemma 9.2.7. *Any basic feasible solution to LP_2 is integral.*

Proof. Consider any basic feasible solution z . Firstly, notice that the characteristic vectors defined by constraints (9.2.9) and (9.2.10) define a laminar family, since all the sets $\mathcal{P}(u)$ are disjoint.

Therefore, the subset of these constraints that are tightly satisfied by z define a laminar family (of mostly disjoint sets). Also, by standard uncrossing arguments (see eg. [121]), we can choose the linearly-independent set of tight rank-constraints (9.2.11) to form a laminar family (in fact even a *chain*).

But then the vector z is defined by a constraint matrix which consists of *two laminar families* on the ground set of vertices. Such matrices are well-known to be *totally unimodular* [121], and this fact is used in proving the integrality of the matroid-intersection polytope. For completeness, we outline a proof of this fact next. This finishes the integrality proof. ■

Proof of TU-ness of Double Laminar Family We now show that such a matrix is *totally unimodular*. For this we use the following classical characterization: A matrix A is totally unimodular if, for each submatrix A' , its rows can be labeled $+1$ or -1 such that every column sum (when restricted to the rows of A') is either $+1$, 0 , or -1 . Consider such a submatrix A' . Clearly, we have chosen some constraints out of two laminar families, so the chosen rows also correspond to some two laminar families.

Consider one of these laminar families \mathcal{L} . We can define a forest by the following rules: We have a node for each set/tight-constraint in \mathcal{L} . Nodes S and T

are connected by a directed edge from S to T , iff $T \subseteq S$, and there exists no tight constraint $T' \in \mathcal{L} \setminus \{S, T\}$ such that $T \subseteq T' \subseteq S$. Then, we can label each set of \mathcal{L} in the following manner: each node of an odd level gets a label $+1$ and labels of an even levels are -1 (say roots has level of 1, and its children have level 2, and so on). By the laminarity, we know that a variable z_v appears in all the tight constraints which correspond to nodes on a path from some root to some other node. By the way we have labeled these constraints, we know that any such sum is either $+1$, or 0 (see Figure 9.2.3).

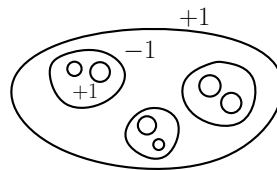


Figure 9.2.3: Labeling the laminar family

Similarly, we can label each set of the second laminar family \mathcal{L}' in the opposite fashion: each node of an odd level gets label -1 and nodes of even levels get $+1$. Again, z_v appears in all the tight constraints corresponding to nodes on a path from some root to some node, and the sum of these labels is either -1 or 0 . Therefore, the total sum corresponding to the column for z_v is either $+1$, 0 , or -1 , which completes the proof.

It is clear that any integral solution feasible for LP_2 is also feasible for MatroidMedian, due to (9.2.11). We now relate the objective in LP_2 to the original MatroidMedian objective:

Lemma 9.2.8. *For any integral solution $C \subseteq V$ to LP_2 , the MatroidMedian objective value under C is at most 3 times that it was paying in the LP_2 solution.*

Proof. We show that each client $u \in V'$ pays in MatroidMedian at most 3 times that in LP_2 . Suppose that $C \cap \mathcal{P}(u) \neq \emptyset$. Then u 's connection cost is identical to its contribution to the LP_2 solution's objective. Therefore, we assume $C \cap \mathcal{P}(u) = \emptyset$.

Suppose that u is not part of a pseudo-center; let $\{u_1, u_2\}$ denote the pseudo-center that u is connected to. By constraint (9.2.10), there is some $v \in C \cap (\mathcal{P}(u_1) \cup \mathcal{P}(u_2))$. The contribution of u is $d(u, \sigma(u))$ in LP_2 and $d(u, v)$ in the actual objective function for MatroidMedian. We will now show that $d(u, v) \leq 3 \cdot d(u, \sigma(u))$.

Without loss of generality let $\sigma(u) = u_1$ and suppose that $v \in \mathcal{P}(u_2)$; the other case of $v \in \mathcal{P}(u_1)$ is easier. From the property of private centers, we know $d(u_2, v) \leq d(u_2, \eta(u_2)) \leq d(u_2, u_1)$. Now if (u_1, u_2) is created as a new pseudo-root in step (iii).1, then we have the property that $d(u_1, u_2) \leq d(u_1, u)$, since we choose the closest leaf to pair up with its parent to form a pseudo-root. Else (u_1, u_2) is the original pseudo-root even before the modifications of step (iii). Thus in that case, by definition $d(u_1, u_2) = d(u_1, \eta(u_1)) \leq d(u_1, u)$. Therefore, $d(u, v) \leq d(u, u_1) + d(u_1, u_2) + d(u_2, v) \leq d(u, u_1) + 2 \cdot d(u_1, u_2) \leq 3 \cdot d(u, u_1) = 3 \cdot d(u, \sigma(u))$.

If u is itself (a singleton) pseudo-center then it must be that $C \cap \mathcal{P}(u) \neq \emptyset$ by (9.2.10), contrary to the above assumption. If u is part of a pseudo-center

$\{u, u'\}$. Then it must be that there is some $v \in C \cap \mathcal{P}(u')$, by (9.2.10). The contribution of u in LP_2 is $d(u, \sigma(u))$, and in MatroidMedian is $d(u, v) \leq d(u, u') + d(u', v) \leq 2 \cdot d(u, u') = d(u, \sigma(u))$ (the second inequality uses property of private centers). ■

To make this result algorithmic, we need to obtain in polynomial-time an extreme point solution to LP_2 . Using the Ellipsoid method (as mentioned in Section 9.1.2) we can indeed obtain *some* fractional optimal solution to LP_2 , which may not be an extreme point. However, such a solution can be converted to an extreme point of LP_2 , using the method in Jain [83]. (Due to the presence of both “ \leq ” and “ \geq ” type constraints in (9.2.9)-(9.2.10) it is not clear whether LP_2 can be cast directly as an instance of matroid intersection.)

Altogether, we obtain an integral solution to the weighted instance from **step (i)** of cost $\leq 12 \cdot \text{LPOpt}$. Combined with the property of **step (i)**, we obtain:

Theorem 9.2.9. *There is a 16-approximation algorithm for the MatroidMedian problem.*

We have not tried to optimize the constant via this approach. However, getting the approximation ratio to match that for usual k -median would require additional ideas.

9.3 MatroidMedian with Penalties

In the MatroidMedian problem with penalties, each client either connects to an open center thereby incurring the connection cost, or pays a penalty. Again, we are given a finite metric space (V, d) , a matroid \mathcal{M} with ground set V and a set of independent sets $\mathcal{J}(\mathcal{M}) \subseteq 2^V$; in addition we are also given a penalty function $p : V \rightarrow \mathbb{R}_{\geq 0}$. The goal is to open centers $S \in \mathcal{J}(\mathcal{M})$ and identify a set of clients C_1 such that $\sum_{u \in C_1} d(u, S) + \sum_{u \in V \setminus C_1} p(u)$ is minimized. Such objectives are also called “prize-collecting” problems. In this section we give a constant factor approximation algorithm for MatroidMedian with penalties, building on the rounding algorithm from the previous section.

Our LP relaxation is an extension of the one (LP₁) for MatroidMedian. In addition to the variables $\{y_v; v \in V\}$ and $\{x_{uv}; u, v \in V\}$, we define for each client an indicator variable h_u whose value equals 1 if client u pays a penalty and is not connected to any open facility. Then, it is straightforward to see that the

following LP is indeed a valid relaxation for the problem.

$$\begin{aligned} \min \quad & \sum_{u \in V} \sum_{v \in V} d(u, v) x_{uv} + \sum_{u \in V} p(u) h_u & (\text{LP}_3) \\ \text{s. t.} \quad & \sum_{v \in V} x_{uv} + h_u = 1 & \forall u \in V \end{aligned} \tag{9.3.13}$$

$$x_{uv} \leq y_v \quad \forall u \in V, v \in V \tag{9.3.14}$$

$$\sum_{v \in S} y_v \leq r_{\mathcal{M}}(S) \quad \forall S \subseteq V \tag{9.3.15}$$

$$x_{uv}, y_v, h_u \geq 0 \quad \forall u, v \in V \tag{9.3.16}$$

Let Opt denote an optimal solution of the given penalty MatroidMedian instance, and let LPOpt denote the LP_3 optimum value. Since LP_3 is a relaxation of our problem, we have that,

Lemma 9.3.1. *The LP_3 cost LPOpt is at most the cost of an optimal solution Opt .*

Let (x^*, y^*, h^*) denote the optimal LP solution. We round the fractional variables to integers in two stages. In the first stage, only x^*, h^* variables are altered such that the modified solution is still feasible but sparse and has a cost that is $O(1)\text{LPOpt}$. This stage is similar in nature to the first stage rounding for MatroidMedian but we need to be careful when comparing the x_{uv} 's and the y_v 's — the primary difficulty is that for any client u , the sum $\sum_v x_{uv}$ is not 1. Typically, this is the case in most LP-based prize-collecting problems, but often, one could argue that if $\sum_v x_{uv} \geq 2/3$, then by scaling we could ensure that it is at

least 1; *therefore* the (scaled) LP would be feasible to the original problem (without penalties). However, in our case (and also k -median with penalties) since we also have packing constraints (the matroid rank constraints), simply scaling the fractional solution is not a viable route.

Once we handle these issues, we show that a new LP can be written for which the modified fractional solution is feasible. The constraints of this LP are identical to that for MatroidMedian; but the objective function is different. Since that polytope is integral (Lemma 9.2.7) we infer that the new LP for MatroidMedian with penalties is integral. This immediately gives us the constant factor approximation for MatroidMedian with penalties. We now get into the details.

Stage I: Sparsifying the LP Solution

Like in the matroid median setting, the goal in this stage is to argue that there exists a *sparse* fractional solution of near optimal cost. This will enable us to write another LP, which will be characterized by an integral polytope.

Step (i): Thresholding Penalties.

Let \mathcal{C} denote the set of clients paying a penalty at most to an extent of $1/4$ in the fractional solution, i.e., $\mathcal{C} = \{u \in V \mid h_u^* < \frac{1}{4}\}$. For each client $u \in V \setminus \mathcal{C}$, we round its h_u^* to one and set x_{uv}^* to zero for all $v \in V$. Let (x^1, h^1, y^*) denote this modified solution. We make the following observations:

Observation 9.3.2. After the above thresholding operation, the following inequali-

ties are satisfied.

1. $\forall u \in \mathcal{C}, \sum_{v \in V} x_{uv}^1 > \frac{3}{4}$
2. $\forall u \in \mathcal{C}, h_u^1 < \frac{1}{4}$
3. $\sum_{u \in V} (\sum_{v \in V} x_{uv}^1 d(u, v) + h_u^1 p(u)) \leq 4 \text{LPOpt}$
4. $\forall u, v \in V, \text{ if } x_{uv}^1 > 0 \text{ then } p(u) \geq d(u, v).$
5. $\forall u, v \in V, \text{ if } x_{uv}^1 > 0 \text{ then } x_{uv}^1 = y_v^* \text{ or } h_u^1 = 0.$

Here, the second-to-last property is true because of the following: For any client $u \in V$ and center $v \in V$, if $x_{uv}^1 > 0$ and $d(u, v) > p(u)$, then we can increase h_u^1 to $h_u^1 + x_{uv}^1$ and set $x_{uv}^1 = 0$. Such a modification maintains feasibility and only decreases the objective function value.

The final property can be seen from the following argument: because $p(u) \geq d(u, v)$, whenever $x_{uv}^1 > 0$, we can increase the connection variable x_{uv}^1 and decrease h_u^1 equally without increasing the objective function until h_u^1 becomes 0 or x_{uv}^1 hits y_v^* .

Step (ii): Clustering Clients.

Let $|\mathcal{C}| = n'$. For each client $u \in \mathcal{C}$, let $D_u = \sum_{v \in V} d_{uv} x_{uv}^1$ denote the fractional connection cost of client u and $X_u^1 = \sum_{v \in V} x_{uv}^1$ denote the total fractional assignment towards connection. Also let $\mathcal{B}(u, R) = \{v \in V \mid d(u, v) \leq R\}$ denote the ball of radius R centered at u . For any vertex u , we say that $\mathcal{B}(u, 4D_u)$ is

the ‘local ball’ centered at u . The following is a direct consequence of Markov’s inequality and Observation 9.3.2-(1).

Lemma 9.3.3. *For any client $u \in \mathcal{C}$, $\sum_{v \in \mathcal{B}(u, 4D_u)} x_{uv}^1 \geq \frac{1}{2}$. In words, each client is fractionally connected to centers in its local ball to an extent of at least $1/2$.*

Now order the clients according to non-decreasing D_u values, and let the ordering be $u_1, u_2, \dots, u_{n'}$. Consider the vertices in the order $u_1, u_2, \dots, u_{n'}$. For a vertex u_i , if there exists a vertex u_j with $j < i$ such that $d(u_i, u_j) \leq 8D_{u_i}$, then we denote this event by $u_i \rightarrow u_j$ and modify the instance by shifting the location of client u_i to u_j .

For each client u_i , define $\pi(u_i) = u_j$ iff $u_i \rightarrow u_j$, and $\pi(u_i) = u_i$ if u_i was not shifted. Let $V' = \pi(\mathcal{C})$ denote the set of clients that maintain their own local balls (i.e. were not shifted in the above process). For each $u \in V'$, let $C_u := \{u' \in \mathcal{C} \mid \pi(u') = u\}$. The new instance consists of $|C_u|$ clients located at each vertex $u \in V'$ having respective penalty values $\{p(u') \mid u' \in C_u\}$.

Observation 9.3.4. For $u, v \in V'$, we have $d(u, v) > 8 \max(D_u, D_v)$.

We obtain a feasible solution (x^2, h^2, y^*) to this modified instance as follows.

For each event $u_i \rightarrow u_j$, do:

Case (i): If $X_{u_i}^1 \leq X_{u_j}^1$: Start with $x_{u_i v}^2 = 0$ for all v . Then, for each vertex v with $x_{u_j v}^1 > 0$, connect u_i to an extent of $x_{u_j v}^1$ to v , i.e. set $x_{u_i v}^2 = x_{u_j v}^1$. Finally, set $h_{u_i}^2 = h_{u_j}^1 \leq h_{u_i}^1$.

Case (ii): If $X_{u_i}^1 > X_{u_j}^1$: Start with $x_{u_i v}^2 = 0$ for all v . For each v with $x_{u_j v}^1 > 0$, connect u_i to an extent of $x_{u_j v}^1$ to v , i.e. set $x_{u_i v}^2 = x_{u_j v}^1$. Since $X_{u_j}^1 < X_{u_i}^1$, we need to further connect u_i to other centers to extent of at least $X_{u_i}^1 - X_{u_j}^1$ in order to avoid increasing h_{u_i} . To this end, set $x_{u_i, w}^2 = x_{u_i, w}^1$ for all $w \in V$ with $x_{u_j, w}^1 = 0$. Observe that client u_i is now connected to extent at least $X_{u_i}^1$; so $h_{u_i}^2 \leq 1 - X_{u_i}^1 = h_{u_i}^1$.

Also if a client is not shifted in the above routine, its x^2, h^2 variables are the same as in x^1, h^1 . The following lemma certifies that the objective of the modified instance is not much more than the original.

Lemma 9.3.5.

$$\sum_{u \in \mathcal{C}} \left[\sum_{v \in V} x_{uv}^2 \cdot d(\pi(u), v) + h_u^2 \cdot p(u) \right] \leq \sum_{u \in \mathcal{C}} \left[10 \sum_{v \in V} x_{uv}^1 \cdot d(u, v) + h_u^1 \cdot p(u) \right].$$

Proof. We prove the inequality term-wise. Any client that is not shifted in the above process maintains its contribution to the objective function. Hence consider a client u_i that is shifted to u_j (ie. $u_i \rightarrow u_j$). It is clear that $h_{u_i}^2 \leq h_{u_i}^1$, so the penalty contribution $h_{u_i}^2 \cdot p(u_i) \leq h_{u_i}^1 \cdot p(u_i)$. There are two cases for the connection costs:

Case (i): $X_{u_i}^1 \leq X_{u_j}^1$.

$$\begin{aligned} \text{In this case we have, } \sum_{v \in V} x_{u_i v}^2 d(\pi(u_i), v) &= \sum_{v \in V} x_{u_i v}^2 d(u_j, v) = \\ \sum_{v \in V} x_{u_j v}^1 d(u_j, v) &= D_{u_j} \leq D_{u_i}. \end{aligned}$$

Case (ii): $X_{u_i}^1 > X_{u_j}^1$.

Here, note that $x_{u_i v}^2 \leq x_{u_i v}^1 + x_{u_j v}^1$ for all $v \in V$. So,

$$\begin{aligned}
& \sum_{v \in V} x_{u_i v}^2 \cdot d(u_j, v) \\
& \leq \sum_{v \in V} x_{u_j v}^1 \cdot d(u_j, v) + \sum_{v \in V} x_{u_i v}^1 \cdot d(u_j, v) \\
& \leq D_{u_j} + \sum_{v \in V} x_{u_i v}^1 \cdot d(u_i, v) + \left(\sum_{v \in V} x_{u_i v}^1 \right) \cdot d(u_j, u_i) \\
& \leq D_{u_j} + D_{u_i} + d(u_j, u_i) \leq 10 \cdot D_{u_i}
\end{aligned}$$

Hence in either case, we can bound the new connection cost by $10 \cdot D_{u_i}$. ■

Thus it follows that (x^2, h^2, y^*) is a feasible LP solution to the modified instance of objective value at most 10 LPOpt . We additionally ensure (by locally changing x^2, h^2) that condition 4 of Observation 9.3.2 holds, namely:

$$\forall u \in V', u' \in C_u, v \in V, \text{ if } x_{u'v}^2 > 0 \text{ then } d(u, v) \leq p(u'). \quad (9.3.17)$$

Note that any feasible integral solution to the modified instance corresponds to one for the original instance, wherein the objective increases by at most an additive term of $8 \cdot \sum_{w \in \mathcal{C}} D_w \leq 8 \cdot \text{LPOpt}$. Hence in the rest of the algorithm we work with this modified instance.

Next we modify the connection-variables (leaving penalty variables h^2 unchanged) of clients exactly as in **step (ii)** of the previous section, and also alter

the coefficients of some x variables just like in the algorithm for MatroidMedian. This results in a disjoint set of private centers $\mathcal{P}(u)$ for each $u \in V'$ (where $\mathcal{P}(u)$ can be thought of as the collection of all private centers for $u' \in C_u$; notice that these are disjoint for different vertices in V'), and new connection variables \tilde{x}^3 such that:

- (a) Each client u' depends only on centers $\mathcal{P}(\pi(u'))$ and centers in the local ball nearest to $\pi(u')$. The connection to the latter type of centers is at most half.
- (b) For any client $u \in V'$ and center $v \in \mathcal{P}(u)$, we have $d(u, v) \leq d(u, w)$ for any other client $w \in V'$.
- (c) The total cost under the modified objective is at most $20 \cdot \text{LPOpt}$ (the factor 2 loss is incurred due to changing the objective coefficients and rearrangements).
- (d) For any $u' \in \mathcal{C}$, $v \in V$ with $\tilde{x}_{u'v}^3 > 0$ we have $d(\pi(u'), v) \leq 3 \cdot p(u')$.
Additionally, for $u' \in \mathcal{C}$, $v \in \mathcal{P}(\pi(u'))$ with $\tilde{x}_{u'v}^3 > 0$ we have $d(\pi(u'), v) \leq p(u')$.

The first three properties above are immediate from the corresponding properties after step (ii) of Section 9.2. The last property uses (9.3.17) and Claim 9.2.4.

We now modify the penalty variables as follows (starting with $h^3 = h^2$ and $x^3 = \tilde{x}^3$). For each client u' , if it is connected to centers in the local-ball of any $w \in V' \setminus \{\pi(u')\}$ then reset $h^3(u') = 0$; and increase the connection-variables

$$\begin{aligned}
& \sum_{u \in M} \left[\sum_{v \in T(u)} d(\pi(u), v) \cdot x_{uv}^3 + p_u \cdot \left(1 - \sum_{v \in T(u)} x_{uv}^3 \right) \right] + \\
& \sum_{u \in \mathcal{C} \setminus M} \left[\sum_{v \in \mathcal{P}(\pi(u))} d(\pi(u), v) x_{uv}^3 + d(\pi(u), \sigma(\pi(u))) \cdot \left(1 - \sum_{v \in \mathcal{P}(\pi(u))} x_{uv}^3 \right) \right],
\end{aligned} \tag{9.3.18}$$

Figure 9.3.4: Modified Objective Function of (x^3, h^3, y^*)

$x^3(u', \cdot)$ to centers in the local-ball of w until client u' is connected to extent one.

(Such a modification is possible since u' is already connected to extent at least half in the local ball of $\pi(u')$, and there is at least half open centers in any local ball.)

Furthermore, using property (d) above, the new objective value of (x^3, h^3, y^*) is at most *thrice* that of (\tilde{x}^3, h^3, y^*) , ie. at most $60 \cdot \text{LPOpt}$.

Observation 9.3.6. Any client u' that has $h^3(u') > 0$ is connected *only* to centers in $\mathcal{P}(\pi(u'))$.

We also apply **step (iii)** from Section 9.2 to obtain a mapping $\sigma : V' \rightarrow V'$ satisfying Claim 9.2.5 (recall that $\eta : V' \rightarrow V'$ maps each client in V' to its closest other client). This increases the objective value by at most factor 2.

