Linear and Convex Programming Based Algorithms for Network Design

by

Xiangkun Shen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in the University of Michigan
2019

Doctoral Committee:

Assistant Professor Viswanath Nagarajan, Chair
Professor Xiuli Chao
Professor Marina A. Epelman
Professor Jon Lee
Professor Seth Pettie

Xiangkun Shen

xkshen@umich.edu

ORCID iD:0000-0002-9753-9461

To my family.

ACKNOWLEDGEMENTS

I am deeply thankful to my advisor Viswanath Nagarajan for the guidance and patience in the past four years. It has been my greatest privilege to work with you as a graduate student. Your endless passion for research and gracious care for students have inspired me to work harder to hopefully become someone like you.

My appreciation also goes to my collaborators, Jon Lee, Anupam Gupta, Amit Kumar, and Martin Koutecký. The discussions of research with you are fascinating.

I also thank my thesis committee members, Xiuli Chao, Marina Epelman, Jon Lee, and Seth Pettie. Your valuable advice and intelligent humor make the writing of this thesis more enjoyable.

The industrial and operations engineering at the University of Michigan is a great home. Many thanks go to my amazing graduate student colleagues and friends Qi, Hao, Zhiyuan, Qianyi, Yiling, Yidu, Minseok, Karmel, Niusha and many others. All the ups and downs we have been sharing make the graduate student life one of the greatest experience for me. Special thanks to my fiancee, Yuanyuan, I am so fortunate in my life to meet you here.

I would like to thank the scalable machine learning group in Yahoo! Research for a summer internship in 2018. Especially to Maxim Sviridenko and Dimitris Fotakis for making it a productive and learning experience.

Finally, I would like to express my deepest gratitude to my parents who encourage and support me to chase my own life and dream.

TABLE OF CONTENTS

Dedication . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Figures . . . . . **vii**

Abstract . . . . . **viii**

Chapter

1 Introduction **1**

2 Max-Cut under Graph Constraints **5**

2.1 Introduction 5
2.1.1 Results and Techniques 6
2.1.2 Related Work 7
2.2 Preliminaries 9
2.2.1 Tree Decomposition 9
2.2.2 Constraint Satisfaction Problem 9
2.2.3 Monadic Second Order Logic 10
2.2.4 Dynamic Program 11
2.2.5 Dynamic Program for CSP 12
2.2.6 Sherali-Adams LP Hierarchy 15
2.3 The Max-Cut Setting 16
2.3.1 Linear Program 17
2.3.2 The Rounding Algorithm 19
2.4 The Max-k-Cut Setting 26
2.4.1 LP Relaxation for Max-k-Cut 27
2.4.2 The Rounding Algorithm 29
2.5 Applications 34
2.5.1 Max-k-Cut Applications 34
2.5.2 Applications with Specific Dynamic Programs 34
2.5.3 Bounded-genus and Excluded-minor Graphs 42
2.6 Conclusion 45

3 Online Covering with ℓ_q-Norm Objectives **46**

3.1 Introduction 46
3.1.1 Results and Techniques 48

3.1.2 Related Work . . . 52
3.2 Preliminaries . . . 53
3.2.1 Dual Packing Problem . . . 54
3.2.2 Disjointedness Assumption on S_es . . . 55
3.3 Algorithm and analysis . . . 57
3.4 Applications . . . 61
3.4.1 Online Buy-at-Bulk Network Design . . . 61
3.4.2 Throughput Maximization with ℓ_p-Norm Capacities . . . 65
3.5 Conclusion . . . 74

4 Stochastic Load Balancing . . . 75

4.1 Introduction . . . 75
4.1.1 Results and Techniques . . . 76
4.1.2 Related Work . . . 78
4.2 Preliminaries . . . 79
4.2.1 Truncated and Exceptional Random Variables . . . 82
4.2.2 Effective Size and Its Properties . . . 82
4.2.3 Non-Adaptive and Adaptive Solutions . . . 85
4.2.4 Useful Probabilistic Inequalities . . . 86
4.3 Makespan Minimization . . . 86
4.3.1 A New Lower Bound . . . 87
4.3.2 The Rounding . . . 89
4.3.3 The Analysis . . . 90
4.4 Budgeted Makespan Minimization . . . 93
4.4.1 Proof of Theorem 4.4.1 . . . 95
4.5 ℓ_q-norm Objectives . . . 99
4.5.1 Useful Bounds . . . 99
4.5.2 Reduction to a Deterministic Scheduling Problem . . . 101
4.5.3 Approximation Algorithm for q-DETSCHED . . . 101
4.5.4 Interpreting the Rounded Solution . . . 103
4.6 Conclusion . . . 104

Bibliography . . . 106

LIST OF FIGURES

Figures

2.1 Examples of (i) a set T_i and (ii) a set in $\mathcal{P}$. 17
2.2 Example of T_j, $T_{\bar{u}}$, and $T_{\bar{v}}$ 23
2.3 Connected components of $G[S_j]$ 42

3.1 Buy-at-bulk network design example 62
3.2 Throughput maximization with ℓ_p-norm capacities example 66

4.1 Deterministic instance example 90
4.2 Example for the matching instance 98

ABSTRACT

This thesis presents linear and convex programming based algorithms for NP-hard discrete optimization problems, mainly with applications in network design. Network design problems aim to find a minimal/maximal weighted subgraph satisfying given properties. The problems studied include maximum cut, buy-at-bulk network design, throughput maximization, and unrelated machine scheduling. This thesis considers different models of input uncertainty: the traditional deterministic setting, the online setting where inputs arrive over time and the stochastic setting where inputs are drawn from some probability distribution. Our approach to these problems involves solving suitable convex relaxations and then using rounding procedures to convert the fractional solutions to integer solutions. The specific contributions of this thesis include (1) approximation algorithms for a constrained variant of the maximum cut problem using the Sherali-Adams LP hierarchy; (2) online primal-dual algorithms for covering and packing with Lq norm objectives; (3) approximation algorithms for stochastic unrelated machine scheduling.

CHAPTER 1

Introduction

In a network design problem, we are given a weighted graph, where we wish to find a subgraph that satisfies certain properties and minimizes/maximizes total weight of the subgraph. Classic network design problems include Steiner tree [38], buy-at-bulk network design [56], throughput maximization [10], and maximum cut [61]. Most problems in this class are $\mathcal{NP}$-hard. Therefore we do not believe there exists any algorithm that is both efficient (polynomial running time) and optimal. One approach is to solve these problems exactly using algorithms that require exponential time in the worst case. A common method here is to formulate them as integer programs: these are optimization problems with a linear objective and constraints, and integrality restrictions on variables. Then many instances can be solved by commercial solvers. However, when the instances become large, this approach becomes very time-consuming. Therefore another practical approach is to deal with this intractability via heuristics, which are efficient algorithms that compute near-optimal solutions.

The primary focus of this dissertation is to design approximation algorithms, which are heuristics with mathematically rigorous performance guarantees on the gap from optimality (referred to as the approximation ratio). An effective approach to design approximation algorithms is to (1) formulate the problem as an integer program, (2) find a valid linear program (LP) or convex program relaxation and (3) design a rounding algorithm that converts a fractional solution of the LP into an integral solution for the original problem. This dissertation presents algorithms based on this approach for network design problems. In addition to the classic deterministic setting, we also apply this approach in the setting of uncertain input (namely online and stochastic optimization).

This thesis presents algorithms for a suite of discrete optimization problems, with emphasis on the following network design problems.

- *Maximum cut*: This problem is defined on an undirected edge-weighted graph and asks to find a subset of vertices that maximizes the weight of edges going out of

this subset [61]. In this thesis, we study a constrained version of the maximum cut problem, where the constraints are defined by graph properties.

- *Buy-at-bulk network design*: In the buy-at-bulk network design problem [56], we are given a graph with a monotone sub-additive cost function on each edge and a collection of source/destination pairs. The goal is to find a path for each source/destination pair such that the total cost on the edges is minimized. We study the online version of this problem.

- *Throughput maximization:* In the throughput maximization problem [10], we are given a directed graph with edge capacities. There are requests with source/destination pairs that arrive online. The goal is to choose a subset of requests to maximize the number of accepted requests while the number of paths using any edge is not allowed to exceed its capacity. In addition to this classic setting, we consider an extension with capacity constraints on subsets of edges.

- *Unrelated machine scheduling:* This problem is a classic scheduling problem where we want to schedule jobs on machines to minimize the maximum load [95]. In this thesis, we take a stochastic optimization approach where the job sizes are random variables with known distribution to deal with the scenarios where there is uncertainty in the job sizes.

In Chapter 2, we study a key aspect of using LP relaxations in designing approximation algorithms, which is the choice of the LP relaxation. For many problems, the natural LP relaxations have been proved to provide only weak bounds (see e.g. [121, 7]). One approach to get around this difficulty is to investigate systematic procedures to strengthen the relaxations. In Chapter 2, we design approximation algorithms based on the Sherali-Adams LP hierarchy for the graph-constrained maximum cut problem. The maximum cut problem is a fundamental network design problem that involves selecting a subset of nodes that maximizes the number of edges crossing its boundary. In many applications, there is a constraint on the subset of nodes that can be chosen. For example, a connected maximum cut problem can arise from community detection [58] and image segmentation [70]. We study a large class of graph-constrained maximum cut problems and present unified approximation algorithms for them. In our setting, the graph-based constraint is defined on a graph G and the solution vertex set S must satisfy some properties (e.g. independent set, connectivity) in G. Our main results are for the case when G is of bounded treewidth, where we obtain a $\frac{1}{2}$-approximation algorithm. The approximation ratio is the best possible via LP-based algorithms even for the unconstrained problem [35]. We also extend the

algorithm to handle the max-k-cut problem under any monadic second-order (MSO) logic constraint [44]. MSO logic is an important concept in fixed-parameter tractability theory that can express a large class of graph properties including Hamiltonicity, 3-colorability, and connectivity [113]. Our main result is a $\frac{1}{2}$-approximation algorithms for graph-MSO-constrained max-k-cut problems, where the constraint graph has bounded treewidth.

In Chapter 3, we study the use of convex programs to solve online optimization problems [26]. Online optimization is a widely used approach in optimization under uncertainty. Here the algorithm has no prior information on the input which is revealed incrementally over time. For example, in a network design problem, when a pair of source/destination request arrives, the algorithm needs to find a path to satisfy this request immediately, without knowledge of any future requests. The performance of an online algorithm is compared to an optimum that knows the entire input upfront. So, in addition to computational complexity, an algorithm needs to deal with the lack of information. The online primal-dual approach is widely used for designing online algorithms and leads to algorithms for covering/packing LPs. There has been a lot of effort in obtaining good online algorithms for various classes of continuous optimization problems, see e.g. [5, 30, 65, 30, 11, 12]. When combined with suitable rounding algorithms, they lead to online algorithms for various problems, e.g. set cover [6], facility location [5], machine scheduling [11], caching [16], and buy-at-bulk network design [56]. In Chapter 3, we design an online algorithm for fractional covering and packing problems with sums of ℓ_q-norm objectives, which significantly extends the online primal-dual approach. We obtain a logarithmic competitive ratio, which is nearly tight even for the special case of a linear objective. As direct applications we obtain (1) improved online algorithms for non-uniform buy-at-bulk network design [34] and (2) the first online algorithm for throughput maximization [10, 30] under ℓ_q-norm edge capacities.

In Chapter 4, we study another approach to deal with input uncertainty, i.e., stochastic optimization. In the stochastic setting, certain parts of the input are represented by random variables, and the algorithm needs to optimize the expected performance [23]. One common approach to design approximation algorithms for stochastic models is to find deterministic surrogates for random variables and then solve a certain deterministic problem with the surrogates. The difficulty, however, is to come up with the correct deterministic surrogate and establish a connection to the resulting deterministic problem. In Chapter 4, we use this approach to give the first constant-factor approximation algorithm for the general stochastic load balancing problem. The deterministic version is well-studied with a known constant-factor approximation even in the most general case [95, 117]. In the stochastic setting, however, the general problem remained open until our work. Prior

results were limited to identical machines where each job is required to have the same random size on all the machines. A constant-factor approximation algorithm for identical machines was given [88] and better results for special classes of job size distributions were known [60]. We utilize an exponential-sized LP relaxation of the stochastic problem and use the solution of the LP to come up with a deterministic surrogate for the job sizes. We note that these techniques can also be extended to some network design problems such as unsplittable flow problem on the line/tree.

CHAPTER 2

Max-Cut under Graph Constraints

2.1 Introduction

The max-cut problem is an extensively studied combinatorial-optimization problem. Given an undirected edge-weighted graph, the goal is to find a subset $S \subseteq V$ of vertices that maximizes the weight of edges in the cut $(S, V \setminus S)$. Max-cut has a 0.878-approximation algorithm [61] which is known to be best-possible assuming the *unique games conjecture* [86]. It also has a number of practical applications, e.g., in circuit layout [37], statistical physics [18] and clustering [81, 94].

In some applications, one needs to solve the max-cut problem under additional constraints on the subset S. Consider, for example, the following community detection problem. The input is an undirected graph $G = (V, E)$ representing, say, a social network (vertices V denote users and edges E denote connections between users), and a weight function $c : \binom{V}{2} \to \mathbb{R}_+$ representing, a dissimilarity measure between pairs of users. The goal is to find a subset $S \subseteq V$ of users that are connected in G while maximizing the weight of edges in the cut $(S, V \setminus S)$. This corresponds to finding a cluster of connected users that is as different as possible from its complement set. Many community detection problems involve such a connectivity constraint; see, e.g., [58]. This "connected max-cut" problem also arises in image segmentation applications [70, 120].

Designing algorithms for constrained versions of max-cut is also interesting from a theoretical standpoint. For max-cut under certain types of constraints (such as cardinality or matroid constraints) good approximation algorithms are known, e.g., [3, 4]. In fact, many of these results have since been extended to the more general setting of submodular objectives [43, 57]. However, not much is known for max-cut under "graph-based" constraints as in the example above.

In this chapter, we study a large class of graph-constrained max-cut problems and present unified approximation algorithms for them. Our results require that the constraint

is defined on a graph G of bounded treewidth. (Treewidth is a measure of how similar a graph is to a tree structure — see Section 2.2 for definitions.) We note however that for a number of constraints, we can combine our algorithm with known decomposition results [47, 48] to obtain essentially the same approximation ratios when the constraint graph G is bounded-genus or excluded-minor.

Definition 2.1.1 (GCMC)**.** *The input to the* graph-constrained max-cut *(*GCMC*) problem consists of (i) an n-vertex graph $G = (V, E)$ with a graph property which implicitly specifies a collection $\mathcal{S}_G$ of vertex subsets, and (ii) symmetric edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$. The* GCMC *problem is then as follows:*

$$\max_{S \in \mathcal{S}_{GG}} \sum_{u \in S, v \notin S} c(u, v). \tag{2.1}$$

In this chapter, we assume that the constraint graph G has bounded treewidth. And we are interested in constraints that are specified by an auxiliary constraint graph, especially the graph constraint that can be expressed in monadic second-order logic (MSO) (see [44]). MSO logic is an important concept in fixed-parameter tractability theory [113]. It can express a large class of graph properties including Hamiltonicity, 3-colorability, connectivity, independent set, vertex cover. Section 2.2 gives more detailed description of MSO logic.

2.1.1 Results and Techniques

Our main result can be stated as follows.

Theorem 2.1.2 (GCMC result)**.** *Consider any instance of the* GCMC *problem on a bounded-treewidth graph $G = (V, E)$. Suppose the constraint $\mathcal{S}_G$ can be expressed by an* MSO *formula. Then we obtain a $\frac{1}{2}$-approximation algorithm for* GCMC*.*

Our algorithm uses a linear-programming relaxation for GCMC based on the dynamic program which is further strengthened via the Sherali-Adams LP hierarchy. The resulting LP has polynomial size whenever the number of dynamic program states associated with a single tree-decomposition node is constant (see Section 2.2 for the formal definition).[1] The rounding algorithm is a natural top-down procedure that randomly chooses a "state" for each tree-decomposition node using the LP's probability distribution conditioned on the choices at its ancestor nodes and their siblings. The complete solution is obtained by combining the chosen states at each tree-decomposition node, which is guaranteed to satisfy constraint $\mathcal{S}_G$ due to properties of the dynamic program.

[1] For other polynomial-time dynamic programs, the LP has *quasi-polynomial* size.

The requirements in Theorem 2.1.2 on the graph constraint $\mathcal{S}_G$ are satisfied by several interesting constraints and thus we obtain approximation algorithms for all these GCMC problems. See Section 2.5 for details.

Theorem 2.1.3 (Applications of GCMC)**.** *There is a $\frac{1}{2}$-approximation algorithm for* GCMC *under the following constraints in a bounded-treewidth graph: independent set, vertex cover, precedence, dominating set, connectivity.*

We note that many other constraints such as connected dominating set, and triangle matching also satisfy our requirement.

For many of the constraints above, we can use known decomposition results [48, 47] to obtain approximation algorithms for GCMC when the constraint graph has bounded genus or excludes some fixed minor (e.g., planar graphs).

Corollary 2.1.3.1. *There is a $(\frac{1}{2} - \epsilon)$-approximation algorithm for* GCMC *under the following constraints in an excluded-minor graph: independent set, vertex cover, dominating set. Here $\epsilon > 0$ is a fixed constant.*

Corollary 2.1.3.2. *There is a $(\frac{1}{2} - \epsilon)$-approximation algorithm for connected max-cut in a bounded-genus graph. Here $\epsilon > 0$ is a fixed constant.*

We also extend these results to the setting of max-k-cut, where we seek to partition the vertices into k parts $\{U_i\}_{i=1}^k$ so as to maximize the weight of edges crossing the partition. In the constrained version, we additionally require each part U_i to satisfy some MSO graph property. We obtain a $\frac{1}{2}$-approximation algorithm even in this setting (k is fixed).

A k-partition of vertex set V is a function $h : V \to [k]$, where the k parts are $U_\alpha = \{v \in V : h(v) = \alpha\}$ for $\alpha \in [k]$. Note that $\cup_{\alpha=1}^k U_\alpha = V$ and $U_1, \cdots, U_k$ are disjoint. When we want to refer to the k parts directly, we also use $\{U_\alpha\}_{\alpha=1}^k$ to denote the k-partition.

Definition 2.1.4 ($\mathsf{GCMC_k}$)**.** *The input to the* graph-constrained max-k-cut *problem consists of (i) an n-vertex graph $G = (V, E)$ with a graph property which implicitly specifies a collection $\mathcal{S}_G$ of vertex k-partitions, and (ii) symmetric edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$. The* $\mathsf{GCMC_k}$ *problem is to find a k-partition in $\mathcal{S}_G$ with the maximum weight of crossing edges:*

$$\max_{h \in \mathcal{S}_G} \sum_{\{u,v\} \in \binom{V}{2}, h(u) \neq h(v)} c(u, v). \tag{2.2}$$

2.1.2 Related Work

For the basic undirected max-cut problem, there is an elegant 0.878-approximation algorithm [61] via semidefinite programming. This is also the best one can hope for, assuming the *unique games conjecture* [86].

Most of the prior work on constrained max-cut has focused on cardinality, matroid and knapsack constraints [3, 4, 43, 57, 92, 93]. Constant-factor approximation algorithms are known for max-cut under the intersection of any constant number of such constraints — these results hold in the substantially more general setting of non-negative submodular functions. The main techniques used here are local search and the multilinear extension [32] of submodular functions. These results made crucial use of certain exchange properties of the underlying constraints, which are not true for the graph-based constraints that we consider.

Closer to our setting, a version of the connected max-cut problem was studied recently in [70], where the connectivity constraint, as well as the weight function, were defined on the *same* graph G. The authors obtained an $O(\log n)$-approximation algorithm for general $\{0,1\}$-weighted graphs, an $O(\log^2 n)$-approximation algorithm for general weighted graphs, and an exact algorithm on bounded-treewidth graphs (which implied a PTAS for bounded-genus graphs); their algorithms relied heavily on the uniformity of the constraint and weight graphs. In contrast, we consider the connected max-cut problem where the connectivity constraint and the weight function are *unrelated*; in particular, our problem generalizes max-cut even when G is a trivial graph (e.g., a star). Moreover, our algorithms work for a much wider class of constraints. We note however that our results require the graph G to have bounded treewidth — this is also necessary because some of the constraints we consider (e.g., independent set) are inapproximable in general graphs [71]. For connected max-cut itself, obtaining a non-trivial approximation ratio when G is a general graph remains an open question.

In terms of techniques, the closest work to ours is [68]. We use ideas from [68] in formulating the (polynomial size) Sherali-Adams LP as well as in the rounding algorithm. There are important differences too, as discussed in Section 2.1.1.

A recent result [35] showed that any polynomial-sized linear program (that only depends on the input size) for unconstrained max-cut has an integrality gap of 1/2. As GCMC generalizes unconstrained Max-Cut, we cannot obtain a better approximation ratio directly using LPs.

Finally, our result adds to a somewhat small list [**?**, 20, 22, 59, 68, 99] of algorithmic results based on the Sherali-Adams [116] LP hierarchy. We are not aware of a more direct approach to obtain a constant-factor approximation algorithm even for connected max-cut when the constraint graph G is a tree.

2.2 Preliminaries

2.2.1 Tree Decomposition

Given an undirected graph $G = (V, E)$, a tree decomposition consists of a tree $\mathcal{T} = (I, F)$ and a collection of vertex subsets $\{X_i \subseteq V\}_{i \in I}$ such that:

- for each $v \in V$, the nodes $\{i \in I : v \in X_i\}$ are connected in $\mathcal{T}$,
- for each edge $(u, v) \in E$, there is some node $i \in I$ with $u, v \in X_i$.

The width of such a tree decomposition is $\max_{i \in I}(|X_i| - 1)$, and the treewidth of G is the smallest width of any tree decomposition for G.

We work with "rooted" tree decompositions, also specifying a root node $r \in I$. The depth d of such a tree decomposition is the length of the longest root-leaf path in $\mathcal{T}$. The depth of any node $i \in I$ is the length of the $r - i$ path in $\mathcal{T}$. For any $i \in I$, the set V_i denotes all the vertices at or below node i, that is

$$V_i := \cup_{k \in \mathcal{T}_i} X_k, \text{where } \mathcal{T}_i = \{k \in I : k \text{ in subtree of } \mathcal{T} \text{ rooted at } i\}.$$

The following result provides a convenient representation of $\mathcal{T}$.

Theorem 2.2.1 (Balanced Tree Decomposition; see [25])**.** *Let $G = (V, E)$ be a graph with tree decomposition $(\mathcal{T} = (I, F), \{X_i | i \in I\})$ of treewidth k. Then G has a rooted tree decomposition $(\mathcal{T}' = (I', F'), \{X'_i | i \in I'\})$ where $\mathcal{T}'$ is a binary tree of depth $2\lceil \log_{\frac{5}{4}}(2|V|) \rceil$ and treewidth at most $3k+2$. Moreover, for all $i \in I$, there is an $i' \in I'$ such that $X_i \subseteq X'_{i'}$. The tree decomposition $\mathcal{T}'$ can be found in $O(|V|)$ time.*

2.2.2 Constraint Satisfaction Problem

Definition 2.2.2 (CSP instance)**.** *An instance of Constraint Satisfaction Problem (CSP) $J = (N, \mathcal{C})$ consists of:*

- *a set N of* boolean variables*,*
- *a set $\mathcal{C}$ of* constraints*, where each constraint $C_U \in \mathcal{C}$ is a $|U|$-ary relation $C_U \subseteq \{0,1\}^U$ on some subset $U \subseteq N$.*

For a vector $x \in \{0,1\}^N$ and a subset R of variables, we denote by $x|_R$ the *restriction of x to R*. A vector $z \in \{0,1\}^N$ *satisfies constraint $C_U \in \mathcal{C}$* if $z|_U \in C_U$. We say that $z \in \{0,1\}^N$ is *a feasible assignment* for the CSP instance J if z satisfies every constraint

$C \in \mathcal{C}$. Let $\mathrm{Feas}(J)$ be the set of all feasible assignments of J. Finally, $\|\mathcal{C}\| = \sum_{C_U \in \mathcal{C}} |C_U|$ denotes the *length of* $\mathcal{C}$.

Definition 2.2.3 (Constraint graph)**.** *The constraint graph of J, denoted $G(J)$, is defined as $G(J) = (N, F)$ where $F = \{\{u, v\} \mid \exists C_U \in \mathcal{C}$ s.t. $\{u, v\} \subseteq U\}$.*

Definition 2.2.4 (Treewidth of CSP)**.** *The treewidth $\mathrm{tw}(J)$ of a CSP instance J is defined as the treewidth of its constraint graph $\mathrm{tw}(G(J))$.*

Definition 2.2.5 (CSP extension)**.** *Let $J = (N, \mathcal{C})$ be a CSP instance. We say that $J' = (N', \mathcal{C}')$ with $N \subseteq N'$ is an extension of J if $\mathrm{Feas}(J) = \big\{z|_N \;\big|\; z \in \mathrm{Feas}(J')\big\}$.*

2.2.3 Monadic Second Order Logic

We briefly introduce MSO over graphs. In *first-order logic* (FO) we have variables for individual vertices/edges (denoted $x, y, \ldots$), equality for variables, quantifiers $\forall, \exists$ ranging over variables, and the standard Boolean connectives $\neg, \wedge, \vee, \implies$. MSO is the extension of FO by quantification over sets (denoted $X, Y, \ldots$). Graph MSO has the binary relational symbol $\mathrm{edge}(x, y)$ encoding edges, and traditionally comes in two flavours, MSO_1 and MSO_2, differing by the objects we are allowed to quantify over: in MSO_1 these are the vertices and vertex sets, while in MSO_2 we can additionally quantify over edges and edge sets. For example, 3-colorability can be expressed in MSO_1 as follows:

$$\exists X_1, X_2, X_3 :$$
$$\big[\, \forall x \, (x \in X_1 \vee x \in X_2 \vee x \in X_3) \wedge \bigwedge\nolimits_{i=1,2,3} \forall x, y \, (x \notin X_i \vee y \notin X_i \;\vee \neg\, \mathrm{edge}(x, y)) \,\big].$$

We remark that MSO_2 can express properties that are not MSO_1 definable. As an example, consider Hamiltonicity on graph $G = (V, E)$; an equivalent description of a Hamiltonian cycle is that it is a connected 2-factor of a graph:

$$\begin{aligned}
\varphi_{\text{ham}} &\equiv \exists F \subseteq E : \varphi_{\text{2-factor}}(F) \wedge \varphi_{\text{connected}}(F), \\
\varphi_{\text{2-factor}}(F) &\equiv (\forall v \in V : \exists e, f \in F : (e \neq f) \wedge (v \in e) \wedge (v \in f)) \wedge \neg(\exists v \in V : \\
&\qquad \exists e, f, g \in F : (e \neq f \neq g) \wedge (v \in e) \wedge (v \in f) \wedge (v \in g)), \\
\varphi_{\text{connected}}(F) &\equiv \neg\big[\exists U, W \subseteq F : (U \cap W = \emptyset) \wedge (U \cup W = V) \wedge \neg\big(\exists \{u, v\} \in E : \\
&\qquad u \in U \wedge v \in W\big)\big] \;.
\end{aligned}$$

We use φ to denote an MSO formula and $G = (V, E)$ for the underlying graph. For a formula φ, we denote by $|\varphi|$ the *size* (number of symbols) of φ.

In order to express constraints on k-vertex-partitions via MSO, we use MSO formulas φ with k free variables $\{U_\alpha\}_{\alpha=1}^k$ where (i) the U_α are enforced to form a partition of the vertex-set V, and (ii) each U_α satisfies some individual MSO constraint φ_α. Because k is constant, the size of the resulting MSO formula is a constant as long as each of the MSO constraints φ_α has constant size.

Connecting CSP and MSO Consider an MSO formula φ with k free variables on graph G (as above). For a vector $t \in \{0,1\}^{V \times [k]}$, we write $G, t \models \varphi$ if and only if φ is satisfied by solution $U_\alpha = \{v \in V : t((v,\alpha)) = 1\}$ for $\alpha \in [k]$.

Definition 2.2.6 ($CSP_\varphi(G)$ instance)**.** *Let G be a graph and φ be an MSO_2-formula with k free variables. By $CSP_\varphi(G)$ we denote the CSP instance $(N, \mathcal{C})$ with $N = \{t((v,\alpha)) \mid v \in V(G), \alpha \in [k]\}$ and with a single constraint $\{t \mid G, t \models \varphi\}$.*

Observe that $\mathrm{Feas}(CSP_\varphi(G))$ corresponds to the set of feasible assignments of φ on G. Also, the treewidth of $CSP_\varphi(G)$ is $|V|k$ which is unbounded. The following result shows that there is an equivalent CSP extension that has constant treewidth.

Theorem 2.2.7 ([89, Theorem 25])**.** *Let $G = (V, E)$ be a graph with $\mathrm{tw}(G) = \tau$ and φ be an MSO_2-formula with k free variables. Then $CSP_\varphi(G)$ has a CSP extension J with $\mathrm{tw}(J) \leq f(|\varphi|, \tau)$ and $\|\mathcal{C}_J\| \leq f(|\varphi|, \tau) \cdot |V|$.*

To be precise, [89, Theorem 25] speaks of MSO_1 over σ_2-structures, which is equivalent to MSO_2 over graphs; cf. the discussion in [89, Section 2.1].

2.2.4 Dynamic Program

The algorithm in this chapter works if the constraint $\mathcal{S}_G$ admits an exact dynamic programming (DP) algorithm of a specific form. Then we will show that any MSO expressible constraint $\mathcal{S}_G$ admits such DP.

Definition 2.2.8 (Dynamic Program)**.** *With a tree decomposition $(\mathcal{T} = (I, F), \{X_i | i \in I\})$, we associate the following:*

1. *For each node $i \in I$, there is a state space Σ_i.*
2. *For each node $i \in I$ and $\sigma \in \Sigma_i$, there is a collection $\mathcal{H}_{i,\sigma} \subseteq 2^{V_i}$ of subsets.*
3. *For each node $i \in I$, its children nodes $\{j, j'\}$ and $\sigma \in \Sigma_i$, there is a collection $\mathcal{F}_{i,\sigma} \subseteq \Sigma_j \times \Sigma_{j'}$ of valid combinations of children states.*

Assumption 2.2.1 (Dynamic Program for $\mathcal{S}_G$)**.** *Let $(\mathcal{T} = (I, F), \{X_i | i \in I\})$ be any tree decomposition. Then there exist Σ_i, $\mathcal{F}_{i,\sigma}$ and $\mathcal{H}_{i,\sigma}$ (see Definition 2.2.8) that satisfy the following conditions:*

1. (bounded state space) *Σ_i and $\mathcal{F}_{i,\sigma}$ are all bounded by constant, that is,* $\max_i |\Sigma_i| = O(1)$ *and* $\max_{i,\sigma} |\mathcal{F}_{i,\sigma}| = O(1)$.

2. (required state) *For each $i \in I$ and $\sigma \in \Sigma_i$, the intersection with X_i of every set in $\mathcal{H}_{i,\sigma}$ is the same, denoted $X_{i,\sigma}$, that is $S \cap X_i = X_{i,\sigma}$ for all $S \in \mathcal{H}_{i,\sigma}$.*

3. *By condition 2, for any leaf $\ell \in I$ and $\sigma \in \Sigma_\ell$, we have $\mathcal{H}_{\ell,\sigma} = \{X_{\ell,\sigma}\}$ or $\emptyset$.*

4. (subproblem) *For each non-leaf node $i \in I$ with children $\{j, j'\}$ and $\sigma \in \Sigma_i$,*

$$\mathcal{H}_{i,\sigma} \quad = \quad \Big\{ X_{i,\sigma} \cup S_j \cup S_{j'} \ : \ S_j \in \mathcal{H}_{j,w_j}, S_{j'} \in \mathcal{H}_{j',w_{j'}}, \, (w_j, w_{j'}) \in \mathcal{F}_{i,\sigma} \Big\}.$$

5. (feasible subsets) *At the root node r, we have $\mathcal{S}_G = \bigcup_{\sigma \in \Sigma_r} \mathcal{H}_{r,\sigma}$.*

2.2.5 Dynamic Program for CSP

In this section we demonstrate that every CSP of bounded treewidth admits a dynamic program that satisfies Assumption 2.2.1.

Consider a CSP instance $J = (V, \mathcal{C})$ with a constraint graph $G = (V, E)$ of bounded treewidth. Let $(\mathcal{T} = (I, F), \{X_i | i \in I\})$ denote a balanced tree decomposition of G (from Theorem 2.2.1). In what follows, we denote the vertex set $V = [n] = \{1, 2, \cdots, n\}$. Let λ be a symbol denoting an unassigned value. For any $W \subseteq V$, define the *set of configurations of W* as:

$$\begin{aligned}\mathcal{K}(W) = \big\{ (z_1, \ldots, z_n) \in \{0, 1, \lambda\}^V \, | \forall C_U \in \mathcal{C} : (U \subseteq W \implies z|_U \in C_U), \\ \forall i \notin W : z_i = \lambda, \, \forall j \in W : z_j \in \{0, 1\} \big\}\end{aligned}$$

Let $k \in \mathcal{K}(W)$ be a configuration and $v \in V$. Because k is a vector, $k(v)$ refers to the v-th element of k.

Definition 2.2.9 (State Operations)**.** *Let $U, W \subseteq V$. Let $k \in \mathcal{K}(U)$ and $p \in \mathcal{K}(W)$.*

- *Configurations k and p are said to be* consistent *if, for each $v \in V$, either $k(v) = p(v)$ or at least one of $k(v), p(v)$ is λ.*
- *If configurations k and p are consistent, define* $[p \cup k](v) = \begin{cases} p(v), & \text{if } k(v) = \lambda; \\ k(v), & \text{otherwise.} \end{cases}$

- *Define* $[k \cap W](v) = \begin{cases} k(v), & \text{if } v \in W; \\ \lambda, & \text{otherwise.} \end{cases}$

We start by defining some useful parameters for the dynamic program.

Definition 2.2.10. *For each node $i \in I$ with children nodes $\{j, j'\}$, we associate the following:*

1. *state space $\Sigma_i = \mathcal{K}(X_i)$.*
2. *for each $\sigma \in \Sigma_i$, there is a collection of partial solutions*

$$\mathcal{H}_{i,\sigma} := \{k \in \mathcal{K}(V_i) \mid k \cap X_i = \sigma\}.$$

3. *for each $\sigma \in \Sigma_i$, there is a collection of valid combinations of children states*

$$\mathcal{F}_{i,\sigma} = \{(\sigma_j, \sigma_{j'}) \in \Sigma_j \times \Sigma_{j'} \mid (\sigma_j \cap X_i) = (\sigma \cap X_j) \text{ and } (\sigma_{j'} \cap X_i) = (\sigma \cap X_{j'})\}.$$

In words, (1) Σ_i is just the set of configurations for the vertices X_i in node i, (2) $\mathcal{H}_{i,\sigma}$ are those configurations for the vertices V_i (in the subtree rooted at i) that are consistent with σ, (3) $\mathcal{F}_{i,\sigma}$ are those pairs of states at the children $\{j, j'\}$ that agree with σ on the intersections $X_i \cap X_j$ and $X_i \cap X_{j'}$ respectively.

Theorem 2.2.11 (Dynamic Program for CSP)**.** *Let $(\mathcal{T} = (I, F), \{X_i | i \in I\})$ be a tree decomposition of a CSP instance $(V, \mathcal{C})$ of bounded treewidth. Then Σ_i, $\mathcal{F}_{i,\sigma}$ and $\mathcal{H}_{i,\sigma}$ from Definition 2.2.8 satisfy the conditions:*

1. (bounded state space) *Σ_i and $\mathcal{F}_{i,\sigma}$ are all bounded by constant, that is,* $\max_i |\Sigma_i| = O(1)$ *and* $\max_{i,\sigma} |\mathcal{F}_{i,\sigma}| = O(1)$.
2. (required state) *For each $i \in I$ and $\sigma \in \Sigma_i$, the intersection with X_i of every vector in $\mathcal{H}_{i,\sigma}$ is the same, in particular $h \cap X_i = \sigma$ for all $h \in \mathcal{H}_{i,\sigma}$.*
3. *By condition 2, for any leaf $\ell \in I$ and $\sigma \in \Sigma_\ell$, we have $\mathcal{H}_{\ell,\sigma} = \{\sigma\}$ or $\emptyset$.*
4. (subproblem) *For each non-leaf node $i \in I$ with children $\{j, j'\}$ and $\sigma \in \Sigma_i$,*

$$\mathcal{H}_{i,\sigma} = \Big\{\sigma \cup h_j \cup h_{j'} \mid h_j \in \mathcal{H}_{j,w_j},\ h_{j'} \in \mathcal{H}_{j',w_{j'}},\ (w_j, w_{j'}) \in \mathcal{F}_{i,\sigma}\Big\}.$$

5. (feasible subsets) *At the root node r, we have* $\mathrm{Feas}(V, \mathcal{C}) = \bigcup_{\sigma \in \Sigma_r} \mathcal{H}_{r,\sigma}$.

Proof. Let $q = O(1)$ denote the treewidth of $\mathcal{T}$. We prove each of the claimed properties.

Bounded state space. Because $|X_i| \leq q+1$, we have $|\Sigma_i| = |\mathcal{K}(X_i)| \leq 3^{q+1} = O(1)$ and $|\mathcal{F}_{i,\sigma}| \leq |\Sigma_j \times \Sigma_{j'}| \leq (3^{q+1})^2 = O(1)$.

Required state. This holds immediately by definition of $\mathcal{H}_{i,\sigma}$ in Definition 2.2.8.

Subproblem. We first prove the "$\subseteq$" inclusion of the statement. Consider any $h \in \mathcal{H}_{i,\sigma} \subseteq \mathcal{K}(V_i)$. Let $h_j = h \cap V_j$, $w_j = h \cap X_j$ and analogously for j'. Observe that for $U \subset W \subseteq V$ we have that $k \in \mathcal{K}(W) \implies k \cap U \in \mathcal{K}(U)$. By this observation, $h_j \in \mathcal{K}(V_j)$. Moreover, $h_j \cap X_j = h \cap X_j = w_j$, which implies $h_j \in \mathcal{H}_{j,w_j}$. Again, the same applies for j' and we have $h_{j'} \in \mathcal{H}_{j',w_{j'}}$. Finally, note that $w_j \cap X_i = h \cap X_j \cap X_i = (h \cap X_i) \cap X_j = \sigma \cap X_j$ and similarly $w_{j'} \cap X_i = \sigma \cap X_{j'}$. So we have $(w_j, w_{j'}) \in \mathcal{F}_{i,\sigma}$.

Now, we prove the "$\supseteq$" inclusion of the statement. Consider any two partial solutions $h_j \in \mathcal{H}_{j,w_j}$ and $h_{j'} \in \mathcal{H}_{j',w_{j'}}$ with $(w_j, w_{j'}) \in \mathcal{F}_{i,\sigma}$. Note that h_j and σ (similarly $h_{j'}$ and σ) are consistent by definition of $\mathcal{F}_{i,\sigma}$. We now claim that h_j and $h_{j'}$ are also consistent: take any $v \in V$ with both $h_{j'}(v), h_{j'}(v) \neq \lambda$, then we must have $v \in V_j \cap V_{j'} \subseteq X_i \cap X_j \cap X_{j'}$ as X_i is a vertex separator, and so $h_j(v) = \sigma(v) = h_{j'}(v)$ by definition of $\mathcal{F}_{i,\sigma}$. Because σ, h_j and $h_{j'}$ are mutually consistent, $h = \sigma \cup h_j \cup h_{j'}$ is well-defined. It is clear from the above arguments that $h \cap X_i = \sigma$. In order to show $h \in \mathcal{H}_{i,\sigma}$ we now only need $h \in \mathcal{K}(V_i)$, that is, h does not violate any constraint that is contained in V_i. For contradiction assume that that there is such a violated constraint C_S with $S \subseteq V_i$. Then S induces a clique in the constraint graph G and thus there must exist a node k among the descendants of i such that $S \subseteq V_k$. But k cannot be in the subtree rooted in j or j', because then C_S would have been violated already in h_j or $h_{j'}$, and also it cannot be that $i = k$, because then C_S would be violated in σ, a contradiction.

Feasible subsets. Clearly, the set $\mathrm{Feas}(V, \mathcal{C})$ of feasible CSP solutions is equal to $\mathcal{K}(V)$. Because $\mathcal{H}_{r,\sigma}$ is those $k \in \mathcal{K}(V)$ with $k \cap X_r = \sigma$, the claim follows. □

Example Here we outline how independent set satisfies the above requirements.

- The state space of each node $i \in I$ consists of all independent subsets of X_i.
- The subsets $\mathcal{H}_{i,\sigma}$ consist of all independent subsets $S \subseteq V_i$ with $S \cap X_i = \sigma$.
- The valid combinations $\mathcal{F}_{i,\sigma}$ consist of all tuples $(w_j, w_{j'})$ where the child states w_j and $w_{j'}$ are "consistent" with state σ at node i. Here "consistent" means both w_j and $w_{j'}$ make the same choice as σ on the vertices of X_i, formally, $w_j \cap X_i = \sigma \cap X_j$ and $w_{j'} \cap X_i = \sigma \cap X_{j'}$.

A formal proof appears in Section 2.5. There, we also discuss a number of other graph constraints satisfying our assumption.

The following result follows from Assumption 2.2.1.

Claim 2.2.12. *For any $S \in \mathcal{S}_G$, there is a collection $\{b(i) \in \Sigma_i\}_{i \in I}$ of states such that:*

- *for each node $i \in I$ with children j and j', $(b(j), b(j')) \in \mathcal{F}_{i,b(i)}$,*
- *for each leaf ℓ we have $\mathcal{H}_{\ell,b(\ell)} \neq \emptyset$, and*
- *$S = \bigcup_{i \in I} X_{i,b(i)}$.*

Moreover, for any vertex $u \in V$, if $\bar{u} \in I$ denotes the highest node in $\mathcal{T}$ containing u then we have: $u \in S$ if and only if $u \in X_{\bar{u},b(\bar{u})}$.

Proof. We define the states $b(i)$ in a top-down manner; we will also define an associated subset $B_i \in \mathcal{H}_{i,b(i)}$ at each node i. At the root, we set $b(r) = \sigma$ such that $S \in \mathcal{H}_{r,\sigma}$: this is well-defined by Assumption 2.2.1(5) because $S \in \mathcal{S}_G$. We also set $B_r = S$. Having set $b(i)$ and $B_i \in \mathcal{H}_{i,b(i)}$ for any node $i \in I$ with children $\{j, j'\}$, we use Assumption 2.2.1(4) to write

$$B_i = X_{i,b(i)} \cup S_j \cup S_{j'} \quad \text{where } S_j \in \mathcal{H}_{j,w_j},\ S_{j'} \in \mathcal{H}_{j',w_{j'}} \text{ and } (w_j, w_{j'}) \in \mathcal{F}_{i,b(i)}.$$

Then we set $b(j) = w_j$, $B_j = S_j$ and $b(j') = w_{j'}$, $B_{j'} = S_{j'}$ for the children of node i. The first condition in the claim is immediate from the definition of states $b(i)$. By induction on the depth of node i, we obtain $B_i \in \mathcal{H}_{i,b(i)}$ for each node i. This implies that $\mathcal{H}_{\ell,b(\ell)} \neq \emptyset$ for each leaf ℓ, which proves the second condition; moreover, by Assumption 2.2.1(3) we have $B_\ell = X_{\ell,b(\ell)}$. Now, by definition of the sets B_i, we obtain $S = B_r = \bigcup_{i \in I} X_{i,b(i)}$ which proves the third condition in the claim.

Because $S = \bigcup_{i \in I} X_{i,b(i)}$, it is clear that if $u \in X_{\bar{u},b(\bar{u})}$ then $u \in S$. In the other direction, suppose $u \notin X_{\bar{u},b(\bar{u})}$: we will show $u \notin S$. Since $\bar{u}$ is the highest node containing u, it suffices to show that $u \notin B_{\bar{u}}$. But this follows directly from Assumption 2.2.1(2) because $B_{\bar{u}} \in \mathcal{H}_{\bar{u},b(\bar{u})}$, $u \in X_{\bar{u}}$ and $u \notin X_{\bar{u},b(\bar{u})}$. □

2.2.6 Sherali-Adams LP Hierarchy

This is one of the several "lift-and-project" procedures that, given a $\{0, 1\}$ integer program, produces systematically a sequence of increasingly tighter convex relaxations. The main idea is to use linear constraints to simulate nonlinear constraints. The Sherali-Adams procedure [116, 101] involves generating stronger *LP relaxations* by adding new variables and constraints.

Given a polytope $P \subseteq \mathbb{R}^n$, Sherali-Adams hierarchy produces systematically a sequence of increasingly tighter convex relaxations of given polytopes. The k^{th} round of this procedure has a variable $y(L)$ for every subset L of at most $k+1$ variables in the original integer program — setting the new variable $y(L)$ to one corresponds to the joint event that all the original variables in S are one. Formally, let $P_0 = L_0 = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a convex polytope contained in $[0,1]^n$, defined by m linear constraints, and let $P = conv(P_0 \cap \{0,1\}^n)$ be the associated 0-1 polytope. Starting from P_0, the k^{th} level P_k of the Sherali-Adams hierarchy is obtained as follows.

First, we multiply constraint $a_\ell^T x - b_\ell \geq 0$ by each product $\prod_{i\in I} x_i \prod_{j\in J}(1-x_j)$,where I, J are disjoint subsets of $\{1,\ldots,n\}$ such that $|I \cup J| \leq k$, to produce a set of polynomial inequalities. We add to this set all the inequalities $\prod_{i\in I} x_i \prod_{j\in J}(1-x_j) \geq 0$, where I, J are disjoint subsets of $\{1,\ldots,n\}$ such that $|I \cup J| \leq \min\{k+1, n\}$.

Then, we replace each square x_i^2 by x_i so that each expression is multilinear, and linearize each product monomial $\prod_{\ell\in L} x_\ell$ by replacing it with new variable $y(L)$. This defines a new, lifted polyhedron L_k in the higher-dimensional space $\mathbb{R}^d$, $d = \binom{n}{1} + \cdots + \binom{n}{k+1}$.

Finally, polyhedron P_k is obtained by projecting L_k back onto $\mathbb{R}^n$:

$$P_k = \{x \in \mathbb{R}^n : \exists y \in L_k,\ \forall i = 1,\ldots,n,\ y(\{i\}) = x_i\}.$$

2.3 The Max-Cut Setting

Here, we consider the GCMC problem where we have $k = 2$ and there is a constraint $\mathcal{S}_G$ for only one side of the cut. Recall the definition from Section 2.1.1, i.e., problem (2.1).

In this section, we prove:

Theorem 2.3.1. *Consider any instance of the* GCMC *problem on a bounded-treewidth graph $G = (V, E)$ with $k = 2$. If the graph constraint $\mathcal{S}_G$ is for only one side of the cut and satisfies Assumption 2.2.1 then we obtain a $\frac{1}{2}$-approximation algorithm.*

Algorithm outline: We start with a balanced tree decomposition $\mathcal{T}$ of graph G, as given in Theorem 2.2.1; recall the associated definitions from Section 2.2. Then we formulate an LP relaxation of the problem using the dynamic program in Assumption 2.2.1. This LP is further strengthened by applying the Sherali-Adams operator for $O(\log n)$ rounds. However, in order to obtain a polynomial-size LP, we only use a specific subset of the variables/constraints from the Sherali-Adams LP. The resulting LP can be solved in polynomial time. The rounding algorithm is a natural top-down procedure that relies on Assumption 2.2.1 and the Sherali-Adams constraints.

2.3.1 Linear Program

We start with some additional notation related to the tree decomposition $\mathcal{T}$ (from Theorem 2.2.1) and our dynamic program assumption (Assumption 2.2.1).

- For any node $i \in I$, T_i is the set consisting of (1) all nodes N on the $r - i$ path in $\mathcal{T}$; (2) children of all nodes in $N \setminus \{i\}$. See also Figure 2.1.
- $\mathcal{P}$ is the collection of all node subsets J such that $J \subseteq T_{\ell_1} \cup T_{\ell_2}$ for some pair of leaf-nodes ℓ_1, ℓ_2. See also Figure 2.1.
- $s(i) \in \Sigma_i$ denotes a state at node i. Moreover, for any subset of nodes $N \subseteq I$, we use the shorthand $s(N) := \{s(k) : k \in N\}$.
- $a(i) \in \Sigma_i$ denotes a state at node i chosen by the algorithm. Similar to $s(N)$, $a(N) := \{a(k) : k \in N\}$.
- $\bar{u} \in I$ denotes the highest tree-decomposition node containing vertex u.

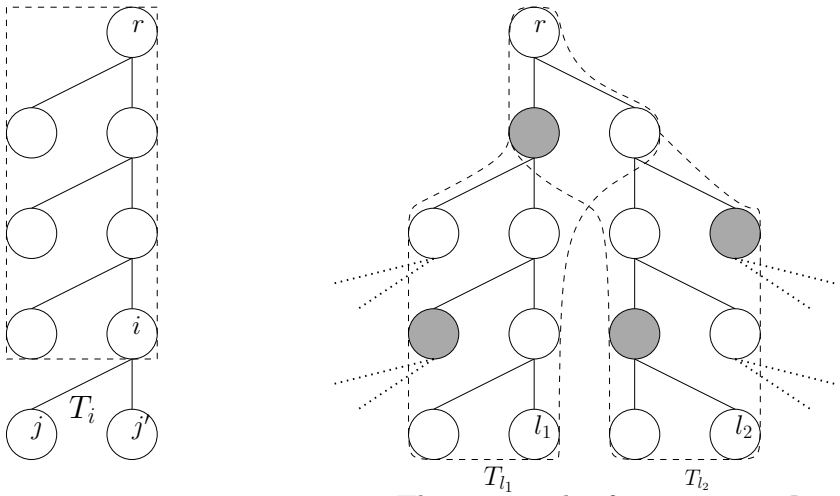

Figure 2.1: Examples of (i) a set T_i and (ii) a set in $\mathcal{P}$.

The variables in our LP are $y(s(N))$ for all $\{s(k) \in \Sigma_k\}_{k \in N}$ and $N \in \mathcal{P}$. Variable $y(s(N))$ corresponds to the probability of the joint event that the solution (in $\mathcal{S}_G$) "induces" state $s(k)$ at each node $k \in N$.

We give a linear program below that has several constraints, numbered (2.3)-(2.8). We use variables z_{uv} defined in constraint (2.3) that measure the probability of an edge (u, v) being cut. The first summation in (2.3) is the probability that the state at $\bar{u}$ selects vertex u and the state at $\bar{v}$ does not select vertex v.

Constraints (2.4) are the Sherali-Adams constraints that enforce consistency among the y variables. They require that the joint distribution imposed by y on any set N of nodes is consistent with the joint distribution imposed by y on any superset of N.

Constraints (2.5)-(2.7) enforce the dynamic program structure from Assumption 2.2.1. Constraint (2.5) requires that some state is selected at the root node. Constraint (2.6) requires that the states chosen at any node i and its children $\{j, j'\}$ form a valid combination. Finally constraint (2.7) requires that the state $s(\ell)$ chosen at any leaf node ℓ must have a non-empty "feasible" subset $\mathcal{H}_{\ell,s(\ell)}$.

$$\text{maximize} \sum_{\{u,v\}\in\binom{V}{2}} c_{uv} z_{uv} \tag{LP}$$

$$z_{uv} = \sum_{\substack{s(\bar{u})\in\Sigma_{\bar{u}} \\ u\in X_{\bar{u},s(\bar{u})}}} \sum_{\substack{s(\bar{v})\in\Sigma_{\bar{v}} \\ v\notin X_{\bar{v},s(\bar{v})}}} y(s(\{\bar{u},\bar{v}\})) + \sum_{\substack{s(\bar{u})\in\Sigma_{\bar{u}} \\ u\notin X_{\bar{u},s(\bar{u})}}} \sum_{\substack{s(\bar{v})\in\Sigma_{\bar{v}} \\ v\in X_{\bar{v},s(\bar{v})}}} y(s(\{\bar{u},\bar{v}\})),$$

$$\forall \{u,v\} \in \binom{V}{2}; \tag{2.3}$$

$$y(s(N)) = \sum_{s(i)\in\Sigma_i} y(s(N\cup\{i\})),$$

$$\forall s(k)\in\Sigma_k,\ \forall k\in N,\ \forall N\in\mathcal{P}, \forall i\notin N\ :\ N\cup\{i\}\in\mathcal{P}; \tag{2.4}$$

$$\sum_{s(r)\in\Sigma_r} y(s(r)) = 1; \tag{2.5}$$

$$y(s(\{i,j,j'\})) = 0, \qquad \forall i\in I,\ s(i)\in\Sigma_i,\ (s(j),s(j'))\notin\mathcal{F}_{i,s(i)}; \tag{2.6}$$

$$y(s(\ell)) = 0, \qquad \forall \text{ leaf } \ell\in I, s(\ell)\in\Sigma_\ell : \mathcal{H}_{\ell,s(\ell)} = \emptyset; \tag{2.7}$$

$$0 \le y(s(N)) \le 1, \qquad \forall N\in\mathcal{P},\ \{s(k)\in\Sigma_k\}_{k\in N}. \tag{2.8}$$

In constraint (2.6), we use j and j' to denote the two children of node $i \in I$.

We note that our base LP (level 0 in the Sherali-Adams hierarchy) involves only variables y for tuples of size at most three. The above LP can be obtained mechanically by applying the Sherali-Adams operator for $O(\log n)$ rounds and then dropping some variables/constraints.

Claim 2.3.2. *Let y be a feasible solution to LP. For any node $i \in I$ with children j, j' and $s(k) \in \Sigma_k$ for all $k \in T_i$,*

$$y(s(T_i)) = \sum_{s(j)\in\Sigma_j} \sum_{s(j')\in\Sigma_{j'}} y(s(T_i\cup\{j,j'\}). \tag{2.9}$$

Proof. Note that $T_i \cup \{j, j'\} \subseteq T_\ell$ for any leaf node ℓ in the subtree below i. So $T_i \cup \{j, j'\} \in \mathcal{P}$ and the variables $y(s(T_i \cup \{j, j'\})$ are well-defined. The claim now follows by two applications of constraint (2.4).

□

2.3.2 The Rounding Algorithm

We start with the root node $r \in I$. Here $\{y(s(r)) : s(r) \in \Sigma_r\}$ defines a probability distribution over the states of r. We sample a state $a(r) \in \Sigma_r$ from this distribution. Then we continue top-down: for any node $i \in I$, given the chosen states $a(k)$ at each $k \in T_i$, we sample states for both children of i *simultaneously* from their joint distribution given at node i. Our algorithm is described in Algorithm 1.

Input : Optimal solution of LP.

Output: A vertex set in $\mathcal{C}_G$.

Sample a state $a(r)$ at the root node by distribution $y(s(r))$;

Do *process all nodes i in $\mathcal{T}$ in order of increasing depth* **:**

  Sample states $a(j), a(j')$ for the children of node i by joint distribution

$$\Pr[a(j) = s(j) \text{ and } a(j') = s(j')] \;=\; \frac{y(s(T_i \cup \{j, j'\}))}{y(s(T_i))}, \tag{2.10}$$

  where $s(T_i) = a(T_i)$.

end

Do *process all nodes i in $\mathcal{T}$ in order of decreasing depth* **:**

  $R_i = X_{i,a(i)} \cup R_j \cup R_{j'}$ where j, j' are the children of i.

end

$R = R_r$;

return R.

Algorithm 1: Rounding Algorithm for LP

Algorithm Analysis

Lemma 2.3.3. (LP) *is a valid relaxation of* GCMC.

Proof. Let $S \in \mathcal{S}_G$ be any feasible solution to the GCMC instance. Let $\{b(i)\}_{i \in I}$ denote the states given by Claim 2.2.12 corresponding to S. For any subset $N \in \mathcal{P}$ of nodes, and

for all $\{s(i) \in \Sigma_i\}_{i \in N}$, set

$$y(s(N)) = \begin{cases} 1, & \text{if } s(i) = b(i) \text{ for all } i \in N; \\ 0, & \text{otherwise.} \end{cases}$$

Clearly constraints (2.4) and (2.8) are satisfied. By the first property in Claim 2.2.12, constraint (2.6) is satisfied. And by the second property in Claim 2.2.12, constraint (2.7) is also satisfied. The last property in Claim 2.2.12 implies that $u \in S \iff u \in X_{\bar{u},b(\bar{u})}$ for any vertex $u \in V$. So any edge $\{u, v\}$ is cut exactly when one of the following occurs:

- $u \in X_{\bar{u},b(\bar{u})}$ and $v \notin X_{\bar{v},b(\bar{v})}$;
- $u \notin X_{\bar{u},b(\bar{u})}$ and $v \in X_{\bar{v},b(\bar{v})}$.

Using the setting of variable z_{uv} in (2.3) it follows that z_{uv} is exactly the indicator of edge $\{u, v\}$ being cut by S. Thus the objective value in (LP) is $c(\delta S)$.

□

Lemma 2.3.4. (LP) *has a polynomial number of variables and constraints. Hence the overall algorithm runs in polynomial time.*

Proof. There are $\binom{n}{2} = O(n^2)$ variables z_{uv}. Because the tree is binary, we have $|T_i| \leq 2d$ for any node i, where $d = O(\log n)$ is the depth of the tree decomposition. Moreover there are only $O(n^2)$ pairs of leaves as there are $O(n)$ leaf nodes. For each pair ℓ_1, ℓ_2 of leaves, we have $|T_{\ell_1} \cup T_{\ell_2}| \leq 4d$. Thus $|\mathcal{P}| \leq O(n^2) \cdot 2^{4d} = poly(n)$. By Assumption 2.2.1, we have $\max |\mathcal{H}_{i,\sigma}| = t = O(1)$, so the number of y-variables is at most $|\mathcal{P}| \cdot t^{4d} = poly(n)$. This shows that (LP) has polynomial size and can be solved optimally in polynomial time. Finally, it is clear that the rounding algorithm runs in polynomial time.

□

Lemma 2.3.5. *The algorithm's solution R is always feasible.*

Proof. Due to Claim 2.3.2, the distributions used in Step 1 and Step 2 are well-defined ; so the states $a(i)$s are well-defined. Moreover, by the choice of these distributions, for each node i, $y(a(T_i)) > 0$.

We now show that for any node $i \in I$ with children j, j' we have $(a(j), a(j')) \in \mathcal{F}_{i,a(i)}$. Indeed, at the iteration for node i (when $a(j)$ and $a(j')$ are set) using the conditional probability distribution in (2.10) and by constraint (2.6), we obtain that $(a(j), a(j')) \in \mathcal{F}_{i,a(i)}$ with probability one.