Let $M = \{u' \in \mathcal{C} \mid h^3(u') > 0\}$ denote the clients that have non-zero penalty variable. For each $u' \in M$ let $T(u') \subseteq \mathcal{P}(\pi(u'))$ denote the centers that client u' is connected to (We may assume that $T(u')$ consists of centers in $\mathcal{P}(u)$ closest to u). The objective of (x^3, h^3, y^*) can then be expressed as in the equation in Figure 9.3.4. From the arguments above, the cost of this solution is at most $120 \cdot$

$$\sum_{u \in M} \left[\sum_{v \in T(u)} d(\pi(u), v) \cdot y_v^* + p_u \cdot \left(1 - \sum_{v \in T(u)} y_v^* \right) \right] \\ + \sum_{u \in \mathcal{C} \setminus M} \left[\sum_{v \in \mathcal{P}(\pi(u))} d(\pi(u), v) \cdot y_v^* + d(\pi(u), \sigma(\pi(u))) \cdot \left(1 - \sum_{v \in \mathcal{P}(\pi(u))} y_v^* \right) \right]$$

Figure 9.3.5: Objective Function for Sparse LP

LPOpt.

Stage II: Reformulating the LP

Reducing center variables y^* . For any $u \in V'$, if $\sum_{v \in \mathcal{P}(u)} y_v^* > 1$ then we reduce the y^* -values in $\mathcal{P}(u)$ one center at a time (starting from the farthest center to u) until $y^*(\mathcal{P}(u)) = 1$. Clearly this does not cause the objective to increase. Additionally, y^* still satisfies the matroid independence constraints. Thus we can ensure that $\sum_{v \in \mathcal{P}(u)} y_v^* \leq 1$ for all $u \in V'$. Additionally, the following two modifications do not increase the objective.

1. *Client* $u' \in M$. For all $v \in T(u')$ we have $d(\pi(u'), v) \leq p(u')$ (property (d) above); set $x^3(u', v) = y_v^*$.
2. *Client* $u \in \mathcal{C} \setminus M$. For all $v \in \mathcal{P}(\pi(u))$ we have $d(\pi(u), v) \leq d(\pi(u), \sigma(\pi(u)))$ (property (b) above); again set $x^3(u, v) = y_v^*$.

Thus we can re-write the objective from (9.3.18) as shown in Figure 9.3.5 (which is just that in Figure 9.3.4 with the x variables replaced by the y variables). Notice

that there are no x -variables in the above expression. Furthermore, y^* satisfies all the constraints (9.2.9)-(9.2.12). We now consider linear program LP_4 with the linear objective (9.3.19) and constraints (9.2.9)-(9.2.12). This can be optimized in polynomial time to obtain an optimal integral solution F (as described in Subsection 9.2). From the reductions in the previous subsection, the objective value of F under (9.3.19) is at most $120 \cdot LPOpt$. Finally, using Lemma 9.2.8 we obtain that F is a feasible solution to MatroidMedian with penalties, of objective at most $360 \cdot LPOpt$.

Theorem 9.3.7. *There is a constant approximation algorithm for MatroidMedian with penalties.*

9.4 The KnapsackMedian Problem

In this section we consider the KnapsackMedian problem. We are given a finite metric space (V, d) , non-negative weights $\{f_i\}_{i \in V}$ and a bound F . The goal is to open centers $S \subseteq V$ such that $\sum_{j \in S} f_j \leq F$ and the objective $\sum_{u \in V} d(u, S)$ is minimized. We can write a LP relaxation (LP_5) of the above problem similar to (LP) in Section 9.1.1, where we replace the constraint (9.1.3) with the knapsack constraint $\sum_{v \in V} f_v y_v \leq F$. In addition, we guess the maximum weight facility f_{max} used in an optimum solution, and if $f_v > f_{max}$ we set $y_v = 0$ (and hence $x_{uv} = 0$ as well). This is clearly possible since there are only n many different choices for f_{max} . Unfortunately LP_5 has an unbounded integrality gap if we do not

allow any violation in the knapsack constraint. In Subsection 9.4, we show that a similar integrality gap persists even if we add the *knapsack-cover (KC) inequalities* to strengthen LP₅, which have often been useful to overcome the gap of the natural LP [33].

However, in the following section, we show that with an *additive* slack of f_{max} in the budget, we can get a constant factor approximation for the knapsack median problem.

The Rounding Algorithm for KnapsackMedian. Let (x^*, y^*) denote the optimal LP solution of LP₅. The rounding algorithm follows similar steps as in MatroidMedian problem. The first stage is identical to Stage I of Section 9.2 modifying x_{uv} variables until we have a collection of disjoint stars with pseudo-roots. The total connection cost of the modified LP solution is at most a constant factor of the optimum LP cost for LP₅. The sparse instance satisfies the budget constraint since y_u variables are never increased. In Stage II, we start with a new LP (LP₆) by replacing the constraint 9.2.11 of LP₂ with the knapsack constraint $\sum_{v \in V} f_v z_v \leq F$. However LP₆ is not integral as opposed to LP₂: it contains the knapsack problem as a special case. We now give an iterative-relaxation procedure that rounds the above LP₆ into an integral solution by violating the budget by at most an additive f_{max} and maintaining the optimum connection cost. The following algorithm iteratively creates the set of open centers \mathcal{C} .

1. Initialize $\mathcal{C} \leftarrow \emptyset$. While $V \neq \emptyset$ do

- (a) Find an extreme point optimum solution \hat{z} to LP_6 .
- (b) If there is a variable $\hat{z}_v = 0$, then remove variable \hat{z}_v , set $V = V \setminus \{v\}$.
- (c) If there is a variable $\hat{z}_v = 1$, then $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$, $V = V \setminus \{v\}$ and $F = F - f_v$.
- (d) If none of (b), (c) holds, and $|V| = 2$ (say $V = \{x_1, x_2\}$) then:
- If $x_1, x_2 \in \mathcal{P}(u)$ for some $u \in V'$. If $d(x_1, u) \leq d(x_2, u)$ then $\mathcal{C} \leftarrow \mathcal{C} \cup \{x_1\}$, else $\mathcal{C} \leftarrow \mathcal{C} \cup \{x_2\}$. Break.
 - If $x_1, x_2 \in \mathcal{P}(u_1) \cup \mathcal{P}(u_2)$ for some pseudo-root $\{u_1, u_2\}$. Then $\mathcal{C} \leftarrow \mathcal{C} \cup \{x_1, x_2\}$, Break.

2. Return \mathcal{C}

The following lemma guarantees that the connection cost is at most the Opt cost of LP_4 and the budget is not exceeded by more than an additive f_{max} .

Lemma 9.4.1. *The above algorithm finds a solution for knapsack median problem that has cost at most Opt cost of LP_6 and that violates the knapsack budget at most by an additive f_{max} .*

Proof. First we show that if the algorithm reaches Step (2), then the solution returned by the algorithm satisfies the guarantee claimed. In Step (c), we always reduce the remaining budget by f_v if we include the center in \mathcal{C} . Thus the budget constraint can only be violated at Step (d). In Step (d), in case of tight V' -constraint, we open only one center among the two remaining centers. Thus the budget constraint

can be violated by at most $\max\{f_{x_1}, f_{x_2}\} \leq f_{max}$. In case of tight pseudo-root, we have $z(x_1) + z(x_2) = 1$ and thus $f_{x_1} \cdot z(x_1) + f_{x_2} \cdot z(x_2) + \max\{f_{x_1}, f_{x_2}\} \leq f_{x_1} + f_{x_2}$. Hence again the budget constraint can be violated by at most an additive f_{max} term. The total cost of LP_6 never goes up in Step (a)-(c). In Step (d), either the nearer center from $\{x_1, x_2\}$ is chosen (in case of tight V' -constraint), or both the centers $\{x_1, x_2\}$ (in case of tight pseudo-root) are opened. Thus the connection cost is always upper bounded by Opt of LP_6 .

To complete the proof we show that the algorithm indeed reaches Step (2). The Steps (1b),(1c) all make progress in the sense that they reduce the number of variables and hence some constraints become vacuous and are removed. Therefore, we want to show whenever we are at an extreme point solution, then either Step (1b),1(c) apply or we have reached (1d) and hence Step (2). Suppose that neither (1b) nor (1c) apply: then there is no $z_v \in \{0, 1\}$. Let the linearly independent tight constraints defining z be: $T \subseteq V'$ from (9.2.9), and R (pseudo-roots) from (9.2.10). From the laminar structure of the constraints and all right-hand-sides being 1, it follows that the sets in $T \cup R$ are all disjoint. Further, each set in $T \cup R$ contains at least two fractional variables. Hence the number of variables is at least $2|T| + 2|R|$. Now count the number of tight linearly independent constraints: There are at most $|T| + |R|$ tight constraints from (9.2.9)-(9.2.10), and one global knapsack constraint. Since at an extreme point, the number of variables must equal the number of tight linearly independent constraints, we obtain $|T| + |R| \leq 1$ and that each set in $T \cup R$

contains exactly two vertices. This is possible only when V is some $\{x_1, x_2\}$.

1. $|T| = 1$. Then there must be some $u \in V'$ with $x_1, x_2 \in \mathcal{P}(u)$.
2. $|R| = 1$. Then there must be some pseudo-root $\{u_1, u_2\}$ with $x_1, x_2 \in \mathcal{P}(u_1) \cup \mathcal{P}(u_2)$.

So in either case, Step (1d) applies. ■

Combining Lemma 9.4.1 with Claim 9.2.6, Lemma 9.2.8 and the property of **step (i)**-Stage-I rounding of Section 9.2, we get the following theorem.

Theorem 9.4.2. *There is a 16-approximation algorithm for the KnapsackMedian problem that violates the knapsack constraint at most by an additive f_{max} , where f_{max} is the maximum weight of any center opened in the optimum solution.*

Using enumeration for centers with cost more than εF , we can guarantee that we do not exceed the budget by more than εF while maintaining a 16-approximation for the connection cost in $n^{O(\frac{1}{\varepsilon})}$ time.

LP Integrality Gap for KnapsackMedian with Knapsack Cover Inequalities

There is a large integrality gap for LP_5 with a hard constraint for the Knapsack bound, from Charikar and Guha [38].

Example 3 ([38]). Consider $|V| = 2$ with $f_1 = N$, $f_2 = 1$, $d(1, 2) = D$ and $F = N$ for any large positive reals N and D . An optimum solution that does not

violate the knapsack constraint can open either center 1 or 2 but not both and hence must pay a connection cost of D . LP_5 can assign $y_1 = 1 - \frac{1}{N}$ and $y_2 = 1$ and thus pay only D/N in the connection cost.

The above example can be overcome by adding *knapsack covering (KC) inequalities* [33]. We now illustrate the use of KC inequalities in the KnapsackMedian problem. KC-inequalities are used for *covering* knapsack problems. Although KnapsackMedian has a packing constraint (at most F weight of open centers), it can be rephrased as a covering-knapsack by requiring “at least $\sum_{v \in V} f_v - F$ weight of closed centers”. Viewed this way, we can strengthen the basic LP as follows.

Define for any subset of centers $S \subseteq V$, $f(S) := \sum_{v \in S} f(v)$. Then to satisfy the knapsack constraint we need to close centers worth of $F' := f(V) - F$. For any subset $S \subseteq V$ of centers with $f(S) < F'$ we can write a KC inequality *assuming* that all the centers in S are closed. Then, the residual covering requirement is:

$$\sum_{v \notin S} \min \{f(v), F' - f(S)\} (1 - y_v) \geq F' - f(S).$$

There are exponential number of such inequalities; however using methods in [33] an FPTAS for the strengthened LP can be obtained. The addition of KC inequalities avoids examples like 3; there $F' = 1$ and setting $S = \emptyset$ yields:

$$\min\{1, 1\} \cdot (1 - y_1) + \min\{1, N\} \cdot (1 - y_2) \geq 1,$$

ie. $y_1 + y_2 \leq 1$. Thus the LP optimum also has value D .

However the following example shows that the integrality gap remains high even with KC inequalities.

Example 4. $V = \{a_i\}_{i=1}^R \cup \{b_i\}_{i=1}^R \cup \{p, q, u, v\}$ with metric distances d as follows: vertices $\{a_i\}_{i=1}^R$ (resp. $\{b_i\}_{i=1}^R$) are at zero distance from each other, $d(a_1, b_1) = d(p, q) = d(u, v) = D$ and $d(a_1, p) = d(p, u) = d(u, a_1) = \infty$. The facility-costs are $f(a_i) = 1$ and $f(b_i) = N$ for all $i \in [R]$, and $f(p) = f(q) = f(u) = f(v) = N$. The knapsack bound is $F = 3N$. Moreover $N > R \gg 1$.

An optimum integral solution must open exactly one center from each of $\{a_i\}_{i=1}^R \cup \{b_i\}_{i=1}^R$, $\{p, q\}$ and $\{u, v\}$ and hence has connection cost of $(R + 2)D$.

On the other hand, we show that the KnapsackMedian LP with KC inequalities has a feasible solution x with much smaller cost. Define $x(a_i) = 1/R$ and $x(b_i) = \frac{N-1}{RN}$ for all $i \in [N]$, and $x_p = x_q = x_u = x_v = \frac{1}{2}$. Observe that the connection cost is $(\frac{R}{N} + 2)D < 3D$. Below we show that x is feasible; hence the integrality gap is $\Omega(R)$.

x clearly satisfies the constraint $\sum_{w \in V} f_w \cdot x_w \leq F$. We now show that x satisfies all KC-inequalities. Recall that $F' = f(V) - F = (R + 1)N + R$ for this instance. Note that KC-inequalities are written only for subsets S with $F' - f(S) > 0$. Also, KC-inequalities corresponding to subsets S with $F' - f(S) \geq N = \max_{w \in V} f_w$ reduce to $\sum_{w \notin S} f_w \cdot y_w \leq F$, which is clearly satisfied by x . Thus the only remaining KC-inequalities are from subsets S with $0 < F' - f(S) < N$,

ie. $f(S) \in [F' - N + 1, F' - 1] = [RN + R + 1, (R + 1)N + R - 1]$. Since all facility-costs are in $\{1, N\}$ and $R < N$, subset S must have *exactly* $R + 1$ cost N facilities. Thus there are exactly three cost- N facilities H in S^c . Since $x_w \leq \frac{1}{2}$ for all $w \in V$, we have $\sum_{w \in H} (1 - x_w) \geq \frac{3}{2}$. The KC-inequality from S is hence:

$$\begin{aligned}
& \sum_{w \in S^c} \min \{f(w), F' - f(S)\} (1 - x_w) \\
& \geq \sum_{w \in H} \min \{f(w), F' - f(S)\} (1 - x_w) \\
& = (F' - f(S)) \cdot \sum_{w \in H} (1 - x_w) > F' - f(S).
\end{aligned}$$

The equality uses $F' - f(S) < N$ and that each facility-cost in H is N , and the last inequality is by $\sum_{w \in H} (1 - x_w) \geq \frac{3}{2}$ which was shown above.

9.5 Bad Example for Local Search with Multiple Swaps

Here we give an example showing that any local search algorithm for the T -server type problem (ie. MatroidMedian under partition matroid of T parts) that uses at most $T - 1$ swaps cannot give an approximation factor better than $\Omega(\frac{n}{T})$; here n is the number of vertices.

The metric is uniform on $T + 1$ locations. There are two servers of each type: Each location $\{2, 3, \dots, T\}$ contains two servers; locations 1 and $T + 1$ contain a single server each. For each $i \in [1, T]$, the two copies of server i are located at locations i (first copy) and $i + 1$ (second copy). There are $m \gg 1$ clients at each

location $i \in [1, T]$ and just one client at location $T + 1$; hence $n = 2T + mT + 1$. The bounds on server-types are $k_i = 1$ for all $i \in [1, T]$. The optimum solution is to pick the *first copy* of each server type and thus pay a connection cost of 1 (the client at location $T + 1$). However, it can be seen that the solution consisting of the *second copy* of each server type is locally optimal, and its connection cost is m (clients at location 1). Thus the locality gap is $m = \Omega(n/T)$.

AdCell: Ad Allocation in Cellular Networks

In this chapter, we develop a novel application of advertisement allocation in cellular networks. With worldwide usage of cellular phones, mobile advertisement is rapidly becoming an attractive alternative to online advertisements. In our model, a wireless service provider (WSP) charges the advertisers for showing their ads. Each advertiser has a valuation for specific types of customers in various times and locations and has a limit on the maximum available budget. Each query is in the form of time and location and is associated with one individual customer. In order to achieve a non-intrusive delivery, only a limited number of ads can be sent to each customer. In this chapter, we show an offline algorithm for this problem, where the bid values, budgets and customer locations are known apriori, and WSP has to decide for each bidder when to show its ads. Here, decisions are based on historical data, however such offline algorithms often help in decision making and also in designing online algorithms. For more details of the online algorithms, the interested readers are referred to our paper [1].

10.1 AdCell: An Introduction

More than 4 billion cellular phones are now being used world-wide, and with increasing popularity of smart phones, mobile advertising holds the prospect of significant growth in recent future. Some research firms [3] estimate mobile advertisements to reach a business worth over 10 billion US dollars by 2012. Given the built-in advertisement solutions from popular smart phone operating systems, such as iAds for Apple's iOS, mobile advertising market is poised with even faster growth.

In the mobile advertising ecosystem, wireless service providers (WSPs) render the physical delivery infrastructure, but so far WSPs have been more or less left out from profiting via mobile advertising because of several challenges. First, unlike web, search, application, and game providers, WSPs typically do not have users' application context, which makes it difficult to provide targeted advertisements. Deep Packet Inspection (DPI) techniques that examine packet traces in order to understand application context, is often not an option because of privacy and legislation issues (i.e., Federal Wiretap Act). Therefore, a targeted advertising solution for WSPs need to utilize *only the information it is allowed to collect by government and by customer via opt-in mechanisms*. Second, without the luxury of application context, targeted ads from WSPs require *non-intrusive delivery methods*. While users are familiar with other ad forms such as banner, search, in-application, and

in-game, push ads with no application context (e.g., via SMS) can be intrusive and annoying if not done carefully. The number and frequency of ads both need to be well-controlled. Third, targeted ads from WSPs should be well personalized such that the users have incentive to read the advertisements and take purchasing actions, especially given the requirement that the number of ads that can be shown to a customer is limited.

Our proposed advertising solution, Adcell, deals with the above challenges. It takes advantage of the detailed real-time location information of users. Location can be tracked upon users' consent. This is already being done in some services offered by WSPs, such as Sprint's Family Location and AT&T's Family Map, thus there is no associated privacy or legal complications. To locate a cellular phone, it must emit a roaming signal to contact some nearby antenna tower, but the process does not require an active call. GSM localization is then done by multi-lateration¹ based on the signal strength to nearby antenna masts [136]. Location-based advertisement is not completely new. Foursquare mobile application allows users to explicitly "check in" at places such as bars and restaurants, and the shops can advertise accordingly. Similarly there are also automatic proximity-based advertisements using GPS or bluetooth. For example, some GPS models from Garmin display ads for the nearby business based on the GPS locations [142]. ShopAlerts by AT&T² is another application along the same line. On the advertiser side, pop-

¹The process of locating an object by accurately computing the time difference of arrival of a signal emitted from that object to three or more receivers.

²<http://shopalerts.att.com/sho/att/index.html?ref=portal>

ular stores such as Starbucks are reported to have attracted significant footfalls via mobile coupons.

To deal with the non-intrusive delivery challenge, we propose user subscription to advertising services that deliver only a *fixed number* of ads per month to its subscribers. The constraint of delivering limited number of ads to each customer adds the main algorithmic challenge in the AdCell model (details in Section 10.1.1). In order to overcome the incentive challenge, the WSP can “pay” users to read ads and purchase based on them through a reward program in the form of credit for monthly wireless bill. To begin with, both customers and advertisers should sign-up for the AdCell-service provided by the WSP. Customers enrolled for the service should sign an agreement that their *location* information will be tracked; but solely for the advertisement purpose. Advertisers (e.g., stores) provide their advertisements and a maximum chargeable budget to the WSP.

The WSP selects proper ads (these, for example, may depend on time and distance of a customer from a store) and sends them (via SMS) to the customers. The WSP charges the advertisers for showing their ad and also for a successful ad. An ad is deemed successful if a customer visits the advertised store. Depending on the service plan, customers are entitled to receive different number of advertisements per month. Several logistics need to be employed to improve AdCell experience and entuse customers into participation. More detailed description of such logistics can be found in an extended version of our paper [1].

10.1.1 AdCell Model & Problem Formulation

In the AdCell model, advertisers bid for individual customers based on their location and time. The triple (k, ℓ, t) where k is a customer, ℓ is a neighborhood (location) and t is a time form a query and there is a bid amount (possibly zero) associated with each query for each advertiser. This definition of query allows advertisers to customize their bids based on customers, neighborhoods and time. We assume a customer can only be in one neighborhood at any particular time and thus at any time t and for each customer k , the queries (k, ℓ_1, t) and (k, ℓ_2, t) are mutually exclusive, for all distinct ℓ_1, ℓ_2 . Neighborhoods are places of interest such as shopping malls, airports, etc. We assume that queries are generated at certain times (e.g., every half hour) and only if a customer stays within a neighborhood for a specified minimum amount of time. The formal problem definition of *Adcell Allocation* is as follows:

Adcell Allocation *There are m advertisers, n queries and s customers. Advertiser i has a total budget B_i and bids $b_{i,j}$ for each query j . Furthermore, for each customer $h \in [1, s]$, let S_h denote the queries corresponding to customer h and c_h denote the maximum number of queries from S_h for which advertisements can be shown. c_h is the capacity associated with customer h and is dictated by the AdCell plan chosen by customer h . Advertiser i pays $b_{i,j}$ if his advertisement is shown for query j and if his budget is not exceeded. That is, if $y_{i,j}$ is an indicator variable*

set to 1, when advertisement for advertiser i is shown on query j , then advertiser i pays a total amount of $\min(\sum_j y_{ij}b_{ij}, B_i)$. The goal of AdCell Allocation is to specify an advertisement allocation plan given $\{m, n, s, S_h, c_h | h \in [1, s]\}$ such that the total payment $\sum_i \min(\sum_j y_{ij}b_{ij}, B_i)$ is maximized.

The AdCell problem is a generalization of the budgeted adword allocation problem [35, 125] with capacity constraint on each customer and thus is NP-hard. Online AdCell problem also generalizes the so-called *online secretary problem* for which no deterministic or randomized online algorithm can get approximation ratio better than $\frac{1}{n}$. The reduction of the *secretary problem* to Adcell problem is as follows: consider a single advertiser with large enough budget and a single customer with a capacity of 1. The queries correspond to secretaries and the bids correspond to the values of the secretaries. So we can only allocate one query to the advertiser.

We can write the AdCell problem as the following integer program in which y_{ij} is the indicator variable which is 1 if query j is assigned to bidder i and 0 otherwise:

$$\text{maximize.} \quad \sum_i \min(\sum_j y_{ij}b_{ij}, B_i) \quad (IP_{BC})$$

$$\forall j \in [n] : \quad \sum_i y_{ij} \leq 1 \quad (\text{Assign})$$

$$\forall h \in [s] : \quad \sum_{j \in S_h} \sum_i y_{ij} \leq c_h \quad (\text{Capacity})$$

$$\forall i \in [m], \forall j \in [m] : \quad y_{ij} \in \{0, 1\}$$

We refer to the variant of the problem explained above as IP_{BC} . The above integer program can be relaxed to obtain a linear programming relaxation LP_{BC} , where we maximize $\sum_i \sum_j y_{ij} u_{ij}$ with the above constraints (F), (C) and additional budget constraint $\sum_j y_{ij} u_{ij} \leq b_i(\mathbf{B})$. We relax the variables $y_{ij} \in \{0, 1\}$ to $y_{ij} \geq 0$ to form the LP relaxation LP_{BC} .

Our algorithm is based on LP rounding. The approximation ratio matches the previously known bounds for adword allocation in offline setting, where there was no capacity constraint [35, 125]. Handling “hard capacities”, those that cannot be violated, is generally tricky in various settings including facility location and many covering problems [46, 63, 113]. The AdCell problem is a generalization of the budgeted adword allocation problem, where we additionally have hard capacity constraints on the number of queries that can be received by any customer. The optimal LP fractional solution for this problem can be seen as a bipartite graph with bidders and persons forming the two partitions. Edges represent the fractional allocations of queries. The essential idea is to first decompose the fractional optimal solution into a forest and apply the rounding technique of [118] over a carefully chosen subset of constraints from the original linear program. The problem is significantly harder than its uncapacitated counterpart. The selected constraints can span customers and bidders from different forests; depending on the current state of the forest, different constraints may need to be chosen and new constraints may need to be added.

10.2 Offline Setting

In this section, we give an offline algorithm for the budgeted adword allocation problem with capacity constraint, that is Adcell with budget and capacity. The capacity constraint restricts how many queries can be assigned to each customer. The instances of a single customer represents a set, and thus sets are disjoint and have an integer capacity associated with them³. We obtain the same approximation factor of $\frac{3}{4}$ for this case as has been shown in [35, 125] without any capacity constraint. Specifically, we prove the following theorem,

Theorem 10.2.1. *Given a fractional optimal solution for LP_{BC} , we can obtain an integral solution for AdCell with budget and capacity constraints that obtains at least a profit of $\frac{4 - \max_i \frac{\max_j b_{i,j}}{B_i}}{4}$ of the profit obtained by optimal fractional allocation and maintains all the capacity constraints exactly.*

Our approximation algorithm is based on carefully rounding a linear programming relaxation for the problem. The essential idea of the proposed rounding is to apply **RandMove** to the variables of a suitably chosen subset of constraints from the original linear program.

Our starting point is the LP relaxation LP_{BC} . The constraints “ $\sum_{i=1}^m y_{i,j} \leq 1$ ” are denoted as Assign constraint, the constraints “ $\sum_{j=1}^n b_{i,j} y_{i,j} \leq B_i$ ” are denoted as Advertiser constraints and finally “ $\sum_{j \in S_h} \sum_{i=1}^m y_{i,j} \leq c_h$ ” are the Capacity

³When the sets may overlap and have integer hard capacities, no non-trivial approximation can be obtained; this can be shown via a reduction from maximum independent set.

constraints. Let Opt denote an optimal solution of the given AdCell problem and let LPOpt denote the LP optimal value. It follows that the LP cost LPOpt is at least the cost of an optimal solution Opt .

10.2.1 Rounding Algorithm

Let y^* denote an LP optimal solution. We begin by simplifying the assignment given by y^* . Consider a bipartite graph $G(\mathcal{B}, \mathcal{J}, E^*)$ with advertisers \mathcal{B} on one side, queries \mathcal{J} on the other side and add an edge (i, j) between an advertiser i and query j , if $y_{i,j}^* \in (0, 1)$. That is, define $E^* = \{(i, j) \mid 1 > y_{i,j}^* > 0\}$. Our first claim is that y^* can be modified without affecting the value of LPOpt such that $G(\mathcal{B}, \mathcal{J}, E^*)$ is a forest. The proof follows from Claim 2.1 of [35]; we additionally show that such assumption of forest structure maintains the capacity constraints.

Lemma 10.2.2. *Bipartite graph $G = (\mathcal{B}, \mathcal{J}, E^*)$ induced by the edges E^* can be converted to a forest maintaining the optimal objective function value.*

Proof. Consider the graph $G = (\mathcal{B}, \mathcal{J}, E^*)$ and consider one connected component of it. We will argue for each component separately and similarly.

Cycle Breaking: Suppose there is a cycle in the chosen component. Since G is bipartite, the cycle has even length. Let the cycle be $C = \langle i_1, j_1, i_2, j_2, \dots, i_l, j_l, i_1 \rangle$, that is consider the cycle to start from an advertiser node. Consider a strictly positive value α and consider the following update of the y^* values over the edges in the cycle C . We add $z_{a,b}$ to edge (a, b) , where

R1. $z_{i_1, j_1} = -\beta$

R2. If we are at an query node j_t , $t \in [1, l]$, then

$$z_{j_t, i_{t+1}} = -z_{i_t, j_t}$$

R3. If we are at an advertiser node i_t , $t \in [1, l]$, then

$$z_{i_t, j_t} = -\frac{b_{i_t, j_{t-1}} z_{j_{t-1}, i_t}}{b_{i_t, j_t}}$$

β is chosen such that after the update, all the variables lie in $[0, 1]$ and at least one variable gets rounded to 0 or 1, thus the cycle is broken. Note that the entire update is a function of z_{i_1, j_1} . For any query node, its total contribution in (Assign) constraint of LP1 remains unchanged. For any advertiser node, except i_1 , its contribution in (Advertiser) constraint and thus in the objective function remains the same. In addition, since the assign constraints remain unaffected, all the capacity constraints are satisfied. For advertiser i_1 , its contribution decreases by $z_{i_1, j_1} b_{i_1, j_1}$ and increases by $z_{j_l, i_1} b_{i_1, j_l} = z_{i_1, j_1} b_{i_1, j_l} \frac{b_{i_2, j_1} b_{i_3, j_2} \dots b_{i_{l-1}, j_{l-2}}}{b_{i_2, j_2} b_{i_3, j_3} \dots b_{i_{l-1}, j_{l-1}}}$. If $b_{i_1, j_1} \leq b_{i_1, j_l} \frac{b_{i_2, j_1} b_{i_3, j_2} \dots b_{i_{l-1}, j_{l-2}}}{b_{i_2, j_2} b_{i_3, j_3} \dots b_{i_{l-1}, j_{l-1}}}$, then instead of adding z_{j_l, i_1} on the last edge, we add some $c < z_{j_l, i_1}$ such that $z_{i_1, j_1} b_{i_1, j_1} = c b_{i_1, j_l}$. Thus, we are able to maintain the objective function exactly. The assign constraint on the last query j_l can only decrease by this change and hence all the capacity constraints are maintained as well.

Otherwise, $b_{i_1, j_1} > b_{i_1, j_l} \frac{b_{i_2, j_1} b_{i_3, j_2} \dots b_{i_{l-1}, j_{l-2}}}{b_{i_2, j_2} b_{i_3, j_3} \dots b_{i_{l-1}, j_{l-1}}}$. In that case, we traverse the cycle in the reverse order, that is, we start by decreasing on z_{i_1, j_l} first and proceed similarly. ■

We now have a collection of trees. There can arise several cases at this stage. For each of these cases, we identify a set of linear constraints and apply our **Rand-Move** step on the variables in the chosen system of linear constraints. We now specify each of these cases and the system of linear constraints associated with that case. For **Rand-Move** to be applicable, we show that our chosen linear system is underdetermined. For the correctness proof, we show that all the assign and capacity constraints are maintained. Some advertiser constraints may get violated, but in the objective an advertiser i can pay at most B_i . We show indeed the loss in the objective is at most $\frac{1}{4}$ of the optimal objective value. Thus, we obtain a $\frac{3}{4}$ -approximation.

Let y denote the LP solution at this stage. There are three main cases to consider:

Case (i). There is a tree with two leaf advertiser nodes.

Case (ii). No tree contains two leaf advertisers, but there is a tree that contains one leaf advertiser.

Case (iii). No tree contains any leaf advertiser nodes.

Case (i). There is a tree with two leaf advertiser nodes. Consider the unique path P connecting the two leaf advertisers say i_0 and i_l . Suppose $P = \langle i_0, j_1, i_1, j_2, i_2, \dots, j_l, i_l \rangle$. Define a x variable for each edge in the path P that

takes values in $[0, 1]$. Consider the following system of linear constraints,

$$x_{i_{t-1},j_t} + x_{i_t,j_t} = y_{i_{t-1},j_t} + y_{i_t,j_t} \quad \forall t \in [1, l] \quad (10.2.1)$$

$$x_{i_t,j_t} b_{i_t,j_t} + x_{i_t,j_{t+1}} b_{i_t,j_{t+1}} = y_{i_t,j_t} b_{i_t,j_t} + y_{i_t,j_{t+1}} b_{i_t,j_{t+1}} \quad \forall t \in [1, l-1] \quad (10.2.2)$$

$$x \in [0, 1]^{2l} \quad (10.2.3)$$

We apply **Rand-Move** on the above linear system.

Lemma 10.2.3. *The linear system defined by Equations 10.2.1 and 10.2.2 is underdetermined, Assign constraints for all queries, Capacity constraints for all sets and Bidder constraints for all advertisers except the two leaf advertisers are maintained.*

Proof. The number of constraints of type 10.2.1 is l and the number of constraints of type 10.2.2 is $l-1$. However the number of variables is $2l$. Constraint 10.2.1 ensures all the assign constraints and hence all the capacity constraints are maintained. Constraint 10.2.2 ensures all the advertisers maintain their budget except probably the two leaf advertisers. ■

Case (ii). No tree contains two leaf advertisers, but there is a tree that contains one leaf advertiser. There are several subcases under it. We first consider four simple subcases.

Subcase (1): *There is a maximal path between two queries, where the two queries belong to the same set and the set-capacity constraint is non-tight.*

Since the path is maximal, the queries at the start and the end of the path are leaf queries and therefore have non-tight assign constraints. Non-tight naturally implies the fact that a constraint is not satisfied by equality. Suppose the maximal path is $P = \langle j_1, i_1, \dots, i_{l-1}, j_l \rangle$ and let the value of the edge-variables associated with this path be $\langle y_{i_1, j_1}, y_{i_1, j_2}, y_{i_2, j_2}, \dots, y_{i_{l-1}, j_{l-1}}, y_{i_{l-1}, j_l} \rangle$. These y values are treated as constants. Define variables $\langle x_{i_1, j_1}, x_{i_1, j_2}, x_{i_2, j_2}, \dots, x_{i_{l-1}, j_{l-1}}, x_{i_{l-1}, j_l} \rangle$ associated with these edges of P . Let S be the set containing the queries j_1 and j_l . Let the capacity of S be c . In the current solution, considering the rounded variables as well, let the total allocation of queries from the set S be $s + y_{i_1, j_1} + y_{i_{l-1}, j_l}$. That is, s is the sum of values of the queries in S other than j_1 and j_l . Consider the following system of linear constraints:

$$x_{i_1, j_1} \leq 1, x_{i_{l-1}, j_l} \leq 1 \quad (10.2.4)$$

$$x_{i_{t-1}, j_t} + x_{i_t, j_t} = y_{i_{t-1}, j_t} + y_{i_t, j_t} \quad \forall t \in [2, l-1] \quad (10.2.5)$$

$$x_{i_t, j_t} b_{i_t, j_t} + x_{i_t, j_{t+1}} b_{i_t, j_{t+1}} = y_{i_t, j_t} b_{i_t, j_t} + y_{i_t, j_{t+1}} b_{i_t, j_{t+1}} \quad \forall t \in [1, l-1] \quad (10.2.6)$$

$$x_{i_1, j_1} + x_{i_{l-1}, j_l} \leq s - c \quad (10.2.7)$$

$$x \in [0, 1]^{l+1} \quad (10.2.8)$$

We apply **Rand-Move** on the above linear system.

Lemma 10.2.4. *The linear system defined for Subcase 1 under Case (ii) is under-determined and **Rand-Move** on it maintains all the constraints, Assign, Bidder, Capacity, of LP-1.*

Proof. Note that, Constraint (10.2.7) is non-tight. In addition, Constraint (10.2.4) implies that the leaf queries have non-tight assignment constraint. Now, the number of variables associated with the above linear-system is $2(l - 1) = 2l - 2$ and the number of tightly satisfied linearly independent constraints are $2l - 3$. Hence, we can employ **Rand-Move**.

Constraint (10.2.5) implies the assignment constraint of the non-leaf queries are maintained. Constraint (10.2.6) implies the budget constraint of the non-leaf advertisers, and therefore all the advertisers considered by this system, are maintained. The capacities of all the sets in which non-leaf queries participates are automatically maintained. In addition, Constraint (10.2.7) implies the capacity constraint of the set involving the leaf queries are maintained as well. ■

Subcase (2): *There is a maximal path between two queries, where the two queries belong to two different sets and both set-capacity constraints are non-tight.*

This is almost similar to Subcase (1). Since the path is maximal, the queries at the start and the end of the path are leaf queries and therefore have non-tight assign constraints. Suppose the maximal path is $P =$

$\langle j_1, i_1, j_2, i_2, \dots, j_{l-1}, i_{l-1}, j_l \rangle$ and let the value of the edge-variables associated with this path be $\langle y_{i_1, j_1}, y_{i_1, j_2}, y_{i_2, j_2}, \dots, y_{i_{l-1}, j_{l-1}}, y_{i_{l-1}, j_l} \rangle$. We treat these values as constants here. Define variables $\langle x_{i_1, j_1}, x_{i_1, j_2}, x_{i_2, j_2}, \dots, x_{i_{l-1}, j_{l-1}}, x_{i_{l-1}, j_l} \rangle$ associated with these edges of P . The set constraint involving the query j_1 is non-tight and suppose the total sum of the values of the queries (rounded and not rounded) belonging to that set is $s + y_{i_1, j_1}$, while its capacity is c . Similarly, the set constraint involving the query j_l is non-tight and suppose the total sum of the values of the queries (rounded and not rounded) belonging to that set is $s' + y_{i_{l-1}, j_l}$, while its capacity is c' . Consider the following linear system.

$$x_{i_1, j_1} \leq 1, x_{i_{l-1}, j_l} \leq 1 \quad (10.2.9)$$

$$x_{i_{t-1}, j_t} + x_{i_t, j_t} = y_{i_{t-1}, j_t} + y_{i_t, j_t} \quad \forall t \in [2, l-1] \quad (10.2.10)$$

$$x_{i_t, j_t} b_{i_t, j_t} + x_{i_t, j_{t+1}} b_{i_t, j_{t+1}} = y_{i_t, j_t} b_{i_t, j_t} + y_{i_t, j_{t+1}} b_{i_t, j_{t+1}} \quad \forall t \in [1, l-1] \quad (10.2.11)$$

$$x_{i_1, j_1} \leq c - s \quad (10.2.12)$$

$$x_{i_{l-1}, j_l} \leq c' - s' \quad (10.2.13)$$

$$x \in [0, 1]^{l+1} \quad (10.2.14)$$

Note that changes in the linear system from Subcase 1. We apply **Rand-Move**

on the above linear system.

Lemma 10.2.5. *The linear system defined for Subcase 2 under Case (ii) is under-determined and **Rand-Move** on it maintains all the constraints, Assign, Bidder, Capacity, of LP-1.*

Proof. The constraints (10.2.12) and (10.2.13) are non-tight and so are 10.2.9. The number of variables associated with the above linear-system is $2(l-1) = 2l-2$ and the number of tightly satisfied linearly independent constraints are $2l-3$. Hence, we employ **Rand-Move**.

Constraint (10.2.10) implies the assignment constraint of the non-leaf queries are maintained. Constraint (10.2.11) implies the budget constraint of the non-leaf advertisers, and therefore all the advertisers considered by this system, are maintained. The constraints (10.2.12), (10.2.13) ensure that we won't violate the capacity constraint of the sets involving the leaf queries j_1 and j_l respectively. ■

Subcase (3): *There is a path (not necessarily maximal path) between two queries, where the two queries belong to the same set, the set-capacity constraint is tight but both the queries have non-tight assignment constraints.*

Suppose the path is $P = \langle j_1, i_1, j_2, i_2, \dots, j_{l-1}, i_{l-1}, j_l \rangle$ and let the value of the edge-variables associated with this path be $\langle y_{i_1, j_1}, y_{i_1, j_2}, y_{i_2, j_2}, \dots, y_{i_{l-1}, j_{l-1}}, y_{i_{l-1}, j_l} \rangle$. We treat these values as constants here. Define variables $\langle x_{i_1, j_1}, x_{i_1, j_2}, x_{i_2, j_2}, \dots, \dots \rangle$

$x_{i_{l-1},j_{l-1}}, x_{i_{l-1},j_l}$ associated with these edges of P . Let the total fractional assignment of query j_1 be $a_1 + y_{i_1,j_1} < 1$ and the total fractional assignment of query j_l be $a_2 + y_{i_{l-1},j_l} < 1$. Here we will apply the **Cycle Breaking** trick. We consider updates $\langle z_{i_1,j_1}, z_{i_1,j_2}, z_{i_2,j_2}, \dots, z_{i_{l-1},j_{l-1}}, z_{i_{l-1},j_l} \rangle$ such that

$$\text{R1. } z_{j_1,i_1} = -\beta$$

R2. If we are at an query node $j_t, t \in [1, l]$, then

$$z_{j_t,i_{t+1}} = -z_{i_t,j_t}$$

R3. If we are at an advertiser node $i_t, t \in [1, l]$, then

$$z_{i_t,j_t} = -\frac{b_{i_t,j_{t-1}}z_{j_{t-1},i_t}}{b_{i_t,j_t}}$$

The value of $\beta > 0$ is so chosen that ensures all the edge-variables remain in $[0, 1]$, $x_{i_{l-1},j_l} \leq 1 - a_2$, $x_{i_1,j_1} \leq 1 - a_1$. The entire update is a function of z_{j_1,i_1} . If $z_{j_1,i_1} \geq z_{j_l,i_{l-1}}$, then we apply the above update. Else, we consider the updates in the reverse direction, starting from the edge (j_l, i_{l-1}) .

Lemma 10.2.6. *The update vector \mathbf{z} is nontrivial and the update maintains all the constraints, Assign, Bidder, Capacity, of LP-1.*

Proof. Clearly, all the advertiser nodes maintain their budget due to rule R3. All the query nodes, except j_1 and j_l maintain their assign constraint. All the sets that do not contain j_1 or j_l thus maintain their capacity constraints. We start the update, by subtracting from the edge (j_1, i_1) if $z_{j_1,i_1} \geq z_{j_l,i_{l-1}}$. Therefore, the set that contains both j_1 and j_l satisfy its capacity reduced. Otherwise, we start subtracting

from the edge (j_l, i_{l-1}) , and again the set containing j_1 and j_l maintains the capacity constraint, since now $z_{j_1, i_1} < z_{j_l, i_{l-1}}$.