We show by induction that for each node $i \in I$, the subset $R_i \in \mathcal{H}_{i,a(i)}$. The base case is when i is a leaf. In this case, due to constraint (2.7) and the fact that $y(a(T_i)) > 0$ we know

that $\mathcal{H}_{i,a(i)} \neq \emptyset$. So $R_i = X_{i,a(i)} \in \mathcal{H}_{i,a(i)}$ by Assumption 2.2.1(3). For the inductive step, consider node $i \in I$ with children j, j' where $R_j \in \mathcal{H}_{j,a(j)}$ and $R_{j'} \in \mathcal{H}_{j',a(j')}$. Moreover, from the property above, $(a(j), a(j')) \in \mathcal{F}_{i,a(i)}$. Now using Assumption 2.2.1(4) we have $R_i = X_{i,a(i)} \cup R_j \cup R_{j'} \in \mathcal{H}_{i,a(i)}$. Thus the final solution $R \in \mathcal{S}_G$.

□

Claim 2.3.6. *A vertex u is contained in solution R if and only if $u \in X_{\bar{u},a(\bar{u})}$.*

Proof. This proof is identical to that of the last property in Claim 2.2.12.

□

In the rest of this section, we show that every edge (u, v) is cut by solution R with probability at least $z_{uv}/2$, which would prove the algorithm's approximation ratio. Lemma 2.3.10 handles the case when $\bar{u} \in T_{\bar{v}}$ (the case $\bar{v} \in T_{\bar{u}}$ is identical). And Lemma 2.3.11 handles the (harder) case when $\bar{u} \notin T_{\bar{v}}$ and $\bar{v} \notin T_{\bar{u}}$.

We first state some useful claims before proving the lemmas.

Observation 2.3.7 (see [68] for a similar use of this principle)**.** *Let X, Y be two jointly distributed $\{0, 1\}$ random variables. Then* $\Pr(X = 1)\Pr(Y = 0) + \Pr(X = 0)\Pr(Y = 1) \geq \frac{1}{2}[\Pr(X = 0, Y = 1) + \Pr(X = 1, Y = 0)]$.

Proof. Let $\Pr(X = 0, Y = 0) = x$, $\Pr(X = 0) = a$, $\Pr(Y = 0) = b$. The probability table is as below:

	$Y=0$	$Y=1$	
$X=0$	x	$a-x$	a
$X=1$	$b-x$	$1+x-a-b$	$1-a$
	b	$1-b$	

Then we want to show $a(1-b) + b(1-a) \geq \frac{1}{2}(a+b-2x)$, i.e. $a+b+2x \geq 4ab$. This is clearly true if $a = 0$ or $b = 0$. When $a, b \neq 0$, consider the covariance matrix of the vector $v = \begin{pmatrix} X \\ Y \end{pmatrix}$ of random variables:

$$M = \mathbf{E}[(v - \mathbf{E}[v]) \cdot (v - \mathbf{E}[v])^T] = \begin{bmatrix} a(1-a) & x - ab \\ x - ab & b(1-b) \end{bmatrix}.$$

As is true of every covariance matrix, $M \succeq 0$. Let $y = \begin{pmatrix} \sqrt{\frac{b}{a}} \\ \sqrt{\frac{a}{b}} \end{pmatrix}$. Then we know $y^T M y \geq 0$ as $M \succeq 0$. Expanding this expression and simplifying directly shows $a+b+2x-4ab \geq 0$.

□

Claim 2.3.8. *For any node i and states $s(k) \in \Sigma_k$ for all $k \in T_i$, the rounding algorithm satisfies* $\Pr[a(T_i) = s(T_i)] = y(s(T_i))$.

Proof. We proceed by induction on the depth of node i. It is clearly true when $i = r$, i.e. $T_i = \{r\}$. Assuming the statement is true for node i, we will prove it for i's children. Let j, j' be the children nodes of i; note that $T_j = T_{j'} = T_i \cup \{j, j'\}$. Then using (2.10), we have

$$\Pr[a(T_j) = s(T_j) \mid a(T_i) = s(T_i)] \quad = \quad \frac{y(s(T_i \cup \{j, j'\}))}{y(s(T_i))}.$$

Combined with $\Pr[a(T_i) = s(T_i)] = y(s(T_i))$ we obtain $\Pr[a(T_j) = s(T_j)] = y(s(T_j))$ as desired. □

Claim 2.3.9. *For any $u, v \in V$, $s(\bar{u}) \in \Sigma_{\bar{u}}$ and $s(\bar{v}) \in \Sigma_{\bar{v}}$, we have*

$$y(s(\{\bar{u}, \bar{v}\})) = \sum_{\substack{s(k) \in \Sigma_k \\ k \in T_i \setminus \bar{u} \setminus \bar{v}}} y(s(T_i \cup \{\bar{u}, \bar{v}\})),$$

where i is the least common ancestor of $\bar{u}$ and $\bar{v}$.

Proof. Since i is the least common ancestor of $\bar{u}$ and $\bar{v}$, we have $T_i \cup \{\bar{u}, \bar{v}\} \in \mathcal{P}$. Then the claim follows by repeatedly applying constraint (2.4). □

Lemma 2.3.10. *Consider any $u, v \in V$ such that $\bar{u} \in T_{\bar{v}}$. Then the probability that edge (u, v) is cut by solution R is z_{uv}.*

Proof. Applying Claim 2.3.8 with node $i = \bar{v}$, for any $\{s(k) \in \Sigma_k : k \in T_{\bar{v}}\}$, we have $\Pr[a(T_{\bar{v}}) = s(T_{\bar{v}})] = y(s(T_{\bar{v}}))$. Let $D_u = \{s(\bar{u}) \in \Sigma_{\bar{u}} \,|\, u \in X_{\bar{u}, s(\bar{u})}\}$ and similarly $D_v = \{s(\bar{v}) \in \Sigma_{\bar{v}} \,|\, v \in X_{\bar{v}, s(\bar{v})}\}$. Because $\bar{u} \in T_{\bar{v}}$,

$$\Pr[u \in R, v \notin R] = \sum_{s(\bar{u}) \in D_u} \sum_{s(\bar{v}) \notin D_v} \sum_{\substack{s(k) \in \Sigma_k \\ k \in T_{\bar{v}} \setminus \bar{u} \setminus \bar{v}}} y(s(T_{\bar{v}})) = \sum_{s(\bar{u}) \in D_u} \sum_{s(\bar{v}) \notin D_v} y(s(\bar{u}, \bar{v})).$$

The last equality above is by repeated application of constraint (2.4). Similarly,

$$\Pr[u \notin R, v \in R] = \sum_{s(\bar{u}) \notin D_u} \sum_{s(\bar{v}) \in D_v} y(s(\bar{u}, \bar{v})),$$

which combined with constraint (2.3) implies $\Pr[|\{u, v\} \cap R| = 1] = z_{uv}$. □

Lemma 2.3.11. *Consider any $u, v \in V$ such that $\bar{u} \notin T_{\bar{v}}$ and $\bar{v} \notin T_{\bar{u}}$. Then the probability that edge (u, v) is cut by solution R is at least $z_{uv}/2$.*

Proof. In order to simplify notation, we define:

$$z^+_{uv} = \sum_{\substack{s(\bar{u})\in\Sigma_{\bar{u}} \\ u\in X_{\bar{u},s(\bar{u})}}} \sum_{\substack{s(\bar{v})\in\Sigma_{\bar{v}} \\ v\notin X_{\bar{v},s(\bar{v})}}} y(s(\{\bar{u},\bar{v}\})), \quad z^-_{uv} = \sum_{\substack{s(\bar{u})\in\Sigma_{\bar{u}} \\ u\notin X_{\bar{u},s(\bar{u})}}} \sum_{\substack{s(\bar{v})\in\Sigma_{\bar{v}} \\ v\in X_{\bar{v},s(\bar{v})}}} y(s(\{\bar{u},\bar{v}\})).$$

Note that $z_{uv} = z^+_{uv} + z^-_{uv}$.

Let $D_u = \{s(\bar{u}) \in \Sigma_{\bar{u}} \,|\, u \in X_{\bar{u},s(\bar{u})}\}$ and $D_v = \{s(\bar{v}) \in \Sigma_{\bar{v}} \,|\, v \in X_{\bar{v},s(\bar{v})}\}$. Let i denote the least common ancestor of nodes $\bar{u}$ and $\bar{v}$, and $\{j, j'\}$ the two children of i. Note that $T_j = T_{j'} = T_i \cup \{j, j'\}$ and $T_{\bar{u}}, T_{\bar{v}} \supseteq T_j$. Because $\bar{u} \notin T_{\bar{v}}$ and $\bar{v} \notin T_{\bar{u}}$, both $\bar{u}, \bar{v}$ are strictly below j and j' in the tree decomposition (see also Figure 2.2).

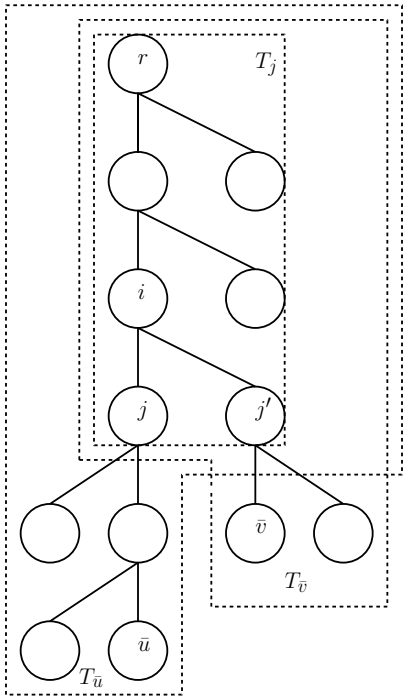

Figure 2.2: Example of T_j, $T_{\bar{u}}$, and $T_{\bar{v}}$

For any choice of states $\{s(k) \in \Sigma_k\}_{k\in T_j}$ define:

$$z^+_{uv}(s(T_j)) = \sum_{s(\bar{u})\in D_u} \sum_{s(\bar{v})\notin D_v} \frac{y(s(T_j \cup \{\bar{u},\bar{v}\}))}{y(s(T_j))},$$

and similarly $z^-_{uv}(s(T_j))$.

In the rest of the proof, we fix states $\{s(k) \in \Sigma_k\}_{k\in T_j}$ and *condition* on the event $\mathcal{E}$ that

$a(T_j) = s(T_j)$. We will show:

$$\Pr[|\{u,v\} \cap R| = 1 \,|\, \mathcal{E}] \;\geq\; \frac{1}{2}\left(z^+_{uv}(s(T_j)) + z^-_{uv}(s(T_j))\right). \tag{2.11}$$

By taking expectation over the conditioning $\mathcal{E}$, this would imply Lemma 2.3.11.

We now define the following indicator random variables (conditioned on $\mathcal{E}$).

$$I_u = \begin{cases} 0 & \text{if } a(\bar{u}) \notin D_u \\ 1 & \text{if } a(\bar{u}) \in D_u \end{cases} \quad \text{and} \quad I_v = \begin{cases} 0 & \text{if } a(\bar{v}) \notin D_v \\ 1 & \text{if } a(\bar{v}) \in D_v \end{cases}.$$

Observe that I_u and I_v (*conditioned* on $\mathcal{E}$) are independent because $\bar{u}, \bar{v} \notin T_j$, and $\bar{v}$ and $\bar{u}$ appear in distinct subtrees under node i. So,

$$\Pr[|\{u,v\} \cap R| = 1 \,|\, \mathcal{E}] \;=\; \Pr[I_u = 1] \cdot \Pr[I_v = 0] + \Pr[I_u = 0] \cdot \Pr[I_v = 1]. \tag{2.12}$$

For any $s(k) \in \Sigma_k$ for $k \in T_{\bar{u}} \setminus T_j$, we have by Claim 2.3.8 and $T_j \subseteq T_{\bar{u}}$ that

$$\Pr[a(T_{\bar{u}}) = s(T_{\bar{u}}) \,|\, a(T_j) = s(T_j)] = \frac{\Pr[a(T_{\bar{u}}) = s(T_{\bar{u}})]}{\Pr[a(T_j) = s(T_j)]} = \frac{y(s(T_{\bar{u}}))}{y(s(T_j))}.$$

Therefore

$$\Pr[I_u = 1] \;=\; \sum_{s(\bar{u}) \in D_u} \sum_{\substack{k \in T_{\bar{u}} \setminus T_j \setminus \{\bar{u}\} \\ s(k) \in \Sigma_k}} \frac{y(s(T_{\bar{u}}))}{y(s(T_j))} = \sum_{s(\bar{u}) \in D_u} \frac{y(s(T_j \cup \{\bar{u}\}))}{y(s(T_j))}.$$

The last equality follows from constraint (2.4). Furthermore, note that $T_j \cup \{\bar{u}, \bar{v}\} \in \mathcal{P}$, we have again by constraint (2.4) that

$$\begin{aligned} \Pr[I_u = 1] \;&=\; \sum_{s(\bar{u}) \in D_u} \frac{y(s(T_j \cup \{\bar{u}\}))}{y(s(T_j))} = \sum_{s(\bar{u}) \in D_u} \sum_{s(\bar{v}) \in \Sigma_{\bar{v}}} \frac{y(s(T_j \cup \{\bar{u}, \bar{v}\}))}{y(s(T_j))} \\ &= \sum_{s(\bar{u}) \in D_u} \sum_{s(\bar{v}) \in D_v} \frac{y(s(T_j \cup \{\bar{u}, \bar{v}\}))}{y(s(T_j))} + z^+_{uv}(s(T_j)). \end{aligned}$$

Similarly,

$$\Pr[I_v = 1] \;=\; \sum_{s(\bar{v}) \in D_v} \frac{y(s(T_j \cup \{\bar{v}\}))}{y(s(T_j))} = \sum_{s(\bar{u}) \in D_u} \sum_{s(\bar{v}) \in D_v} \frac{y(s(T_j \cup \{\bar{u}, \bar{v}\}))}{y(s(T_j))} + z^-_{uv}(s(T_j)).$$

and

$$\Pr[I_u = 0] \;=\; \sum_{s(\bar{u})\notin D_u} \frac{y(s(T_j \cup \{\bar{u}\}))}{y(s(T_j))} = \sum_{s(\bar{u})\notin D_u} \sum_{s(\bar{v})\notin D_v} \frac{y(s(T_j \cup \{\bar{u},\bar{v}\}))}{y(s(T_j))} + z^{-}_{uv}(s(T_j)).$$

Now define $\{0,1\}$ random variables X and Y jointly distributed as:

	$Y=0$	$Y=1$
$X=0$	$\Pr[I_u=0] - z^{-}_{uv}(s(T_j))$	$z^{-}_{uv}(s(T_j))$
$X=1$	$z^{+}_{uv}(s(T_j))$	$\Pr[I_u=1] - z^{+}_{uv}(s(T_j))$

Note that $\Pr[X=1] = \Pr[I_u=1]$ and $\Pr[Y=1] = \Pr[I_u=1] - z^{+}_{uv}(s(T_j)) + z^{-}_{uv}(s(T_j)) = \Pr[I_v=1]$. So, applying Observation 2.3.7 and using (2.12) we have:

$$\Pr[|\{u,v\}\cap R| = 1 \,|\, \mathcal{E}] \;\geq\; \frac{1}{2}\left(\Pr[X=0,Y=1] + \Pr[X=1,Y=0]\right),$$

which implies (2.11).

□

Now We show that the dynamic-program structure for CSP can be combined with the algorithm in this section to obtain a $\frac{1}{2}$-approximation algorithm.

Theorem 2.3.12. *There is a $\frac{1}{2}$-approximation algorithm for* GCMC *with $k=2$ when the constraint $\mathcal{S}_G$ is for one-side of the cut and given by any* MSO *formula on a bounded-treewidth graph.*

Proof. The proof uses Theorem 2.3.1 as a black-box. Note that the constraint $\mathcal{S}_G$ corresponds to feasible assignments to $CSP_\varphi(G)$ as in Definition 2.2.6. Consider the CSP extension ϑ obtained after applying Theorem 2.2.7 to $CSP_\varphi(G)$. Then ϑ has variables $V' \supseteq V$ and bounded treewidth. We obtain an extended weight function $c : \binom{V'}{2} \to \mathbb{R}_+$ from c by setting $c'(u,v) = c(u,v)$ if $u,v \in V$ and $c'(u,v) = 0$ otherwise. We now consider a new instance of GCMC_I on vertices V' and constraint ϑ. Due to the bounded-treewidth property of ϑ, we can apply Theorem 2.2.11 which proves that Assumption 2.2.1 is satisfied by the dynamic program in Definition 2.2.8. Combined with Theorem 2.3.1, we obtain the claimed result. □

We note that the choice of variables in the Sherali-Adams LP as well as the rounding algorithm are similar to those used in [68] for the sparsest-cut problem on bounded-treewidth graphs. An important difference in our result is that we apply the Sherali-Adams hierarchy to a non-standard LP that is defined using the dynamic program. (If we were to apply

Sherali-Adams to the standard LP, then it is unclear how to enforce the constraint $\mathcal{S}_G$ during the rounding algorithm.) Another difference is that our rounding algorithm needs to make a correlated choice in selecting the states of sibling nodes in order to satisfy constraint $\mathcal{S}_G$ — this causes the number of variables in the Sherali-Adams LP to increase, but it still remains polynomial because one can ensure that the tree decomposition has constant degree.

2.4 The Max-k-Cut Setting

In this section, we generalize the setting to any constant k, i.e. problem (2.2). Recall the formal definition from Section 2.1.1. Here the graph property $\mathcal{S}_G$ is expressed as an MSO formula with k free variables on graph G. Our main result is the following:

Theorem 2.4.1. *There is a $\frac{1}{2}$-approximation algorithm for any* GCMC *instance with constant k when the constraint $\mathcal{S}_G$ is given by any* MSO *formula on a bounded-treewidth graph.*

Remark 2.4.2. *The complexity of Theorem 2.4.1 in terms of the treewidth τ, length $|\varphi|$ of φ, depth d of a tree decomposition of G, and maximum degree r of a tree decomposition of G, is s^{dr}, where s is the number of states of the dynamic program, namely $f(|\varphi|, \tau)$ for f from Theorem 2.2.7. From the perspective of parameterized complexity [51] our algorithm is an* XP algorithm *parameterized by τ, i.e., it has runtime $n^{g(\tau)}$ for some computable function g.*

Let $G = (V, E)$ be the input graph (assumed to have bounded treewidth) and φ be any MSO formula with k free variables. Recall the CSP instance $CSP_\varphi(G)$ on variables $\{y(v, \alpha) : v \in V, \alpha \in [k]\}$ from Definition 2.2.6. Feasible solutions to $CSP_\varphi(G)$ correspond to feasible k-partitions in $\mathcal{S}_G$. Now consider the CSP extension ϑ obtained after applying Theorem 2.2.7 to $CSP_\varphi(G)$. Note that ϑ is defined on variables $V' \supseteq \{(v, \alpha) : v \in V, \alpha \in [k]\}$ and has bounded treewidth. Let $\mathcal{T}$ denote the tree decomposition for ϑ. Below we utilize the dynamic program from Definition 2.2.10 applied to ϑ: recall the quantities Σ_i, $\mathcal{F}_{i,\sigma}$ etc. We will also refer to the variables in V' as vertices, especially when referring to the tree decomposition $\mathcal{T}$; note that these are different from the vertices V in the original graph G.

Claim 2.4.3. *Let $\{U_\alpha\}_{\alpha=1}^k$ be a k-partition satisfying $\mathcal{S}_G$. There is a collection of states $\{b[i] \in \Sigma_i\}_{i \in I}$ such that:*

- *for each node $i \in I$ with children j and j', $(b[j], b[j']) \in \mathcal{F}_{i,b[i]}$,*

- *for each leaf ℓ we have $\mathcal{H}_{\ell,b[\ell]} \neq \emptyset$,*
- $U_\alpha = \{v \in V : b_{\mathcal{T}}((v,\alpha)) = 1\}$ *for all* $\alpha \in [k]$*, where* $b_{\mathcal{T}} = \bigcup_{i \in I} b[i]$.

Moreover, for any vertex $(v,\alpha) \in V'$, if $\overline{v\alpha} \in I$ denotes the highest node in $\mathcal{T}$ containing (v,α) then we have: $v \in U_\alpha$ if and only if $b[\overline{v\alpha}]((v,\alpha)) = 1$.

Proof. By definition of CSP ϑ, we know that it has some feasible solution $t \in \{0,1\}^{V'}$ where $U_\alpha = \{v \in V : t((v,\alpha)) = 1\}$ for all $\alpha \in [k]$. Now, using Theorem 2.2.11(5) we have $t \in \bigcup_{\sigma \in \Sigma_r} \mathcal{H}_{r,\sigma}$.

We define the states $b[i]$ in a top-down manner. We will also define an associated vector $t_i \in \mathcal{H}_{i,b[i]}$ at each node i. At the root, we set $b[r] = \sigma$ such that $t \in \mathcal{H}_{r,\sigma}$: this is well-defined because $t \in \bigcup_{\sigma \in \Sigma_r} \mathcal{H}_{r,\sigma}$. We also set $t_r = t$. Having set $b[i]$ and $t_i \in \mathcal{H}_{i,b[i]}$ for any node $i \in I$ with children $\{j, j'\}$, we use Theorem 2.2.11(4) to write $h_i = b[i] \cup h_j \cup h_{j'}$ where $h_j \in \mathcal{H}_{j,w_j}$, $h_{j'} \in \mathcal{H}_{j',w_{j'}}$ and $(w_j, w_{j'}) \in \mathcal{F}_{i,b[i]}$. Then we set $b[j] = w_j$, $t_j = h_j$ and $b[j'] = w_{j'}$, $t_{j'} = h_{j'}$ for the children of node i. The first condition in the claim is immediate from the definition of states $b[i]$. By induction on the depth of node i, we obtain $t_i \in \mathcal{H}_{i,b[i]}$ for each node i. This implies that $\mathcal{H}_{\ell,b[\ell]} \neq \emptyset$ for each leaf ℓ, which proves the second condition; moreover, by Theorem 2.2.11(3) we have $t_\ell = b[\ell]$. Now, by definition of the vectors t_i, we obtain $t = t_r = \bigcup_{i \in I} b[i] = b_{\mathcal{T}}$ which, combined with $U_\alpha = \{v \in V : t((v,\alpha)) = 1\}$ for all $\alpha \in [k]$, proves the third condition in the claim.

Because $t = \bigcup_{i \in I} b[i]$, it is clear that if $b[\overline{v\alpha}]((v,\alpha)) = 1$ then $v \in U_\alpha$. In the other direction, suppose $b[\overline{v\alpha}]((v,\alpha)) \neq 1$: we will show $v \notin U_\alpha$. Since $\overline{v\alpha}$ is the highest node containing (v,α), it suffices to show that $t_{\overline{v\alpha}}((v,\alpha)) \neq 1$. But this follows directly from Theorem 2.2.11(2) because $t_{\overline{v\alpha}} \in \mathcal{H}_{\overline{v\alpha},b[\overline{v\alpha}]}$, $(v,\alpha) \in X_{\overline{v\alpha}}$ and $b[\overline{v\alpha}]((v,\alpha)) \neq 1$. □

2.4.1 LP Relaxation for Max-k-Cut

We start with some additional notation related to the tree decomposition $\mathcal{T}$ (from Theorem 2.2.1) and the dynamic program for CSP (from Theorem 2.2.11).

- For any node $i \in I$, T_i is the set consisting of (1) all nodes N on the $r - i$ path in $\mathcal{T}$, and (2) children of all nodes in $N \setminus \{i\}$.
- $\mathcal{P}$ is the collection of all node subsets J such that $J \subseteq T_{\ell_1} \cup T_{\ell_2}$ for some pair of leaf-nodes ℓ_1, ℓ_2.
- $s[i] \in \Sigma_i$ denotes a state at node i. Moreover, for any subset of nodes $N \subseteq I$, we use the shorthand $s[N] := \{s[k] : k \in N\}$.

- $a[i] \in \Sigma_i$ denotes a state at node i chosen by the algorithm. Similar to $s[N]$, for any subset $N \subseteq I$ of nodes, $a[N] := \{a[k] : k \in N\}$.
- $\overline{v\alpha} \in I$ denotes the highest tree-decomposition node containing vertex $(v, \alpha) \in V'$.

The LP that we use here is a generalization of that in Section 2.3.1. The variables are $y(s[N])$ for all $\{s[k] \in \Sigma_k\}_{k \in N}$ and $N \in \mathcal{P}$. Variable $y(s[N])$ corresponds to the probability of the joint event that the solution (in $\mathcal{S}_G$) "induces" state $s[k]$ at each node $k \in N$. Variable $z_{uv\alpha}$ corresponds to the probability that edge $(u, v) \in E$ is cut by part α of the k-partition.

In constraint (2.16), we use j and j' to denote the two children of node $i \in I$. We note that constraints (2.14)-(2.18) which utilize the dynamic-program structure, are identical to the constraints (2.4)-(2.8) in Section 2.3.1. This allows us to essentially reuse many of the claims proved in Section 2.3.1, which are stated below.

$$\text{maximize } \frac{1}{2} \sum_{\{u,v\} \in \binom{V}{2}} c_{uv} \sum_{\alpha=1}^{k} z_{uv\alpha} \tag{LP}$$

$$z_{uv\alpha} = \sum_{\substack{s[\overline{u\alpha}] \in \Sigma_{\overline{u\alpha}},\, s[\overline{v\alpha}] \in \Sigma_{\overline{v\alpha}} \\ s[\overline{u\alpha}]((u,\alpha)) \neq s[\overline{v\alpha}]((v,\alpha))}} y(s[\{\overline{u\alpha}, \overline{v\alpha}\}]),\ \forall \{u, v\} \in \binom{V}{2}, \forall \alpha \in [k]; \tag{2.13}$$

$$y(s[N]) = \sum_{s[i] \in \Sigma_i} y(s[N \cup \{i\}]),$$
$$\forall s[k] \in \Sigma_k,\ \forall k \in N,\ \forall N \in \mathcal{P},\ \forall i \notin N : N \cup \{i\} \in \mathcal{P}; \tag{2.14}$$

$$\sum_{s[r] \in \Sigma_r} y(s[r]) = 1; \tag{2.15}$$

$$y(s[\{i, j, j'\}]) = 0,\ \forall i \in I,\ \forall s[i] \in \Sigma_i,\ \forall (s[j], s[j']) \notin \mathcal{F}_{i,s[i]}; \tag{2.16}$$

$$y(s[\ell]) = 0,\ \forall \text{ leaf } \ell \in I,\ \forall s[\ell] \in \Sigma_\ell : \mathcal{H}_{\ell,s[\ell]} = \emptyset; \tag{2.17}$$

$$0 \leq y(s[N]) \leq 1,\ \forall N \in \mathcal{P},\ \forall s[k] \in \Sigma_k \text{ for } k \in N. \tag{2.18}$$

Claim 2.4.4. *Let y be feasible to* (LP). *For any node $i \in I$ with children j, j' and $s[k] \in \Sigma_k$ for all $k \in T_i$,*

$$y(s[T_i]) = \sum_{s[j] \in \Sigma_j} \sum_{s[j'] \in \Sigma_{j'}} y(s[T_i \cup \{j, j'\}]).$$

Proof. Note that $T_i \cup \{j, j'\} \subseteq T_\ell$ for any leaf node ℓ in the subtree below i. So $T_i \cup \{j, j'\} \in \mathcal{P}$ and the variables $y(s[T_i \cup \{j, j'\}])$ are well-defined. The claim follows by two applications of (2.14). □

Lemma 2.4.5. (LP) *has a polynomial number of variables and constraints.*

Proof. There are $\binom{n}{2} \cdot k = O(kn^2)$ variables $z_{uv\alpha}$. Because the tree is binary, we have $|T_i| \leq 2d$ for any node i, where $d = O(\log n)$ is the depth of the tree decomposition. Moreover there are only $O(n^2)$ pairs of leaves as there are $O(n)$ leaf nodes. For each pair ℓ_1, ℓ_2 of leaves, we have $|T_{\ell_1} \cup T_{\ell_2}| \leq 4d$. Thus $|\mathcal{P}| \leq O(n^2) \cdot 2^{4d} = poly(n)$. By Theorem 2.2.11, we have $\max |\mathcal{H}_{i,\sigma}| = O(1)$, so the number of y-variables is at most $|\mathcal{P}| \cdot (\max |\mathcal{H}_{i,\sigma}|)^{4d} = poly(n)$. This shows that (LP) has polynomial size and can be solved optimally in polynomial time. Finally, it is clear that the rounding algorithm runs in polynomial time. □

Lemma 2.4.6. (LP) *is a valid relaxation of* GCMC.

Proof. Let k-partition $\{U_\alpha\}_{\alpha=1}^k$ be any feasible solution to $\mathcal{S}_G$. Let $\{b[i]\}_{i \in I}$ denote the states given by Claim 2.4.3 corresponding to $\{U_\alpha\}_{\alpha=1}^k$. For any subset $N \in \mathcal{P}$ of nodes, and for all $\{s[i] \in \Sigma_i\}_{i \in N}$, set

$$y(s[N]) = \begin{cases} 1, & \text{if } s[i] = b[i] \text{ for all } i \in N; \\ 0, & \text{otherwise.} \end{cases}$$

Clearly constraints (2.14) and (2.18) are satisfied. By the first property in Claim 2.4.3, constraint (2.16) is satisfied. And by the second property in Claim 2.4.3, constraint (2.17) is also satisfied. The last property in Claim 2.4.3 implies that $v \in U_\alpha \iff b[\overline{v\alpha}]((v,\alpha)) = 1$ for any vertex $v \in V$. So any edge $\{u, v\}$ is cut by U_α exactly when $b[\overline{u\alpha}]((u,\alpha)) \neq b[\overline{v\alpha}]((v,\alpha))$. Using the setting of variable $z_{uv\alpha}$ in (2.13) it follows that $z_{uv\alpha}$ is exactly the indicator of edge $\{u, v\}$ being cut by U_α. Finally, the objective value is exactly the total weight of edges cut by the k-partition $\{U_\alpha\}_{\alpha=1}^k$ where the coefficient $\frac{1}{2}$ comes from the fact that that summation counts each cut-edge twice. Thus (LP) is a valid relaxation. □

2.4.2 The Rounding Algorithm

This is a top-down procedure, exactly as Algorithm 1. The only difference is now we need to find a vertex partition of the vertices. Our algorithm is formally described in Algorithm 2.