Since, $y_{i_1, j_1} < 1 - a_1$, $y_{i_{l-1}, j_l} < 1 - a_2$ and all the other variables are in $(0, 1)$, we can always find a $\beta > 0$ such that either $x_{i_1, j_1} = 1 - a_1$ or $x_{i_{l-1}, j_l} = 1 - a_2$, or one of them is rounded down to 0, or some other variable in the path is rounded to 0 or 1. ■

Subcase (4): *There is a maximal path with an advertiser on one side, an query in another with the set containing it being non-tight.*

Since we are considering a maximal path, the two end-points must be leaf nodes. Suppose the maximal path is $P = \langle j_1, i_1, j_2, i_2, \dots, j_{l-1}, i_{l-1} \rangle$ and let the value of the edge-variables associated with this path be $\langle y_{i_1, j_1}, y_{i_1, j_2}, y_{i_2, j_2}, \dots, y_{i_{l-1}, j_{l-1}} \rangle$. Let the set in which the query j_1 belongs be S and let it have a total assignment from the rounded and yet to be rounded variables equalling $s + y_{i_{l-1}, j_{l-1}}$. In addition, let its capacity be c . Consider the following linear system:

$$x_{i_1, j_1} \leq 1 \quad (10.2.15)$$

$$x_{i_{t-1}, j_t} + x_{i_t, j_t} = y_{i_{t-1}, j_t} + y_{i_t, j_t} \quad \forall t \in [2, l-1] \quad (10.2.16)$$

$$x_{i_t, j_t} b_{i_t, j_t} + x_{i_t, j_{t+1}} b_{i_t, j_{t+1}} = y_{i_t, j_t} b_{i_t, j_t} + y_{i_t, j_{t+1}} b_{i_t, j_{t+1}} \quad \forall t \in [1, l-2] \quad (10.2.17)$$

$$x_{i_1, j_1} \leq c - s \quad (10.2.18)$$

$$x \in [0, 1]^{l+1} \quad (10.2.19)$$

We apply **Rand-Move** on the above linear system.

Lemma 10.2.7. *The linear system defined for Subcase 4 under Case (ii) is under-determined and **Rand-Move** on it maintains the constraints, Assign, Capacity, of LP-1 as well as the Bidder constraint except possibly for the one leaf advertiser.*

Proof. The constraints 10.2.15 and 10.2.18 are non-tight. The number of tightly satisfied linear independent constraints is therefore at most $(l-2) + (l-2) = 2l-4$, whereas the number of variables is $2l-3$. Hence **Rand-Move** can be applied.

Constraint 10.2.16 and 10.2.15 ensure that all the assign constraints for the queries are maintained. Constraint 10.2.17 ensures the advertiser constraints are maintained for all the advertisers except possibly for i_{l-1} . Constraint 10.2.16 and 10.2.18 ensure that all the capacity constraints are maintained. ■

As long as Case (i) or (1-4) subcases of Case (ii) apply, we continue applying

them. Also at any time, if we find the linear-system composed of all the tightly satisfied linearly independent constraints of LP-1 for any tree becomes underdetermined, we apply **Rand-Move**. When neither subcase (1)-(4) or Case (i) apply, or **Rand-Move** can not be applied to the whole system, we have the following properties of the resulting forest structure:

1. (Case 1 does not apply): No two leaves are advertisers. So there can be at most one leaf that is an advertiser in any tree.
2. (Subcase 3 does not apply): No two queries that are non-tight and belong to the same set with tight capacity are in the same tree. Therefore, each tree can contain only one non-tight query from a tight set.
3. (**Rand-Move** does not apply to the LP_{BC} constraints for any single tree): The number of tightly satisfied linearly independent constraints from each tree is at least as many as the number of variables.
4. (Subcase 1 and 2 do not apply): No two leaves that are queries belong to the same set. Also among the leaves that are queries, at most one can belong to a set that has non-tight capacity constraint. In essential, there can be only one leaf that is an query and that belongs to a set that has non-tight capacity constraint.
5. (Subcase 4 does not apply): If there is a leaf node that is an advertiser in a tree, all other leaf nodes must be queries and must be part of sets that have

tight capacity constraint.

Subcase (5): *None of subcases (1)-(4) apply.*

This is the most nontrivial subcase. Denote the tree that contains a leaf advertiser node by T_1 and let i_1 be the advertiser that is a leaf. Consider a maximal path starting from i_1^1 . Since Case (i) or Subcases (1-4) do not apply, the other leaf end-point is an query, say $j_{l_1}^1$, that belongs to set S_1 and set S_1 has tight capacity constraint. Of course, the query $j_{l_1}^1$ has non-tight assign constraint since it is a leaf node. Let the path be as follows:

$$P_1 = \langle i_1^1, j_1^1, i_2^1, j_2^1, \dots, i_{l_1-1}^1, j_{l_1-1}^1, i_{l_1}^1, j_{l_1}^1 \rangle.$$

Since subcase 3 does not apply, tree T_1 does not contain any other non-tight query from S_1 . Now capacities are always integer and set S_1 has tight capacity constraint. This implies that set S_1 must contain another non-tight query and that non-tight query must belong to a different tree. Denote this second tree by T_2 and call this another non-tight query of S_1 by j_1^2 . If T_2 contains a leaf node that is an advertiser, consider the path from j_1^2 to that advertiser node. Say the path is,

$$P_2 = \langle j_1^2, i_1^2, j_2^2, \dots, i_{l_2-2}^2, j_{l_2-1}^2, i_{l_2-1}^2, j_{l_2}^2, i_{l_2}^2 \rangle.$$

Consider a combined path $\langle P_1, P_2 \rangle$.

$$\langle P_1, P_2 \rangle = \langle i_1^1, j_1^1, \dots, j_{l_1-1}^1, i_{l_1}^1, \underbrace{j_{l_1}^1, j_1^2, i_1^2, j_2^2, \dots, j_{l_2}^2, i_{l_2}^2} \rangle.$$

Essentially this combined path is thought of a single path ending at two leaf advertisers. We apply the rounding of Case (i) in this scenario with a slight change in handling the job nodes. We rewrite the linear system for convenience.

$$x_{i_{t-1}^1, j_t^1} + x_{i_t^1, j_t^1} = y_{i_{t-1}^1, j_t^1} + y_{i_t^1, j_t^1} \forall t \in [1, l_1 - 1] \quad (10.2.20)$$

$$x_{i_{t-1}^2, j_t^2} + x_{i_t^2, j_t^2} = y_{i_{t-1}^2, j_t^2} + y_{i_t^2, j_t^2} \forall t \in [2, l_2] \quad (10.2.21)$$

$$x_{i_{l_1}^1, j_{l_1}^1} \leq 1, x_{i_1^2, j_1^2} \leq 1 \quad (10.2.22)$$

$$x_{i_{l_1}^1, j_{l_1}^1} + x_{i_1^2, j_1^2} \leq y_{i_{l_1}^1, j_{l_1}^1} + y_{i_1^2, j_1^2} \quad (10.2.23)$$

$$x_{i_t^a, j_t^a} b_{i_t^a, j_t^a} + x_{i_t^a, j_{t+1}^a} b_{i_t^a, j_{t+1}^a} = y_{i_t^a, j_t^a} b_{i_t^a, j_t^a} + y_{i_t^a, j_{t+1}^a} b_{i_t^a, j_{t+1}^a}$$

$$\forall (t, a) \in ([2, l_1], 1) \cup ([1, l_2 - 1], 2) \quad (10.2.24)$$

$$x \in [0, 1]^{2l_1 + 2l_2 - 2} \quad (10.2.25)$$

We apply **Rand-Move** as usual. Note that, essentially we are assuming $j_{l_1}^1$ and j_1^2 as a single node while writing the constraint 10.2.23.

Lemma 10.2.8. *The linear system defined above is underdetermined and Assign constraints for all queries, advertiser constraints for all advertisers except i_1^1 and $i_{l_2}^2$ and Capacity constraints for all sets are maintained.*

Proof. Again the number of linearly independent tightly satisfied constraints are $(l_1 - 1) + (l_2 - 1) + 1 + (l_1 - 1) + (l_2 - 1) = 2l_1 + 2l_2 - 3$ from 10.2.20, 10.2.21, 10.2.23 and 10.2.24. The number of variables is $2l_1 + 2l_2 - 2$. Thus **Rand-Move** can be applied. From constraints 10.2.20, 10.2.21, 10.2.22 we get that all the assignment constraints and all the capacity constraints except for set S are satisfied. Constraint 10.2.23 ensures that the capacity constraint of the set S is maintained. Constraint 10.2.24 maintains all the advertiser constraints except for advertisers i_1^1 and $i_{l_2}^2$. ■

When, the above does not apply, then in T_2 there is no leaf node that is an advertiser. If there is a leaf node that is a query but the query is in a set that has non-tight capacity constraint, then we consider that path P'_2 (say) (we use the same symbols as in P_2 for P'_2 , but it is not to be confused with P_2 , since we are considering P'_2 when no such path like P_2 exists): $P'_2 = \langle j_1^2, i_1^2, j_2^2, \dots, i_{l_2}^2, j_{l_2}^2 \rangle$.

Consider a combined path $\langle P_1, P'_2 \rangle$ as before, that is we treat j_1^1 and j_1^2 as a single node while maintaining their total contribution to the set S . Note because of considering the combined path $\langle P_1, P'_2 \rangle$, this becomes identical to the subcase 4. So we apply the rounding on this combined path as in subcase 4. The correctness of this rounding step also follows from Lemma 10.2.7.

Otherwise, all the leaf nodes in T_2 are queries and the sets containing them have tight capacity constraint. Follow a maximal path from j_1^2 to one such leaf node, say j_l^2 , and let it belong to set S_2 . Denote the maximal path by P''_2 .

Since subcase 3 does not apply to T_2 , T_2 does not contain another non-tight

query from S_2 . But, the capacity of S_2 is integer and thus it must have another non-tight query. Call that query to be j_1^3 and denote the tree containing it to be T_3 . If T_3 happens to be same as T_1 , then consider the path P' in T_1 between j_1^3 and $j_{l_1}^1$. Now consider the combined path $\langle P', P_2'' \rangle$. In this combined path the two end-points belong to two non-tight queries from set S_2 that has tight capacity constraint. Thus, this is identical to subcase 3 and we apply the rounding of subcase 3. The correctness follows again from Lemma 10.2.6.

Otherwise, T_3 is a tree different from both T_1 and T_2 and we continue similarly from j_1^3 . Thus, if at any point of time, we reach a leaf node that is an advertiser or an query in a non-tight set, or an query in a tight-set but for which the another non-tight query belongs to a tree already visited, we can continue our rounding.

However, it may happen that a tight set contains more than two non-tight queries. In that case, it is possible to visit a tight set more than twice in our process. So suppose we are at tree T_g and while considering maximal path, $P_i = \langle j_1^g, i_1^g, j_2^g, \dots, i_{l_g-1}^g, j_{l_g}^g \rangle$, we get to $j_{l_g}^g$ that belongs to a set S^g that is already visited. That is, we have already seen two non-tight queries as end-points (one at the end of a maximal path and the other as the start of a maximal path in two consecutive trees) of two maximal paths say in T_h and T_{h+1} , $h + 1 < g$. Let the maximal paths that have been considered in trees $T_{h+1}, T_{h+2}, \dots, T_g$ be $P_{h+1}, P_{h+2}, \dots, P_g$. Consider the combined path $\langle P_{h+1}, P_{h+2}, \dots, P_g \rangle$ and note that in this combined path the two end-points belong to two non-tight queries from

set S_g that has tight capacity constraint. Thus we apply the rounding of subcase 4. Indeed it is not required to visit a non-tight query for the third time as an end-point of a maximal path. If at any time in this process, we visit a third non-tight query from a set with tight capacity constraint, we can write a combined path with two end-points containing non-tight queries from that set and apply rounding of subcase 3.

Otherwise, all the trees visited are different and we keep on continuing this process. Since the number of trees are at most $\min\{n, m\}$, this process must terminate in some tree T^t and at some leaf query node $j_{l_t}^t$ within a tight set S_t . Since S_t has at least two non-tight queries, the other non-tight query, say j , must belong to some tree $T^{t'}$, $t' < t$. Considering a path from j to $j_{l_t}^t$ and then following the maximal paths in $T^{t'+1}, T^{t'+2}, \dots, T^t$, we again get a combined path on which we can apply rounding of subcase 3.

Case (iii). No tree contains any leaf advertiser nodes. This case is similar to Case (ii). We start with a leaf query, possibly with a leaf query that is in a non-tight set if one exists, and obtain a combined path on which we can apply one of Subcases (1)-(4).

This completes the description of the rounding method. At every step, the entire rounding procedure takes $poly(n, m)$ time and at each step we either make a constraint tight or round a variable. Thus we are guaranteed to complete rounding all the variables to integers in polynomial number of steps.

From the above discussion and Lemma 10.2.3-10.2.8, we get the following,

Lemma 10.2.9. *The rounding procedure maintains all the assign and the capacity constraints. An advertiser node maintains the advertiser constraint as long as in the current fractional solution, it is connected to two or more queries with nonzero fractional values.*

Now, we need to prove that our expected approximation ratio is $\frac{4 - \max_i \frac{b_{i,max}}{B_i}}{4}$, where $b_{i,max} = \max_j b_{i,j}$. We can always assume $b_{i,max} \leq B_i$ without loss of generality for all i , we get a $3/4$ approximation. If bids are small, that is $\max_i \frac{b_{i,max}}{B_i} \leq \varepsilon$, then we get a $(4 - \varepsilon)/4$ approximation.

Theorem [10.2.1]. *Given a fractional optimal solution for LP1, we can obtain an integral solution for AdCell with capacity constraints on disjoint sets that obtains at least a profit of $\frac{4 - \max_i \frac{b_{i,max}}{B_i}}{4}$ of the profit obtained by optimal fractional allocation and maintains all the capacity constraints exactly.*

Proof. Let P_i^0 denote the payment made by advertiser i as assigned by LP1. In our rounding process, when an edge-variable gets rounded to 0 or 1, it is removed permanently or assigned permanently. The forest structure that we consider always contains only the fractional edge-variables. If the advertiser i never has degree 1 in the forest, then by our rounding procedure its final payment is same as P_i^0 . Therefore, suppose at some stage s , advertiser i becomes a leaf node and let a be the so far rounded payment on i and let b be the unique query assigned to advertiser i with fractional assignment p and bid d . Note that, all a, b, p, d are random variables.

If P_i^s denote the total payment (fractional and integral) done by advertiser i at the end of the s th iteration, then we have

$$P_i^s = a + dp = P_i^0$$

Once an advertiser becomes a leaf node, it only takes part in **Rand-Move**. Let $P_i^{s+1}, P_i^{s+2}, \dots, P_i^t$ denote the payment rounded on advertiser i at the end of the iterations $s + 1, s + 2, \dots, t$. Assume t is the last iteration. Then we have from property **[P1]** of **Rand-Move** that

$$\mathbb{E} \left[P_i^g | P_i^{g-1} = a + dp_{g-1} \right] = a + dp_{g-1}$$

for $g > s$. Thus

$$\begin{aligned} \mathbb{E} [P_i^g] &= \int_x \mathbb{E} \left[P_i^g | P_i^{g-1} = a + dx \right] \Pr \left[P_i^{g-1} = a + dx \right] = \\ &\int_x a + dx \Pr \left[P_i^{g-1} = a + dx \right] = \mathbb{E} \left[P_i^{g-1} \right]. \end{aligned}$$

Hence we have

$$\mathbb{E} [P_i^t] = \mathbb{E} [P_i^{t-1}] = \dots = \mathbb{E} [P_i^s] = a + dp = P_i^0$$

Then it directly follows from the above,

With probability $1 - p$ the rounded payment on advertiser i is a and with probability p the rounded payment is $a + d$, since $\mathbb{E} [P_i^t] = a \Pr [\text{edge } (i, b) \text{ is rounded to } 0] + (a + d) \Pr [\text{edge } (i, b) \text{ is rounded to } 1]$.

Thus the final expected profit from advertiser i is $(1 - p) \min \{B_i, a\} + p \min \{B_i, a + d\}$. The profit obtained from i in the optimal LP solution is $\min \{B_i, a + dp\}$. Therefore, by the linearity of expectation, the expected approximation ratio is the maximum possible value of

$$\frac{(1 - p) \min \{B_i, a\} + p \min \{B_i, a + d\}}{\min \{B_i, a + dp\}}.$$

This part of the proof is similar to the analysis of Theorem 1 of [125]. Let $b_{i,max} = \max_j b_{i,j}$. We can assume without loss of generality that $b_{i,max} \leq B_i$ for all i . It is easy to see that if $a > B_i$ or $a + d < B_i$, then the above approximation ratio is 1. Hence assume, $a < B_i < a + d$. We thus have the approximation ratio to be

$$r = \frac{a(1 - p) + pB_i}{\min \{B_i, a + dp\}}$$

Now considering the two cases, $B_i \leq / > a + dp$, we get the following result:

$$\frac{(1 - p) \min \{B_i, a\} + p \min \{B_i, a + d\}}{\min \{B_i, a + dp\}} \leq \frac{4 - \max_i \frac{b_{i,max}}{B_i}}{4}$$

Since we can assume without loss of generality $b_{i,max} \leq B_i$ for all i , we get a $3/4$ approximation. If bids are small, that is $\max_i \frac{b_{i,max}}{B_i} \leq \varepsilon$, then we get a $(4 - \varepsilon)/4$ approximation. ■

Covering with Hard Capacities

In this chapter, we consider classical covering problems, such as vertex cover and set cover with hard capacities. In hard-capacitated vertex (set) cover problem each vertex (set) has a capacity and the number of edges (elements) it covers cannot exceed that given capacity. In addition, the available copies of each vertex (set) is bounded. The problem was first studied by Chuzhoy and Naor [46], where they gave a 3-approximation algorithm for the vertex cover problem with hard capacities on *unweighted simple* graphs. This result was later improved to a 2-approximation by Gandhi et al. [63]. In contrast, for weighted graphs, the problem is as hard as the set cover problem. These capacitated covering problems belong to the general paradigm of submodular covering. Using a result by Wolsey [138], logarithmic approximation algorithms can be derived for them.

The set cover hardness precludes the possibility of a constant factor approximation for the hard-capacitated vertex cover problem on weighted graphs. However, it was not known whether a better than logarithmic approximation is possible on

unweighted but *multigraphs*, i.e., graphs that may contain parallel edges. Neither the approach of Chuzhoy and Naor, nor the follow-up work of Gandhi et al. can handle the case of multigraphs. In fact, achieving a constant factor approximation for hard-capacitated vertex cover problem on unweighted multigraphs was posed as an open question in Chuzhoy and Naor’s work.

In this chapter, we describe the first constant factor approximation algorithm for the vertex cover problem with hard capacities on unweighted multigraphs. We also show an $O(f)$ -approximation algorithm for the unweighted set cover problem with hard capacities, where an element belongs to at most $f \in \mathbb{N}$ sets. Vertex cover is a special case of the above set cover problem where $f = 2$. A crucial ingredient of our method is a reduction to the multi-set multi-cover problem for which we obtain a new approximation algorithm.

11.1 Capacitated Vertex Cover and Set Cover Problem

We are given a ground set of elements $\mathcal{U} = \{a_1, a_2, \dots, a_n\}$ and a collection of subsets of \mathcal{U} , $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$. Each set $S \in \mathcal{S}$ has a weight $\tilde{w}(S) \in \mathbb{R}^+$ and a positive integral capacity $k(S) \in \mathbb{N}$. In addition, for each set $S \in \mathcal{S}$, at most $m(S) \in \mathbb{N}$ copies of it are available. A solution for the capacitated covering problem contains $x(S)$ copies of set $S \in \mathcal{S}$, where $x(S) \in \{0, 1, 2, \dots, m(S)\}$ such that there is an assignment of at most $x(S)k(S)$ elements to set S and all elements are covered by the assignment. The goal is to pick a solution that minimizes

$$\sum_{S \in \mathcal{S}} \tilde{w}(S)x(S).$$

A special case of the capacitated set cover problems is the *vertex cover problem with hard capacities*. In this version of the problem, we are given a graph $G = (V, E)$ with n vertices and m edges, a weight function $\tilde{w} : V \rightarrow \mathbb{R}^+$, a capacity of $k(v) \in \mathbb{N}$ and the number of available copies $m(v) \in \mathbb{N}$ for each vertex $v \in V$. The goal again is to pick $x(v) \in [0, m(v)]$ copies of v and orient every edge to one of its end-points such that at most $k(v)x(v)$ edges are oriented towards v and $\sum_{v \in V} \tilde{w}(v)x(v)$ is minimized.

Capacitated problems come in two flavors: soft-capacitated in which each set has an unbounded number of copies and hard-capacitated where the number of available copies of each set is restricted. Soft-capacitated problems appear easier than their hard-capacitated counterparts and have been studied under the context of vertex cover [64, 70], facility location [43, 103, 104], k-median [25] etc.

Hard-capacitated vertex cover and set cover problems were first studied by Chuzhoy and Naor in [46]. They show when the weight function is arbitrary, the vertex cover problem with hard capacities is as hard as the set cover problem. Therefore, assuming $P \neq NP$, nothing better than a $\log n$ -approximation can be obtained [59, 101], whereas for the uncapacitated vertex cover problem, the best known approximation ratio is $2 - \frac{\log \log n}{2 \log n}$ [23, 78]. A 2-approximation is also known for the case of soft capacities [70]. On the other hand, the hard-capacitated set cover problem belongs to the general paradigm of submodular covering for

which Wolsey gave a $(1 + \log S_{max})$ -approximation, where S_{max} is the maximum number of elements covered by any set [138]. When each element belongs to at most f sets, there exists a simple f -approximation algorithm for uncapacitated set cover problem. However, in presence of hard capacities, nothing better than the general $\log n$ bound is known.

Interestingly, while $\log n$ hardness seems to apply to the vertex cover problem in presence of hard capacities, if we consider simple unweighted graphs, much better results can be achieved. Chuzhoy and Naor in [46] gave a 3-approximation algorithm for hard-capacitated vertex cover problem on unweighted simple graphs, which was later improved to a 2-approximation by Gandhi et al. [63]. Vertex cover is a special case of set cover problem where $f = 2$. This naturally raises the question whether it is possible to obtain an $O(f)$ approximation for unweighted set cover problem with hard capacities, where each element belongs to at most f sets. The approaches of [46, 63] do not extend to case when $f > 2$. In fact, the results of [46, 63] only hold for *simple* graphs. *Obtaining a constant factor approximation algorithm for the hard-capacitated vertex cover problem for unweighted multigraphs was posed as an open question in [46]*. In this paper, we resolve that question, and extending our approach, we also obtain an $O(f)$ -approximation for the unweighted set cover problem with hard capacities for arbitrary values of f .