Lemma 2.4.7. *The algorithm's solution* $\{U_\alpha\}_{\alpha=1}^k$ *is always feasible.*

Proof. Due to Claim 2.4.4, the distributions used in Step 2 and Step 3 are well-defined; so the states $a[i]$s are well-defined. Moreover, by the choice of these distributions, for each node i, $y(a[T_i]) > 0$.

Input : Optimal solution of LP.
Output: A vertex partition of V in $\mathcal{S}_G$.

Sample a state $a[r]$ at the root node by distribution $y(s[r])$.
Do *process all nodes i in $\mathcal{T}$ in order of increasing depth* **:**
  Sample states $a[j], a[j']$ for the children of node i by joint distribution

$$\Pr[a[j] = s[j] \text{ and } a[j'] = s[j']] = \frac{y(s[T_i \cup \{j, j'\}])}{y(s[T_i])}, \tag{2.19}$$

  where $s[T_i] = a[T_i]$.
end
Do *process all nodes i in $\mathcal{T}$ in order of decreasing depth* **:**
  $h_i = a[i] \cup h_j \cup h_{j'}$ where j, j' are the children of i.
end
Set $U_\alpha = \{v \in V : h_r((v, \alpha)) = 1\}$ for all $\alpha \in [k]$.
return k-partition $\{U_\alpha\}_{\alpha=1}^k$.

Algorithm 2: Rounding Algorithm for LP

We now show that for any node $i \in I$ with children j, j' we have $(a[j], a[j']) \in \mathcal{F}_{i,a[i]}$. Indeed, at the iteration for node i (when $a[j]$ and $a[j']$ are set), using the conditional probability distribution (2.19) and constraint (2.16), we have $(a[j], a[j']) \in \mathcal{F}_{i,a[i]}$ with probability one.

We show by induction that for each node $i \in I$, $h_i \in \mathcal{H}_{i,a[i]}$. The base case is when i is a leaf. In this case, due to constraint (2.17) and the fact that $y(a[T_i]) > 0$ we know that $\mathcal{H}_{i,a[i]} \neq \emptyset$. So $h_i = a[i] \in \mathcal{H}_{i,a[i]}$ by Theorem 2.2.11(3). For the inductive step, consider node $i \in I$ with children j, j' where $h_j \in \mathcal{H}_{j,a[j]}$ and $h_{j'} \in \mathcal{H}_{j',a[j']}$. Moreover, from the property above, $(a[j], a[j']) \in \mathcal{F}_{i,a[i]}$. Now using Theorem 2.2.11(4) we have $h_i = a[i] \cup h_j \cup h_{j'} \in \mathcal{H}_{i,a[i]}$. Finally, using $h_r \in \mathcal{H}_{r,a[r]}$ at the root node and Theorem 2.2.11(5), it follows that $h_r \in \text{Feas}(\vartheta)$. Now let h' denote the restriction of h_r to the variables $\{(v, \alpha) : v \in V, \alpha \in [k]\}$. Then, using the CSP extension result (Theorem 2.2.7) we obtain that h' is feasible for $CSP_\varphi(G)$. In other words, the k-partition $\{U_\alpha\}_{\alpha=1}^k$ satisfies $\mathcal{S}_G$. □

Claim 2.4.8. *For any node i and states $s[k] \in \Sigma_k$ for all $k \in T_i$, the rounding algorithm satisfies* $\Pr[a[T_i] = s[T_i]] = y(s[T_i])$.

Proof. We proceed by induction on the depth of node i. It is clearly true when $i = r$, i.e. $T_i = \{r\}$. Assuming the statement is true for node i, we will prove it for i's children. Let j, j' be the children nodes of i; note that $T_j = T_{j'} = T_i \cup \{j, j'\}$. Then using (2.19), we have

$$\Pr[a[T_j] = s[T_j] \mid a[T_i] = s[T_i]] = \frac{y(s[T_i \cup \{j, j'\}])}{y(s[T_i])}.$$

Combined with $\Pr[a[T_i] = s[T_i]] = y(s[T_i])$ we obtain $\Pr[a[T_j] = s[T_j]] = y(s[T_j])$ as desired. □

Lemma 2.4.9. *Consider any $u, v \in V$ and $\alpha \in [k]$ such that $\overline{u\alpha} \in T_{\overline{v\alpha}}$. Then the probability that edge (u, v) is cut by $U_\alpha = \{v \in V : h_r((v, \alpha)) = 1\}$ is $z_{uv\alpha}$.*

Proof. Applying Claim 2.4.8 with node $i = \overline{v\alpha}$, for any $\{s[k] \in \Sigma_k : k \in T_{\overline{v\alpha}}\}$, we have $\Pr[a[T_{\overline{v\alpha}}] = s[T_{\overline{v\alpha}}]] = y(s[T_{\overline{v\alpha}}])$. Let $D_{u\alpha} = \{s[\overline{u\alpha}] \in \Sigma_{[\overline{u\alpha}]} \mid s[\overline{u\alpha}]((u, \alpha)) = 1\}$ and similarly $D_{v\alpha} = \{s[\overline{v\alpha}] \in \Sigma_{[\overline{v\alpha}]} \mid s[\overline{v\alpha}]((v, \alpha)) = 1\}$. Because $\overline{u\alpha} \in T_{\overline{v\alpha}}$,

$$\Pr[u \in U_\alpha, v \notin U_\alpha] = \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \sum_{s[\overline{v\alpha}] \notin D_{v\alpha}} \sum_{\substack{s[k] \in \Sigma_k \\ k \in T_{[\overline{v\alpha}]} \setminus \{\overline{u\alpha}\} \setminus \{\overline{v\alpha}\}}} y(s[\overline{v\alpha}])$$

$$= \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \sum_{s[\overline{v\alpha}] \notin D_{v\alpha}} y(s[\{\overline{u\alpha}, \overline{v\alpha}\}]).$$

The last equality above is by repeated application of LP constraint (2.14) where we use $T_{\overline{v\alpha}} \in \mathcal{P}$. Similarly,

$$\Pr[u \notin U_\alpha, v \in U_\alpha] = \sum_{s[\overline{u\alpha}] \notin D_{u\alpha}} \sum_{s[\overline{v\alpha}] \in D_{v\alpha}} y(s[\{\overline{u\alpha}, \overline{v\alpha}\}]),$$

which combined with constraint (2.13) implies $\Pr[|\{u, v\} \cap U_\alpha| = 1] = z_{uv\alpha}$. □

Lemma 2.4.10. *Consider any $u, v \in V$ and $\alpha \in [k]$ such that $\overline{u\alpha} \notin T_{\overline{v\alpha}}$ and $\overline{v\alpha} \notin T_{\overline{u\alpha}}$. Then the probability that edge (u, v) is cut by $U_\alpha = \{v \in V : h_r((v, \alpha)) = 1\}$ is at least $z_{uv\alpha}/2$.*

Proof. In order to simplify notation, we define:

$$z^+_{uv\alpha} = \sum_{\substack{s[\overline{u\alpha}] \in \Sigma_{\overline{u\alpha}},\, s[\overline{v\alpha}] \in \Sigma_{\overline{v\alpha}} \\ s[\overline{u\alpha}]((u,\alpha))=1, s[\overline{v\alpha}]((v,\alpha))=0}} y(s[\{\overline{u\alpha}, \overline{v\alpha}\}]),$$

$$z^-_{uv\alpha} = \sum_{\substack{s[\overline{u\alpha}] \in \Sigma_{\overline{u\alpha}},\, s[\overline{v\alpha}] \in \Sigma_{\overline{v\alpha}} \\ s[\overline{u\alpha}]((u,\alpha))=0, s[\overline{v\alpha}]((v,\alpha))=1}} y(s[\{\overline{u\alpha}, \overline{v\alpha}\}]).$$

Note that $z_{uv} = z^+_{uv\alpha} + z^-_{uv\alpha}$.

Let $D_{u\alpha} = \{s[\overline{u\alpha}] \in \Sigma_{[\overline{u\alpha}]} \mid s[\overline{u\alpha}]((u, \alpha)) = 1\}$ and $D_{v\alpha} = \{s[\overline{v\alpha}] \in \Sigma_{[\overline{v\alpha}]} \mid s[\overline{v\alpha}]((v, \alpha)) = 1\}$. Let i denote the least common ancestor of nodes $\overline{u\alpha}$ and $\overline{v\alpha}$, and $\{j, j'\}$ the two children of i. Note that $T_j = T_{j'} = T_i \cup \{j, j'\}$ and $T_{\overline{u\alpha}}, T_{\overline{v\alpha}} \supseteq T_j$. Because $\overline{u\alpha} \notin T_{\overline{v\alpha}}$ and $\overline{v\alpha} \notin T_{\overline{u\alpha}}$, both $\overline{u\alpha}$ and $\overline{v\alpha}$ are strictly below j and j' (respectively) in the tree decomposition.

For any choice of states $\{s[k] \in \Sigma_k\}_{k \in T_j}$ define:

$$z^+_{uv\alpha}(s[T_j]) = \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \sum_{s[\overline{v\alpha}] \notin D_{v\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\}])}{y(s[T_j])},$$

and similarly $z^-_{uv\alpha}(s[T_j])$.

In the rest of the proof, we fix states $\{s[k] \in \Sigma_k\}_{k \in T_j}$ and *condition* on the event $\mathcal{E}$ that $a[T_j] = s[T_j]$. We will show

$$\Pr\left[|\{u, v\} \cap U_\alpha| = 1 \,|\, \mathcal{E}\right] \;\geq\; \frac{1}{2}\left(z^+_{uv\alpha}(s[T_j]) + z^-_{uv\alpha}(s[T_j])\right). \tag{2.20}$$

By taking expectation over the conditioning $\mathcal{E}$, this would imply Lemma 2.3.11.

We now define the following indicator random variables (conditioned on $\mathcal{E}$).

$$I_{u\alpha} = \begin{cases} 0 & \text{if } a[\overline{u\alpha}] \notin D_{u\alpha} \\ 1 & \text{if } a[\overline{u\alpha}] \in D_{u\alpha} \end{cases} \quad \text{and} \quad I_{v\alpha} = \begin{cases} 0 & \text{if } a[\overline{v\alpha}] \notin D_{v\alpha} \\ 1 & \text{if } a[\overline{v\alpha}] \in D_{v\alpha} \end{cases}$$

Observe that $I_{u\alpha}$ and $I_{v\alpha}$ (*conditioned* on $\mathcal{E}$) are independent because $\overline{u\alpha}, \overline{v\alpha} \notin T_j$, and $\overline{u\alpha}$ and $\overline{v\alpha}$ appear in distinct subtrees under node i. So,

$$\Pr[|\{u, v\} \cap U_\alpha| = 1 \,|\, \mathcal{E}] = \Pr[I_{u\alpha} = 1] \Pr[I_{v\alpha} = 0] + \Pr[I_{u\alpha} = 0] \Pr[I_{v\alpha} = 1]. \tag{2.21}$$

For any $s[k] \in \Sigma_k$ for $k \in T_{\overline{u\alpha}} \setminus T_j$, we have by Claim 2.4.8 and $T_j \subseteq T_{\overline{u\alpha}}$ that

$$\Pr[a[T_{\overline{u\alpha}}] = s[T_{\overline{u\alpha}}] \,|\, a[T_j] = s[T_j]] \;=\; \frac{y(s[T_{\overline{u\alpha}}])}{y(s[T_j])}.$$

Therefore $\Pr[I_{u\alpha} = 1]$ equals

$$\sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \sum_{\substack{k \in T_{\overline{u\alpha}} \setminus T_j \setminus \{\overline{u\alpha}\} \\ s[k] \in \Sigma_k}} \frac{y(s[T_{\overline{u\alpha}}])}{y(s[T_j])} = \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}\}])}{y(s[T_j])}.$$

The last equality follows by repeatedly using LP constraint (2.14) and the fact that $T_{\overline{u\alpha}} \in \mathcal{P}$. Furthermore, note that $T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\} \in \mathcal{P}$; again by constraint (2.14),

$$\Pr[I_{u\alpha} = 1] = \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}\}])}{y(s[T_j])} = \sum_{s[\overline{u\alpha}] \in D_{u\alpha}} \sum_{s[\overline{v\alpha}] \in \Sigma_{\overline{v\alpha}}} \frac{y(s[T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\}])}{y(s[T_j])}$$

$$= \sum_{s[\overline{u\alpha}]\in D_{u\alpha}} \sum_{s[\overline{v\alpha}]\in D_{v\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\}])}{y(s[T_j])} + z^{+}_{uv\alpha}(s[T_j]).$$

Similarly,

$$\begin{aligned}\Pr[I_{v\alpha} = 1] &= \sum_{s[\overline{v\alpha}]\in D_{v\alpha}} \frac{y(s[T_j \cup \{\overline{v\alpha}\}])}{y(s[T_j])} \\ &= \sum_{s[\overline{u\alpha}]\in D_{u\alpha}} \sum_{s[\overline{v\alpha}]\in D_{v\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\}])}{y(s[T_j])} + z^{-}_{uv\alpha}(s[T_j]).\end{aligned}$$

$$\begin{aligned}\Pr[I_{u\alpha} = 0] &= \sum_{s[\overline{u\alpha}]\notin D_{u\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}\}])}{y(s[T_j])} \\ &= \sum_{s[\overline{u\alpha}]\notin D_{u\alpha}} \sum_{s[\overline{v\alpha}]\notin D_{v\alpha}} \frac{y(s[T_j \cup \{\overline{u\alpha}, \overline{v\alpha}\}])}{y(s[T_j])} + z^{-}_{uv\alpha}(s[T_j]).\end{aligned}$$

Now define $\{0, 1\}$ random variables X and Y jointly distributed as:

	$Y = 0$	$Y = 1$
$X = 0$	$\Pr[I_{u\alpha} = 0] - z^{-}_{uv\alpha}(s[T_j])$	$z^{-}_{uv\alpha}(s[T_j])$
$X = 1$	$z^{+}_{uv\alpha}(s[T_j])$	$\Pr[I_{u\alpha} = 1] - z^{+}_{uv\alpha}(s[T_j])$

Note that $\Pr[X = 1] = \Pr[I_{u\alpha} = 1]$ and $\Pr[Y = 1] = \Pr[I_{u\alpha} = 1] - z^{+}_{uv\alpha}(s[T_j]) + z^{-}_{uv\alpha}(s[T_j]) = \Pr[I_{v\alpha} = 1]$. So, applying Observation 2.3.7 and using (2.21) we have $\Pr[|\{u, v\} \cap U_\alpha| = 1 \,|\, \mathcal{E}]$ is at least

$$\frac{1}{2}\left(\Pr[X = 0, Y = 1] + \Pr[X = 1, Y = 0]\right),$$

which implies (2.20). □

Lemma 2.4.11. *For any $u, v \in V$, the probability that edge (u, v) is cut by the k-partition $\{U_\alpha\}_{\alpha=1}^{k}$ is at least $\frac{1}{4}\sum_{\alpha=1}^{k} z_{uv\alpha}$.*

Proof. Edge (u, v) is cut by $\{U_\alpha\}_{\alpha=1}^{k}$ if and only if $u \in U_\alpha$ and $v \in U_\beta$ for some $\alpha \neq \beta$. Enumerating all partition parts and applying Lemmas 2.4.9 and 2.4.10, we get that the probability is at least $\frac{1}{4}\sum_{\alpha=1}^{k} z_{uv\alpha}$. The extra factor of $\frac{1}{2}$ is because any cut edge (u, v) is cut by the partition twice: by the parts containing u and v. □

From Lemmas 2.4.6, 2.4.7 and 2.4.11, we obtain Theorem 2.4.1.

2.5 Applications

2.5.1 Max-k-Cut Applications

We claim that MSO_2 is powerful enough to model various graph properties; to that end, consider the following formulae, meant to model that a set S is a vertex cover (φ_{vc}), an independent set (φ_{is}), a dominating set (φ_{ds}), and a connected set (φ_{conn}), respectively:

$$\varphi_{\text{vc}}(S) \equiv \forall \{u,v\} \in E : (u \in S) \vee (v \in S).$$
$$\varphi_{\text{is}}(S) \equiv \forall \{u,v\} \in E : \neg\left((u \in S) \wedge (v \in S)\right).$$
$$\varphi_{\text{ds}}(S) \equiv \forall v \in V : \exists u \in S : (v \notin S) \implies \{u,v\} \in E.$$
$$\varphi_{\text{conn}}(S) \equiv \neg\big[\exists U, V \subseteq S : U \cap V = \emptyset \wedge U \cup V = S \wedge \neg\big(\exists \{u,v\} \in E : u \in U \wedge v \in V\big)\big].$$

We argue as follows: φ_{vc} is true if every edge has at least one endpoint in S; φ_{is} is true if every edge does not have both endpoints in S; φ_{ds} is true if for each vertex v not in S there is a neighbor u in S; finally, φ_{conn} is true if there does not exist a partition U, V of S with an edge going between U and V.

We also show how to handle the precedence constraint. Let G be a directed graph; we require S to satisfy that, for each arc $(u,v) \in E$, either $v \notin S$, or $u, v \in S$. This can be handled directly with CSP constraints: we have a binary variable for each vertex with the value 1 indicating that a vertex is selected for S; then, for each arc $(u,v) \in E$, we have a constraint $C_{(u,v)} = \{(1,0),(0,0),(1,1)\}$.

It is known [44] that many other properties are expressible in MSO_2, such as that S is k-colorable, k-connected (both for fixed $k \in \mathbb{N}$), planar, Hamiltonian, chordal, a tree, not containing a list of graphs as minors, etc. It is also known how to encode directed graphs into undirected graphs in an "MSO-friendly" way [44], which allows the expression of various properties of directed graphs. Our results also extend to so-called *counting* MSO, where we additionally have a predicate of the form $|X| = p \bmod q$ for a fixed integer $q \in \mathbb{N}$.

2.5.2 Applications with Specific Dynamic Programs

For MSO expressible constraints, by Theorem 2.4.1, we automatically get that there exists a $\frac{1}{2}$-approximation algorithm via MSO and CSP dynamic program. However, this does not directly gives an actual algorithm description. In this section, we show a number of graph constraints that satisfy Assumption 2.2.1 by explicitly giving the required dynamic program structure, and thereby obtain $\frac{1}{2}$-approximation algorithms for GCMC under these constraints (on bounded-treewidth graphs). The constraints we consider includes indepen-

dent set, vertex cover, precedence, dominating set and connectivity.

Recall that the underlying graph G is given by its tree decomposition from Theorem 2.2.1, i.e. $(\mathcal{T} = (I, F), \{X_i | i \in I\})$. Recall also the definition of a dynamic program on this tree decomposition, as given in Definition 2.2.8. We first have some simple but useful observations.

Observation 2.5.1. *For any node i with children nodes j_1, j_2, if $u, v \in V_i$ and $(u, v) \in E$, then either $u, v \in X_i$ or $u, v \in V_j$ for some $j \in \{j_1, j_2\}$.*

Proof. Since $(u, v) \in E$, by the tree decomposition, there is some node $i' \in I$ with $u, v \in X_{i'}$. If node i' is an ancestor of node i or node i itself, $u, v \in X_i$ by the first property of the tree decomposition. If node i' is strictly below node i, then $u, v \in X_{i'}$ implies $u, v \in V_j$ for some $j \in \{j_1, j_2\}$.

□

Observation 2.5.2. *For any node i with children nodes j_1, j_2, if $S_j \subseteq V_j$ for $j \in \{j_1, j_2\}$, $S_j \cap X_i = S_j \cap X_i \cap X_j$.*

Proof. For any vertex $v \in S_j \cap X_i$, $v \in X_i$ and $v \in V_j$. By the first property of the tree decomposition, $X_i \cap (V_j \setminus X_j) = \emptyset$. Hence $v \in X_j$.

□

Independent Set Given graph $G = (V, E)$ and edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$, we want to maximize $c(\delta S)$ where S is an independent set in G. Recall that $S \subseteq V$ is an independent set if there is no edge of G induced on S.

For each node $i \in I$ define state space $\Sigma_i = \{\sigma \subseteq X_i \,|\, \sigma \text{ is an independent set}\}$. For each node $i \in I$ and $\sigma \in \Sigma_i$, we define:

- set $X_{i,\sigma} = \sigma$.
- collection $\mathcal{H}_{i,\sigma} = \{S \subseteq V_i \,|\, X_i \cap S = \sigma \text{ and } S \text{ is an independent set in } G[V_i]\}$.
- $\mathcal{F}_{i,\sigma} = \{(w_{j_1}, w_{j_2}) \,|\, \text{for each } j \in \{j_1, j_2\},\ w_j \in \Sigma_j \text{ such that } w_j \cap X_i = \sigma \cap X_j\}$ which denotes valid combinations. Note that the condition $w_j \cap X_i = \sigma \cap X_j$ enforces w_j to agree with σ on vertices of $X_i \cap X_j$.

We next show that these satisfy all the conditions in Assumption 2.2.1. Note that as all subgraphs we consider are induced graphs, a set S is independent in the induced graph if and only if it is independent in G.

Assumption 2.2.1, part 1. We have $t = \max |\Sigma_i| \leq 2^k$. Also $p = \max |\mathcal{F}_{i,\sigma}| \leq t^2$ because each node has at most two children. For constant k, both $t, p = O(1)$.

Assumption 2.2.1, part 2. By definition, for any $S \in \mathcal{H}_{i,\sigma}$ we have $S \cap X_i = \sigma = X_{i,\sigma}$.

Assumption 2.2.1, part 3. For any leaf $\ell \in I$ and $\sigma \in \Sigma_\ell$ it is clear that $\mathcal{H}_{\ell,\sigma} = \{X_{\ell,\sigma}\}$.

Assumption 2.2.1, part 4. Consider now any non-leaf node i and $\sigma \in \Sigma_i$. Let

$$\mathcal{Z} \;=\; \{X_{i,\sigma} \cup S_{j_1} \cup S_{j_2} \;:\; S_{j_1} \in \mathcal{H}_{j_1,w_{j_1}}, S_{j_2} \in \mathcal{H}_{j_2,w_{j_2}}, (w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}\}. \tag{2.22}$$

We first prove $\mathcal{H}_{i,\sigma} \subseteq \mathcal{Z}$. For any $S \in \mathcal{H}_{i,\sigma}$ and child $j \in \{j_1, j_2\}$ let $S_j = S \cap V_j$ and $w_j = S \cap X_j$; because S is independent in $G[V_i]$, S_j is also an independent set in $G[V_j]$, and $S_j \in \mathcal{H}_{j,w_j}$. Note that $S \cap X_i = X_{i,\sigma} = \sigma$. Because $V_i = X_i \cup V_{j_1} \cup V_{j_2}$, we have $S = X_{i,\sigma} \cup S_{j_1} \cup S_{j_2}$. Moreover, we have $\sigma \cap X_j = S \cap X_i \cap X_j = w_j \cap X_i$ for each $j \in \{j_1, j_2\}$. So we have $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$ and hence $S \in \mathcal{Z}$.

We next prove $\mathcal{Z} \subseteq \mathcal{H}_{i,\sigma}$. Consider any $S = X_{i,\sigma} \cup S_{j_1} \cup S_{j_2}$ as in (2.22). For $j \in \{j_1, j_2\}$ by definition of $\mathcal{F}_{i,\sigma}$ and $\mathcal{H}_{j,w_j}$, we have $\sigma \cap X_j = w_j \cap X_i = (S_j \cap X_j) \cap X_i$. By Observation 2.5.2 and $S_j \subseteq V_j$, we have $X_i \cap S_j = X_i \cap X_j \cap S_j = \sigma \cap X_j$. Thus we have $X_i \cap S = \sigma$. It just remains to prove that S is an independent set in $G[V_i]$. Because S_{j_1}, S_{j_2} and $X_{i,\sigma}$ are independent sets, if S were not independent then we must have an edge (u, v) where $u, v \in S$. By Observation 2.5.1, $u, v \in X_i$ or $u, v \in V_j$ for some $j \in \{j_1, j_2\}$, which is a contradiction to the fact that S_{j_1}, S_{j_2}, and $X_{i,\sigma}$ are all independent sets. So $S \in \mathcal{H}_{i,\sigma}$.

Assumption 2.2.1, part 5. This follows directly from the definition of $\mathcal{H}_{i,\sigma}$.

Vertex Cover Given graph $G = (V, E)$ and edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$ we want to maximize $c(\delta S)$ where S is a vertex cover in G. Recall that $S \subseteq V$ is a vertex cover if S contains at least one end-point of each edge in E.

We can use the independent set application as a black box for the vertex cover application. This is because S is an independent set of G if and only if $V \setminus S$ is a vertex cover of G and $c(\delta S) = c(\delta(V \setminus S))$.

Precedence Given a directed graph $G = (V, E)$ and edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$, we want to maximize $c(\delta S)$ where S is set of vertices that satisfies precedence constraints in G. A subset $S \subseteq V$ is said to satisfy precedence constraints if, for each arc $(u, v) \in E$, either $v \notin S$ or both $u, v \in S$. We assume that the undirected version of graph G (obtained by dropping arc directions) has bounded treewidth, and is given by a tree decomposition as in Theorem 2.2.1.

For each node $i \in I$, define state space $\Sigma_i = \{\sigma \subseteq X_i \,|\, \sigma \text{ satisfies precedence constraints in } G[X_i]\}$. For each node $i \in I$ and $\sigma \in \Sigma_i$, we define:

- set $X_{i,\sigma} = \sigma$.
- $\mathcal{H}_{i,\sigma} = \{S \subseteq V_i \,|\, X_i \cap S = \sigma \text{ and } S \text{ satisfies precedence constraint in } G[V_i]\}$.
- $\mathcal{F}_{i,\sigma} = \{(w_{j_1}, w_{j_2}) \,|\, \text{for each } j \in \{j_1, j_2\},\ w_j \in \Sigma_j \text{ such that } w_j \cap X_i = \sigma \cap X_j\}$ which denotes valid combinations. Note that the condition $w_j \cap X_i = \sigma \cap X_j$ enforces w_j to agree with σ on vertices of $X_i \cap X_j$.

We next show that these satisfy all the conditions in Assumption 2.2.1.

Assumption 2.2.1 part 1. We have $t = \max |\Sigma_i| \leq 2^k$ and $p = \max |\mathcal{F}_{i,\sigma}| \leq t^2$. So for constant k, both $t, p = O(1)$.

Assumption 2.2.1 part 2. By definition, for any $S \in \mathcal{H}_{i,\sigma}$ we have $S \cap X_i = \sigma = X_{i,\sigma}$.

Assumption 2.2.1 part 3. For any leaf $\ell \in I$ and $\sigma \in \Sigma_\ell$ it is clear that $\mathcal{H}_{\ell,\sigma} = \{X_{\ell,\sigma}\}$.

Assumption 2.2.1 part 4. Consider any non-leaf node i and $\sigma \in \Sigma_i$. Let $\mathcal{Z}$ be as in (2.22) with the new definitions of $\mathcal{H}$ and $\mathcal{F}$ for precedence (as above).

We first prove $\mathcal{H}_{i,\sigma} \subseteq \mathcal{Z}$. For any $S \in \mathcal{H}_{i,\sigma}$ and child $j \in \{j_1, j_2\}$ let $S_j = S \cap V_j$ and $w_j = S \cap X_j$. Because S satisfies precedence constraints in $G[V_i]$, S_j satisfies precedence constraints in $G[V_j]$; and with $S_j \cap X_j = S \cap V_j \cap X_j = S \cap X_j = w_j$, we have $S_j \in \mathcal{H}_{j,w_j}$. Note that $S \cap X_i = X_{i,\sigma} = \sigma$. Because $V_i = X_i \cup V_{j_1} \cup V_{j_2}$, we have $S = X_{i,\sigma} \cup S_{j_1} \cup S_{j_2}$. Moreover, we have $\sigma \cap X_j = S \cap X_i \cap X_j = w_j \cap X_i$ for each $j \in \{j_1, j_2\}$. So we have $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$ and hence $S \in \mathcal{Z}$.

We next prove $\mathcal{Z} \subseteq \mathcal{H}_{i,\sigma}$. Consider any $S = X_{i,\sigma} \cup S_{j_1} \cup S_{j_2}$ as in (2.22). For $j \in \{j_1, j_2\}$ by definition of $\mathcal{F}_{i,\sigma}$ and $\mathcal{H}_{j,w_j}$, we have $\sigma \cap X_j = w_j \cap X_i = (S_j \cap X_j) \cap X_i$. By Observation 2.5.2 and $S_j \subseteq V_j$, we have $X_i \cap S_j = X_i \cap X_j \cap S_j = \sigma \cap X_j$. Thus we have $X_i \cap S = \sigma$. It just remains to prove that S satisfies precedence constraints in $G[V_i]$. For any edge $(u, v) \in G[V_i]$, by Observation 2.5.1, it must be that at least one of X_i, V_{j_1} or V_{j_2} contains both u and v. By definition of σ and $\mathcal{H}_{i,\sigma}$, we obtain that $v \in S \implies u \in S$. That is, S satisfies precedence constraints in $G[V_i]$.

Assumption 2.2.1 part 5. This follows directly from the definition of $\mathcal{H}_{i,\sigma}$ at the root.

Dominating Set Given graph $G = (V, E)$ and edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$, we want to maximize $c(\delta S)$ where S is a dominating set in G. Recall that $S \subseteq V$ is a dominating set if every vertex in V is either in S or a neighbor of some vertex in S. For a vertex subset $S \subseteq V$, we use $N(S)$ to denote the set of vertices in S as well as neighbors of S.

For each node $i \in I$ (other than the root), define the state space

$$\Sigma_i = \{(B_i, Y_i) \,|\, B_i \subseteq X_i,\, Y_i \subseteq X_i\}.$$

Here, state $\sigma = (B_i, Y_i)$ specifies which subset B_i of the vertices (in X_i) are included in the solution and what subset Y_i of the vertices (in X_i) *need not* be dominated.

For the root r, define $\Sigma_r = \{(B_r, Y_r) \,|\, B_r \subseteq X_r,\, Y_r = \emptyset\}$.

For each node $i \in I$ and $\sigma = (B_i, Y_i) \in \Sigma_i$, we define:

- set $X_{i,\sigma} = B_i$
- $\mathcal{H}_{i,\sigma} = \{S \subseteq V_i |\, X_i \cap S = B_i, S \text{ is a dominating set of } V_i \setminus Y_i \text{ in } G[V_i]\}$
- $\mathcal{F}_{i,\sigma}$ consists of (w_{j_1}, w_{j_2}) where for $j \in \{j_1, j_2\}$, $w_j = (B_j, Y_j) \in \Sigma_j$ such that $B_i \cap X_j = B_j \cap X_i$ and $V_i \setminus Y_i \subseteq (V_{j_1} \setminus Y_{j_1}) \cup (V_{j_2} \setminus Y_{j_2}) \cup N(B_i)$. Note that for some states there may be no such pair (w_{j_1}, w_{j_2}) : in this case $\mathcal{F}_{i,\sigma}$ is empty.

Assumption 2.2.1, part 1. For each node i, the possible number of vertex subsets B_i, Y_i is at most 2^k. So we have $t = \max |\Sigma_i| \leq 2^{2k} = O(1)$ and $p = \max |\mathcal{F}_{i,\sigma}| \leq t^2 = O(1)$.

Assumption 2.2.1, part 2. This follows directly from the definition of $\mathcal{H}_{i,\sigma}$ and $X_{i,\sigma}$.

Assumption 2.2.1, part 3. For any leaf $\ell \in I$ and $\sigma = (B_\ell, Y_\ell) \in \Sigma_\ell$ it is clear that $\mathcal{H}_{\ell,\sigma} = \{X_{\ell,\sigma}\}$ if B_ℓ is a dominating set of $V_\ell \setminus Y_\ell$ and $\emptyset$ otherwise.

Assumption 2.2.1, part 4. Consider a non-leaf node $i \in I$ and $\sigma = (B_i, Y_i) \in \Sigma_i$. Let $\mathcal{Z}$ be as in (2.22) with the new definitions of $\mathcal{H}$ and $\mathcal{F}$ for dominating set (as above). To reduce notation we just use j to denote a child of i; we will not specify $j \in \{j_1, j_2\}$ each time.

We first prove $\mathcal{H}_{i,\sigma} \subseteq \mathcal{Z}$. For any $S \in \mathcal{H}_{i,\sigma}$, let $S_j = V_j \cap S$ and $B_j = X_j \cap S$. Let $Y_j = X_j \setminus N(S_j)$. Let $w_j = (B_j, Y_j)$. We will show that $S_j \in \mathcal{H}_{j,w_j}$ and $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$.

1. $S_j \in \mathcal{H}_{j,w_j}$. By definition of B_j, we have $X_j \cap S_j = X_j \cap V_j \cap S = X_j \cap S = B_j$. We only need to prove S_j is a dominating set of $V_j \setminus Y_j$ in $G[V_j]$. For any $v \in V_j \setminus Y_j$, we consider the following cases:

   - $v \in X_i$. Because $v \in V_j$, we have $v \in X_j$. Because $v \notin Y_j$ and $v \in X_j$, $v \in X_j \setminus Y_j = X_j \setminus (X_j \setminus N(S_j)) = N(S_j) \cap X_j$.
   - $v \notin X_i$. So $v \notin Y_i$, which implies $v \in V_i \setminus Y_i$. So v is dominated by S, i.e. there is some $u \in S$ with $(u, v) \in E$. Then by Observation 2.5.1, because $v \in V_j$ and $v \notin X_i$, we must have $u \in V_j$. And using $S_j = S \cap V_j$, we have $u \in S_j$, i.e. $v \in N(S_j)$.

Therefore we obtain that S_j dominates $V_j \setminus Y_j$. Thus $S_j \in \mathcal{H}_{j,w_j}$.

2. $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$. We have $B_i \cap X_j = B_j \cap X_i$ and $Y_j = X_j \setminus N(S_j)$ by definition of w_j. It remains to show that $V_i \setminus Y_i \subseteq (V_{j_1} \setminus Y_{j_1}) \cup (V_{j_2} \setminus Y_{j_2}) \cup N(B_i)$. For all $v \in V_i \setminus Y_i$, we have v is dominated by $S \subseteq V_i$, i.e. there is $u \in S$ with $(u, v) \in E$. By Observation 2.5.1, we have the following two cases:

   - If $u \in X_i$, then we have $u \in B_i$ and so $v \in N(B_i)$.
   - If $u, v \in V_j$, then we have $u \in S_j$ and $v \in N(S_j)$. Therefore $v \notin X_j \setminus N(S_j) = Y_j$, i.e. $v \in V_j \setminus Y_j$.

     Therefore, for all $v \in V_i \setminus Y_i$, we have $v \in (V_{j_1} \setminus Y_{j_1}) \cup (V_{j_2} \setminus Y_{j_2}) \cup N(B_i)$. Thus $V_i \setminus Y_i \subseteq (V_{j_1} \setminus Y_{j_1}) \cup (V_{j_2} \setminus Y_{j_2}) \cup N(B_i)$ as desired.

Next we prove $\mathcal{Z} \subseteq \mathcal{H}_{i,\sigma}$. Consider any $S \in \mathcal{Z}$ given by $S = B_i \cup S_{j_1} \cup S_{j_2}$ as in (2.22). The fact that $S \cap X_i = B_i$ follows exactly as in the case of an independent-set constraint. It remains to show that S is a dominating set of $V_i \setminus Y_i$. By the definition of $\mathcal{F}_{i,\sigma}$, $V_i \setminus Y_i \subseteq (V_{j_1} \setminus Y_{j_1}) \cup (V_{j_2} \setminus Y_{j_2}) \cup N(B_i)$. Because S_j is a dominating set of $V_j \setminus Y_j$ and B_i is a dominating set of $N(B_i)$, we have $V_i \setminus Y_i$ is dominated by $B_i \cup S_{j_1} \cup S_{j_2} = S$. Thus we have S is a dominating set of $V_i \setminus Y_i$. So $S \in \mathcal{H}_{i,\sigma}$.

Assumption 2.2.1, part 5. By our definition of Σ_r, any solution given by $\mathcal{H}_{r,\sigma}$ requires all vertices to be dominated. Thus this assumption is satisfied.

Connectivity Given graph $G = (V, E)$ and edge-weights $c : \binom{V}{2} \to \mathbb{R}_+$, we want to maximize $c(\delta S)$ where S is a connected vertex-set in G. We say that a vertex subset $S \subseteq V$ is connected if the induced graph $G[S]$ is connected.

For each node $i \in I$ (other than the root), define the state space

$$\Sigma_i = \{(B_i, P_i) \mid B_i \subseteq X_i,\ P_i \text{ is a partition of } B_i\}.$$

Here, a state $\sigma = (B_i, P_i)$ specifies (1) the subset $B_i \subseteq X_i$ that is included in the solution and (2) the connectivity pattern of B_i in the subgraph $G[V_i]$ which is given by partition P_i where all vertices in the same part are connected in the solution induced on V_i. Recall that any partition corresponds to an equivalence relation: vertices u, v are related if and only if u, v lie in the same part. We also need the notion of a partition union. Given two partitions Q and R, their union $P = Q \cup R$ is the partition given by the transitive closure of the union of the equivalence relations of Q and R.

At the root, we set $\Sigma_r = \{(B_r, P_r) \mid B_r \subseteq X_r,\ P_r = \{B_r\}\}$.

For each node $i \in I$ and $\sigma = (B_i, P_i) \in \Sigma_i$, we define:

- set $X_{i,\sigma} = B_i$.
- $\mathcal{H}_{i,\sigma} = \{S \subseteq V_i \,|\, X_i \cap S = B_i$, each part of P_i is connected in $G[S]$ and every connected component of $G[S]$ contains some vertex of $B_i\}$.
- partition $\bar{P}_i$ denotes the connected components in $G[B_i]$.
- $\mathcal{F}_{i,\sigma}$ consists of (w_{j_1}, w_{j_2}) where for $j \in \{j_1, j_2\}$, $w_j = (B_j, P_j) \in \Sigma_j$ such that $B_i \cap X_j = B_j \cap X_i$ and each part of P_j contains some vertex of B_i, and P_i is satisfied[2] by $\bar{P}_i \cup P_{j_1} \cup P_{j_2}$. Note that for some states there may be no such pair (w_{j_1}, w_{j_2}) : in this case $\mathcal{F}_{i,\sigma}$ is empty.

Assumption 2.2.1, part 1. For each node i, the possible number of vertex subsets B_i is at most 2^k and the possible number of partitions P_i is at most k^k. So $t = \max |\Sigma_i| \leq k^{k+1} = O(1)$ and $p = \max |\mathcal{F}_{i,\sigma}| \leq t^2 = O(1)$.

Assumption 2.2.1, part 2. This follows directly from the definition of $\mathcal{H}_{i,\sigma}$ and $X_{i,\sigma}$.

Assumption 2.2.1, part 3. For any leaf $\ell \in I$ and $\sigma = (B_\ell, P_\ell) \in \Sigma_\ell$ it is clear that $\mathcal{H}_{\ell,\sigma} = \{X_{\ell,\sigma}\}$ if each part of P_ℓ is connected in $G[B_\ell]$; $\mathcal{H}_{\ell,\sigma} = \emptyset$ otherwise.

Assumption 2.2.1, part 4. Consider a non-leaf node $i \in I$ and $\sigma = (B_i, P_i) \in \Sigma_i$. To reduce notation we just use j to denote a child of i; we will not specify $j \in \{j_1, j_2\}$ each time. Let $\mathcal{Z}$ be as in (2.22) with the new definitions of $\mathcal{H}$ and $\mathcal{F}$ for connectivity (as above).

We first prove $\mathcal{H}_{i,\sigma} \subseteq \mathcal{Z}$. For any $S \in \mathcal{H}_{i,\sigma}$, let $S_j = V_j \cap S$ and $B_j = X_j \cap S$. Let P_j be a partition of B_j with a part $C \cap B_j$ for every connected component C in $G[S_j]$. Let $w_j = (B_j, P_j)$. We will show that $S_j \in \mathcal{H}_{j,w_j}$ and $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$.

1. $S_j \in \mathcal{H}_{j,w_j}$. By definition of B_j, we have $X_j \cap S_j = X_j \cap V_j \cap S = X_j \cap S = B_j$. By definition of P_j, each part of P_j is connected in $G[S_j]$. We only need to prove each connected component of $G[S_j]$ has at least one vertex of B_j. We show the following stronger claim.

   Claim 2.5.3. *There is at least one vertex of $B_j \cap B_i$ in each connected component C of $G[S_j]$.*

   Proof. Suppose that C has some vertex $u \in B_i$. Then we have $u \in S_j \cap X_i = S_j \cap X_i \cap X_j$ by Observation 2.5.2. This implies $u \in B_i \cap B_j$.

[2] A partition P is said to be satisfied by another partition P' if every pair of elements in the same part of P also lie in the same part of P'.

Now suppose (for contradiction) that C does not contain any vertex of B_i. By $S \in \mathcal{H}_{i,\sigma}$ we know that in the (larger) graph $G[S]$, component C has to be connected to some vertex $u \in B_i$. Then there is a path π in $G[S]$ from some vertex $u' \in C$ to u such that u' is the *only* vertex of C on π (see also Figure 2.3). Let (u', v') be the first edge of π, so $u' \in C \subseteq S_j$ and $v' \in S \setminus S_j$. By Observation 2.5.1, because $(u', v') \in E$ and $u', v' \in V_i$, we have either $u', v' \in X_i$ or $u, v \in V_{j'}$ for some $j' \in \{j_1, j_2\}$. We consider two cases:

- $u' \in X_i$. Because $u' \in C \subseteq S_j \subseteq S$ and $B_i = X_i \cap S$, $u' \in B_i$ which is a contradiction.
- $u', v' \in V_{j'}$. This means that $v' \in S \cap V_{j'} = S_{j'}$. As $v' \in S \setminus S_j$ we have $j' \neq j$. So u' appears in both children of node i. By the first property of the tree decomposition, $u' \in X_i$. The remaining proof is the same as the first case.

So we obtain a contradiction in either case above.

□

Based on this claim, each component C in $G[S_j]$ has at least one vertex of $B_j \cap B_i$. Therefore $S_j \in \mathcal{H}_{j,w_j}$.

2. $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$. We have $B_i \cap X_j = B_j \cap X_i$ by definition of w_j. Because we already proved that each connected component of $G[S_j]$ has at least one vertex of $B_j \cap B_i$, we know that each part of partition P_j has at least one vertex of B_i. By the tree decomposition we have $G[V_i] = G[X_i] \cup G[V_{j_1}] \cup G[V_{j_2}]$, so $G[S] = G[B_i] \cup G[S_{j_1}] \cup G[S_{j_2}]$. Hence partition P_i is satisfied by $\bar{P}_i \cup P_{j_1} \cup P_{j_2}$. Thus $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$.

Next we prove $\mathcal{Z} \subseteq \mathcal{H}_{i,\sigma}$. Consider any $S \in \mathcal{Z}$ given by $S = B_i \cup S_{j_1} \cup S_{j_2}$ as in (2.22). The fact that $S \cap X_i = B_i$ follows exactly as in the case of an independent-set constraint. Because P_i is satisfied by $\bar{P}_i \cup P_{j_1} \cup P_{j_2}$ and S_j connects up each part of P_j, it follows that $G[S] = G[B_i] \cup G[S_{j_1}] \cup G[S_{j_2}]$ connects up each part of P_i. It remains to show that each connected component of $G[S]$ has a vertex of B_i. Because $(w_{j_1}, w_{j_2}) \in \mathcal{F}_{i,\sigma}$ we know that each part of P_j has a B_i-vertex. By $S_j \in \mathcal{H}_{j,w_j}$, we know that each component of $G[S_j]$ contains some vertex $u \in B_j$, and this vertex u is connected to some vertex $v \in B_i$ (as each part of P_j contains a B_i-vertex); so every component of $G[S_j]$ contains some vertex of B_i. Hence each component of $G[S] = G[B_i] \cup G[S_{j_1}] \cup G[S_{j_2}]$ contains some vertex of B_i.

Assumption 2.2.1, part 5. By our definition of Σ_r, any solution given by $\mathcal{H}_{r,\sigma}$ requires all chosen vertices to be connected. Thus this assumption is satisfied.

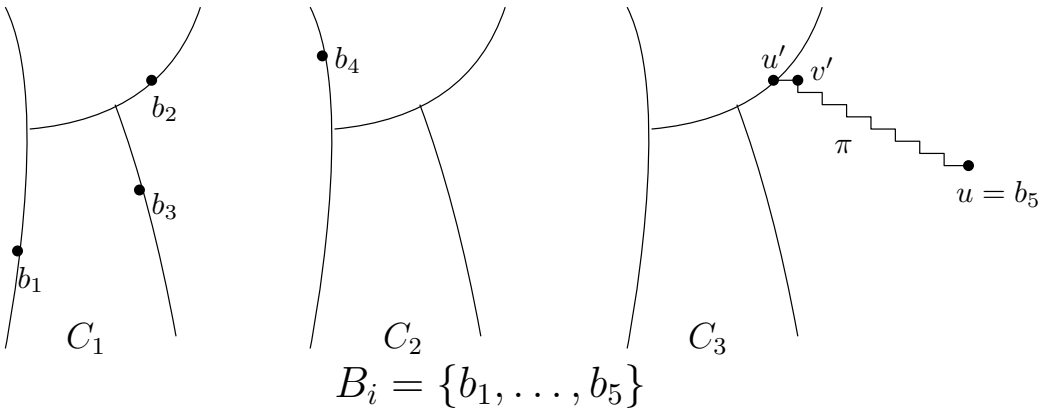

Figure 2.3: Connected components of $G[S_j]$

2.5.3 Bounded-genus and Excluded-minor Graphs

In this section, we extend our results beyond constant treewidth graph by combining known decomposition results and prove Corollary 2.1.3.1 and 2.1.3.2. We consider two larger class of graphs: excluded-minor graphs and bounded-genus graphs, with a small loss in the approximation ratio.

Excluded-minor graph A minor of a graph G is a graph obtained from G by contracting edges, deleting edges, and deleting isolated vertices. An H minor is a minor isomorphic to graph H. An H-minor-free graph is a graph where H is not a minor of it. For example, the famous planar graphs are $(K_5, K_{3,3})$-minor-free graph, where K_5 is complete graph of five vertices and $K_{3,3}$ is complete bipartite graph of three vertices on each part.

The following decomposition can convert any excluded-minor graph into graphs of bounded treewidth.

Theorem 2.5.4. *[47] For a fixed graph H, there exists a constant c_H such that, for any integer $h \geq 1$ and for every H-minor-free graph G, the vertices of G can be partitioned into $h+1$ sets such that any h of those sets induce a graph of treewidth at most $c_H h$. Furthermore, such partition can be found in polynomial time.*

We first state a simple claim.

Claim 2.5.5. *Let $V_1, \ldots, V_h$ be a partition of V and $S' \subseteq V$. Then*

1. *there is some $i \in \{1, \ldots, h\}$ such that $c(\delta(S' \setminus V_i)) \geq (1 - \frac{2}{h})c(\delta S')$.*
2. *there is some $i \in \{1, \ldots, h\}$ such that $c(\delta(S' \cup V_i)) \geq (1 - \frac{2}{h})c(\delta S')$.*

Proof. Because $V_1, \ldots, V_h$ is a partition of V, we have $\sum_{i=1}^{h} c(\delta(S' \cap V_i)) \leq 2c(\delta S')$. So $\min_i c(\delta(S' \cap V_i)) \leq \frac{2}{h} c(\delta S')$, which implies $\max_i c(\delta(S' \setminus V_i)) \geq (1 - \frac{2}{h}) c(\delta S')$. This proves the first statement as $c(\delta S') \leq c(\delta(S' \setminus V_i)) + c(\delta(S' \cap V_i))$.

The second statement can be shown by a reduction to the first. Let $S'' = V \setminus S'$. Since the cut function is symmetric, we have

$$c(\delta(S' \cup V_i)) = c(\delta\left(V \setminus (S' \cup V_i)\right)) = c(\delta(S'' \cap (V \setminus V_i))) = c(\delta(S'' \setminus V_i)).$$

By the first property applied to the subset S'',

$$\max_i \, c(\delta(S'' \setminus V_i)) \geq (1 - \frac{2}{h}) \cdot c(\delta S'') = (1 - \frac{2}{h}) \cdot c(\delta S')$$

This proves the second statement.

□

Algorithm for GCMC **under an independent set constraint.** We first partition V into $\{V_i\}_{i=1}^{h}$ using Theorem 2.5.4. Then, for each $i = 1, \cdots, h$ we solve the GCMC instance on the constraint graph $G[V \setminus V_i]$ (which has bounded treewidth) using Theorem 4.1.1 to obtain solution S_i. Finally, we output the solution $S = \arg\max_{S \in \{S_i\}} c(\delta S_i)$. Clearly S is a feasible independent set.

In order to bound the objective value, let S^* be the optimal solution of the GCMC instance on G. For each $i = 1, \cdots, h$, $S^* \setminus V_i$ is a feasible solution to the GCMC instance on constraint graph $G[V \setminus V_i]$. Since we use a $\frac{1}{2}$-approximation algorithm for GCMC on bounded-treewidth graphs, $c(\delta S_i) \geq \frac{1}{2} c(\delta(S^* \setminus V_i))$. By Claim 2.5.5, we have $\max_i \, c(\delta(S^* \setminus V_i)) \geq (1 - \frac{2}{h}) c(\delta S^*)$. So we obtain $c(\delta S) \geq \frac{1}{2}(1 - \frac{2}{h}) c(\delta S^*)$.

Algorithm for GCMC **under dominating-set constraint.** The algorithm for a dominating-set constraint relies on a slight variant of this approach. They also start with the partition of V into $\{V_i\}_{i=1}^{h}$ using Theorem 2.5.4. For each $i = 1, \cdots, h$ we consider the graph G_i obtained by contracting V_i to a single vertex v_{new}. The edge weights on G_i are defined naturally as

$$c_i(u, v) = \begin{cases} c(u, v), \text{ if } u, v \in V \setminus V_i \\ \sum_{w \in V_i} c(u, w), \text{ if } u \in V \setminus V_i \text{ and } v = v_{new} \end{cases} . \tag{2.23}$$

Note that G_i is also a bounded-treewidth graph: its treewidth is at most one more than that of $G[V \setminus V_i]$ because adding v_{new} to every node gives a valid tree decomposition.

Next, we solve the GCMC instance on constraint graph G_i with the additional requirement that v_{new} be part of the solution. This requirement can be achieved by adding constraints $y(s(r)) = 0$ for all root states $s(r) \not\ni v_{new}$ to (LP). Let S'_i be the solution obtained by our $\frac{1}{2}$-approximation algorithm on G_i; and let $S_i = S'_i \setminus \{v_{new}\} \cup V_i$. Finally we output solution $S = \arg\max_{S \in \{S_i\}} c(\delta S_i)$. Since S'_i is a feasible dominating set in G_i, it follows that S_i is also a feasible dominating set in G.

In order to bound the objective value, let S^* be the optimal solution of the GCMC instance on G. For each $i = 1, \cdots, h$, $S^* \setminus V_i \cup \{v_{new}\}$ is a feasible solution to the GCMC instance on constraint graph $G[V \setminus V_i]$ for dominating-set. Since we use a $\frac{1}{2}$-approximation algorithm for GCMC on bounded-treewidth graphs, $c_i(\delta S'_i) \geq \frac{1}{2} c_i(\delta(S^* \setminus V_i \cup \{v_{new}\})) = \frac{1}{2} c(\delta(S^* \cup V_i))$; the last equality is by definition of graph G_i. Since $v_{new} \in S'_i$ and $S_i = S'_i \setminus \{v_{new}\} \cup V_i$, we have $c(\delta S_i) = c_i(\delta S'_i) \geq \frac{1}{2} c(\delta(S^* \cup V_i))$. By Claim 2.5.5, we have $\max_i c(\delta(S^* \cup V_i)) \geq (1 - \frac{2}{h}) c(\delta S^*)$. So $c(\delta S) \geq \frac{1}{2}(1 - \frac{2}{h}) c(\delta S^*)$.

Choosing h to be a large enough constant in Theorem 2.5.4, we obtain a $(\frac{1}{2} - \epsilon)$-approximation algorithm for GCMC under independent-set, vertex-cover or dominating-set constraints in an excluded minor graph. This proves Corollary 2.1.3.1.

Bounded-genus graph A 2-cell embedding of a graph G in a surface Σ (two-dimensional manifold) is a drawing of the vertices as points in Σ and the edges as curves in Σ such that no two points coincide, two curves intersect only at shared endpoints, and every face (region) bounded by edges is an open disk. The genus of a graph is defined as the minimum genus of a surface in which G can be 2-cell embedded. A graph has bounded genus if its genus is O(1). It is known that every bounded genus graph is an excluded-minor graph [2]. Planar graphs are special bounded genus graph with genus 0.

For bounded-genus graph, recall the following decomposition of any bounded-genus graph into graphs of bounded treewidth.

Theorem 2.5.6. *[48] For a bounded-genus graph G and an integer h, the edges of G can be partitioned in h color classes $E_1, \ldots, E_h$ such that contracting all the edges in any color class leads to a graph with treewidth $O(h)$. Further, the color classes have the following property: if edge $e = (u, v)$ is in class i, then every edge e' such that $e \cap e' \neq \emptyset$ is in class $i - 1$ or i or $i + 1$.*

The algorithm for a connectivity constraint is as follows. For each $i = 1, \cdots h$, solve the GCMC instance on the graph G/E_i (contract E_i) which has bounded treewidth, to obtain solution S_i. Let $S'_i = \{v | v \in S_i$ or v is contracted to some vertex of $S_i\}$; note that S'_i is connected in G. We output the solution that maximizes $c(\delta S'_i)$. The proof of Corollary

2.1.3.2 using Theorem 2.5.6 is identical to the proof in [70] for the "uniform" connected max-cut problem. Although our edge-weights c are defined on a complete graph, the proof of [70] works directly, and we omit the details.

2.6 Conclusion

In this chapter we obtained $\frac{1}{2}$-approximation algorithms for graph-MSO-constrained max-k-cut problems, where the constraint graph has bounded treewidth. Our approach was based on a strong LP relaxation that utilized a dynamic programming structure in the constraints as well as the Sherali-Adams hierarchy. Getting an approximation ratio better than $\frac{1}{2}$ for any of these problems is an interesting question, even for a specific MSO-constraint; this is likely to require the use of SDP relaxations.

Regarding Remark 2.4.2, could our algorithm be improved to an FPT algorithm (runtime $g(\tau)n^{O(1)}$ for some function g)? If not, is there an FPT algorithm parameterized by the (more restrictive) tree-depth of G?

Credits: The results in this chapter are from "Approximating graph-constrained max-cut" [115], obtained jointly with Jon Lee, and Viswanath Nagarajan and "Approximating Max-Cut under Graph-MSO Constraints" [90], obtained jointly with Martin Koutecký, Jon Lee, and Viswanath Nagarajan.

CHAPTER 3

Online Covering with ℓ_q-Norm Objectives

3.1 Introduction

Online algorithms are widely used to deal with optimization under uncertainty. They model the situation where the input arrives as a sequence and the algorithm must respond to each incoming input immediately. Moreover, they take into account that at any point in time future input is unknown. In this situation, we wish to design algorithms that always return good a solution for any possible future.

Formally, an online algorithm receives a sequence of requests $\sigma = \sigma(1), \ldots, \sigma(m)$. These requests must be served in the order of occurrence. When serving request $\sigma(t)$, an online algorithm does not know requests $\sigma(t')$ with $t' > t$. Serving the requests incurs cost, and the goal is to minimize the total cost paid on the entire request sequence. This process can be viewed as a request answer game. An adversary generates requests, and an online algorithm has to serve them one at a time. The performance of online algorithms is usually evaluated using competitive analysis [118].

Definition 3.1.1 (Competitive ratio)**.** *The competitive ratio of an online algorithm ALG is the worst case (i.e., maximum) over possible futures σ of the ratio: $ALG(\sigma)/OPT(\sigma)$, where $ALG(\sigma)$ is the cost of ALG on σ and $OPT(\sigma)$ is the least possible cost on σ.*

We note that competitive analysis is a strong worst-case performance measure. Here an online algorithm ALG is compared to an optimal offline algorithm OPT that knows the entire request sequence σ in advance and can serve it with minimum cost.

An important approach to design online algorithms is using potential functions. The potential function Φ is typically served as the amortized online cost on requests σ. Here $\Phi(t)$ is the value of potential function after request $\sigma(t)$, i.e., $\Phi(t) - \Phi(t-1)$ is the change of the potential function that occurs during the processing of $\sigma(t)$. In the analysis using a

potential function, we usually show that for any request $\sigma(t)$,

$$ALG_t + \Phi(t) - \Phi(t-1) \leq c \cdot OPT_t.$$

There are results based on potential functions, such as paging [1], k-serve [21], list update [118], and routing [8]. The difficult part in a competitive analysis using a potential function is to construct the potential function Φ, which is often mysterious and show the above inequality for all requests.

The online primal-dual approach is another widely used approach for online problems. This involves solving a discrete optimization problem online as follows (i) formulate a linear programming relaxation and obtain a primal-dual online algorithm for it; (ii) obtain an online rounding algorithm for the resulting fractional solution. While this is similar to a linear programming (LP) based approach for offline optimization problems, a key difference is that solving the LP relaxation in the online setting is highly non-trivial. (Recall that there are general polynomial time algorithms for solving LPs offline.) So there has been a lot of effort in obtaining good online algorithms for various classes of LPs: see [5, 30, 65] for pure covering LPs, [30] for pure packing LPs and [11] for certain mixed packing/covering LPs. Such online LP solvers have been useful in obtaining online algorithms for various problems, eg. set cover [6], facility location [5], machine scheduling [11], caching [16] and buy-at-bulk network design [56].

Recently, [12] initiated a systematic study of online fractional covering and packing with *convex* objectives; see also the full versions [13, 28, 36]. These papers obtained good online algorithms for a large class of fractional convex covering problems. They also demonstrated the utility of this approach via many applications that could not be solved using just online LPs. However these results were limited to convex objectives $f : \mathbb{R}^n_+ \to \mathbb{R}_+$ satisfying a monotone gradient property, i.e. $\nabla f(z) \geq \nabla f(y)$ pointwise for all $z, y \in \mathbb{R}^n$ with $z \geq y$. There are however many natural convex functions that do not satisfy such a gradient monotonicity condition. Note that this condition requires the Hessian $\nabla^2 f(x)$ to be pointwise non-negative in addition to convexity which only requires $\nabla^2 f(x)$ to be positive semidefinite.