Capacitated problems arise naturally in applications where there are resource constraints. Guha et al. [70] describes a research project undertaken by a bio-

technology company Glycodata for drug design and improvement of known drugs, where the underlying problem was exactly a capacitated vertex cover instance. Motivated by crew-scheduling [56, 117] and by the fact that servers often have limits on how many jobs can be assigned to them, [41, 88, 137, 139, 144] study the classical unrelated parallel machine scheduling [96] and generalized assignment problem [122] with hard capacity bounds on machines. Recently, [1] considers a capacitated version of the budgeted allocation problem [35, 125] that arises in mobile advertising scenario. Capacitated facility location and k -median problems have been an active area of research [22, 25, 43, 43, 47, 66, 102–104, 113, 143] and frequently appear in applications involving placement of warehouses [48], caches on the web [18, 97, 107] and as a subroutine in several network design protocols [71, 72, 86]. Non-metric capacitated facility location problem is a generalization of the hard-capacitated set cover problem for which Bar-Ilan et al. [22] gave an $O(\log n + \log m)$ -approximation. In this problem, there are m facilities and n clients; there is a cost associated for opening each facility and each client connects to one of the open facility paying a connection cost while the number of clients that can be assigned to an open facility remains bounded by its capacity. When, the connection costs are either 0 or ∞ , we get the set cover problem with hard capacities.

In several set cover instances that occur in practice, an element only belongs to few sets. In such settings, a f -approximation for set cover is much desirable.

In [88], Khuller et al. study a scheduling problem motivated by issues of energy savings in data centers. In data centers, each data is replicated a *small* number of times (generally 3 to 4 times). Each job has a data requirement and thus can be executed only on 3 to 4 machines. These data centers are provisioned to handle high work loads during peak demand periods. However, since the work load on modern cloud computing platform is very cyclical with infrequent peaks and deep valleys, much energy can be saved, if the machines can be shut down selectively during the low work period. This observation led to the following question. Suppose, there are m machines \mathcal{M} and n jobs \mathcal{J} , where each job can be processed on a subset of machines and its processing time may be machine-dependent. Each machine consumes a fixed amount of energy to be activated and the goal is to minimize the total cost of activation while maintaining a bound on the maximum load (sum of the processing times of the jobs assigned to a machine). When processing times are arbitrary, no approximation is possible without violating the maximum load [96, 122]. In [88], Khuller et al. provide a $(\ln n + 1)$ approximation algorithm that violates the maximum load by a factor of 2. On the other hand, if each job has some fixed processing time p , then, as we know, each job can be scheduled only on f ($= 3$ or 4) machines; we obtain the hard-capacitated set cover problem with elements belonging to at most f sets.

Our algorithms for the hard-capacitated versions of both vertex cover (Section 11.2) and set cover (Section 11.4) are based on rounding linear programming (LP)

relaxations. We utilize the LP-structure to decompose the problem into two simpler instances. Rounding on each of these instances exploits their specific nature. The approach of [46, 63] cannot handle the vertex cover problem with hard capacities on multigraphs. Their algorithms are randomized in nature and presence of parallel edges makes some of the random variables *positively correlated*. This hinders the application of required concentration inequalities that are needed to obtain the desired result with their procedure. For the same reason, their algorithm does not extend to set cover problem for $f > 2$. In this paper, we are able to overcome that barrier. In the process, we also develop a new algorithm for multi-set multi-cover problem (Section 11.3),

Related Works

Set cover and vertex cover are probably two most well-studied NP-hard problems and the reader is referred to the surveys in [42, 79, 135]. Both of their hard-capacitated versions are examples of more general submodular covering problem for which Wolsey gave a logarithmic approximation algorithm [138]. Hence [138] gave the first non-trivial approximation algorithm results for both the set cover and vertex cover problem with hard capacities. See [22] for a generalization of Wolsey's approach.

A problem closely related to hard-capacitated set cover problem is the capacitated facility location problem. When the connection cost forms a metric, the

problem is known as metric capacitated facility location problem. For this problem, Pál et al. [113] gave a $(9 + \varepsilon)$ -approximation using local search which was later improved to $(8 + \varepsilon)$ by [102] and then to $(6 + \varepsilon)$ by [66, 143]. In contrast, no constant factor approximation algorithm is known for metric k -median problem in presence of capacities, even when the capacities are soft. Bartal et al. [25] gave a bicriteria approximation algorithm for soft-capacities where the number of facilities opened is slightly more than k . Chuzhoy and Rabani [45] provided a different bi-criteria approximation for the hard-capacitated case where capacities are allowed to be violated by a constant factor.

A variation of set cover problem is the multi-set multi-cover problem. In this problem, each set is in fact a multi-set and each element has a demand. Multi-set multi-cover problem is also an instance of submodular covering problem, and therefore a $\log n$ -approximation result is known for it [89, 138]. Our results in this paper for hard-capacitated vertex cover and set cover problem are based on providing a new algorithm for the multi-set multi-cover problem.

11.2 Vertex Cover on Multigraphs with Hard Capacities

In this section, we describe a constant factor approximation algorithm for the vertex cover problem with hard capacities on unweighted multigraphs. We assume each vertex has unit multiplicity ($m(v) = 1, \forall v \in V(G)$), that is, for each vertex, exactly one copy is available. The more general case, where each vertex may have

arbitrary number of copies is handled in Section 11.4. Our starting point is the following linear programming relaxation.

$$\text{minimize } \sum_{v \in V} x(v) \quad (\text{LP}_{\text{VC}})$$

subject to

$$y(e, u) + y(e, v) = 1 \quad \forall e = (u, v) \in E, \quad (11.2.1)$$

$$y(e, v) \leq x(v), y(e, u) \leq x(u) \quad \forall e = (u, v) \in E, \quad (11.2.2)$$

$$\sum_{e=(u,v)} y(e, v) \leq k(v)x(v) \quad \forall v \in V, \quad (11.2.3)$$

$$0 \leq x(v), y(e, v), y(e, u) \leq 1 \quad \forall v \in V, \forall e = (u, v) \in E. \quad (11.2.4)$$

Here $x(v)$ is an indicator variable, which is 1 if vertex v is chosen and 0 otherwise. Variables $y(e, u)$ and $y(e, v)$ are associated with edge $e = (u, v)$. $y(e, u) = 1$ ($y(e, v) = 1$) indicates edge e is assigned to vertex u (v). Constraints (11.2.1) ensure each edge is covered by at least one of its end-vertices. Constraints (11.2.2) imply an edge cannot be covered by a vertex v , if v is not chosen in the solution. The total number of edges covered by a vertex v is at most $k(v)$ if v is chosen and 0 otherwise (constraints (11.2.3)). We relax the variables $x(v), y(e, v)$ to take value in $[0, 1]$ in order to obtain the desired LP-relaxation. The optimal solution of LP_{VC} denoted by $\text{LP}_{\text{VC}}(\text{OPT})$ clearly is a lower bound on the actual optimal cost OPT .

Lemma 11.2.1. $LP_{VC}(OPT) \leq OPT$, where OPT denotes the optimal cost for the hard-capacitated vertex cover problem on unweighted multigraphs with unit multiplicities.

11.2.1 Rounding Algorithm

Let (x^*, y^*) denote an optimal fractional solution of LP_{VC} . We create a bipartite graph $H = (A, B, E(H))$, where A represents the vertices of G , B represents the edges of G ¹ and the links $E(H)$ correspond to the (e, v) variables $e \in B, v \in A$ with non-zero y^* value². Each $v \in A(H)$ is assigned a weight of $x^*(v)$. Each link (e, v) is assigned a weight of $y^*(e, v)$. We now modify the link weights in a suitable manner to decompose the link sets of H into two graphs H_1 and H_2 . The special properties of H_1 and H_2 make the rounding process relatively simpler.

- H_1 is a forest.
- In H_2 , if $(e, v) \in E(H_2)$, then weight of link (e, v) is equal to the weight of v .

A moment's reflection shows the usefulness of such a property, essentially, in H_2 , we can ignore the hard capacity constraints altogether.

We now describe the decomposition procedure and the rounding steps in each of H_1 and H_2 .

¹We often refer a vertex in $B(H)$ by edge-vertex to indicate it belongs to $E(G)$.

²In order to avoid confusion between edges of G with edges of H , we refer to $E(H)$ by links.

Step 1. Constructing H_1 and H_2 . H_1 and H_2 contain the same set of vertices as H . We start by setting $E(H_1) = E(H)$ and $E(H_2) = \emptyset$. We remove all links and vertices from H_1 with weight 0. Further, for any link (e, v) , if $y^*(e, v) = x^*(v)$, we move (e, v) from H_1 to H_2 . Therefore, after this initial stage, for all links $(e, v) \in E(H_1)$, $y^*(e, v) < x^*(v)$ and for all links $(e', v') \in E(H_2)$, $y^*(e', v') = x^*(v')$.

While there is a cycle $\mathbf{C} = (v_1, e_1, v_2, e_2, \dots, v_l, e_l, v_{l+1} = v_1)$ in H_1 , we select an $\varepsilon > 0$, and set $y^*(v_i, e_i) = y^*(v_i, e_i) + \varepsilon$ and $y^*(v_{i+1}, e_i) = y^*(v_{i+1}, e_i) - \varepsilon$ for $i = 1, 2, \dots, l$. The choice of ε is such that after modification, all link weights satisfy constraints (11.2.2) and (11.2.4), and at least one of them is tight. That is, for at least one $e_j \in \mathbf{C}$, either $y^*(v_j, e_j) = x^*(v_j)$ or $y^*(v_{j+1}, e_j) = x^*(v_{j+1})$ or $y^*(v_j, e_j) = 0$ or $y^*(v_{j+1}, e_j) = 0$. We can always find such an $\varepsilon > 0$. New y^* is a feasible solution for LP_{VC} . We move all links (e', v') that satisfy $y^*(e', v') = x^*(v')$ to H_2 , and drop any link whose weight becomes 0. Any isolated node is dropped as well. Choice of ε guarantees that at least one link from H_1 is either dropped or moved; so the cycle is broken.

Proceeding in this fashion, after at most $|E(H_1)|$ steps, we get H_1 and H_2 such that

- H_1 is a forest and for each node $v \in A(H_1)$ and link $(e, v) \in E(H_1)$,
 $y^*(e, v) < x^*(v)$.
- In H_2 , for each node $v \in A(H_2)$ and link $(e, v) \in E(H_2)$, $y^*(e, v) = x^*(v)$.

Since, \mathbf{x}^* does not change, objective function value of LP_{VC} remains unchanged in the process.

Step 2. Rounding on H_2 .

We discard all isolated vertices from H_2 . Let $\eta \geq 2$ be the desired approximation factor. We select all vertices in $A(H_2)$ with value of x^* at least $\frac{1}{\eta}$. Let us denote the chosen vertices by \mathcal{D} . Then,

$$\mathcal{D} = \{v \mid v \in A(H_2), x^*(v) \geq \frac{1}{\eta}\}.$$

For every edge-vertex $e = (u, v) \in B(H_2)$, if v (or u) is in \mathcal{D} , and $(e, v) \in E(H_2)$ (or $(e, u) \in E(H_2)$), then we set $y^*(e, v) = 1$ (or $y^*(e, u) = 1$). That is, we assign e to v , if the link (e, v) is in $E(H_2)$ and v is in \mathcal{D} , else if $u \in \mathcal{D}$ and $(e, u) \in E(H_2)$, the edge e is assigned to u .

Observation 11.2.2. From constraints (11.2.3), $\sum_{e=(u,v)} y(e, v) \leq x(v)k(v)$. Therefore, $\sum_{e=(u,v)} \frac{y(e,v)}{x(v)} \leq k(v)$, and hence in H_2 , after the assignment of edges to vertices in \mathcal{D} , all vertices maintain their capacity.

In fact, in H_2 , capacity constraints become irrelevant. *Whenever, we decide to pick a vertex in $A(H_2)$, we can immediately cover all the links in $E(H_2)$ incident on it.*

All edges with both links in $E(H_2)$ get covered at this stage. In addition, if $e \in B(H_2)$ has only one link $(e, v) \in E(H_2)$, but $x^*(v) = y^*(e, v) \geq \frac{1}{\eta}$, then since

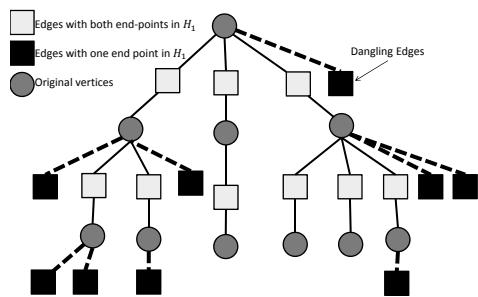


Fig 1a. Structure of H_1 , dangling edges are colored black and connected by dashed lines, edges with both end-points in H_1 are colored white and connected by solid lines.

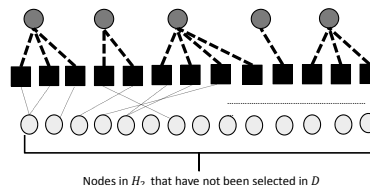


Fig 1b. Structure of H_1 after the edges with two end points in H_1 have been assigned.

$v \in \mathcal{D}$, e gets covered. Therefore, the uncovered edges after this step either have no link in $E(H_2)$ or are fractionally covered to an extent less than $\frac{1}{\eta}$ in H_2 .

Step 3. Rounding on H_1 .

H_1 is a forest; edge-vertices in H_1 either have both or one link in $E(H_1)$. Remember that, while the vertices of H_1 and H_2 may overlap, the link sets are disjoint. Edge-vertices in $B(H_1)$ with only one link in H_1 are called *dangling* edges. We root H_1 arbitrarily at some node of $A(H_1)$. This naturally defines a parent-child relationship. Figure (1a) depicts the structure of H_1 . Dangling edges are shown by dashed lines.

Step 3a. Rounding edges with both links in H_1 .

Algorithm (1) describes the procedure to assign edge-vertices that have both links in $E(H_1)$.

We first select a collection of \mathcal{D}' vertices from $A(H_1) \setminus \mathcal{D}$ with x^* value at least $\frac{1}{\eta}$. Edge-vertices in $B(H_1)$ that have their children in \mathcal{D}' get assigned to their children. In Algorithm (1), for each vertex $v \in A(H_1)$, we use $T(v)$ to denote the set of children edge-vertices that are not assigned in step (4). We select $t(v) =$

Algorithm 1 Assigning edges with two links in H_1

- 1: **let** $\mathcal{D}' = \{v \in A(H_1) \mid x^*(v) \geq \frac{1}{\eta}\}$, **select** all the vertices in \mathcal{D}' .
 - 2: **for each** edge-vertex e with two links in H_1 **do**
 - 3: **if** the child vertex of e is selected in \mathcal{D}' **then**
 - 4: **assign** e to the selected child vertex.
 - 5: **end if**
 - 6: **end for**
 - 7: **let** $T(v)$ denote the set of unassigned edge-vertices incident on $v \in A(H_1)$ with both links in H_1 .
 - 8: **select** any $t(v) = \lceil \sum_{e=(u,v) \in T(v)} y^*(e, u) \rceil$ vertices from the children of the edge-vertices in $T(v)$, and **assign** the corresponding $t(v)$ edge-vertices in $T(v)$ to these selected children vertices. If v' is a newly selected vertex in this step and there are edges that have links incident on v' in $E(H_2)$, then assign those edges to v' as well.
 - 9: **assign** the remaining edge-vertices from $T(v)$ to v .
-

$\lceil \sum_{e=(u,v) \in T(v)} y^*(e, u) \rceil$ vertices from the children of the edge-vertices in $T(v)$.

We assign the corresponding $t(v)$ edge-vertices in $T(v)$ to these newly selected children vertices. Rest of the edges in $T(v)$ are assigned to v .

Step 3b. *Rounding dangling edges, i.e., with one link in H_1 .*

After Algorithm 1 finishes, let $L(v)$ denote the set of unassigned dangling edge-vertices connected to v , and let $l(v) = \sum_{e=(u,v), e \in L(v)} y^*(e, u)$. $L(v)$ are the leaf edge-vertices of H_1 . We first have a lemma that shows after the edge-assignment in Step 2 and 3a, we still can safely assign at least $|L(v)| - \lceil l(v) \rceil$ edges from $L(v)$ to v without violating its capacity.

Lemma 11.2.3. *Each vertex $v \in A(H_1)$ can be assigned $|L(v)| - \lceil l(v) \rceil$ leaf edge-vertices without violating its capacity.*

Proof. Suppose, v belongs to H_2 as well and is selected in H_2 . Then,

$$\sum_{(e,v) \in E(H_2)} x_v^* + \sum_{(e,v) \in H_1} y^*(e,v) \leq k(v)x^*(v).$$

Thus,

$$\sum_{(e,v) \in E(H_1)} y^*(e,v) = (k(v) - |\{(e,v) \in E(H_2), \forall e\}|) x^*(v).$$

Now, $(k(v) - |\{(e,v) \in E(H_2), \forall e\}|)$ is an integer, and we denote it by $k'(v)$.

Let us assume $v \in \mathcal{D}'$ first. Let the fractional value of the link connecting v to its parent edge-vertex in H_1 be b . The capacity of v is $k'(v) \geq \lceil b + |\mathbf{T}(v)| - t(v) + |\mathbf{L}(v)| - l(v) \rceil$. The number of edges assigned to v is $1 + |\mathbf{T}(v)| - \lceil t(v) \rceil + |\mathbf{L}(v)| - \lceil l(v) \rceil$.

If $t(v)$ and $l(v)$ are both integers, then clearly $1 + |\mathbf{T}(v)| - \lceil t(v) \rceil + |\mathbf{L}(v)| - \lceil l(v) \rceil < k'(v)$.

If $t(v)$ is an integer, but $l(v)$ is not an integer, then $k'(v) \geq |\mathbf{T}(v)| - t(v) + |\mathbf{L}(v)| - \lceil l(v) \rceil$ which is again at least the number of edges assigned to v . Similarly, the capacity constraint holds when $l(v)$ is an integer, but $t(v)$ is not.

If $l(v)$ and $t(v)$ are both non-integers, then $\lceil t(v) \rceil + \lceil l(v) \rceil > \lfloor l(v) + t(v) \rfloor + 1$. Capacity $k'(v) \geq |\mathbf{T}(v)| + |\mathbf{L}(v)| - \lfloor t(v) + l(v) \rfloor$, and the number of edges assigned to v is at most $1 + |\mathbf{T}(v)| - \lceil t(v) \rceil + |\mathbf{L}(v)| - \lceil l(v) \rceil \leq |\mathbf{T}(v)| + |\mathbf{L}(v)| - \lfloor t(v) + l(v) \rfloor$. Thus, in all cases, the capacity constraint of v is maintained.

If $v \notin \mathcal{D}'$, then $|\mathbf{L}(v)| = 0$, because otherwise leaf edge-vertices are assigned

to v at least to an extent of $1 - \frac{1}{\eta} > \frac{1}{\eta}$. Therefore, $x^*(v) > \frac{1}{\eta}$, leading to a contradiction. Hence, $|L(v)|$ must be 0. In this case, at most one parent edge-vertex can be assigned to v , hence its capacity constraint is maintained. ■

The edge-vertices in $L(v)$ are leaves of H_1 , they are connected to v and have their other link in $E(H_2)$. We first pick *one vertex* from $A(H_2)$ such that it covers at least one edge from $L(v)$. Let us denote this vertex by $h2(v)$ and let it cover $p2(v) \geq 1$ parallel edges $(v, h2(v))$. If $l(v) \leq p2(v)$, then following Lemma 11.2.3, rest of the edge-vertices of $L(v)$ can be assigned to v , and we exactly do that.

Else, $l(v) > p2(v)$. Let $R(v)$ denote the vertices of $A(H_2) \setminus h2(v)$ that are end-points of edges in $L(v)$. If we pick enough vertices from $R(v)$ such that they cover at least $l'(v) = l(v) - p2(v) + 1$ leaf-edges, then again from Lemma 11.2.3, rest of the edges from $L(v)$ can be assigned to v . We let $l'(v) = 0$, if $l(v) \leq p2(v)$.

We scale up all the x^* variables of $\bigcup_{v \in A(H_1)} R(v)$ by a factor of $\frac{1}{1-\frac{1}{\eta}}$. We also scale up the corresponding y^* link variables by a factor of $\frac{1}{1-\frac{1}{\eta}}$. Let (\bar{x}, \bar{y}) denote this scaled up variables. Then,

$$\begin{aligned} \sum_{\substack{e=(u,v) \in \\ L(v) \setminus (v, h2(v))}} \bar{y}(e, u) &= \frac{(l(v) - p2(v)x^*(h2(v)))}{\left(1 - \frac{1}{\eta}\right)} \\ &\geq \frac{\left(l(v) - \frac{p2(v)}{\eta}\right)}{\left(1 - \frac{1}{\eta}\right)} \\ &> l(v) - p2(v) + 1 = l'(v), \end{aligned} \quad (11.2.5)$$

where the last inequality follows from the fact that $l(v) > p2(v) \geq 1$. We now have the following multi-set multi-cover problem (MM).