One of the goals in this chapter is to expand the class of convex programs with good online algorithms. To this end, we focus on convex functions f that are sums of different ℓ_q-norms. This is a canonical class of convex functions with non-monotone gradients, and prior results are not applicable; see Section 3.1.1 for more details. Another goal in this chapter is to obtain better online algorithms for discrete optimization using such convex relaxations. Here, we show that sum of ℓ_q-norm objectives arise naturally as relaxations

of some network design/routing problems for which our result leads to better online algorithms.

We show that covering programs with sums of ℓ_q-norm objectives (and their dual packing programs) admit an online algorithm with a logarithmic competitive ratio. This result is nearly tight because there is a logarithmic lower bound even for online covering LPs (which corresponds to an ℓ_1 norm objective).

We also provide two applications of our fractional solver. The first is a covering application: we obtain improved competitive ratios (by two logarithmic factors) for online non-uniform buy-at-bulk problems. The second is a packing application: we obtain the first poly-logarithmic online algorithm for throughput maximization with "group" edge capacities where there is an ℓ_p-norm constraint on the flows through some subsets of edges.

Given that we achieve log-competitive online algorithms for sums of ℓ_q-norms, a natural question is whether such a result holds for all norms. Recall that any norm is a convex function. It turns out that a log-competitive algorithm is not possible for general norms. This follows from a result in [13] which shows an $\Omega(q \log d)$ lower bound for minimizing the objective $\|Bx\|_q$ under covering constraints (where B is a non-negative matrix). It is still an interesting open question to identify the correct competitive ratio for general norm functions.

3.1.1 Results and Techniques

We consider the online covering problem

$$\begin{aligned} \min \quad & \sum_{e=1}^{r} c_e \|x(S_e)\|_{q_e} \\ \text{s.t.} \quad & Ax \geq \mathbf{1}, \\ & x \geq \mathbf{0}, \end{aligned} \tag{P}$$

where each $S_e \subseteq [n] := \{1, 2, \cdots n\}$, $q_e \geq 1$, $c_e \geq 0$ and A is a non-negative $m \times n$ matrix. For any $x \in \mathbb{R}^n$ and $S \subseteq [n]$, we use $x(S) \in \mathbb{R}^{|S|}$ to denote the vector with coordinates $(x_i)_{i \in S}$; moreover, given any $q \geq 1$ we use $\|x(S)\|_q = \left(\sum_{i \in S} x_i^q\right)^{1/q}$. For any subset $S \subseteq [n]$ we use $\overline{S} := [n] \setminus S$. We also consider the dual of the above convex program, which is the following packing problem:

$$\begin{aligned} \max \quad & \sum_{k=1}^{m} y_k \\ \text{s.t.} \quad & A^T y = \mu, \end{aligned} \tag{D}$$

$$\sum_{e=1}^{r} \mu_e = \mu,$$
$$\|\mu_e(S_e)\|_{p_e} \leq c_e, \quad \forall e \in [r],$$
$$\mu_e(\overline{S}_e) = \mathbf{0}, \quad \forall e \in [r],$$
$$y \geq \mathbf{0}.$$

The values p_e above satisfy $\frac{1}{p_e} + \frac{1}{q_e} = 1$; so $\|\cdot\|_{p_e}$ is the dual norm of $\|\cdot\|_{q_e}$. This dual can be derived from (P) using Lagrangian duality; see Section 3.2.

Our framework captures the classic setting of packing/covering LPs when $r = n$ and for each $e \in [n]$ we have $S_e = \{e\}$ and $q_e = 1$. Our first main result is:

Theorem 3.1.2 (Online covering/packing result)**.** *There is an $O(\log d + \log \rho)$-competitive online algorithm for* (P) *and* (D) *where the covering constraints in* (P) *and variables y in* (D) *arrive over time. Here $\rho = \frac{\max\{a_{ij}\}}{\min\{a_{ij}\}}$, and d is the maximum of the row-sparsity of A and $\max_{e=1}^{r} |S_e|$.*

We note that this bound is also the best possible, even in the linear case [30] when we require monotone primal and dual variables. For just the covering problem, a better $O(\log d)$ bound is known for linear objectives [65] and for monotone-gradient convex functions [12]: these results involve non-monotone dual variables. Obtaining a similar $O(\log d)$ bound for our covering program (P) remains an open question.

The algorithm in Theorem 3.1.2 is the natural extension of the primal-dual approach for online LPs [30]. We use the gradient $\nabla f(x)$ at the current primal solution x as the cost function, and use this to define a multiplicative update for the primal. Simultaneously, the dual solution y is increased additively. This algorithm is in fact identical to the one in [12] for convex functions with monotone gradients. Our contribution here is in the analysis of this algorithm, which requires new ideas to deal with non-monotone gradients.

Limitations of previous approaches in handling ℓ_q-norm objectives. The general convex covering problem is

$$\min \{f(x) \,:\, Ax \geq \mathbf{1},\, x \geq \mathbf{0}\},$$

where $f : \mathbb{R}_+^n \to \mathbb{R}_+$ is a convex function and $A \in \mathbb{R}_+^{m \times n}$. Its dual is:

$$\max \left\{ \sum_{k=1}^{m} y_k - f^*(\mu) \,:\, A^T y = \mu,\, y \geq \mathbf{0} \right\},$$

where $f^*(\mu) = \max_{x \in \mathbb{R}^n_+} \{\mu^T x - f(x)\}$ is the Fenchel conjugate of f. When f is the sum of ℓ_q-norms, these primal-dual convex programs reduce to (P) and (D).

We restrict the discussion of prior techniques to functions f with $\max_{x \in \mathbb{R}^n_+} \frac{x^T \nabla f(x)}{f(x)} \leq 1$ because this condition is satisfied by sums of ℓ_q norms.[1] At a high level, the analysis in [12] uses the gradient monotonicity to prove a *pointwise upper bound* $A^T y \leq \nabla f(\bar{x})$ where $\bar{x}$ is the final primal solution. This allows them to lower bound the dual objective by $\sum_{k=1}^m y_k$ because $f^*(\nabla f(\bar{x})) \leq 0$ for any $\bar{x}$ (see Lemma 4(d) in [12]). Moreover, proving the pointwise upper bound $A^T y \leq \nabla f(\bar{x})$ is similar to the task of showing dual feasibility in the *linear* case [30, 65] where $\nabla f(\bar{x})$ corresponds to the (fixed) primal cost coefficients.

Below we give a simple example with an ℓ_q-norm objective where the pointwise upper bound $A^T y \leq \nabla f(\bar{x})$ is not satisfied by the online primal-dual algorithm unless the dual solution y is scaled down by a large (i.e. polynomial) factor. This means that one cannot obtain a sub-polynomial competitive ratio for (P) using this approach directly.

Consider an instance with objective function $f(x) = \|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$. So the gradient $\nabla f(x) = x/\|x\|_2$ which is not monotone. There are $m = \sqrt{n}$ covering constraints, where the k^{th} constraint is $\sum_{i=m(k-1)+1}^{km} x_i \geq 1$. Note that each variable appears in only one constraint. Let P be the value of the primal objective and D be the value of the dual objective at any time. Suppose that the rate of increase of the primal objective is at most α times that of the dual; α corresponds to the competitive ratio in the online primal-dual algorithm. Upon arrival of any constraint k, it follows from the primal updates that all the variables $\{x_i\}_{i=m(k-1)+1}^{km}$ increase from 0 to $\frac{1}{m}$. So the increase in P due to constraint k is $(\sqrt{k} - \sqrt{k-1})\frac{1}{\sqrt{m}}$ for iteration k. This means that the increase in D is at least $\frac{1}{\alpha}(\sqrt{k} - \sqrt{k-1})\frac{1}{\sqrt{m}}$, and so $y_k \geq \frac{1}{\alpha}(\sqrt{k} - \sqrt{k-1})\frac{1}{\sqrt{m}}$. Finally, since $\bar{x} = \frac{1}{m}\mathbf{1}$, we know that $\nabla f(\bar{x}) = \frac{1}{m}\mathbf{1}$ (recall $n = m^2$). On the other hand, $(A^T y)_1 = y_1 \geq \frac{1}{\alpha\sqrt{m}}$. Therefore, in order to guarantee $A^T y \leq \nabla f(\bar{x})$ we must have $\alpha \geq \sqrt{m} = n^{1/4}$.

Our approach to handle ℓ_q-norm objectives. First, we show that by duplicating variables and using an online separation oracle approach one can ensure that the sets $\{S_e\}_{e=1}^r$ are disjoint. The use of a separation oracle in the online context is similar to [5]. The disjoint structure of S_es allows for a simple expression for ∇f which is useful in the later analysis. Then we utilize the specific form of the primal-dual convex programs (P) and (D) and an explicit expression for ∇f to show that the dual y is approximately feasible. In particular we show that $\|y^T A(S_e)\|_{p_e} \leq O(\log d\rho) \cdot c_e$ for each $e \in [r]$; here $A(S_e)$ denotes the submatrix of A with columns from S_e. Note that this is a weaker requirement than upper

[1] The result in [12] also applies to other convex functions with monotone gradients, but the competitive ratio depends exponentially on $\max_{x \in \mathbb{R}^n_+} \frac{x^T \nabla f(x)}{f(x)}$.

bounding $A^T y$ pointwise by $\nabla f(\bar{x})$.

In order to bound $\|y^T A(S_e)\|_{p_e}$, we analyze each $e \in [r]$ separately. We partition the steps of the algorithm into *phases* where phase j corresponds to steps where $\Phi_e = \sum_{i \in S_e} x_i^{q_e} \approx \theta^j$; here $\theta > 1$ is a parameter that depends on q_e. The number of phases can be bounded using the fact that Φ_e is monotonically increasing. By triangle inequality we upper bound $\|y^T A(S_e)\|_{p_e}$ by $\sum_j \|y_{(j)}^T A(S_e)\|_{p_e}$ where $y_{(j)}$ denotes the dual variables that arrive in phase j. And in each phase j, we can upper bound $\|y_{(j)}^T A(S_e)\|_{p_e}$ using the differential equations for the primal and dual updates.

We also provide two applications of Theorem 3.1.2, one using the result for the covering problem (P) and another using the packing problem (D).

Non-uniform multicommodity buy-at-bulk. This is a well-studied network design problem in the offline setting [39, 40]. The setting is as follows. We are given an undirected (or directed) graph $G = (V, E)$ with a monotone sub-additive cost function $g_e : \mathbb{R}_+ \to \mathbb{R}_+$ on each edge $e \in E$ and a collection $\{(s_i, t_i)\}_{i=1}^m$ of m source/destination pairs. The goal is to find an $s_i - t_i$ path P_i for each $i \in [m]$ such that the objective $\sum_{e \in E} g_e(load_e)$ is minimized; here $load_e$ is the number of paths using e.

In its online version, the source-destination pairs arrive incrementally over time and we need to select the path for each pair immediately upon arrival. The first poly-logarithmic competitive ratio for the online problem was obtained recently in [56]. A key step in this result was a fractional online algorithm for a specific mixed packing-covering LP. By utilizing Theorem 3.1.2 we improve the competitive ratio of this step from $O(\log^3 n)$ to $O(\log n)$ which is also the best possible. Combined with the other steps in [56], we obtain:

Theorem 3.1.3 (Application for buy-at-bulk network design)**.** *There is an $O(\alpha\beta\gamma \cdot \log^3 n)$-competitive ratio for non-uniform multicommodity buy-at-bulk, where α is the "junction tree" approximation ratio, β is the integrality gap of the natural LP relaxation for single-source instances, and γ is the competitive ratio for single-source instances.*

See Section 3.1.1 for more details on the parameters α, β and γ. The corresponding competitive ratio in [56] was $O(\alpha\beta\gamma \cdot \log^5 n)$. In particular, for undirected multicommodity buy-at-bulk we obtain an $O(\log^9 n)$ competitive ratio, improving over the $O(\log^{11} n)$ ratio in [56].

The main idea in Theorem 3.1.3 is a reformulation of the LP from [56] as a pure covering program where the objective is a sum of ℓ_1 and ℓ_∞ norms. This reformulation uses the equivalence of maximum-flow and minimum-cut. The resulting covering program has an

exponential number of constraints: but we still obtain a polynomial-time online algorithm using a suitable separation oracle.

Throughput maximization with ℓ_p-norm capacities. The online problem of maximizing throughput subject to edge capacities is a classic online optimization problem [10, 30]. Here we are given a directed graph with m edges and edge capacities $u(e)$. Source/destination requests (s_i, t_i) arrive in an online fashion. An algorithm needs to select a subset of requests to accept and assign a path to each accepted request so that the load on each edge e is at most u_e. The goal is to maximize the number of accepted requests. We consider a natural generalization where there are capacity constraints on *groups* of edges: each such constraint requires the ℓ_{p_j}-norm of the loads on some edge-subset S_j to be at most a given capacity c_j.

Theorem 3.1.4 (Application for throughput maximization)**.** *The throughput maximization problem with ℓ_p-norm capacities admits a randomized $O(\log m)$-competitive algorithm when:*

1. *the capacity $c_j = \Omega(\log m)\,|S_j|^{1/p_j}$ for each group j, or*
2. *each capacity may be violated by an $O(\log^{1+1/p} m)$ factor where $p = \min_j p_j$.*

The first algorithm above also works for arbitrary capacities when each capacity may be violated by a factor of $O(\log m \cdot \max_j |S_j|^{1/p_j})$. So the second algorithm achieves a better capacity violation (note that $|S_j|$ can be polynomial in m). On the other hand, the first algorithm runs in polynomial time whereas the second algorithm takes exponential time. The two algorithms rely on different convex relaxations that have the form of our dual program (D) and so Theorem 3.1.2 can be used directly. While both relaxations have an exponential number of variables, the first relaxation can be solved in polynomial time using an efficient separation oracle (shortest path). Both algorithms use the natural randomized rounding to obtain integral solutions from the fractional relaxations.

We note that some "high capacity" assumption is required (regardless of running time) to obtain any sub-polynomial competitive ratio even in the usual throughput maximization problem where each $|S_j| = 1$ [10, 19].

3.1.2 Related Work

The online primal-dual framework for linear programs [31] is fairly well understood. Tight results are known for the class of packing and covering LPs [30, 65], with competitive ratio $O(\log d)$ for covering LPs and $O(\log d\rho)$ for packing LPs; here d is the row-sparsity and ρ

is the ratio of the maximum to minimum entries in the constraint matrix. Such LPs are very useful because they correspond to the LP relaxations of many combinatorial optimization problems. Combining the online LP solver with suitable online rounding schemes, good online algorithms have been obtained for many problems, eg. set cover [6], group Steiner tree [5], caching [16] and ad-auctions [29]. Online algorithms for LPs with mixed packing and covering constraints were obtained in [11]; the competitive ratio was improved in [12]. Such mixed packing/covering LPs were also used to obtain an online algorithm for capacitated facility location [11]. A more complex mixed packing/covering LP was used recently in [56] to obtain online algorithms for non-uniform buy-at-bulk network design: as an application of our result, we obtain a simpler and better (by two log-factors) online algorithm for this problem.

There have also been a number of results utilizing the online primal-dual framework with *convex* objectives for specific problems, eg. matching [50], caching [104], energy-efficient scheduling [49, 63] and welfare maximization [24, 76]. All of these results involve separable convex/concave functions. Recently, [12] considered packing/covering problems with general (non-separable) convex objectives, but (as discussed previously) this result requires a monotone gradient assumption on the convex function. The sum of ℓ_q-norm objectives considered in this chapter does not satisfy this condition. While our primal-dual algorithm is identical to [12], we need new techniques in the analysis.

All the results above (as well as ours) involve convex objectives and linear constraints. We note that [54] obtained online primal-dual algorithms for certain semidefinite programs (i.e. involving non-linear constraints). While both our result and [54] generalize packing/covering LPs, they are not directly comparable.

We also note that online algorithms with ℓ_q-norm objectives have been studied previously for many scheduling problems, eg. [9, 17]. More recently [13] used ideas from the online primal-dual approach in an online algorithm for unrelated machine scheduling where the objective is the sum of ℓ_p-norm of loads and startup costs. These results use different techniques and are not directly comparable to ours.

3.2 Preliminaries

Recall the primal covering problem (P) and its dual packing problem (D). In the online setting, the constraints in the primal and variables in the dual arrive over time. We need to maintain monotonically increasing primal (x) and dual (y) solutions.

3.2.1 Dual Packing Problem

We first describe how (D) can be derived as the Lagrangian dual of (P). Let $f_e(x) = c_e\|x(S_e)\|_{q_e}$ for each $e \in [r]$ and $f(x) = \sum_{e=1}^{r} f_e(x)$.

The Lagrangian dual of problem (P) is given by

$$\begin{aligned}
&\sup_{y\geq 0} \quad \inf_{x\geq 0} \left(\sum_{e=1}^{r} c_e\|x(S_e)\|_{q_e} + y^T(\mathbf{1} - Ax) \right) \\
=&\sup_{y\geq 0} \left(\sum_{k=1}^{m} y_k \; - \; \sup_{x\geq 0} \left((A^T y)^T x - \sum_{e=1}^{r} c_e\|x(S_e)\|_{q_e} \right) \right) \\
=&\sup_{y\geq 0} \left(\sum_{k=1}^{m} y_k - f^*(A^T y) \right) \qquad (3.1)
\end{aligned}$$

where $f^*(\cdot)$ is the conjugate function of $f(\cdot)$. Let $\mu = A^T y$. $\mu \geq \mathbf{0}$ since $y \geq \mathbf{0}$ and A is a nonnegative matrix. By the Moreau-Rockafellar formula [108, §6.8][119, Thm 3.2], since $f(x)$ is closed, proper, continuous and convex, we have

$$f^*(\mu) = f_1^*(\mu) \oplus \cdots \oplus f_r^*(\mu) = \inf_{\mu_1+\cdots+\mu_r=\mu} \left\{ \sum_{e=1}^{r} f_e^*(\mu_e) \right\}, \qquad \forall \mu \in \mathbb{R}^n,$$

where $\oplus$ is the infimal convolution.

The Fenchel conjugate of f is $f^*(\mu) = \max_{x\in\mathbb{R}^n_+}\{\mu^T x - f(x)\}$ [27, §3.3.1]. Since $f_e(x) = c_e\|x(S_e)\|_{q_e}$,

$$f_e^*(\mu_e) = \begin{cases} 0, & \text{if } \|\mu_e(S_e)\|_{p_e} \leq c_e \text{ and } \mu_e(\overline{S}_e) = \mathbf{0}, \\ \infty, & \text{otherwise.} \end{cases}$$

Problem (3.1) can then be reformulated as

$$\begin{aligned}
\max \quad & \sum_{k=1}^{m} y_k \\
\text{s.t.} \quad & A^T y = \mu, \\
& \sum_{e=1}^{r} \mu_e = \mu, \\
& \|\mu_e(S_e)\|_{p_e} \leq c_e, \quad \forall e \in [r], \\
& \mu_e(\overline{S}_e) = \mathbf{0}, \quad \forall e \in [r], \\
& y \geq \mathbf{0},
\end{aligned}$$

which is exactly the packing problem (D).

Note that strong duality holds since the Slater's condition holds [27, §5.2.3], that is, there is $x \in \mathbb{R}^n$ such that $Ax > 1$ and $x > 0$.

3.2.2 Disjointedness Assumption on S_es

We next show that one can assume that the sets $\{S_e\}_{e=1}^r$ are *disjoint* without loss of generality. This leads to a much simpler expression for ∇f that will be used in Section 3.3. An online algorithm for the covering problem (P) is said to be *primal-dual* if it also maintains dual variables in (D) and the primal objective is bounded in terms of the dual objective.

Lemma 3.2.1. *Suppose there is a polynomial time α-competitive algorithm $\mathcal{A}$ for the covering problem* (P) *with disjoint S_e. Then, there is a polynomial time $O(\alpha)$-competitive algorithm for* (P) *on general instances. Moreover, if algorithm $\mathcal{A}$ is primal-dual and maintains monotonically non-decreasing dual variables, then there is a polynomial time $O(\alpha)$-competitive algorithm for the packing problem* (D) *on general instances.*

Proof. Let $\mathcal{A}$ denote an α-competitive algorithm for the covering problem (P) with disjoint S_e. We assume that it is a minimal algorithm, that is when constraint k arrives it stops increasing x when $\sum_{i=1}^n a_{ki} x_i = 1$. (Any online algorithm can be ensured to be of this form.)

Given an instance $\mathcal{P}_I$ of the covering problem (P) with general $\{S_e\}_{e=1}^r$, we define an instance $\mathcal{P}_J$ with disjoint S'_e as follows. For each variable x_i, we introduce r copies $x_i^{(1)}, \ldots, x_i^{(r)}$ where $x_i^{(e)}$ corresponds to the possible occurrence of x_i in S_e. So there are nr variables in $\mathcal{P}_J$. For each $e \in [r]$ we set S'_e to consist of the variables $x_i^{(e)}$ for $i \in S_e$. So $\{S'_e\}_{e=1}^r$ are disjoint. For each constraint $a_k^T x \geq 1$ in instance $\mathcal{P}_I$, we introduce a family of r^n constraints in instance $\mathcal{P}_J$ which corresponds to all combinations of the $x_i^{(e)}$ variables, namely

$$\sum_{i=1}^n a_{ki} \cdot x_i^{(e_i)} \geq 1, \quad \forall e_1 \in [r],\, e_1 \in [r], \cdots e_n \in [r]. \tag{3.2}$$

If $\bar{x}$ is a feasible solution of $\mathcal{P}_I$, then $x_i^{(e)} = \bar{x}_i$, for all $e \in [r]$ and $i \in [n]$ is a feasible solution to $\mathcal{P}_J$ with the same objective value. Conversely, if x is a feasible solution for $\mathcal{P}_J$ then $\bar{x}_i = \min_{e=1}^r x_i^{(e)}$ for all $i \in [n]$ is a feasible solution for $\mathcal{P}_I$ with at most the same objective value. Hence, instances $\mathcal{P}_I$ and $\mathcal{P}_J$ share the same optimal value. So an α-competitive algorithm for $\mathcal{P}_J$ also leads to one for $\mathcal{P}_I$. However, this is not a polynomial time reduction as there are exponentially many constraints in J. In order to deal with this, we use a separation oracle based algorithm (see Algorithm 3), as in [5].

Input : The kth covering constraint.
Output: Current solution $\bar{x}$.

When the k^{th} covering constraint $\sum_{i=1}^{n} a_{ki}x_i \geq 1$ arrives in $\mathcal{P}_I$
while $\sum_{i=1}^{n} a_{ki} \cdot \min_{e=1}^{r} x_i^{(e)} < \frac{1}{2}$ **do**
    let $e_i = \arg\min_{e=1}^{e} x_i^{(e)}$ for all $i \in [n]$;
    add constraint $\sum_{i=1}^{n} a_{ki} \cdot x_i^{(e_i)} \geq 1$ to instance $\mathcal{P}_J$ and run algorithm $\mathcal{A}$;
end
Output current solution $\bar{x}_i = 2 \cdot \min_{e=1}^{r} x_i^{(e)}$ for all $i \in [n]$;

Algorithm 3: Separation Oracle Based Algorithm for General S_e

It is obvious that the output solution is feasible for instance $\mathcal{P}_I$. As x is an α-competitive solution to $\mathcal{P}_J$, the output solution is 2α-competitive for $\mathcal{P}_I$. It remains to show that Algorithm 3 runs in polynomial time upon arrival of any constraint k. For this, define potential function $\psi = \sum_{i=1}^{n}\sum_{e=1}^{r} a_{ki} \cdot x_i^{(e)}$ which is monotone non-decreasing. We know that $\max_{i,e} a_{ki}x_i^{(e)} \leq 1$ since algorithm $\mathcal{A}$ is minimal. So $\psi \leq rn$. In each iteration of Algorithm 3, $\sum_{i=1}^{n} a_{ki} \cdot x_i^{(e_i)}$ increases by at least $\frac{1}{2}$, i.e. ψ also increases by at least $\frac{1}{2}$. So the number of iterations is bounded by $2rn$ which is polynomial. This completes the first part of the proof.

Let $\mathcal{D}_I$ and $\mathcal{D}_J$ be the dual programs for $\mathcal{P}_I$ and $\mathcal{P}_J$ respectively. By strong duality and the fact that $\mathcal{P}_I$ and $\mathcal{P}_J$ share the same optimal value, $\mathcal{D}_I$ and $\mathcal{D}_J$ also have the same optimal value. Let $\mu' \in \mathbb{R}^{nr}$ denote the μ-variables in $\mathcal{D}_J$. Recall from (3.2) that each constraint k in $\mathcal{P}_I$ corresponds to r^n constraints in $\mathcal{P}_J$: let ${y'}_{k,\ell}$ for $\ell \in [r^n]$ denote the dual variables in $\mathcal{D}_J$ for these constraints. Given a feasible dual solution μ', y' for $\mathcal{D}_J$, we can obtain a feasible solution for $\mathcal{D}_I$ by setting $y_k = \sum_\ell {y'}_{k,\ell}$, for each $e \in [r]$,

$$\mu_e(i) = \begin{cases} \mu'(e,i) & \text{if } i \in S_e \\ 0 & \text{if } i \notin S_e \end{cases},$$

and $\mu = \sum_{e=1}^{r} \mu_e$. Note also that the objective value of (y, μ) in $\mathcal{D}_I$ equals that of (y', μ') in $\mathcal{D}_J$, Furthermore, since the online algorithm maintains monotone variables y', the corresponding y-variables are also monotone. The running time is polynomial (same as for the primal instance). Finally, as the algorithm is primal-dual, we obtain an $O(\beta)$-competitive ratio for the dual problem $\mathcal{D}_I$ as well. This completes the second part of the proof. □

Henceforth we will assume that the sets $\{S_e\}_{e=1}^{r}$ are disjoint. Our algorithm in this case (Section 3.3) is primal-dual and maintains monotone duals. Using Lemma 3.2.1 we would then obtain online algorithms for both covering and packing on general instances.

With disjoint $\{S_e\}_{e=1}^{r}$, the constraints $\sum_{e=1}^{r} \mu_e = \mu$, $\|\mu_e(S_e)\|_{p_e} \leq c_e$, and $\mu_e(\overline{S}_e) = \mathbf{0}$

for $e \in [r]$ are equivalent to $\|\mu(S_e)\|_{p_e} \leq c_e$ for $e \in [r]$ and $\mu(\cap_e \overline{S}_e) = \mathbf{0}$. Then the dual packing problem (D) simplifies to:

$$\max \left\{ \sum_{k=1}^{m} y_k \,:\, A^T y = \mu,\, \|\mu(S_e)\|_{p_e} \leq c_e \,\forall e \in [r],\, \mu(\cap_e \overline{S}_e) = \mathbf{0}, y \geq \mathbf{0} \right\}. \qquad \text{(DD)}$$

This is the dual program that will be used in Section 3.3. We show below (for completeness) that weak duality holds for the primal program (P) and its dual (DD). We note that strong duality also holds because (P) satisfies Slater's condition; however we do not use this fact in Section 3.3.

Lemma 3.2.2. *For any pair of feasible solutions x to* (P) *and (y, μ) to* (DD)*, we have*

$$\sum_{e=1}^{r} c_e \|x(S_e)\|_{q_e} \geq \sum_{k=1}^{m} y_k.$$

Proof. This follows from the following inequalities:

$$\begin{aligned} \sum_{k=1}^{m} y_k = y^T \mathbf{1} \leq y^T A x = \mu^T x \leq \sum_{e=1}^{r} \sum_{i \in S_e} \mu_i \cdot x_i \leq \sum_{e=1}^{r} \|\mu(S_e)\|_{p_e} \cdot \|x(S_e)\|_{q_e} \\ \leq \sum_{e=1}^{r} c_e \cdot \|x(S_e)\|_{q_e}. \end{aligned}$$

The first inequality is by primal feasibility; the second inequality is by $x \geq 0$, $\mu_i \geq 0$ and $\mu_i = 0$ if $i \in \cap_e \overline{S}_e$. The third inequality is by Hölder's inequality. The last inequality is by dual feasibility. □

3.3 Algorithm and analysis

Let $f(x) = \sum_{e=1}^{r} c_e \|x(S_e)\|_{q_e}$ denote the primal objective in (P). The algorithm is shown in Algorithm 4.

In order to ensure that the gradient ∇f is defined, the primal solution x starts off as $\delta \cdot \mathbf{1}$ where $\delta > 0$ is arbitrarily small. So we assume that the initial primal value is zero.

It is clear that the algorithm maintains a feasible and monotonically non-decreasing primal solution x. The dual solution (y, μ) is also monotonically non-decreasing, but not necessarily feasible. We will show that (y, μ) is $O(\log \rho d)$-approximately feasible, i.e. the packing constraints in (DD) are violated by at most an $O(\log \rho d)$ factor.

Lemma 3.3.1. *The primal objective $f(x)$ is at most twice the dual objective $\sum_{k=1}^{m} y_k$.*

Input : The kth covering constraint.
Output: Current solution x, y, μ.

When the k^{th} request $\sum_{i=1}^{n} a_{ki}x_i \geq 1$ arrives
Let τ be a continuous variable denoting the current time;
while *the constraint is unsatisfied, i.e.,* $\sum_{i=1}^{n} a_{ki}x_i < 1$ **do**
    For each i with $a_{ki} > 0$, increase x_i at rate
    $\frac{\partial x_i}{\partial \tau} = \frac{a_{ki}x_i + \frac{1}{d}}{\nabla_i f(x)} = \frac{a_{ki}x_i + \frac{1}{d}}{c_e x_i^{q_e - 1}} \|x(S_e)\|_{q_e}^{q_e - 1}$;
    ; // If $\nabla_i f(x) = 0$, increase x_i at rate $\frac{\partial x_i}{\partial \tau} = \infty$;
    Increase y_k at rate $\frac{\partial y_k}{\partial \tau} = 1$;
    Set $\mu = A^T y$;
end

Algorithm 4: Algorithm for ℓ_q-norm packing/covering

Proof. We will show that the rate of increase of the primal is at most twice that of the dual. Consider the algorithm upon the arrival of some constraint k. Then

$$\frac{df(x)}{d\tau} = \sum_{i:a_{ki}>0} \nabla_i f(x) \cdot \frac{\partial x_i}{\partial \tau} = \sum_{i:a_{ki}>0} (a_{ki}x_i + \frac{1}{d}) \leq 2.$$

The inequality comes from the fact that (i) the process for the kth constraint is terminated when $\sum_i a_{ki}x_i = 1$ and (ii) the number of non-zeroes in constraint k is at most d. Also it is clear that the dual objective increases at rate one, which finishes the proof. □

Lemma 3.3.2. *The dual solution (y, μ) is $O(\log \rho d)$-approximately feasible, i.e.*

$$\mu(\cap_e \overline{S}_e) = 0, \tag{3.3}$$

and

$$\|\mu(S_e)\|_{p_e} \;\leq\; O(\log \rho d) \cdot c_e, \qquad \forall e \in [r]. \tag{3.4}$$

Proof. First we prove (3.3). For any $i \in \cap_e \overline{S}_e$ we always have $\nabla_i f(x) = 0$: we will show that $\mu_i = 0$ always. Consider the arrival of any constraint $\sum_{i=1}^{n} a_{ki}x_i \geq 1$. If $a_{ki} = 0$ then $\frac{\partial \mu_i}{\partial \tau} = 0$. If $a_{ki} > 0$ then x_i increases at ∞ rate: so the constraint will be satisfied without increasing y_k, so μ_i also stays 0.

In order to prove (3.4), fix any $e \in [r]$. When $q_e = 1$, the corresponding part of the objective function is reduced to the linear case $c_e \sum_{i \in S_e} x_i$ and we want to prove $\|\mu(S_e)\|_\infty \leq O(\log \rho d) \cdot c_e$ for all $e \in [r]$. It is equivalent to prove that $\mu_i \leq O(\log \rho d) \cdot c_e$

for all $i \in S_e$. In this case, we have

$$\frac{\partial x_i}{\partial \tau} = \frac{a_{ki}\, x_i + \frac{1}{d}}{c_e}, \quad \frac{\partial y_k}{\partial \tau} = 1, \quad \frac{\partial \mu_i}{\partial \tau} = a_{ki} \Rightarrow d\mu_i = \frac{c_e\, a_{ki}}{a_{ki}\, x_i + \frac{1}{d}} dx_i.$$

This means that the increase in μ_i over the entire algorithm is:

$$\Delta\mu_i \leq \int_0^{\frac{1}{\min\{a_{ij}\}}} \frac{c_e\, a_{ki}}{a_{ki}\, x_i + \frac{1}{d}} dx_i = c_e \cdot \ln\left(\frac{a_{ki} \cdot d}{\min\{a_{ij}\}} + 1\right) = O(\log \rho d) \cdot c_e.$$

The case $q_e > 1$ is the main part of the analysis. In order to prove the desired upper bound on $\|\mu(S_e)\|_{p_e}$ we use a potential function $\Phi = \sum_{i \in S_e} (x_i^{q_e})$. Let phase zero denote the period where $\Phi \leq \zeta := (\frac{1}{\max\{a_{ij}\} \cdot d^2})^{q_e}$, and for each $j \geq 1$, phase j is the period where $\theta^{j-1} \cdot \zeta \leq \Phi < \theta^j \cdot \zeta$. Here $\theta > 1$ is a parameter depending on q_e that will be determined later. Note that $\Phi \leq d(\frac{1}{\min\{a_{ij}\}})^{q_e}$ as variable x_i will never be increased beyond $1/\min_{j=1}^m a_{ij}$. So the number of phases is at most $3q_e \cdot \log(d\rho)/\log\theta$. Next, we bound the increase in $\|\mu(S_e)\|_{p_e}$ for each phase separately.

For any phase, we have the following equalities

$$\frac{\partial x_i}{\partial \tau} = \frac{a_{ki} x_i + \frac{1}{d}}{c_e x_i^{q_e - 1}} \|x(S_e)\|_{q_e}^{q_e - 1}, \qquad \frac{\partial y_k}{\partial \tau} = 1, \qquad \frac{\partial \mu_i}{\partial \tau} = a_{ki},$$

$$\Rightarrow d\mu_i = \frac{c_e\, a_{ki}\, x_i^{q_e - 1}}{(\sum_{j \in S_e} x_j^{q_e})^{1 - \frac{1}{q_e}} (a_{ki} x_i + \frac{1}{d})} dx_i. \tag{3.5}$$

Phase zero. Suppose that each x_i increases to α_i in phase zero. From (3.5) we have

$$d\mu_i \leq \frac{d\, c_e\, a_{ki}\, x_i^{q_e - 1}}{(\sum_{j \in S_e} x_j^{q_e})^{1 - \frac{1}{q_e}}} dx_i \qquad \Rightarrow \qquad \frac{1}{d\, c_e\, a_{ki}} d\mu_i \leq \frac{x_i^{q_e - 1}}{(\sum_{j \in S_e} x_j^{q_e})^{1 - \frac{1}{q_e}}} dx_i.$$

This means that the increase $\Delta\mu_i$ in μ_i (during phase zero) can be bounded as

$$\frac{1}{d\, c_e\, a_{ki}} \Delta\mu_i \leq \int_\delta^{\alpha_i} \frac{x_i^{q_e - 1}}{(\sum_{j \in S_e} x_j^{q_e})^{1 - \frac{1}{q_e}}} dx_i \leq \int_0^{\alpha_i} 1 dx_i \leq \alpha_i.$$

Since in phase zero, $\Phi \leq (\frac{1}{\max\{a_{ij}\} \cdot d^2})^{q_e}$, we know that each $\alpha_i \leq \frac{1}{\max\{a_{ij}\} \cdot d^2}$. So $\Delta\mu_i \leq \frac{c_e}{d}$ and at the end of phase zero, we have $\|\mu(S_e)\|_{p_e} \leq \|\mu(S_e)\|_1 \leq c_e$. The last inequality is because $d \geq \max_e |S_e|$.

Phase $j \geq 1$. Let Φ_0 and Φ_1 be the value of Φ at the beginning and end of this phase

respectively. In phase j, suppose that each x_i increases from s_i to t_i. Then,

$$d\mu_i = \frac{c_e\, a_{ki}\, x_i^{q_e-1}}{(\sum_{j\in S_e} x_j^{q_e})^{1-\frac{1}{q_e}}(a_{ki}x_i + \frac{1}{d})}dx_i \le \frac{c_e x_i^{q_e-2}}{(\sum_{j\in S_e} x_j^{q_e})^{1-\frac{1}{q_e}}}dx_i.$$

So the increase $\Delta\mu_i$ in μ_i during this phase is:

$$\Delta\mu_i \le \int_{s_i}^{t_i} \frac{c_e x_i^{q_e-2}}{(\sum_{j\in S_e} x_j^{q_e})^{1-\frac{1}{q_e}}}dx_i.$$

Note that variables $x_{i'}$ for $i' \neq i$ can also increase in this phase: so we cannot directly bound the above integral. This is precisely where the potential Φ is useful. We know that throughout this phase, $\sum_{i\in S_e} x_i^{q_e} \ge \Phi_0$. So the increase in μ_i during this phase is:

$$\Delta\mu_i \le c_e \int_{s_i}^{t_i} \frac{x_i^{q_e-2}}{\Phi_0^{1-\frac{1}{q_e}}}dx_i = c_e \frac{t_i^{q_e-1} - s_i^{q_e-1}}{(q_e-1)\Phi_0^{1-\frac{1}{q_e}}} = c_e \frac{t_i^{q_e-1} - s_i^{q_e-1}}{(q_e-1)\Phi_0^{\frac{1}{p_e}}}.$$

Above we used the assumption that $q_e > 1$ in evaluating the integral. Now,

$$\begin{aligned}(\Delta\mu_i)^{p_e} &\le \frac{c_e^{p_e}}{(q_e-1)^{p_e}\,\Phi_0} \cdot \left(t_i^{q_e-1} - s_i^{q_e-1}\right)^{p_e} \\ &\le \frac{c_e^{p_e}}{(q_e-1)^{p_e}\,\Phi_0} \cdot \left(t_i^{(q_e-1)p_e} - s_i^{(q_e-1)p_e}\right) \\ &= \frac{c_e^{p_e}}{(q_e-1)^{p_e}\,\Phi_0} \cdot \left(t_i^{q_e} - s_i^{q_e}\right).\end{aligned}$$

The first inequality above uses the fact that $(z_1+z_2)^{p_e} \ge z_1^{p_e} + z_2^{p_e}$ for any $p_e \ge 1$ and $z_1, z_2 \ge 0$, with $z_1 = s_i^{q_e-1}$ and $z_2 = t_i^{q_e-1} - s_i^{q_e-1}$. The last equality uses $\frac{1}{p_e} + \frac{1}{q_e} = 1$.

We can now bound

$$\sum_{i\in S_e}(\Delta\mu_i)^{p_e} \le \frac{c_e^{p_e}}{(q_e-1)^{p_e}\,\Phi_0} \cdot \sum_{i\in S_e}(t_i^{q_e} - s_i^{q_e}) = \frac{c_e^{p_e}\,(\Phi_1 - \Phi_0)}{(q_e-1)^{p_e}\,\Phi_0} \le \frac{c_e^{p_e}(\theta-1)}{(q_e-1)^{p_e}}.$$

Let vector $\mu^{(\mathrm{j})} \in \mathbb{R}^{S_e}$ denote the increase in variables $\{\mu_i : i \in S_e\}$ during phase j. It follows from the above that $\|\mu^{(\mathrm{j})}\|_{p_e} \le \frac{c_e}{q_e-1}(\theta-1)^{1/p_e}$.

Combining across phases. Note that the final vector $\mu = \sum_{j\ge 0}\mu^{(\mathrm{j})}$. By triangle inequality, we have

$$\|\mu\|_{p_e} \le \sum_{j\ge 0}\|\mu^{(\mathrm{j})}\|_{p_e} \le c_e + \sum_{j\ge 1}\|\mu^{(\mathrm{j})}\|_{p_e} \le c_e\left(1 + \frac{3q_e(\theta-1)^{1/p_e}}{(q_e-1)\log\theta}\cdot\log(d\rho)\right). \quad (3.6)$$

To complete the proof we show next that for any $q_e > 1$, there is some choice of $\theta > 1$ such that the right-hand-side above is $O(\log(d\rho)) \cdot c_e$.

Case 1: $q_e \geq 2$**.** In this case, setting $\theta = 2$, we have $\frac{3q_e}{(q_e-1)}(\theta-1)^{1/p_e}/\log\theta \leq 6$.

Case 2: $1 < q_e < 2$**.** We set $\theta = 1 + (q_e-1)^{-\epsilon p_e}$, where $\epsilon = \frac{1}{-\log(q_e-1)} > 0$. We have

$$\frac{(\theta-1)^{\frac{1}{p_e}}}{\log\theta} \leq \frac{(\theta-1)^{\frac{1}{p_e}}}{\log(q_e-1)^{-\epsilon p_e}} = \frac{(q_e-1)^{-\epsilon}}{\log(q_e-1)^{-\epsilon p_e}} = \frac{(q_e-1)^{-\epsilon}}{-\epsilon p_e \log(q_e-1)} = \frac{(q_e-1)^{-\epsilon}}{p_e} = \frac{2}{p_e}.$$

The first inequality above uses that $\theta - 1 = (q_e-1)^{-\epsilon p_e} > 1$. Thus we have

$$\frac{3q_e(\theta-1)^{1/p_e}}{(q_e-1)\log\theta} \leq \frac{6q_e}{(q_e-1)p_e} = 6,$$

where the last equality uses $\frac{1}{p_e} + \frac{1}{q_e} = 1$.

So in either case we have that the right-hand-side of (3.6) is at most $(1 + 6\log(d\rho))c_e$. □

Combining Lemmas 3.2.2, 3.3.1 and 3.3.2, we obtain Theorem 3.1.2.

3.4 Applications

3.4.1 Online Buy-at-Bulk Network Design

In the non-uniform multicommodity buy-at-bulk problem, we are given a directed or undirected graph $G = (V, E)$ with a monotone sub-additive cost function $g_e : \mathbb{R}_+ \to \mathbb{R}_+$ on each edge $e \in E$ and a collection $\{(s_i, t_i)\}_{i=1}^m$ of m source/destination pairs. The goal is to find an $s_i - t_i$ path P_i for each $i \in [m]$ such that the objective $\sum_{e\in E} g_e(load_e)$ is minimized; here $load_e$ is the number of paths using e. An equivalent view of this problem involves two costs c_e and ℓ_e for each edge $e \in E$ and the objective $\sum_{e\in\cup P_i} c_e + \sum_{e\in E} \ell_e \cdot load_e$. In the online setting, the pairs (s_i, t_i) arrive over time and we need to decide on the path P_i immediately after the i^{th} pair arrives.

Example In Figure 3.1, there are three requests. (s_1, t_1) is satisfied by the yellow path, (s_2, t_2) is satisfied by the blue path and (s_3, t_3) is satisfied by the red path. The number on each edge is the load on that edge.

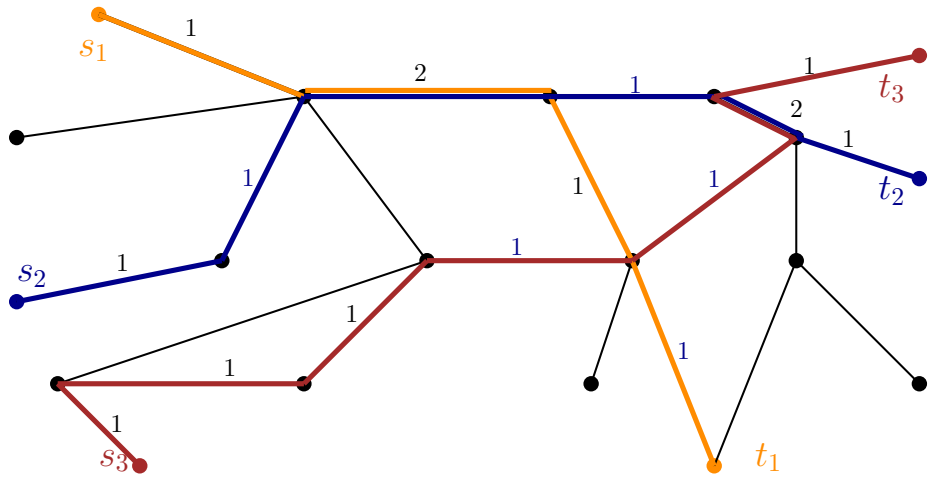

Figure 3.1: Buy-at-bulk network design example

Recently, [56] gave a modular online algorithm for non-uniform buy-at-bulk problems with competitive ratio $O(\alpha\beta\gamma \cdot \log^5 n)$ where:

- β is the integrality gap of the natural LP relaxation for single-source instances, where all s_is correspond to the same node.
- γ is the competitive ratio of an online algorithm for single-source instances.
- α is the "junction tree" approximation ratio. A junction-tree is a specific solution structure (introduced in [40]) that enables a reduction from multicommodity to single-source instances. In such a solution, the m pairs are partitioned into groups and each group $S \subseteq [m]$ corresponds to a root vertex $r \in V$ such that the path for each pair in S goes through r. There is no sharing of costs across groups: in particular, we view the solution for each group as using a distinct copy of the graph. The value α is the worst-case ratio of the cost of a junction-tree solution to the optimum.

One of the main components in the result in [56] was an $O(\log^3 n)$-competitive fractional online algorithm for a certain mixed packing/covering LP. Here we show that Theorem 3.1.2 can be used to provide a better (and tight) $O(\log n)$-competitive ratio for the same LP. This leads to the improved $O(\alpha\beta\gamma \cdot \log^3 n)$-competitive ratio stated in Theorem 3.1.3.

The LP relaxation. We now describe the LP relaxation used in [56]. Let $\mathcal{T} = \{(s_i, t_i) : i \in [m]\}$ denote the set of all sources/destinations. For each $i \in [m]$ and root $r \in V$ variable z_{ir} denotes the extent to which both s_i and t_i route to/from r: this corresponds to assigning pair i to the group (in the junction-tree solution) with root r. For each $r \in V$ and $e \in E$, variable x_{er} denotes the extent to which edge e is used in the routing to root r: this

corresponds to whether/not edge e is used in the junction-tree solution for root r. For each $r \in V$ and $u \in \mathcal{T}$, variables $\{f_{r,u,e} : e \in E\}$ represent a flow between r and u.

$$\begin{aligned}
\min \quad & \sum_{r\in V}\sum_{e\in E} c_e \cdot x_{e,r} \;+\; \sum_{r\in V}\sum_{e\in E} \ell_e \cdot \sum_{u\in\mathcal{T}} f_{r,u,e} \\
\text{s.t.} \quad & \sum_{r\in V} z_{ir} \geq 1, \qquad \forall i \in [m], \\
& \{f_{r,s_i,e} : e \in E\} \text{ is a flow from } s_i \text{ to } r \text{ of } z_{ir} \text{ units}, \qquad \forall r \in V, \forall i \in [m], \\
& \{f_{r,t_i,e} : e \in E\} \text{ is a flow from } r \text{ to } t_i \text{ of } z_{ir} \text{ units}, \qquad \forall r \in V, \forall i \in [m], \\
& f_{r,u,e} \leq x_{e,r}, \qquad \forall u \in \mathcal{T}, \forall e \in E, \forall r \in V, \\
& x, f, z \geq 0.
\end{aligned}$$

The above LP is not of packing or covering type due to the flow constraints: there are both positive and negative signs on variables. The online algorithm in [56] for this LP uses various ideas and has competitive ratio $O(D \cdot \log n)$ w.r.t. the optimal *integral* solution; here D is an upper bound on the length of any $s_i - t_i$ path (note that D can be as large as n). Using a height reduction operation, they could ensure that $D = O(\log n)$ while incurring an additional $O(\log n)$-factor loss in the objective. This lead to the $O(\log^3 n)$ factor for the fractional online algorithm. Here we provide an improved $O(\log n)$-competitive algorithm that does not require any bound on the path-lengths and that also guarantees the approximation relative to the optimal *fractional* solution.

For any $r \in V$ and $u \in \mathcal{T}$, let $\mathsf{MC}(r, u)$ denote the $u - r$ (resp. $r - u$) minimum cut in the graph with edge capacities $\{f_{r,u,e} : e \in E\}$ if u is a source (resp. destination). By the max-flow min-cut theorem, it follows that $z_{ir} \leq \min\{\mathsf{MC}(r, s_i)\,,\, \mathsf{MC}(r, t_i)\}$. Using this, we can combine the first three constraints of the above LP into the following:

$$\sum_{r\in V} \min\{\mathsf{MC}(r, s_i)\,,\, \mathsf{MC}(r, t_i)\} \;\geq\; 1, \qquad \forall i \in [m].$$

For a fixed $i \in [m]$, this constraint is equivalent to the following. For each $r \in V$, pick either an $s_i - r$ cut (under capacities $f_{r,s_i,\star}$) or an $r - t_i$ cut (under capacities $f_{r,t_i,\star}$), and check if the total cost of these cuts is at least one. Moreover, given values for the f-variables, it is optimal for the LP to set $x_{er} = \max_{u\in\mathcal{T}} f_{r,u,e}$ for all $e \in E$ and $r \in V$.

This leads to the following equivalent reformulation that eliminates the x and z variables. We use the notation $f_{r,u}(S) = \sum_{e\in S} f_{r,u,e}$ for any subset $S \subseteq E$, and $r \in V$, $u \in \mathcal{T}$.

$$\min \quad \sum_{r\in V}\sum_{e\in E} c_e \cdot \left(\max_{u\in\mathcal{T}} f_{r,u,e}\right) \; + \; \sum_{r\in V}\sum_{e\in E} \ell_e \cdot \sum_{u\in\mathcal{T}} f_{r,u,e}$$

$$\text{s.t.} \quad \sum_{r\in R_s} f_{r,s_i}(S_r) \; + \; \sum_{r\in R_t} f_{r,t_i}(T_r) \geq 1, \qquad \forall i \in [m],\ \forall (R_s, R_t) \text{ partition of } V,$$

$$\forall S_r : s_i - r \text{ cut}, \forall r \in R_s,\ \forall T_r : r - t_i \text{ cut}, \forall r \in R_t,$$

$$f \geq 0.$$

Note that $\ell_{\log(n)}$-norm is a constant approximation for ℓ_∞. Therefore we can reformulate the above objective function (at the loss of a constant factor) as the sum of $\ell_{\log(n)}$ and ℓ_1 norms. Our fractional solver applies to this convex covering problem, and yields an $O(\log n)$-competitive ratio (note that $\rho = 1$ for this instance). In order to get a polynomial running time, we can use the natural "separation oracle" approach (as in Section 3.2) to produce violated covering constraints.

Input : The i^{th} request (s_i, t_i).
Output: Current solution f.

When the i^{th} request (s_i, t_i) arrives
while $\sum_{r\in V} \min\{\mathsf{MC}(r, s_i), \mathsf{MC}(r, t_i)\} < \frac{1}{2}$ **do**
  For each $r \in V$, compute $\mathsf{MC}(r, s_i)$ and $\mathsf{MC}(r, t_i)$ and the respective cuts S_r and T_r;
  Let $R_s = \{r \in V : \mathsf{MC}(r, s_i) \leq \mathsf{MC}(r, t_i)\}$ and $R_t = V \setminus R_s$;
  Run Algorithm 4 with constraint $\sum_{r\in R_s} f_{r,s_i}(S_r) + \sum_{r\in R_t} f_{r,t_i}(T_r) \geq 1$;
end

Algorithm 5: Separation Oracle Based Algorithm for Buy-at-Bulk

Each iteration of Algorithm 5 runs in polynomial time since the minimum cuts can be computed in polynomial time. In order to bound the number of iterations, consider the potential $\psi = \sum_{e\in E}(f_{r,s_i,e} + f_{r,t_i,e})$. Note that $0 \leq \psi \leq 2|E|$ and each iteration increases ψ by at least $\frac{1}{2}$. So the number of iterations is at most $4|E|$.

Results for specific buy-at-bulk problems. Using existing results from offline and single-source versions of these problems, Theorem 3.1.3 implies the following:

- For undirected edge-weighted buy-at-bulk we obtain an $O(\log^9 n)$-competitive ratio in polynomial time using $\alpha = O(\log n)$ [41], $\beta = O(\log n)$ [42] and $\gamma = O(\log^4 n)$ [105]. This improves upon the $O(\log^{11} n)$-competitive ratio that follows from [56].

- For undirected node-weighted buy-at-bulk we obtain an $O(\log^9 n)$-competitive ratio in quasi-polynomial time using $\alpha = O(\log n)$ [41], $\beta = O(\log n)$ [41] and $\gamma = O(\log^4 n)$ [56, 5]. This again improves upon the $O(\log^{11} n)$-competitive ratio that follows from [56]. The quasi-polynomial runtime is due to the online single-source algorithm that relies on the height-reduction technique for directed Steiner problems [73].

- As discussed in [56], we can also obtain the same competitive ratios for the prize-collecting variants of these problems, where pairs may be left disconnected by paying a penalty in the objective. So our result implies an $O(\log^9 n)$-competitive ratio here as well.

3.4.2 Throughput Maximization with ℓ_p-Norm Capacities

The online problem of maximizing multicommodity flow was studied in [10, 30]. In this problem, we are given a directed graph $G = (V, E)$ with edge capacities $u(e)$. Requests (s_i, t_i) arrive in an online fashion. The algorithm needs to accept a subset of these requests and choose an $s_i - t_i$ path for each accepted request i. The number of paths using any edge e (referred to as the *load* of edge e) is not allowed to exceed its capacity $u(e)$. The goal is to maximize the number of accepted requests.

Here we consider an extension with capacity constraints on subsets of edges. In particular, we are also given a number of groups where the j^{th} group consists of a subset $S_j \subseteq E$ and requires the ℓ_{p_j}-norm of the loads of these edges to be at most c_j, i.e. $\sum_{e \in S_j} L_e^{p_j} \leq c_j^{p_j}$ where L_e denotes the load of edge e. The objective is again to maximize the number of accepted requests. Note that if each $|S_j| = 1$ then we recover the classic setting of individual edge capacities.

Example In Figure 3.2, there are four requests. (s_1, t_1) is satisfied by the yellow path, (s_2, t_2) is satisfied by the blue path, (s_3, t_3) is not satisfied, and (s_4, t_4) is satisfied by the red path. There are also three group of edges where we have ℓ_p-norm constraints, S_1, S_2, and S_3. Suppose we use ℓ_2-norm for all the groups. Then S_1 has load $\|(1, 0, 1, 1)\|_2 = \sqrt{3}$, S_2 has load $\|(1, 0, 1, 0, 0)\|_2 = \sqrt{2}$ and S_3 has load $\|(0, 3, 2, 2, 1, 0, 1, 1, 1, 0, 0)\|_2 = \sqrt{21}$.

In this section we assume (without loss of generality) that the subsets S_j form a partition of E. By subdividing edges if necessary, we can ensure that the subsets S_j are disjoint. If $\cup_j S_j \subsetneq E$ then we can just add a dummy group consisting of edges $E \setminus \cup_j S_j$ and assign a very high capacity to the dummy group. We denote the number of edges by m; as the

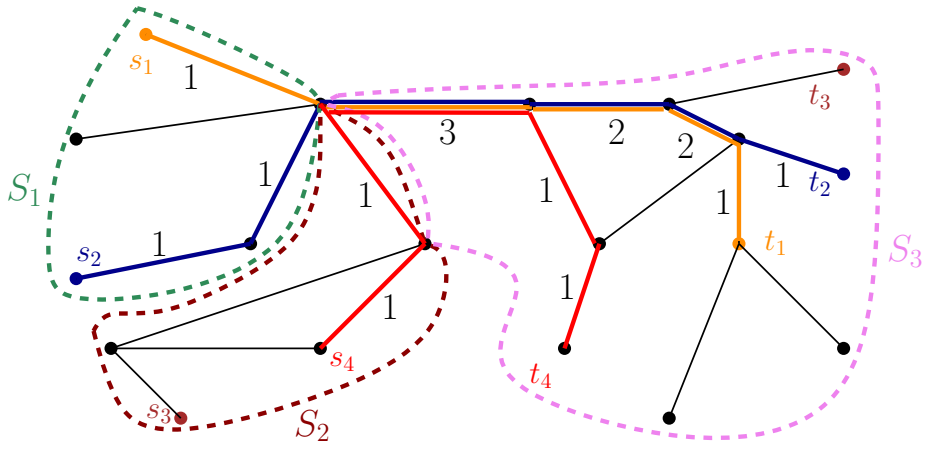

Figure 3.2: Throughput maximization with ℓ_p-norm capacities example

groups are disjoint, the number of groups is at most m. We also use i to index requests, j to index groups and e to index edges.

We provide two online algorithms for this problem. The first algorithm (in Section 3.4.2.1) runs in polynomial time and achieves an $O(\log m)$-competitive ratio when each $c_j = \Omega(\log m) \cdot |S_j|^{1/p_j}$. Without the high-capacity assumption, this implies an $O(\log m)$-competitive ratio while violating capacities by an $O(m^{1/p} \cdot \log m)$ factor, where $p = \min_j p_j$. The second algorithm (in Section 3.4.2.2) allows for any capacities and achieves an $O(\log m)$-competitive ratio while violating capacities by an $O(\log^{1+1/p} m)$ factor, where $p = \min_j p_j$. The second algorithm provides a better capacity violation than the first (for arbitrary capacities). However, the second algorithm does not run in polynomial time. The two algorithms rely on different convex relaxations, both of which correspond to our dual problem (D). We note that in the absence of a high-capacity assumption (or some capacity violation), there is no sub-polynomial randomized competitive ratio even in the special case where $|S_j| = 1$ [19].

A randomized (α, β)-bicriteria competitive algorithm finds a solution that (i) violates each capacity constraint by at most factor β with probability one, and (ii) has expected objective value at least $\frac{1}{\alpha}$ times the offline optimum.

3.4.2.1 Polynomial-time $(O(\log m), O(m^{1/p} \log m))$ bicriteria algorithm

Here we prove the first part of Theorem 3.1.4, which is restated below:

Theorem 3.4.1. *Assume that $c_j = \Omega(\log m) \cdot |S_j|^{1/p_j}$ for each j. Then there is a polynomial-time randomized $O(\log m)$-competitive online algorithm for throughput maximization with ℓ_p-norm capacities, where m is the number of edges in the graph.*

In particular, we will show that (i) the algorithm's solution satisfies all capacities with probability one and (ii) has expected objective at least an $O(\log m)$ fraction of the optimum.