For each $v \in A(H_1)$ with $l'(v) > 0$, we create an element $a(v)$. For each vertex $u \in \bigcup_{v \in A(H_1)} R(v)$, we create a multi-set $S(u)$. If there are $d(v, u)$ leaf edge-vertices in $L(v) \setminus (v, h2(v))$ incident upon u , then we include $a(v)$ in $S(u)$, $d(v, u)$ times. Each element $a(v)$ has a requirement of $r(a(v)) = \lfloor l'(v) \rfloor$. The goal is to pick minimum number of sets such that each element $a(v)$ is covered $\lfloor l'(v) \rfloor$ times counting multiplicities.

Note that, since the original graph is multigraph, $d(v, u)$ can be greater than 1.

Lemma 11.2.4. *If we set $z(S(u)) = \bar{x}_u, \forall u \in \bigcup_{v \in A(H_1)} R(v)$, then \mathbf{z} is a feasible fractional solution for the above stated multi-set multi-cover problem*

Proof. Consider any element $a(v)$. The total fractional coverage of element $a(v)$ from \mathbf{z} is

$$\begin{aligned}
 \sum_{S(u) \ni a(v)} d(v, u) z(S(u)) &= \sum_{u \in \bigcup_{v \in A(H_1)} R(v)} \bar{x}_u \\
 &= \sum_{\substack{e=(u,v) \\ L(v) \setminus (v, h2(v))}} \bar{y}(e, u) \\
 &> l(v) - p2(v) + 1 \quad (\text{from Equation 11.2.5}) \\
 &= l'(v) > r(a(v)),
 \end{aligned}$$

■

If, s is the number of vertices $v \in A(H_1)$ with $l'(v) > 0$, then we can obtain an integral solution with a cost of $O(\log s) \sum_{u \in \bigcup_{v \in A(H_1)} R(v)} \bar{x}_u$. This $\log s$ factor “in a sense” is unavoidable because of the hardness of approximating set cover. We instead, obtain an algorithm where the total number of sets picked is close to $s + \sum_{u \in \bigcup_{v \in A(H_1)} R(v)} \bar{x}_u$. In Section 11.3, we prove the following theorem.

Theorem 11.3.5. *Given any feasible fractional solution \bar{x} with cost F for multi-set multi-cover problem with N elements, there is a polynomial time rounding algorithm that rounds the fractional solution to a feasible integral solution with cost at most $21N + 32F$.*

The algorithm for assigning the leaf edge-vertices in $L(v)$ is given in Algorithm (2).

Since, each vertex $v \in A(H_1)$ covers at most $|L(v)| - \lceil l(v) \rceil$ leaf edge-vertices, by Lemma 11.2.3 the capacity of all the vertices in H_1 are maintained. We now proceed to analyze the cost.

Theorem 11.2.5. *There exists a polynomial time algorithm achieving an approximation factor of 34 for the hard-capacitated vertex cover problem with unit multiplicity on unweighted multigraphs.*

Proof. The capacities of all the vertices in H_1 and H_2 are maintained. The cost paid while rounding the vertices in H_2 is

$$\eta \sum_{u \in \mathcal{D}} x^*(u). \tag{11.2.6}$$

Algorithm 2 Assigning edges with only one link in H_1

- 1: **for each** vertex $v \in A(H_1)$ with $|L(v)| \geq 1$ **do**
 - 2: select the vertex $h_2(v)$ that covers at least an edge-vertex from $L(v)$ and assign the corresponding edge-vertices to $h_2(v)$.
 - 3: **end for**
 - 4: **for each** vertex $v \in A(H_1)$ with $l(v) \leq p_2(v)$ **do**
 - 5: assign all the remaining edge-vertices (at most $|L(v)| - \lceil l(v) \rceil$) to v
 - 6: **end for**
 - 7: **for each** vertex $v \in A(H_1)$ with $l'(v) > 1$ **do**
 - 8: scale up the x^* variables in $\bigcup_{v \in A(H_1)} R(v)$ by a factor of $\frac{1}{1-\frac{1}{\eta}}$ and denote it by \bar{x} .
 - 9: **end for**
 - 10: create the MM instance $(\{(a(v), d(v))\}, \{S(u)\})$, and round the fractional solution \bar{x} to obtain an integral solution.
 - 11: **for each** u such that $S(u)$ is chosen by MM algorithm **do**
 - 12: select u and assign all the leaf-edges incident on u to it.
 - 13: **end for**
 - 14: **for each** $v \in A(H_1)$ with $l'(v) > 1$ **do**
 - 15: assign all the remaining leaf edge-vertices of $L(v)$ (at most $|L(v)| - \lceil l(v) \rceil$) to it.
 - 16: **end for**
-

From H_1 , vertices are chosen in two phases. First, for selecting vertices in \mathcal{D}' ,

we pay at most

$$\eta \sum_{v \in \mathcal{D}'/\mathcal{D} \text{ s.t. } L(v)=0 \text{ and } T(v)=0} x^*(v) + \frac{1}{\left(1 - \frac{1}{\eta}\right)} \sum_{v \in \mathcal{D}'/\mathcal{D} \text{ s.t. } L(v) \geq 1 \text{ or } T(v) \geq 1} x^*(v). \quad (11.2.7)$$

Vertices with $|L(v)| \geq 1$ must have fractional value at least $\left(1 - \frac{1}{\eta}\right)$. Vertices with $|T(v)| \geq 1$, also must have fractional value at least $1 - \frac{1}{\eta}$, since none of its children edge-vertices were assigned in step (4) of Algorithm (1). The number of vertices

picked in step (8) of Algorithm 1 is at most

$$\begin{aligned} & |\{v \in \mathcal{D}' \text{ s.t. } |T(v)| \geq 1\}| + \sum_{v \in A(H_1) \setminus \mathcal{D}' \cup \mathcal{D}} x^*(v) \\ & \leq \frac{1}{\left(1 - \frac{1}{\eta}\right)} \sum_{v \in \mathcal{D}' \text{ s.t. } T(v) \geq 1} x^*(v) + \sum_{v \in A(H_1) \cup A(H_2) \setminus \mathcal{D}' \cup \mathcal{D}} x^*(v). \end{aligned} \quad (11.2.8)$$

In Algorithm 2, we further select some vertices from $A(H_2)$. Let $R = \bigcup_{\substack{v \in A(H_1) \\ \text{s.t. } |L(v)| \geq 1}} \{R(v) \cup h_2(v)\}$. The cost paid for selecting vertices from R while rounding on H_1 is at most s for selecting the vertices $h_2(v)$ for all v and $21s + 32 \sum_{\substack{u \in \bigcup_{v \in A(H_1)} R(v) \\ \text{s.t. } l'(v) > 1}} \bar{x}(u)$ from Theorem 11.3.5. Therefore, the cost paid for selecting vertices from H_2 while rounding on H_1 is at most

$$\begin{aligned} & 22s + \frac{32}{1 - \frac{1}{\eta}} \sum_{\substack{u \in \bigcup_{v \in A(H_1)} R(v) \\ \text{s.t. } l'(v) > 1}} x^*(u) \\ & \leq \frac{22}{\left(1 - \frac{1}{\eta}\right)} \sum_{v \in \mathcal{D}' \text{ s.t. } |L(v)| \geq 1} x^*(v) \\ & + \frac{32}{1 - \frac{1}{\eta}} \sum_{v \in A(H_1) \cup A(H_2) \setminus \mathcal{D}' \cup \mathcal{D}'} x^*(v). \end{aligned} \quad (11.2.9)$$

Therefore, the total cost from Equation (11.2.6), (11.2.7), (11.2.8) and (11.2.9) is at most

$$\eta \sum_{\substack{v \in \mathcal{D}' \cup \mathcal{D}' \\ \text{s.t. } x^*(v) < 1 - \frac{1}{\eta}}} x^*(v) + \frac{23}{\left(1 - \frac{1}{\eta}\right)} \sum_{\substack{v \in \mathcal{D}' \cup \mathcal{D}' \\ \text{s.t. } x^*(v) \geq 1 - \frac{1}{\eta}}} x^*(v) + \left(\frac{32}{\left(1 - \frac{1}{\eta}\right)} + 1 \right) \sum_{\substack{v \in A(H_1) \cup A(H_2) \\ \setminus \mathcal{D}' \cup \mathcal{D}'}} x^*(v)$$

Setting $\eta = 34$, we thus obtain a 34-approximation. ■

11.3 Multi-Set Multi-cover

In the multi-set multi-cover problem (MM), we are given a ground set of N elements U and a collection of multi-sets \mathcal{S} of U , $\mathcal{S} = \{S_1, S_2, \dots, S_M\}$. Each multi-set $S \in \mathcal{S}$ contains $M(S, e)$ copies of element $a \in U$. Each element a has a demand of $r(a)$ and needs to be covered $r(a)$ times. The objective is to minimize the number of chosen sets that satisfy the demands of all the elements.

For MM, Kolliopoulos and Young gave a $\log n$ -approximation [89]. The same bound also follows from [138], and in general, this $\log n$ factor cannot be improved. However, if we plug in one of these algorithms in previous section, we cannot get better than $\log n$ -approximation for the hard-capacitated vertex cover problem. In this section we provide a new algorithm for MM that selects at most $21N + 32F$ sets, when an optimal solution selects F sets. As we saw in the last section, this result played a crucial role in obtaining a constant approximation.

The following is a linear program relaxation for MM.

$$\text{minimize } \sum_{S \in \mathcal{S}} x(S) + |U| \quad (\text{LP}_1)$$

$$\text{subject to } \sum_{a \in S} M(a, S)x(S) \geq r(a) \quad \forall a \in U \quad (11.3.10)$$

$$0 \leq x(S) \leq 1 \quad \forall S \in \mathcal{S} \quad (11.3.11)$$

11.3.1 Rounding Algorithm

Let x^* denote the LP optimal solution. Rounding algorithm has several steps.

Step 1. Selecting sets with high fractional value. First, we pick all sets $S \in \mathcal{S}$ such that $x^*(S) \geq \alpha > 0$, where $\frac{1}{\alpha}$ is the desired approximation factor. Denote the chosen sets by \mathcal{H} . Each element a now has a residual requirement of $r(a) - \sum_{a \in S, S \in \mathcal{H}} M(S, a)$. Clearly the fractional solution x^* projected on the sets $\mathcal{S} \setminus \mathcal{H}$ is a feasible solution for the residual problem. For each element $a \in \mathcal{U}$, let $\bar{r}(a) = r(a) - \sum_{a \in S, S \in \mathcal{H}} M(S, a)$ be the residual requirement. For some $\beta > 0$ (to be set later), let $y(S) = \beta x^*(S)$, for each $S \in \mathcal{S} \setminus \mathcal{H}$. We have for all elements $a \in \mathcal{U}$, $\sum_{a \in S, S \in \mathcal{S} \setminus \mathcal{H}} M(S, a)y(S) \geq \beta \bar{r}(a)$.

Note that after this step, we have a fractional solution with cost

$$|H| + \sum_{S \in \mathcal{S} \setminus \mathcal{H}} y(S) + |U| \leq \frac{1}{\alpha} \sum_{S \in H} x^*(S) + \beta \sum_{S \in \mathcal{S} \setminus \mathcal{H}} x^*(S) + |U|.$$

For notational simplicity, we denote $\mathcal{C} = \mathcal{S} \setminus \mathcal{H}$. Next, we proceed to round the variables $y(S)$ for $S \in \mathcal{C}$.

Step 2. Rounding into powers of 2. For each multiplicity $M(S, a)$, $\forall S \in \mathcal{C}, a \in \mathcal{U}$, we round it to the highest power of 2 lesser than or equal to $M(S, a)$ and denote it by $M^1(S, a)$. For each requirement $\bar{r}(a)$, $\forall a \in \mathcal{U}$, consider the lowest power of 2 greater than or equal to $\bar{r}(a)$ and denote it by $\bar{r}^1(a)$. Clearly, if $\sum_{a \in S, S \in \mathcal{C}} M(S, a)y(S) \geq \beta \bar{r}(a)$, then $\sum_{a \in S, S \in \mathcal{C}} M^1(S, a)4y(S) \geq \beta \bar{r}^1(a)$. We

denote $\mathbf{y}^1 = 4\mathbf{y}$.

Step 3. Division into small and big elements. First, for each element if there is a set that completely satisfies its requirement, we pick the set. We continue the process as long as no element can be covered entirely by a single set. Thus after this procedure, for all elements a , and for all sets S , $M^1(S, a) < \bar{r}^1(a)$ and hence $M^1(S, a) \leq \frac{\bar{r}^1(a)}{2}$. Now for each element a , we divide the sets in \mathcal{C} containing a into *big* sets ($Big(a)$) and *small* sets ($Small(a)$). A set $S \in \mathcal{C}$ is said to be a big set for a , if $M^1(S, a) \geq \frac{1}{18 \ln n} \bar{r}^1(a)$, otherwise it is called a small set, i.e.,

$$Big(a) = \{S \in \mathcal{C} \mid M^1(S, a) \geq \frac{1}{18 \ln n} \bar{r}^1(a)\}$$

$$Small(a) = \{S \in \mathcal{C} \mid M^1(S, a) < \frac{1}{18 \ln n} \bar{r}^1(a)\}$$

Now, we decompose elements into *big* and *small*. An element is *small* if it is covered to an extent of $\bar{r}^1(a)$ by the sets in $Small(a)$. Else, the element is covered at least to an extent of $(\beta - 1)\bar{r}^1(a)$ by the sets in $Big(a)$ and we call it a *big* element. This follows from the inequality

$$\sum_{a \in S, S \in \mathcal{C} \cap Big(a)} M^1(S, a) y^1(S) \quad + \quad \sum_{a \in S, S \in \mathcal{C} \cap Small(a)} M^1(S, a) y^1(S) \geq \beta \bar{r}^1(a).$$

Therefore, either the sets in $Small(a)$ cover a to an extent of $\bar{r}^1(a)$, or the sets in $Big(a)$ cover a to an extent of $(\beta - 1)\bar{r}^1(a)$. Let $\beta_1 = \beta - 1$. In the first case,

we refer a as a small element, otherwise it is a big element.

Step 4. Covering small elements. We employ simple independent randomized rounding for covering small elements. We pick each set $S \in \mathcal{C}$ with probability γy_S^1 , for some $\gamma \geq 2$.

Lemma 11.3.1. *All small elements are covered in Step 4 with probability at least $\left(1 - \frac{1}{n^{1/3}}\right)$.*

Proof. Consider a small element a and define random variable X_S^a for each small set $S \in \text{Small}(a)$ as follows:

$$\begin{aligned} X_S^a &= M^1(a, S), \text{ if } S \text{ is picked} \\ &= 0, \quad \text{otherwise} \end{aligned}$$

Then $X^a = \sum_{S \in \text{Small}(a)} X_S^a$ denotes the number of times a is covered by the sets in $\text{Small}(a)$. We have $\mathbb{E}[X^a] = \gamma \bar{r}^1(a)$. X^a is a sum of independent random variables, where each random variable X_S^a takes values between $[0, \frac{1}{18 \ln n} \bar{r}^1(a)]$.

We apply the following version of the Chernoff-Hoeffding inequality.

Theorem 11.3.2 (The Chernoff-Hoeffding Bound). *Given n independent random variables X_1, X_2, \dots, X_n each taking values between 0 and 1, if $X = \sum_{i=1}^n X_i$ and $\mathbb{E}[X] = \mu$ then for any $\delta > 0$*

$$\Pr[X < (1 - \delta)\mu] \leq e^{-\mu\delta^2/2},$$

where e is the base of the natural logarithm.

We define $Z_S^a = \frac{X_S^a}{\frac{\bar{r}^1(a)}{18 \ln n}}$. Then $Z_S^a \in [0, 1]$. We apply the Chernoff-Hoeffding bound to $\sum_{S \in \text{Small}(a)} Z_S^a$. We have $\mathbb{E} \left[\sum_{S \in \text{Small}(a)} Z_S^a \right] = \gamma 18 \log n$.

$$\begin{aligned} \Pr \left[\sum_{S \in \text{Small}(a)} Z_S^a < 18 \log n \right] &= \Pr [X_S^a < \bar{r}^1(a)] \\ &\leq e^{-\gamma 18 \log n \frac{(1-\frac{1}{2})^2}{2}} < \frac{1}{n^{4/3}} \end{aligned}$$

Thus by union bound, all small elements are covered the required number of times with probability at least $\left(1 - \frac{1}{n^{1/3}}\right)$. ■

Step 5. Covering big elements. For each big element, we consider only the big sets containing it. For each such big element and big set we have $\frac{1}{18 \ln n} r_a^1 < M^1(S, a) \leq \frac{r_a^1}{2}$. Since, multiplicities are powers of 2, there are at most $l = \ln \ln n + 3$ different values of multiplicities of the sets for each element a . Let $T_1^a, T_2^a, \dots, T_l^a$ denote the collection of these sets with multiplicities $\frac{\bar{r}^1(a)}{2}, \frac{\bar{r}^1(a)}{2^2}, \dots, \frac{\bar{r}^1(a)}{2^l}$ respectively. That is, $T_i^a = \{S \in \text{Big}(a) \mid M(S, a) = \frac{\bar{r}^1(a)}{2^i}\}$. Set $\beta_1 \geq 3$. For each $i = 1, 2, \dots, l$, if $\sum_{S \in T_i^a} y^1(S) > i$ and the number of sets that have been picked from T_i^a is less than $\frac{\sum_{S \in T_i^a} y^1(S)}{(\beta_1 - 2)}$, pick new sets from T_i^a such that the total number of chosen sets from T_i^a is $\left\lceil \frac{\sum_{S \in T_i^a} y^1(S)}{(\beta_1 - 2)} \right\rceil$.

We now show that each big element gets covered the required number of times and the total cost is bounded by a constant factor of the optimal cost.

Lemma 11.3.3. *Each big element a is covered $r(a)$ times by the chosen sets.*

Proof. Consider a big element a that is not covered after Step 4. Clearly, there is no set in \mathcal{S} such that $M(S, a) > \frac{\bar{r}^1(a)}{2}$. Now, a must satisfy the following inequality

$$\sum_{S \in \text{Big}(a)} M(a, S) y^1(S) \geq \beta_1 \bar{r}^1(a),$$

and thus it also satisfies the inequality below

$$\sum_{i=1}^l \frac{\bar{r}^1(a)}{2^i} \sum_{S \in T_i^a} y^1(S) \geq \beta_1 \bar{r}^1(a).$$

Call $R_i^a = \sum_{S \in T_i^a} y^1(S)$, for $i = 1, 2, \dots, l$. We pick at least $\lceil R_i^a / (\beta_1 - 2) \rceil$ sets from T_i^a unless $R_i^a \leq i$. If for all i , $R_i^a > i$, then taking $\beta_1 \geq 3$, element a is covered at least to an extent of $\sum_{i=1}^l \frac{\bar{r}^1(a)}{2^i} R_i^a / (\beta_1 - 2) = \frac{\beta_1}{\beta_1 - 2} \bar{r}^1(a) > 3\bar{r}^1(a)$. Otherwise, there are some i , for which $R_i^a \leq i$, and it is possible that we do not pick any set from T_i^a . The total fractional coverage coming from the sets in T_i^a with $R_i^a \leq i$ is at most

$$\bar{r}^1(a) \sum_{i=1}^l \frac{i}{2^i} < 2\bar{r}^1(a).$$

Therefore,

$$\sum_{i=1}^l \frac{\bar{r}^1(a)}{2^i} \sum_{S \in T_i^a, R_i^a > i} y^1(S) \geq (\beta_1 - 2) \bar{r}^1(a).$$

We set $\beta = 3$. Thus, element a is covered to an extent of at least \bar{r}_a^1 . The remaining coverage requirement of element a is fulfilled by the sets chosen in \mathcal{H} .

Thus all the big elements are covered. ■

Lemma 11.3.4. *The expected number of sets selected in Step 4 is at most $2ln'$, where n' are the number of big elements that are not covered after Step 5.*

Proof. Consider an element a . For each $T_i^a, i = 1, 2, \dots, l$, compute the probability that the number of sets chosen in Step 4 is less than $R_i^a/(\beta_1 - 2)$, where R_i^a as defined in the previous lemma is $\sum_{S \in T_i^a} y^1(S)$. We define an indicator random variable $X_i^a(S)$ for each set $S \in T_i^a$.

$$\begin{aligned} X_i^a(S) &= 1, \text{ if } S \text{ is selected,} \\ &= 0, \text{ otherwise.} \end{aligned}$$

Then $X_i^a = \sum_{S \in T_i^a} X_i^a(S)$ denote the number of sets chosen from T_i^a in Step 4.

Now, $\Pr[X_i^a(S) = 1] = \gamma y^1(S)$, where $\gamma \geq 3$. Therefore, $E[X_i^a] = \gamma R_i^a$.

Hence, by the Chernoff-Hoeffding bound,

$$\Pr \left[X_i^a < \frac{R_i^a}{(\beta_1 - 2)} \right] \leq e^{-\frac{\gamma R_i^a}{2} \left(1 - \frac{1}{\gamma(\beta_1 - 2)}\right)^2}.$$

With $\beta_1 = 3, \gamma = 2$, we get $\Pr[X_i^a < R_i^a] \leq e^{-\frac{1}{4}R_i^a} = 1.284^{-R_i^a}$. If $R_i^a > i$ and $X_i^a < R_i^a$, we pick at most $R_i^a + 1$ sets. The expected number of sets picked

in Step 5 to cover a is at most

$$\sum_{i=1, R_i^a \geq i}^l (R_i^a + 1) 1.284^{-R_i^a} \leq \sum_{i=1}^l \frac{i+1}{1.284^i} \leq \frac{1}{(1 - \frac{1}{1.284})} + \frac{1}{1.284 (1 - \frac{1}{1.284})^2} \leq 21.$$

Thus, the expected number of sets selected in Step 5 is at most $21n'$, where n' is the number of big elements that get covered in Step 5. ■

Theorem 11.3.5. *The algorithm returns a solution with expected cost at most $21N + 32F$, where $F = \sum_S x^*(S)$, and covers all the elements with probability at least $1 - \frac{1}{n^{1/3}}$.*

Proof. From Lemma 11.3.1 and 11.3.3, we know all the big elements are covered and all the small elements are covered with probability at least $1 - \frac{1}{n^{1/3}}$.

Step 1. The total number of sets picked is at most $|\mathcal{H}|$ where \mathcal{H} are the sets each with fractional value at least α . Thus, $|\mathcal{H}| < \frac{1}{\alpha} \sum_{S \in \mathcal{H}} x_S^*$.

Step 4. The total expected cost incurred in the randomized rounding step is at most

$$\sum_{S \in \mathcal{S} \setminus \mathcal{H}} \gamma y_S^1 = \sum_{S \in \mathcal{S} \setminus \mathcal{H}} 2y_S^1 = \sum_{S \in \mathcal{S} \setminus \mathcal{H}} 8y_S = \sum_{S \in \mathcal{S} \setminus \mathcal{H}} 8\beta x_S^*. \text{ Now } \beta_1 = 3 \text{ and } \beta = \beta_1 + 1 = 4. \text{ Hence, the expected cost is at most } 32 \sum_{S \in \mathcal{S} \setminus \mathcal{H}} x_S^*.$$

Step 5. From Lemma 11.3.4, the expected number of sets picked is at most $21n'$, where n' are the big elements that are not covered by Step 4.

Setting $\alpha = \frac{1}{32}$, we get the desired result. ■

We have not tried to optimize the constants of our approach, but reducing the approximation ratio substantially to 2 or 3 may require significant new ideas.

11.4 Set Cover with Hard Capacity Constraints

In this section, we consider the unweighted set cover problem, where each set has a hard capacity. We first consider the case, where each set has a single copy ($m(S) = 1, \forall S$). Next, this is extended to handle arbitrary multiplicities for each set. The main result in this section is an $O(f)$ approximation for the set cover problem with hard capacity constraints where each element belongs to at most f sets. As a corollary, we obtain a constant factor approximation algorithm for the vertex cover problem with hard capacity where arbitrary number of copies of each vertex may be available.

The algorithm in this section follows the same basic steps as in Section 11.2. We start with the natural LP-relaxation similar to LP_{VC} .

$$\text{minimize } \sum_{S \in \mathcal{S}} x(S) \quad (\text{LP}_{\text{SC}})$$

subject to

$$\sum_{S \ni a} y(a, S) = 1 \quad \forall a \in \mathcal{U}, \quad (11.4.12)$$

$$y(a, S) \leq x(S), \quad \forall a \in \mathcal{U}, a \in S, \quad (11.4.13)$$

$$\sum_{a \in S} y(a, S) \leq k(S)x(S) \quad \forall S \in \mathcal{S}, \quad (11.4.14)$$

$$0 \leq x(S) \leq 1 \quad \forall S \in \mathcal{S}, \quad (11.4.15)$$

$$0 \leq y(a, S) \leq 1 \quad \forall a \in \mathcal{U}. \quad (11.4.16)$$

The rounding algorithm is similar to the one described in Section 11.2. Here we highlight the main changes. From the LP optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$, we create a bipartite graph $H = (A, B, E(H))$, where A represents the sets, B represents the elements and links in H represent whether a particular element is fractionally covered by a set in the LP solution, that is, $A = \{S \in \mathcal{S}\}$, $B = \{a \in \mathcal{U}\}$, $E(H) = \{(a, S) \mid y^*(a, S) > 0\}$. Each vertex $S \in A$ has an associated weight of $x^*(S)$, and each link (a, S) has an associated weight of $y^*(a, S)$. We now modify the link weights and in the process decompose H into two graphs H_1 and H_2 , where H_1 is a forest and in H_2 all the link weights are equal to the weights of the corresponding incident vertex in A . This step is exactly same as Step 1 in Section 11.2.

Step 2. Rounding on H_2 .

We discard all the isolated vertices in H_2 and we select all the vertices in $A(H_2)$ with x^* value equal or greater than $\min(\frac{1}{\eta}, \frac{1}{2f})$. Recall that η will be the desired approximation ratio. Let us denote these chosen vertices by \mathcal{D} . Then,

$$\mathcal{D} = \{S \mid S \in A(H_2), x^*(S) \geq \min(\frac{1}{\eta}, \frac{1}{2f})\}.$$

For every element $a \in B(H_2)$ with a contained in the sets $\{S_a^1, S_a^2, \dots, S_a^f\} \in B(H_2)$, if either one of these sets, say S_a^i is in \mathcal{D} and also $(a, S_a^i) \in E(H_2)$, then we set the corresponding $y(a, S_a^i)$ variable to 1. Here sets play the role of vertices in the vertex cover problem and elements correspond to edges. Thus, following Observation 11.2.2, all the capacities of the sets in \mathcal{D} are maintained.

If all f links of an element a belong to $E(H_2)$, then after this step, a is covered. Otherwise, if the total fractional contribution of the links connecting a in H_2 is at least $\min(\frac{f-1}{\eta}, \frac{f-1}{2f})$, then again a is covered. We now proceed to H_1 .

Step 3. Rounding on H_1 . H_1 is a forest, it contains the vertices in $A(H_1)$ and elements that have at least one link in $E(H_1)$. We call an element *dangling* if it has at least one link in $E(H_2)$ and at least one link in $E(H_1)$. We root each tree in H_1 to some arbitrary set. Trees naturally define a parent-child relationship.

Step 3a. Rounding elements with all f connections in H_1 .

In H_1 , we define \mathcal{D}' as

$$\mathcal{D}' = \{S \mid S \in \mathbf{A}(H_1) \setminus \mathcal{D}, x^*(S) \geq \min(\frac{1}{\eta}, \frac{1}{2f})\}.$$

For each element in $\mathbf{B}(H_1)$, if at least one of its *children* set is selected in \mathcal{D}' , we assign a to it. Define $\mathbf{T}(S)$ to be the collection of elements contained in S that are not yet assigned and have all the links in $\mathbf{E}(H_1)$. Consider, any such element $a' \in \mathbf{T}(S)$. Since a' has not been covered, none of its children sets are picked. Denoting these children sets by $C(a')$, all $S \in C(a')$ have fractional value strictly less than $\min(\frac{1}{2f}, \frac{1}{\eta})$. Can $S \in C(a')$ have any child element a'' in H_1 that is not yet unassigned ? a'' must have at least one link either in $\mathbf{E}(H_1)$ or $\mathbf{E}(H_2)$ with fractional value at least $\min(\frac{1}{\eta}, \frac{1}{2f})$, and thus gets assigned. Since a' is not covered by any of at most $(f-1)$ children sets in H_1 , we have $x^*(S) \geq 1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})$.

We now pick $t(S) = \lceil \sum_{a' \in \mathbf{T}(S)} \sum_{a' \in S' \setminus S} y^*(a, S) \rceil$ sets one from each of the children sets of $t(S)$ elements in $\mathbf{T}(S)$. Rest of the elements in $\mathbf{T}(S)$ are assigned to S . Whenever, we pick a set in this stage, if there is any element in this set that is connected to it by a link in H_2 , we assign that element to the set.

Step 3b. *Rounding dangling elements, i.e, with not all f connections in H_1 .*

Define $\mathbf{L}(S)$ as the collection of dangling elements connected to S that are not covered in the previous steps and $l(S) = \sum_{a \in S} \sum_{a \in S', (a, S') \in \mathbf{E}(H_2)} y^*(a, S')$. Note that any S , with $|\mathbf{L}(S)| > 0$ must have $x^*(S) \geq 1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})$. We have a Lemma analogous to Lemma 11.2.3.

Lemma 11.4.1. *Each set $S \in A(H_1)$ can be assigned $|L(S)| - \lceil l(S) \rceil$ dangling elements without violating its capacity.*

Proof. Suppose, S belongs to H_2 as well and is selected in H_2 . Then,

$$\sum_{as.t.(a,S) \in E(H_2)} x^*(S) + \sum_{as.t.(a,S) \in E(H_1)} y^*(a, S) \leq k(S)x^*(S).$$

Thus,

$$\sum_{a|(a,S) \in E(H_1)} y^*(a, S) = (k(S) - |\{a \mid (a, S) \in H_2\}|) x^*(S).$$

Now, $(k_S - |\{a \mid (a, S) \in E(H_2)\}|)$ is an integer, and we denote it by $k'(S)$.

Let us assume $S \in \mathcal{D}'$ first. Let the fractional value of the link connecting S to its parent edge-vertex be b . The capacity of S is $k'(S) \geq \lceil b + |T(S)| - t(S) + |L(S)| - l(S) \rceil$. The number of elements assigned to S is at most $1 + |T(S)| - \lceil t(S) \rceil + |L(S)| - \lceil l(S) \rceil$.

Now, following a similar argument as in Lemma 11.2.3, we get the desired result. ■

The elements in $L(S)$ have at least one link in $E(H_2)$ and other than S (which is the parent node for the elements of $L(S)$ in H_1), may be connected to some sets (that appear as their children) in $A(H_1)$. We first pick one set other than S from $A(H_2)$ such that it covers at least one element from $L(S)$. Let us denote this set by $h_2(S)$ and the elements of $L(S)$ that it covers by $P_2(S)$. Let $|P_2(S)| = p_2(S)$. If $l(S) \leq p_2(S)$, then rest of the elements of $L(S)$ can be assigned to S (by Lemma

11.4.1), and we exactly do that. Else, $l(S) > p2(S)$.

Consider all sets in $A(H_1) \cup A(H_2)$ that contain the elements of $L(S)$ except S and $h2(S)$. Denote these sets by $R(S)$. Therefore, any set in $R(S)$ is connected by at most one link from $E(H_1)$ (because of the tree structure); rest of the links are from $E(H_2)$. Hence, if we pick a set in $R(S)$, we can assign all the elements it connects to both in $E(H_1)$ and $E(H_2)$ without violating its capacity³.

We scale up all the x^* variables of $\bigcup_{S \in A(H_1)} R(S)$ by a factor of $\frac{1}{1 - \min(\frac{f-1}{\eta}, \frac{1}{2f})}$.

We also scale up the corresponding y^* link variables by a factor of $\frac{1}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})}$.

Let (\bar{x}, \bar{y}) denote this scaled up variables.

Lemma 11.4.2. *After scaling up \bar{y} satisfies $\sum_{s.t. a \in L(S) \setminus P2(S), S' \in R(S)} \bar{y}(a, S') \geq l(s) - p2(S) + 1$.*

Proof.

$$\begin{aligned} \sum_{\substack{(a, S') \\ s.t. a \in L(S) \setminus P2(S), S' \in R(S)}} \bar{y}(a, S') &= \frac{\left(l(S) - \sum_{a \in P2(S)} \sum_{S' \ni a, S' \neq S} y^*(a, S') \right)}{\left(1 - \min\left(\frac{f-1}{\eta}, \frac{f-1}{2f}\right) \right)} \\ &\geq \frac{\left(l(S) - p2(S) \min\left(\frac{f-1}{\eta}, \frac{f-1}{2f}\right) \right)}{\left(1 - \min\left(\frac{f-1}{\eta}, \frac{f-1}{2f}\right) \right)} \\ &> l(S) - p2(S) + 1, \end{aligned}$$

where the last inequality follows from the fact that $l(S) > p2(S) > 1$. ■

We set $l'(S) = 0$ if $l(S) \leq p2(S)$, else we set $l'(S) = l(S) - p2(S) + 1$. If

³ this holds because any set S' that has at least one link fractionally connected to it in $E(H_1)$ has capacity $k'(S') \geq 1$.

we can pick enough sets from $R(S)$ such that at least $\lfloor l'(S) \rfloor$ elements from $L(S)$ are covered by the sets picked from $R(S)$, then from Lemma 11.4.1, the remaining elements can be assigned to S .

We thus arrive to the MM problem (Section 11.3).

For each $S \in A(H_1)$ with $l'(S) > 1$, we create an element $a(S)$. For each set $S' \in \bigcup_{S \in A(H_1)} R(S)$, we create a multi-set $T(S')$. If there are $d(S, S')$ elements in $L(S) \setminus P2(S)$ incident upon S' , then we create $d(S, S')$ copies of $a(S)$ in $T(S')$. Each element $a(S)$ has a requirement of $r(S) = \lfloor l'(S) \rfloor$. The goal is to pick minimum number of sets such that each element $a(S)$ is covered $\lfloor l'(S) \rfloor$ times counting multiplicities.

We solve the MM problem and for each selected set $T(S')$, we include S' in the solution. If there are $d(S, S')$ copies of $a(S)$ in $T(S')$, then there are $d(S, S')$ elements from $L(S) \setminus P2(S)$ that are contained in S' . We let S' cover all these elements. The number of elements that are not covered from $L(S)$ is at most $|L(S)| - \lfloor l'(S) \rfloor - p2(S)$, which is at most $L(S) - \lceil l(S) \rceil$. By, Lemma 11.4.1, these elements can be covered by S and therefore we assign them to S . Each element S' covers all the elements linked to it in $E(H_2)$ and possibly one extra element that is linked in $E(H_1)$. Since capacities are always integers, S' maintains its capacity.

Theorem 11.4.3. *There exists a polynomial time algorithm achieving an approximation factor of $\max(65, 2f)$ for the set cover problem with hard capacities with unit multiplicities, where each element is contained in at most f sets.*

Proof. The capacities of all the sets in H_1 and H_2 are maintained.

The cost paid while rounding the sets in H_2 is

$$\max(2f, \eta) \sum_{S \in \mathcal{D}} x^*(S). \quad (11.4.17)$$

From H_1 , sets are chosen in two phases. First, for selecting vertices in \mathcal{D}' , we pay at most

$$\max(2f, \eta) \sum_{\substack{S \in \mathcal{D}'/\mathcal{D} \\ s.t., |L(S)|=0 \text{ and } |T(S)|=0}} x^*(S) + \frac{1}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} \sum_{\substack{S \in \mathcal{D}'/\mathcal{D} \\ s.t., |L(S)| \geq 1 \text{ or } |T(S)| \geq 1}} x^*(S). \quad (11.4.18)$$

The sets with $|L(S)| \geq 1$ or $|T(S)| \geq 1$ must have fractional value at least $(1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f}))$. The number of sets picked to satisfy the requirement of $t(S)$ for all S is at most

$$\begin{aligned} & |\{S \text{ s.t. } |T(S)| \geq 1\}| + \sum_{S \in A(H_1) \setminus \mathcal{D}' \cup \mathcal{D}} x^*(S) \\ & \leq \frac{1}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} \sum_{\substack{S \in \mathcal{D}'/\mathcal{D} \\ s.t., |T(S)| \geq 1}} x^*(S) \\ & + \sum_{S \in A(H_1) \cup A(H_2) \setminus \mathcal{D}' \cup \mathcal{D}} x^*(S). \end{aligned} \quad (11.4.19)$$

We further select sets from $A(H_1)$ and $A(H_2)$ to satisfy the requirements from $L(S)$. Let $R = \bigcup_{\substack{S \in A(H_1) \\ s.t., |L(S)| \geq 1}} \{R(S) \cup h_2(S)\}$. The cost paid for selecting sets from R while rounding on H_1 is at most s for selecting the sets $h_2(S)$ for all S

and $21s + 32 \sum_{S' \in \cup_{S \in A(H_1)} R(S)} \bar{x}(S)$ from Theorem 11.3.5. Here $s = |\{S \in A(H_1) \text{ s.t. } |L(S)| \geq 1\}|$. Therefore, the cost paid in this step is at most

$$\begin{aligned}
& 22s + \frac{32}{1 - \min(\frac{1}{\eta}, \frac{1}{2f})} \sum_{\substack{S' \in \cup_{S \in A(H_1)} R(S) \\ \text{s.t. } |L(S)| > 1}} x^*(S) \\
\leq & \frac{22}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} \sum_{S \in \mathcal{D}' \text{ s.t. } |L(S)| \geq 1} x^*(S) \\
& + \frac{32}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} \sum_{S \in A(H_1) \cup A(H_2) \setminus \mathcal{D} \cup \mathcal{D}'} x^*(S). \quad (11.4.20)
\end{aligned}$$

Therefore, the total cost from Equation (11.4.17), (11.4.18), (11.4.19) and (11.4.20) is at most

$$\begin{aligned}
& \max(\eta, 2f) \sum_{\substack{S \in \mathcal{D} \cup \mathcal{D}' \text{ s.t.} \\ x^*(S) < 1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})}} x^*(S) \\
& + \frac{23}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} \sum_{\substack{S \in \mathcal{D}' \cup \mathcal{D} \text{ s.t.} \\ x^*(S) \geq 1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})}} x^*(v) \\
& + \left(\frac{32}{1 - \min(\frac{f-1}{\eta}, \frac{f-1}{2f})} + 1 \right) \sum_{v \in A(H_1) \cup A(H_2) \setminus \mathcal{D} \cup \mathcal{D}'} x^*(v)
\end{aligned}$$

We can adjust the value of η according to the value of f , in general, by setting $\eta = 65$, we obtain a $\max(65, 2f)$ -approximation. ■

11.4.1 Hard-Capacitated Set Cover with Arbitrary Multiplicities

Given an instance of hard-capacitated set cover with arbitrary multiplicities where each element belongs to at most f sets, we reduce it to an instance of unit multiplicity by slightly increasing the value of f . First, we solve the following natural LP-relaxation, where set S has multiplicity $m(S)$.

$$\text{minimize } \sum_{S \in \mathcal{S}} x(S) \quad (\text{LP}_{\text{SC-Mult}})$$

$$\text{subject to} \quad (11.4.21)$$

$$\sum_{S \ni a} y(a, S) = 1 \quad \forall a \in \mathcal{U}, \quad (11.4.22)$$

$$y(a, S) \leq x(S), \quad \forall a \in \mathcal{U}, a \in S, \quad (11.4.23)$$

$$\sum_{a \in S} y(a, S) \leq k(S)x(S) \quad \forall S \in \mathcal{S}, \quad (11.4.24)$$

$$0 \leq x(S) \leq m(S) \quad \forall S \in \mathcal{S}, \quad (11.4.25)$$

$$0 \leq y(a, S) \leq 1 \quad \forall a \in \mathcal{U}. \quad (11.4.26)$$

Let $(\mathbf{x}^*, \mathbf{y}^*)$ be an optimal solution of the above LP. We construct a bipartite graph $H(A, B, E(H))$, where A contains sets, possibly multiple copies of them, B contains the elements and links are created based on non-zero components of \mathbf{y}^* . For each set $S \in \mathcal{S}$ with $x^*(S) > 0$, we create $\lceil x^*(S) \rceil$ copies of S in A . Each one of them except the first one gets a weight of 1, while the first one gets a weight

of $x^*(S) - \lfloor x^*(S) \rfloor$. We denote the weights of the sets by w . Therefore the total weight of all the sets in A equals $\sum_S x^*(S)$. Next, for each element a , we create a vertex a in B . Let a be contained in sets $S_a^1, S_a^2, \dots, S_a^f$ with fractional values $y^*(a, S_a^1), y^*(a, S_a^2), \dots, y^*(a, S_a^f)$ respectively. Consider, one of these sets, say S_a^i . Let there be l copies of S_a^i in A . Denote them by $S_{a,1}^i, S_{a,2}^i, \dots, S_{a,l}^i$ and their weights by $w(S_{a,1}^i) = h, w(S_{a,2}^i) = w(S_{a,3}^i) = \dots = w(S_{a,l}^i) = 1$. The fractional capacity of $S_{a,j}^i, j \in [1, l]$, is $w(S_{a,j}^i)k(S_a^i)$.

We start with $S_{a,1}^i$ and create a link $(a, S_{a,1}^i)$. Let the current weight of the links connected to $S_{a,1}^i$ be W_1 . We set the weight of $(a, S_{a,1}^i)$ as $z(a, S_{a,1}^i) = \min(y^*(a, S_a^i), w(S_{a,1}^i), W_1 - w(S_{a,1}^i)k(S_a^i))$. We set $y^*(a, S_a^i) = y^*(a, S_a^i) - z(a, S_{a,1}^i)$ and if $y^*(a, S_a^i) > 0$, we proceed to $S_{a,2}^i$.

We again create a link $(a, S_{a,2}^i)$. Let the current weight of the links connected to $S_{a,2}^i$ be W_2 , then we set the weight of $(a, S_{a,2}^i)$ as $z(a, S_{a,2}^i) = \min(y^*(a, S_a^i), w(S_{a,2}^i), W_2 - w(S_{a,2}^i)k(S_a^i)) = \min(y^*(a, S_a^i), W_2 - w(S_{a,2}^i)k(S_a^i))$.

A link is never made to a copy $S_{a,j}^i, j \geq 3$, unless the $(j - 1)$ -th copy is completely filled up to its fractional capacity which is at least 1. Therefore, element a may have links to at most 3 copies of S_a^i . We repeat the same procedure for all the other sets.

Hence, in the created bipartite graph an element may be linked to at most $3f$ sets. Also, the vectors (\mathbf{w}, \mathbf{z}) satisfy the constraints of LP_{SC} . Each set in the

modified instance now has multiplicity 1, therefore from Theorem 11.4.3, we get a $\max(65, 6f)$ approximation algorithm for it.

Theorem 11.4.4. *There exists a polynomial time algorithm achieving an approximation factor of $\max(65, 6f)$ for the set cover problem with hard capacities and arbitrary multiplicities, where each element is contained in at most f sets.*

Corollary 11.4.5. *There exists a polynomial time algorithm achieving an approximation factor of 38 for the vertex cover problem with hard capacities and arbitrary multiplicities in multigraph.*

Proof. We reduce the vertex cover with arbitrary multiplicities to a unit multiplicity instance. Thus, after the reduction, we have $f \leq 6$. Therefore, if we set $\eta = 38$ in Theorem 11.4.3, we get a 38-approximation. ■

There are several interesting questions to pursue. The scheduling model of [52], which is a generalization of [21, 51] nicely captures the aspects of energy usage in data centers. Demaine et al. gave a logarithmic approximation in [52], however their problem can be reduced to a hard-capacitated set covering instance with *multiple* capacity constraints. This raises the question, can we design an $O(f)$ -approximation algorithm for set cover problem in presence of multiple capacity constraints. Given the fact that each job can be executed only on a few machines in data centers, such an $O(f)$ -approximation algorithm could be quite useful. Also, designing an $O(f)$ -approximation algorithm for submodular covering problem will be an interesting future direction.

CHAPTER 12

Conclusion

In this thesis, we make significant contributions towards designing new techniques in approximation algorithms. While our techniques are very general and can have numerous applications, in this thesis we illustrate the power of these methods through a variety of resource allocation problems. Resource allocation problems are ubiquitous in computer science and come in a rich variety. We showcase four such different applications: scheduling jobs on parallel machines, fair allocation problems, server location problems in networking and advertisement allocations. In several cases, we develop new models and algorithms and in other cases our techniques lead to significant better results. All of our algorithms are based on rounding fractional optimal solutions of linear programming relaxations and most of them rely on probabilistic analysis.

In Chapter 2, we introduce a new rounding methodology which is a generalization of two previously extremely well-studied rounding methods, namely, dependent rounding and iterative relaxations. Using this rounding, we get improved

results for several job-scheduling problems in Chapter 4 and 5, max-min fair allocation problem in Chapter 6, overlay network design problem in Chapter 8 and adcell allocation problem in Chapter 10.

In Chapter 3, we describe our contribution towards the algorithmic aspects of a powerful probabilistic tool, the Lovász Local Lemma. This result leads to the first algorithms for several combinatorial optimization problems. Using it, we also obtain the first constant-factor approximation algorithm for the Santa Claus problem (Chapter 7), which was a major open question in the area of approximation algorithms.

In Chapter 9, we develop an approximation algorithm for a server allocation problem in content distribution networks. The MatroidMedian is a significant generalization of the well-studied k -median problem. The algorithm is based on linear programming rounding that exploits the structure of the underlying constraint matrix. The applications in Chapter 9, as well as in Chapter 10 and Chapter 4 can be viewed as generalizations of well-known optimization problems to handle capacity constraints. In Chapter 11, we study general capacitated covering problems and settle an open question in this area.

12.1 Future Directions

The thesis leads to several interesting questions and directions. Probabilistic methods are in the heart of techniques that we develop in this thesis. The role that

probability and randomness play in modern computer science cannot be overemphasized. From the highly theoretical notion of probabilistic theorem proving, to the very practical applications of cryptography and web search ranking, sophisticated probabilistic techniques have been developed in the last two decades for a broad range of challenging computing applications. One such powerful probabilistic tool is the Lovász's Local Lemma (LLL). While developing a constructive version for LLL, we introduce a proof-concept, namely, *LLL distribution*. This is the distribution that one obtains conditioned on avoiding certain bad events. We believe the concept of LLL distribution will find many applications in randomized algorithms and exploring further properties of it will be extremely interesting.

Probably, one of the most well-known probabilistic concentration inequality that applies to sum of independent and bounded random variables is The Chernoff-Hoeffding bound. The random variables, instead of being independent, can satisfy negative correlation and yet, the bound holds. We show when our new rounding method is applied to variables of a LP, that can equivalently be seen as fractional values on edges of a bipartite graph, then such negative correlation property holds. Exploiting this property, we characterize the integrality gap of a configuration LP for the max-min fair allocation problem near optimally. This triggers the question, for what kind of LP and for which set of variables our rounding method can guarantee such negative correlation property ? An answer to this question may have several implications in approximation algorithms design.

In the area of resource allocation, our applications lead to many new directions. First, in the area of scheduling, earlier to our works, research on energy minimization mainly concentrated on identical or related machine models. Whereas, for modern data centers, such machine models are not appropriate. Machines in data centers may have varying computation power and CPU speed, and a data may not be replicated on all machines. As a consequence, a job or a query can be executed only on a subset of machines. The fundamental model of parallel machines that captures these aspects is the *unrelated parallel machines* (UPM). A salient feature of our scheduling works in this thesis is that we consider UPM. It will be extremely interesting to study different aspects of energy savings such as scaling processor speed, introducing sleep states with different energy requirements in this machine model. Of course, with multicores, a machine can handle several jobs at a time, and a more realistic model is to consider online jobs. Altogether, these lead to a broad spectrum of problems.

The concept of capacitated covering that we study in this thesis can be very useful in the domain of energy-efficient scheduling as well. As we point out in Chapter 11, a class of such scheduling problems can be modeled using multiple capacity constraints on sets. It will be an interesting future direction to extend our methods to handle multiple capacity constraints and understand the hardness of such problems.

In the area of fair allocation, an obvious question is to better understand the

limitation of approximability of the max-min fair allocation problem and getting a better constant bound for the Santa Claus problem. In the area of networking, resource constraints lead to several interesting questions. Each open server may have an upper bound on the number of clients that it can handle or a client may need to be connected to several servers in order to ensure a fault-resilient networking. In adcell allocation, it will be interesting to develop online stochastic models and new approximation algorithms in that framework.

To conclude, the theory of approximation algorithms is a vibrant area of research and our contributions aim towards providing new generic tools that can have wide-applicability. These methods have already been successfully applied to several problems on resource allocations in this thesis. While, we have provided improved solutions for a collection of them, our research has opened up many new possibilities for future works and has created a premise for a large variety of optimization problems; they are both of theoretical interest and of practical significance.

Bibliography

- [1] S. Aalei, M. Hajiaghayi, V. Liaghat, D. Pei, and B. Saha. Adcell-ad allocation in cellular networks. *European Symposium on Algorithms*, 2011.
- [2] A. Ageev and M. Sviridenko. Pipage rounding: a new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.
- [3] M. Agrawal. Overview of mobile advertising. <http://www.telecomcircle.com/2009/11/overview-of-mobile-advertising>.
- [4] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA '07: Proceedings of the Nineteenth annual ACM Symposium on Parallel algorithms and architectures*, pages 289–298, 2007.
- [5] N. Alon. A parallel algorithmic version of the local lemma. *Random Structures & Algorithms*, 2:367–378, 1991.

- [6] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm. *J. of Algorithms*, 7:567–583, 1986.
- [7] N. Alon and J. Grytczuk. Breaking the rhythm on graphs. *Discrete Mathematics*, 308(8):1375–1380, 2008.
- [8] N. Alon, J. Grytczuk, M. Haluszczak, and O. Riordan. Nonrepetitive colorings of graphs. *Random Structures & Algorithms*, 21(3-4):336–346, 2002.
- [9] N. Alon and J. H. Spencer. *The Probabilistic Method, Third Edition*. John Wiley & Sons, Inc., 2008.
- [10] K. Andreev, B. Maggs, A. Meyerson, and R. Sitaraman. Designing overlay multicast networks for streaming. In *SPAA*, pages 149–158, 2003.
- [11] M. Andrews. Approximation algorithms for the edge-disjoint paths problem via Ræcke decompositions. In *FOCS*, pages 277–286, 2010.
- [12] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming*, pages 1–36, 2002.
- [13] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for euclidean k-medians and related problems. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, 1998.

- [14] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.
- [15] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets Hypergraph Matchings. In *Proc. APPROX-RANDOM*, pages 10–20, 2009.
- [16] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *STOC '07: Proceedings of the thirty-ninth annual ACM Symposium on Theory of computing*, pages 114–121, 2007.
- [17] Y. Azar and A. Epstein. Convex programming for scheduling unrelated parallel machines. In *Proc. of the ACM Symposium on Theory of Computing*, pages 331–337. ACM, 2005.
- [18] I. D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *SODA*, pages 661–670, 2001.
- [19] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 520–529, 2004.
- [20] N. Bansal and M. Sviridenko. The Santa Claus problem. In *STOC '06: Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 31–40, 2006.

- [21] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA '06*, pages 364–367, 2006.
- [22] J. Bar-Ilan, G. Kortsarz, and D. Peleg. Generalized submodular cover problems and applications. *Theor. Comput. Sci.*, 250:179–200, January 2001.
- [23] R. Bar-yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.
- [24] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 184, 1996.
- [25] Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum k-clustering in metric spaces. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 11–20, 2001.
- [26] M. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC '09: Proceedings of the 41st annual ACM Symposium on Theory of computing*, pages 543–552, 2009.
- [27] J. Beck. An algorithmic approach to the lovász local lemma. *Random Structures & Algorithms*, 2:343–365, 1991.

- [28] I. Bezáková and V. D. Allocating indivisible goods. *SIGecom Exch.*, 5(3):11–18, 2005.
- [29] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.
- [30] S. J. Brams and A. D. Taylor. Fair division - from cake-cutting to dispute resolution. In *Cambridge University Press*, 1996.
- [31] B. Bresar, J. Grytczuk, S. Klavzar, S. Niwczyk, and I. Peterin. Nonrepetitive colorings of trees. *Discrete Mathematics*, 307(2):163–172, 2007.
- [32] H. Bronnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14:463–479, 1995.
- [33] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA '00: Proceedings of the eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.
- [34] D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *FOCS '09: 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009.
- [35] D. Chakrabarty and G. Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap. *SIAM J. Comput.*, 39(6). 2189-2211, 2010.

- [36] D. Chakrabarty, E. Grant, and J. Knemann. On column-restricted and priority covering integer programs. In *Proc. Integer Programming and Combinatorial Optimization(IPCO)*, 2010.
- [37] K. Chandrasekaran, N. Goyal, and B. Haeupler. Deterministic algorithms for the Lovász local lemma. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10*, pages 992–1004, 2010.
- [38] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005.
- [39] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.
- [40] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005.
- [41] Z. Chi, G. Wang, X. Liu, and J. Liu. Approximating scheduling machines with capacity constraints. In *FAW '09: Proceedings of the Third International Frontiers of Algorithmics Workshop*, pages 283–292, 2009. Corrected version available as arXiv:0906.3056.
- [42] N. Christofides and S. Korman. A computational survey of methods for the set covering problem. *Mathematics of Operations Research*, 1975.

- [43] F. A. Chudak and D. P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Math. Program.*, 102(2):207–222, 2005.
- [44] J. Chuzhoy and P. Codenotti. Resource minimization job scheduling. In *APPROX*, 2009.
- [45] J. Chuzhoy, A. Gupta, J. S. Naor, and A. Sinha. On the approximability of some network design problems. In *SODA '05*, pages 943–951, 2005.
- [46] J. Chuzhoy and J. S. Naor. Covering problems with hard capacities. In *Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS '02*, pages 481–489, 2002.
- [47] J. Chuzhoy and Y. Rabani. Approximating k-median with non-uniform capacities. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05*, pages 952–958, 2005.
- [48] G. Cornuejols, M. Fisher, and G. Nemhauser. On the uncapacitated location problem. *Annals of Discrete Mathematics*, 1:163–178, 1977.
- [49] J. D. Currie. Pattern avoidance: themes and variations. *Theor. Comput. Sci.*, 339(1):7–18, 2005.
- [50] A. Czumaj and C. Scheideler. Coloring non-uniform hypergraphs: A new algorithmic approach to the general Lovász local lemma. In *SODA '00: Pro-*

ceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 30–39, 2000.

- [51] E. D. Demaine, M. Ghodsi, M. T. Hajiaghayi, A. S. Sayedi-Roshkhar, and M. Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '07, pages 46–54, 2007.
- [52] E. D. Demaine and M. Zadimoghaddam. Scheduling to minimize power consumption using submodular functions. In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, SPAA '10, pages 21–29, 2010.
- [53] B. Doerr. Linear discrepancy of basic totally unimodular matrices. *Electr. J. Comb.*, 2000.
- [54] T. Ebenlendr, M. Křícal, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *SODA '08: Proceedings of the Nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 483–490, 2008.
- [55] P. Erdos and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and Finite Sets*, volume 11 of *Colloq. Math. Soc. J. Bolyai*, pages 609–627. North-Holland, 1975.

- [56] M. M. Etschmaier and D. F. X. Mathaisel. Airline scheduling: An overview. *Transportation Science*, 19(2):127–138, 1985.
- [57] S. Eubank, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, and N. Wang. Structural and algorithmic aspects of massive social networks. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 711–720, 2004.
- [58] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. System Sci.*, 69(3):485–497, 2004.
- [59] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
- [60] U. Feige. On allocations that maximize fairness. In *SODA '08: Proceedings of the Nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, 2008.
- [61] U. Feige. On estimation algorithms vs approximation algorithms. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS*, volume 2 of *LIPICs*, pages 357–363. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.
- [62] L. Fleischer, K. Jain, and D. P. Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. In *In 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 339–347, 2001.

- [63] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72:16–33, 2006.
- [64] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding in bipartite graphs. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 323–332, 2002.
- [65] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 53:324–360, 2006.
- [66] N. Garg, R. Khandekar, and V. Pandit. Improved approximation for universal facility location. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05*, pages 959–960, 2005.
- [67] M. X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 273–282, 2006.
- [68] F. Grandoni, R. Ravi, and M. Singh. Iterative rounding for multi-objective optimization problems. In *ESA '09: Proceedings of the 17th Annual European Symposium on Algorithms*, 2009.
- [69] J. Grytczuk. Thue type problems for graphs, points, and numbers. *Discrete Mathematics*, 308(19):4419–4429, 2008.

- [70] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering with applications. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 858–865, 2002.
- [71] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 603–, 2000.
- [72] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. In *STOC*, pages 389–398, 2001.
- [73] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Segev. Scheduling with outliers. In *Proc. APPROX*, 2009. Full version available as arXiv:0906.2020.
- [74] A. Gupta and K. Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008.
- [75] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the lovasz local lemma. In *FOCS*, pages 397–406, 2010.
- [76] M. Hajiaghayi, R. Khandekar, and G. Kortsarz. The red-blue median problem and its generalization. *European Symposium on Algorithms*, 2010.
- [77] P. Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995.

- [78] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *Siam Journal on Computing*, 11:555–556, 1982.
- [79] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.
- [80] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [81] T. Ibaraki and N. Katoh. *Resource allocation problems: algorithmic approaches*. MIT Press, Cambridge, MA, USA, 1988.
- [82] S. Iwata and J. B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237, 2009.
- [83] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 448, 1998.
- [84] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 731–740, 2002.
- [85] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

- [86] D. R. Karger and M. Minkoff. Building steiner trees with incomplete global knowledge. In *FOCS*, pages 613–623, 2000.
- [87] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2:113–129, 1987.
- [88] S. Khuller, J. Li, and B. Saha. Energy efficient scheduling via partial shut-down. In *SODA*, pages 1360–1372, 2010.
- [89] S. G. Kolliopoulos and N. E. Young. Tight approximation results for general covering integer programs. In *IEEE Symposium on Foundations of Computer Science*, pages 522–528, 2001.
- [90] H. Konishi, T. Quint, and J. Wako. The on the shapley-scarf economy: Case of multiple types of indivisible goods. *Journal of Mathematical Economics*, 35:2001, 2000.
- [91] R. Krishnaswamy, A. Kumar, V. Nagarajan, Y. Sabharwal, and B. Saha. The matroid median problem. In *SODA*, pages 1117–1130, 2011.
- [92] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. A unified approach to scheduling on unrelated parallel machines. *Journal of the ACM*, 56(5), 2009.
- [93] A. Kündgen and M. J. Pelsmayer. Nonrepetitive colorings of graphs of bounded tree-width. *Discrete Mathematics*, 308(19):4473–4478, 2008.

- [94] L. C. Lau, J. S. Naor, M. R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 651–660, 2007.
- [95] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and jobshop scheduling in $O(\text{congestion} + \text{dilation})$ steps. *Combinatorica*, 14:167–186, 1994.
- [96] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
- [97] B. Li, X. Deng, M. J. Golin, and K. Sohraby. On the optimal placement of web proxies in the internet: The linear topology. In *Proceedings of the IFIP TC-6 Eighth International Conference on High Performance Networking*, pages 485–495, 1998.
- [98] J.-H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Inf. Process. Lett.*, 44(5):245–249, 1992.
- [99] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.
- [100] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15:1036–1053, 1986.

- [101] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41:960–981, September 1994.
- [102] M. Mahdian and M. Pal. Universal facility location. In *in Proc. of European Symposium of Algorithms 03*, pages 409–421, 2003.
- [103] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *APPROX*, pages 229–242, 2002.
- [104] M. Mahdian, Y. Ye, and J. Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. In *RANDOM-APPROX*, pages 129–140, 2003.
- [105] D. Marx and M. Schaefer. The complexity of nonrepetitive coloring. *Discrete Appl. Math.*, 157(1):13–18, 2009.
- [106] I. Mclean and A. Urken. Classics of social choice. In *U. Michigan Press, Ann Arbor, MI*.
- [107] A. Meyerson, K. Munagala, and S. A. Plotkin. Web caching using access statistics. In *SODA*, pages 354–363, 2001.
- [108] M. Molloy and B. Reed. Further algorithmic aspects of the Local Lemma. In *STOC '98: Proceedings of the 30th annual ACM Symposium on Theory of Computing*, pages 524–529, 1998.

- [109] M. Molloy and B. Reed. *Graph Colouring and the Probabilistic Method*. Springer-Verlag, 2001.
- [110] R. Moser. A constructive proof of the Lovász Local Lemma. In *Proc. ACM Symposium on Theory of Computing*, pages 343–350, 2009.
- [111] R. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. Technical Report Corr.abs/0903.0544, Computing Research Repository, 2009.
- [112] R. A. Moser. Derandomizing the Lovász Local Lemma more effectively. *CoRR*, abs/0807.2120, 2008.
- [113] M. Pál, E. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 329–338, 2001.
- [114] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.
- [115] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 86–92, 2000.

- [116] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.
- [117] R. Rushmeier, K. Hoffman, and M. Padberg. Recent advances in exact optimization of airline scheduling problems. Technical report, George Mason University, 1995.
- [118] B. Saha and A. Srinivasan. A new approximation technique for resource-allocation problems. *ICS*, pages 342–357, 2010.
- [119] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News*, 33(3):32–49, Sept. 2002.
- [120] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, 1995.
- [121] A. Schrijver. *Combinatorial optimization*. Springer, 2003.
- [122] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.
- [123] M. Singh and L. C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 661–670, 2007.

- [124] M. Skutella. Convex quadratic and semidefinite relaxations in scheduling. *Journal of the ACM*, 46(2):206–242, 2001.
- [125] A. Srinivasan. Budgeted allocations in the full-information setting. *APPROX '08*. 247-253, 2008.
- [126] A. Srinivasan. Improved approximations of packing and covering problems. In *STOC '95: Proceedings of the twenty-seventh annual ACM Symposium on Theory of computing*, pages 268–276, 1995.
- [127] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 588–597, 2001.
- [128] A. Srinivasan. New approaches to covering and packing problems. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete algorithms*, pages 567–576, 2001.
- [129] A. Srinivasan. An extension of the Lovász Local Lemma, and its applications to integer programming. *SIAM Journal on Computing*, 36:609–634, 2006.
- [130] T. Stephen, L. Tuncel, and H. Luss. On equitable resource allocation problems: a lexicographic minimax approach. *Oper. Res.*, 47(3):361–376, 1999.
- [131] C. S. Tang. A max-min allocation problem: its solutions and applications. *Oper. Res.*, 36(2):359–367, 1988.

- [132] A. Thue. Über unendliche Zeichenreihen. *Norske Vid Selsk. Skr. I. Mat. Nat. Kl. Christiana*, 7:1–22, 1906.
- [133] L. Tsai. Asymptotic analysis of an algorithm for balanced parallel processor scheduling. *SIAM J. Comput.*, 21(1):59–64, 1992.
- [134] K. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *STOC '10*, pages 641–648, 2010.
- [135] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.
- [136] S. Wang, J. Min, and B. K. Yi. Location based services for mobiles: Technologies and standards. *ICC '08*, 2008.
- [137] G. Woeginger. A comment on scheduling two parallel machines with capacity constraints. *Discrete Optimization*, 2(3):269–272, 2005.
- [138] L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:358–393, 1982.
- [139] H. Yang, Y. Ye, and J. Zhang. An approximation algorithm for scheduling two parallel machines with capacity constraints. *Discrete Appl. Math.*, 130(3):449–467, 2003.
- [140] F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS '95*, page 374, 1995.
- [141] H. P. Young. Equity. In *Princeton University Press, Princeton, NJ*, 1994.

- [142] F. Zahradnik. Garmin to offer free, advertising-supported traffic detection and avoidance. <http://gps.about.com/b/2008/09/15/garmin-to-offer-free-advertising-supported-traffic-detection-and-avoidance.htm>.
- [143] J. Zhang, B. Chen, and Y. Ye. Multi-exchange local search algorithm for capacitated facility location problem. In *Mathematics of Operations Research*, pages 389–403, 2004.
- [144] J. Zhang and Y. Ye. On the Budgeted MAX-CUT problem and its Application to the Capacitated Two-Parallel Machine Scheduling. Technical report.