In a fractional version of the problem, a request can be satisfied by several paths and the allocation of bandwidth can be in the range $[0, 1]$ instead of being restricted to $\{0, 1\}$. For request (s_i, t_i), let $\mathcal{P}_i$ be the set of simple paths from s_i to t_i. Variable $f_{i,P}$ is defined to be the amount of flow on the path P for request (s_i, t_i). The total profit is the (fractional) number of requests served. The complete fractional relaxation is given below:

$$\max \quad \sum_i \sum_{P \in \mathcal{P}_i} f_{i,P} \tag{3.7}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}_i} f_{i,P} \leq 1, \qquad \forall i \tag{3.8}$$

$$\sum_i \sum_{P \in \mathcal{P}_i : e \in P} f_{i,P} = \mu_e, \qquad \forall e \tag{3.9}$$

$$\|\mu(S_j)\|_{p_j} \leq c_j, \qquad \forall j \tag{3.10}$$

$$f \geq 0. \tag{3.11}$$

Constraint (3.8) ensures that at most one path is selected for each request, (3.9) assigns to each variable μ_e the load on edge e and (3.10) is the capacity constraint on each group. It is clear that when each $f_{i,P} \in \{0, 1\}$ we obtain an exact formulation of the routing problem. Rewriting constraint (3.8) as $\sum_{P \in \mathcal{P}_i} f_{i,P} = \nu_i$ and $\nu_i \leq 1$, we have the following equivalent relaxation:

$$\max \quad \sum_i \sum_{P \in \mathcal{P}_i} f_{i,P} \tag{3.12}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{P}_i} f_{i,P} = \nu_i, \qquad \forall i \tag{3.13}$$

$$\sum_i \sum_{P \in \mathcal{P}_i : e \in P} f_{i,P} = \mu_e, \qquad \forall e \tag{3.14}$$

$$\|\nu\|_\infty \leq 1, \tag{3.15}$$

$$\|\mu(S_j)\|_{p_j} \leq c_j, \qquad \forall j \tag{3.16}$$

$$f \geq 0. \tag{3.17}$$

Note that this corresponds to the (dual) packing program (D). In particular, if z_i and x_e are the primal variables corresponding to constraints (3.13) and (3.14) respectively, the primal

problem is:

$$\begin{aligned} \min \quad & \sum_j c_j \|x(S_j)\|_{q_j} + \sum_i z_i \\ \text{s.t.} \quad & z_i + \sum_{e \in P} x_e \geq 1, && \forall i,\ \forall P \in \mathcal{P}_i \\ & x, z \geq 0. \end{aligned}$$

This is in the form of (P), so Theorem 3.1.2 can be applied. However, each request is associated with an exponential number of constraints and to obtain a polynomial-time algorithm we again need to apply a separation oracle. This is based on computing shortest paths in a modified graph: we add a vertex s_i' and edge (s_i', s_i) to graph G. Let z_i be the length of edge (s_i', s_i) and x_e be the length of each edge $e \in E$, and let H denote this edge-weighted graph.

Input : The i^{th} request (s_i, t_i).
Output: Current solution x, z.
When the i^{th} request (s_i, t_i) arrives
while *shortest $s_i' - t_i$ path in H has length less than $\frac{1}{2}$* **do**
    Let $P \in \mathcal{P}_i$ be the path corresponding to the shortest $s_i' - t_i$ path in H;
    Run Algorithm 4 with request $z_i + \sum_{e \in P} x_e \geq 1$;
end

Algorithm 6: Online Algorithm for Throughput Maximization

The shortest path algorithm runs in polynomial time and it finds a constraint with $z_i + \sum_{e \in P} x_e < \frac{1}{2}$ (if any). To see that the number of iterations of Algorithm 6 is polynomial, define potential function $\psi = z_i + \sum_{e \in E} x_e$. We know that $\psi \leq m + 1$ the number of edges in H since our algorithm is minimal, that is, each iteration terminates with $z_i + \sum_{e \in P} x_e = 1$. In each iteration, ψ increases by at least $\frac{1}{2}$. So the total number of iteration is at most $2m$. Finally, by doubling the variables z, x we have a feasible solution and the objective increases by factor two.

Using Algorithm 4, we obtain an $O(\log m)$-competitive online algorithm for the fractional relaxation (3.12)-(3.17). To get an integer solution, we use a simple randomized rounding algorithm. For each request i, choose a path $P \in \mathcal{P}_i$ with probability $\frac{f_{i,P}}{8}$, and choose no path with the remaining probability $1 - \frac{1}{8}\sum_{P \in \mathcal{P}_i} f_{i,P}$. For each request i and edge e, let $X_{i,e} = 1$ if the path chosen for request i contains edge e and $X_{i,e} = 0$ otherwise. Let $X_e = \sum_i X_{i,e}$ denote the load on each edge $e \in E$; note that X_e is the sum of independent $0 - 1$ random variables. Also, by the rounding algorithm and constraint (3.14), $\mathbb{E}(X_e) = \frac{\mu_e}{8}$ for each edge e. We use the following standard concentration inequality.

Theorem 3.4.2 (Chernoff Bound 1)**.** *Let $X = \sum_i X_i$ where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Then*

$$\Pr[X \geq (1+\epsilon)\mathbb{E}[X]] \leq e^{-\frac{\epsilon^2}{2+\epsilon}\mathbb{E}[X]} \quad \textit{for all } \epsilon > 0.$$

Let $\delta = 12 \log m$. Using this result on $X_e = \sum_i X_{i,e}$ with $\epsilon = 1 + 2\delta/\mu_e$,

$$\Pr\left[X_e > \frac{\mu_e}{4} + \frac{\delta}{4}\right] = \Pr\left[X_e > (1+\epsilon)\mathbb{E}[X_e]\right] \leq e^{-\frac{(1+\frac{2\delta}{\mu_e})^2}{2+(1+\frac{2\delta}{\mu_e})}\frac{\mu_e}{8}} \leq e^{-\frac{\mu_e+2\delta}{8}} \leq e^{-\frac{\delta}{4}} = \frac{1}{m^3}.$$

Then by union bound over all m edges, we obtain that $X_e \leq \frac{\mu_e}{4} + \frac{\delta}{4}$ for all $e \in E$, with probability at least $1 - 1/m^2$. Conditioned on this "good event" we have

$$\begin{aligned}\sum_{e \in S_j} X_e^{p_j} &\leq \sum_{e \in S_j} (\frac{\mu_e}{4} + \frac{\delta}{4})^{p_j} \leq \sum_{e \in S_j} 2^{p_j} (\frac{\mu_e^{p_j}}{4^{p_j}} + \frac{\delta^{p_j}}{4^{p_j}}) \\ &= \frac{1}{2^{p_j}} (\|\mu(S_j)\|_{p_j}^{p_j} + |S_j|\delta^{p_j}) < c_j^{p_j}, \ \forall j. \end{aligned} \tag{3.18}$$

where the last inequality is by constraint (3.16) and the assumption $c_j = \Omega(\log m) \cdot |S_j|^{\frac{1}{p_j}}$.

In order to guarantee that we always find a feasible solution (i.e. satisfy all the capacities) we simply terminate the algorithm when *any* capacity constraint is about to be violated. Below ALG denotes the number of paths selected in the randomized rounding and $\overline{ALG}$ is the number of paths selected before the algorithm is terminated.

To prove the $O(\log m)$-competitive ratio, let OPT be the offline optimal value of the throughput maximization instance. We first assume $OPT = \Omega(\log m)$ and handle the case $OPT = O(\log m)$ later. Define the following random variables:

- For each request i, $A_i = 1$ if the rounding satisfies request i and $A_i = 0$ otherwise.
- $I = 0$ if the rounding satisfies all capacities and $I = 1$ otherwise.
- $ALG = \sum_i A_i$ the number of requests satisfied by the rounding.
- $\overline{ALG} = ALG$ if $I = 0$ and $\overline{ALG} = 0$ otherwise.
- $G = \min(ALG, OPT) - OPT \cdot I$.

Note that $\mathbb{E}[ALG] = \sum_i \mathbb{E}[A_i] = \sum_i \sum_{P \in \mathcal{P}_i} \frac{f_{i,P}}{8} \geq \frac{OPT}{O(\log m)}$ because our fractional online algorithm is $O(\log m)$-competitive. Moreover, assuming $OPT = \Omega(\log m)$ we have $\mathbb{E}[ALG] = \Omega(1)$.

By definition of G, we have $\overline{ALG} \geq G$ because:

1. if the rounded solution is feasible ($I = 0$) then $\overline{ALG} = ALG \geq \min(ALG, OPT)$,

2. if the rounded solution is infeasible ($I = 1$) then $\overline{ALG} \geq 0 \geq G$.

Now we have

$$\begin{aligned}\mathbb{E}[\overline{ALG}] \geq \mathbb{E}[G] &= \mathbb{E}[\min(ALG, OPT)] - OPT \cdot \mathbb{E}[I] \\ &\geq \mathbb{E}[\min(ALG, OPT)] - \frac{OPT}{m^2}. \end{aligned} \tag{3.19}$$

The last inequality is by (3.18) which implies $\mathbb{E}[I] \leq \frac{1}{m^2}$.

We now use the Chernoff bound on the lower tail.

Theorem 3.4.3 (Chernoff Bound 2)**.** *Let $X = \sum_i X_i$ where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Then*

$$\Pr[X \leq (1-\epsilon)\mathbb{E}[X]] \leq e^{-\frac{\epsilon^2}{2}\mathbb{E}[X]} \quad \textit{for all } 0 < \epsilon < 1.$$

Recall $\mathbb{E}[ALG] = \Omega(1)$ (using our assumption on OPT) and $A_i \in \{0, 1\}$ for all i. By choosing $\epsilon = \frac{1}{2}$ in Theorem 3.4.3, with constant probability we have $ALG \geq \frac{1}{2}\mathbb{E}[ALG] \geq \frac{OPT}{O(\log m)}$. Finally, using $ALG \geq 0$ we have $\mathbb{E}[\min(ALG, OPT)] = \frac{OPT}{O(\log m)}$. Combined with (3.19) we obtain $\mathbb{E}[\overline{ALG}] = \frac{OPT}{O(\log m)}$.

We now handle the case $OPT = O(\log m)$. Note that in this case, just selecting a single path is an $O(\log m)$-competitive solution. The overall algorithm runs with probability half either the above rounding or the greedy choice of selecting any path for the first request. Note that selecting any path $P \in \mathcal{P}_i$ leads to a feasible solution because of our capacity assumption. Finally, the expected objective is $OPT/O(\log m)$, which completes the proof of Theorem 3.4.1.

3.4.2.2 An $(O(\log m), O(\log^{1+1/p} m))$ bicriteria algorithm

We now prove the second part of Theorem 3.1.4 (restated below).

Theorem 3.4.4. *There is a randomized $(O(\log m), O(\log^{1+1/p} m))$-bicriteria competitive online algorithm for throughput maximization with ℓ_p-norm capacities, where m is the number of edges in the graph and $p = \min_j p_j$.*

For this result we further strengthen the dual continuous relaxation to

$$\max \quad \sum_i \sum_{P \in \mathcal{Q}_i} f_{i,P} \tag{3.20}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{Q}_i} f_{i,P} \leq 1, \qquad \forall i \tag{3.21}$$

$$\sum_i \sum_{P \in \mathcal{Q}_i} |S_j \cap P| f_{i,P} \leq c_j^{p_j}, \qquad \forall j \tag{3.22}$$

$$\sum_i \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P} = \mu_e, \qquad \forall e \tag{3.23}$$

$$\|\mu(S_j)\|_{p_j} \leq c_j, \qquad \forall j \tag{3.24}$$

$$f \geq 0. \tag{3.25}$$

Here $\mathcal{Q}_i$ is the set of simple paths P between s_i and t_i such that $|S_j \cap P| \leq c_j^{p_j}$ for all groups j.

Lemma 3.4.5. *Convex program* (3.20)-(3.25) *is a relaxation of the throughput maximization problem.*

Proof. We first observe that only paths in $\cup_i \mathcal{Q}_i$ may be used in any feasible solution for the throughput maximization problem. Suppose (for a contradiction) that some $s_i - t_i$ path $P \in \mathcal{P}_i \setminus \mathcal{Q}_i$ is used. Then it is clear that the load induced by path P alone violates some capacity c_j, which contradicts the feasibility. This justifies using only $f_{i,P}$ variables corresponding to the $\mathcal{Q}_i$s.

Now, note that any feasible solution to the throughput maximization problem corresponds to a solution (f, μ) with each $f_{i,P} \in \{0, 1\}$ that satisfies constraints (3.21), (3.23) and (3.24). The objective value (number of accepted requests) is clearly (3.20). We only need to show that the new constraints (3.22) are also satisfied. We know that the ℓ_{p_j} load on each edge subset S_j is at most c_j. That is,

$$\sum_e (\sum_i \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P})^{p_j} \leq c_j^{p_j}, \qquad \forall j.$$

Then, using the fact that each $f_{i,P} \in \{0, 1\}$ we have for any group j,

$$\begin{aligned} \sum_{e \in S_j} (\sum_i \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P})^{p_j} &\geq \sum_{e \in S_j} \sum_i \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P} = \sum_i \sum_{e \in S_j} \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P} \\ &= \sum_i \sum_{P \in \mathcal{Q}_i} |S_j \cap P| f_{i,P}. \end{aligned}$$

Hence we obtain $\sum_i \sum_{P \in \mathcal{Q}_i} |S_j \cap P| f_{i,P} \leq c_j^{p_j}$ for all j, as desired. □

Now, the relaxation (3.20)-(3.25) can be recast as

$$\max \quad \sum_i \sum_{P \in \mathcal{Q}_i} f_{i,P} \tag{3.26}$$

$$\text{s.t.} \quad \sum_{P \in \mathcal{Q}_i} f_{i,P} = \nu_i, \qquad \forall i \tag{3.27}$$

$$\sum_i \sum_{P \in \mathcal{Q}_i} |S_j \cap P| f_{i,P} = \lambda_j, \qquad \forall j \tag{3.28}$$

$$\sum_i \sum_{P \in \mathcal{Q}_i : e \in P} f_{i,P} = \mu_e, \qquad \forall e \tag{3.29}$$

$$\|\nu\|_\infty \leq 1, \tag{3.30}$$

$$\|\mu(S_j)\|_{p_j} \leq c_j, \qquad \forall j \tag{3.31}$$

$$\|\lambda_j\|_\infty \leq c_j^{p_j}, \qquad \forall j \tag{3.32}$$

$$f \geq 0. \tag{3.33}$$

This is exactly in the form of our dual program (D). If z_i, y_j and x_e are primal variables corresponding to (3.27), (3.28) and (3.29) respectively, the primal program is

$$\begin{aligned} \min \quad & \sum_j c_j \|x(S_j)\|_{q_j} + \sum_j c_j^{p_j} y_j + \sum_i z_i \\ \text{s.t.} \quad & z_i + \sum_j |S_j \cap P| y_j + \sum_{e \in P} x_e \geq 1, \qquad \forall i, \forall P \in \mathcal{Q}_i, \\ & x, y, z \geq 0. \end{aligned}$$

We can apply Theorem 3.1.2 to this primal formulation to obtain an $O(\log m)$ -competitive online algorithm. However, there are an exponential number of constraints and in this case we are not aware of an efficient separation oracle. The separation problem corresponds to resource constrained shortest path [103] and to the best of our knowledge, no efficient (approximation) algorithms are known. So the running time of our fractional online algorithm is exponential.

The online randomized rounding algorithm is the same as section 3.4.2.1. We again use Chernoff bound to prove that with high probability the load on each edge e is small. Recall that for each request i and edge e, random variable $X_{i,e}$ the the indicator if the path chosen for request i contains edge e, and $X_e = \sum_i X_{i,e}$ is the load on edge e. We also use $Y_{i,P}$ as the indicator that path $P \in \mathcal{Q}_i$ is selected for request i. We analyze the following two cases separately.

Case 1: $\mu_e \geq 1$**.** Let $\delta = 12 \log m$. By Chernoff bound (Theorem 3.4.2), we have

$$\Pr[X_e > \frac{\mu_e}{4} + \frac{\delta}{4}] \leq e^{-\frac{(1+\frac{2\delta}{\mu_e})^2}{2+(1+\frac{2\delta}{\mu_e})}\frac{\mu}{8}} \leq e^{-\frac{\mu_e+2\delta}{8}} \leq e^{-\frac{\delta}{4}} = \frac{1}{m^3}.$$

So, with probability $1-\frac{1}{m^3}$ we have $X_e \leq (3\log m)\mu_e$, which implies $X_e^{p_j} \leq (3\log m)^{p_j}\mu_e^{p_j}$ for all j.

Case 2: $\mu_e \leq 1$**.** Let random variable $R_e = 1$ if some path using e is selected, and $R_e = 0$ otherwise. Note that $R_e \leq \sum_i \sum_{P\in\mathcal{Q}_i: e\in P} Y_{i,P}$. Again by Chernoff bound (Theorem 3.4.2),

$$\Pr[X_e > 3\log m] \leq \frac{1}{m^3}.$$

Conditioned on $X_e \leq 3\log m$, we have $X_e \leq 3\log m \cdot R_e$ because $X_e = 0$ if and only if $R_e = 0$. Then

$$X_e^{p_j} \leq (3\log m)^{p_j} \cdot R_e \leq (3\log m)^{p_j} \sum_i \sum_{P\in\mathcal{Q}_i: e\in P} Y_{i,P}, \qquad \forall j.$$

Let $\mathcal{E}$ denote the event that $X_e \leq (3\log m)\cdot\max\{\mu_e, 1\}$ for all edges e. It follow from above that $\Pr[\mathcal{E}] \geq 1 - \frac{1}{m^2}$. Moreover, conditioned on $\mathcal{E}$, we have for each group j:

$$\begin{aligned}
\sum_{e\in S_j} X_e^{p_j} &\leq \sum_{e\in S_j} (3\log m)^{p_j}\mu_e^{p_j} + \sum_{e\in S_j}(3\log m)^{p_j}\sum_i \sum_{P\in\mathcal{Q}_i: e\in P} Y_{i,P} \\
&= (3\log m)^{p_j}\sum_{e\in S_j}\mu_e^{p_j} + (3\log m)^{p_j}\sum_i\sum_{e\in S_j}\sum_{P\in\mathcal{Q}_i: e\in P} Y_{i,P} \\
&= (3\log m)^{p_j}\sum_{e\in S_j}\mu_e^{p_j} + (3\log m)^{p_j}\sum_i\sum_{P\in\mathcal{Q}_i} |S_j\cap P|\cdot Y_{i,P} \\
&\leq (3\log m)^{p_j}\cdot c_j^{p_j} \;+\; (3\log m)^{p_j}\sum_i\sum_{P\in\mathcal{Q}_i} |S_j\cap P|\cdot Y_{i,P}. \qquad (3.34)
\end{aligned}$$

where the last inequality is by constraint (3.24).

By the constraints (3.22) and definition of $\mathcal{Q}_i$, we have:

$$\mathbb{E}\left[\sum_i\sum_{P\in\mathcal{Q}_i} |S_j\cap P|\cdot Y_{i,P}\right] \leq c_j^{p_j} \quad \text{and} \quad |S_j\cap P| \leq c_j^{p_j} \text{ for all } P\in\cup_i\mathcal{Q}_i.$$

Note that the random variables $\sum_{P\in\mathcal{Q}_i}|S_j\cap P|\cdot Y_{i,P}$ are independent across requests i and bounded between 0 and $c_j^{p_j}$. By Chernoff bound (Theorem 3.4.2) and union bound over

groups j, it follows that with probability at least $1 - \frac{1}{m^2}$,

$$\sum_i \sum_{P \in \mathcal{Q}_i} |S_j \cap P| \cdot Y_{i,P} \leq (3 \log m) c_j^{p_j}, \qquad \forall j.$$

Let $\mathcal{F}$ denote the above event. Combined with (3.34), conditioned on $\mathcal{E}$ and $\mathcal{F}$ we obtain,

$$\sum_{e \in S_j} X_e^{p_j} \leq (3 \log m)^{p_j} \cdot c_j^{p_j} + (3 \log m)^{p_j} \cdot (3 \log m) c_j^{p_j} \leq 2(3 \log m)^{p_j + 1} \cdot c_j^{p_j}, \ \forall j.$$

As $\Pr[\mathcal{E} \wedge \mathcal{F}] \geq 1 - \frac{2}{m^2}$, it follows that the capacities are violated by at most an $O(\log^{1+1/p} m)$ factor with high probability. Here $p = \min_j p_j$.

The proof of the $O(\log m)$ competitive ratio and ensuring that the capacity bounds hold with probability one, are identical to that in Section 3.4.2.1. This completes the proof of Theorem 3.4.4.

3.5 Conclusion

In this chapter, we obtained a nearly tight $O(\log d + \log \rho)$-competitive algorithm for fractional online covering problems with ℓ_q-norm objectives and its dual packing problem. We also demonstrated the applicability of this result in two settings: non-uniform buy-at-bulk network design and throughput maximization under ℓ_p-norm capacities. It leads to an improved result (by two logarithmic factors) for the former problem and the first poly-logarithmic result for the latter one. Identifying online algorithms for other classes of convex programs is an interesting direction. Another open question is to design online algorithms for more combinatorial optimization problems using convex program relaxations.

Credits: The results in this chapter are from "Online Covering with Sum of ℓ_q-Norm Objectives" [109], obtained jointly with Viswanath Nagarajan.

CHAPTER 4

Stochastic Load Balancing

4.1 Introduction

Deterministic optimization assumes perfect information of the input. However, uncertainty is an inevitable issue for almost all decision problems. It can be caused by many reasons, such as the errors and noises in data measurements, the model parameters, or the inherent nature of the problem. Dealing with uncertainty in the input is a fundamental issue in practical optimization problems. Because estimates based on historical data are usually available, stochastic optimization is a prominent approach in this area, where certain parts of the input are represented by random variables and the algorithm needs to optimize the expected performance [23]. Stochastic optimization has been studied for more than 60 years [45, 100, 83, 62, 84, 72, 110]. Recent years have witnessed a surge of interest in solving combinatorial optimization problems under uncertainty [52, 46, 88, 80, 67, 66, 97]. It is well known that many combinatorial optimization problems are $\mathcal{NP}$-hard. The stochastic setting introduces considerable additional complications and makes some simple questions even $\#\mathcal{P}$-hard [46, 88, 97]. Therefore, it is very unlikely that these stochastic problems admit efficient algorithms that can solve them exactly. Again, a principled way to deal with the computational intractability is to design polynomial time approximation algorithms.

In this chapter, we consider the problem of scheduling jobs on machines to minimize the maximum load (i.e., the problem of *makespan minimization*). This is a classic $\mathcal{NP}$-hard problem, with Graham's list scheduling algorithm for the identical machines being one of the earliest approximation algorithms known. If the job sizes are deterministic, the problem is fairly well-understood, with PTASes for the identical [74] and related machines cases [75], and a constant-factor approximation and APX-hardness [95, 117] for the case of unrelated machines. Given we understand the basic problem well, it is natural to consider settings which are less stylized, and one step closer to modeling real-world scenarios: *what*

can we do if there is uncertainty in the job sizes?

We take a stochastic optimization approach where the job sizes are random variables with known distributions. In particular, the size of each job j on machine i is given by a random variable X_{ij}. Throughout this chapter we assume that the sizes of different jobs are independent of each other. Given just this information, an algorithm has to assign these jobs to machines, say resulting in jobs J_i being assigned to each machine i. The expected makespan of this assignment is

$$\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} X_{ij}\right], \tag{4.1}$$

where m denotes the number of machines. The goal for the algorithm is to minimize this expected makespan. Observe that the entire assignment of jobs to machines is done up-front without knowledge of the actual outcomes of the random variables, and hence there is no adaptivity in this problem.

4.1.1 Results and Techniques

Our main result is:

Theorem 4.1.1 (Stochastic load balancing result)**.** *There is an $O(1)$-approximation algorithm for the problem of finding an assignment to minimize the expected makespan on unrelated machines.*

Our work naturally builds on the edifice of [88]. However, we need several new ideas to achieve this. As mentioned above, the prior result for identical machines used the notion of effective size, which depends on the number m of machines available. When machines are not identical, consider just the "restricted assignment" setting where each job needs to choose from a specific subset of machines: here it is not even clear how to define the effective size of each job. Instead of working with a single deterministic value as the effective size of any random variable X_{ij}, we use all the $\beta_k(X_{ij})$ values for $k = 1, 2, \cdots m$.

Then we show that in an optimal solution, for every k-subset of machines, the total β_k effective size of jobs assigned to those machines is at most some bound depending on k. Such a property for $k = m$ was also used in the algorithm for identical machines. We then formulate a linear program (LP) relaxation that enforces such a "volume" constraint for *every* subset of machines. Although our LP relaxation has an exponential number of constraints, it can be solved in polynomial time using the ellipsoid algorithm and a suitable separation oracle.

Finally, given an optimal solution to this LP, we show how to carefully choose the right parameter for effective size of each job and use it to build an approximately optimal schedule. Although our LP relaxation has an exponential number of constraints (and it seems difficult to preserve them all), we show that it suffices to satisfy a small subset of these constraints in the integral solution. Roughly, our rounding algorithm uses the LP solution to identify the "correct" deterministic size for each job and then applies an existing algorithm for deterministic scheduling [117].

Budgeted Makespan Minimization. In this problem, each job j has a *reward* r_j (having no relationship to other parameters such as its size), and we are given a target reward value R. The goal is to assign some subset $S \subseteq [n]$ of jobs whose total reward $\sum_{j \in S} r_j$ is at least R, and to minimize the expected makespan of this assignment. Clearly, this generalizes the basic makespan minimization problem (by setting all rewards to one and the target $R = n$).

Theorem 4.1.2 (Budgeted makespan minimization result)**.** *There is an $O(1)$-approximation algorithm for the budgeted makespan minimization problem on unrelated machines.*

To solve this, we extend the ideas for expected makespan scheduling to include an extra constraint about high reward. We again write a similar LP relaxation. Rounding this LP requires some additional ideas on top of those in Theorem 4.1.1. The new ingredient is that we need to round solutions to an assignment LP with two linear constraints. To do this without violating the budget, we utilize a "reduction" from the Generalized Assignment Problem to bipartite matching [117] as well as certain adjacency properties of the bipartite matching polytope [15].

Minimizing ℓ_q Norms. Finally, we consider the problem of stochastic load balancing under ℓ_q norms. Note that given some assignment, we can denote the "load" on machine i by $L_i := \sum_{j \in J_i} X_{ij}$, and the "load vector" by $\mathbf{L} = (L_1, L_2, \ldots, L_m)$. The expected makespan minimization problem is to minimize $\mathbb{E}[\|\mathbf{L}\|_\infty]$. The *$q$-norm minimization problem* is the following: find an assignment of jobs to machines to minimize

$$\mathbb{E}\big[\|\mathbf{L}\|_q\big] = \mathbb{E}\left[\left(\sum_{i=1}^{m}\left(\sum_{j \in J_i} X_{ij}\right)^q\right)^{1/q}\right].$$

Our result for this setting is the following:

Theorem 4.1.3 (Stochastic q-norm minimization result)**.** *There is an $O(\frac{q}{\log q})$-approximation algorithm for the stochastic q-norm minimization problem on unrelated machines.*

The main idea here is to reduce this problem to a suitable instance of deterministic q-norm minimization with additional side constraints. We then show that existing techniques for deterministic q-norm minimization [14] can be extended to obtain a constant-factor approximation for our generalization as well. We also need to use/prove some probabilistic inequalities to relate the deterministic sub-problem to the stochastic problem. We note that using general polynomial concentration inequalities [87, 114] only yields an approximation ratio that is exponential in q. We obtain a much better $O(q/\log q)$-approximation factor by utilizing the specific form of the norm function. Specifically, we use the convexity of norms, a second-moment calculation and a concentration bound [82] for the q^{th} moment of sums of independent random variables.

We note that Theorem 4.1.3 implies a constant-factor approximation for any fixed $q \geq 1$. However, our techniques do not extend directly to provide an $O(1)$-approximation algorithms for all q-norms.

4.1.2 Related Work

Stochastic load-balancing problems are common in real-world systems where the job sizes are indeed not known, but given the large amounts of data, one can generate reasonable estimates for the distribution. Moreover static (non-adaptive) assignments are preferable in many applications as they are easier to implement.

Inspired by work on scheduling and routing problems in several communities, Kleinberg, Rabani, and Tardos first posed the problem of approximating the expected makespan in 1997 [88]. They gave a constant-factor approximation for the *identical machines* case, i.e., for the case where for each job j, the sizes $X_{ij} = X_{i'j}$ for all $i, i' \in [m]$. A key concept in their result was the *effective size* of a random variable (due to Hui [77]) which is a suitably scaled logarithm of the moment generating function. This effective size (denoted β_m) depended crucially on the number m of machines. Roughly speaking, the algorithm in [88] solved the *deterministic* makespan minimization problem by using the effective size $\beta_m(X_j)$ of each job j as its deterministic size. The main part of their analysis involved proving that the resulting schedule also has small *expected* makespan when viewed as a solution to the stochastic problem. See Section 4.2 for a more detailed discussion.

Goel and Indyk [60] considered the stochastic load balancing problem on identical machines (same setting as [88]) but for specific job-size distributions. For Poisson distributions they showed that Graham's algorithm achieves a 2-approximation, and for Exponential distributions they obtained a PTAS. Despite these improvements and refined understanding of the identical machines case, the above stochastic load-balancing problem

has remained open, even for the related machines setting. Recall that *related machines* refers to the case where each machine i has a *speed* s_i, and the sizes for each job j satisfy $X_{ij}\, s_i = X_{i'j}\, s_{i'}$ for all $i, i' \in [m]$.

Kleinberg et al. [88] also considered stochastic versions of knapsack and bin-packing: given an overflow probability p, a feasible single-bin packing here corresponds to any subset of jobs such that their total size exceeds one with probability at most p. [60] gave better/simpler algorithms for these problems, under special distributions.

Recently, Deshpande and Li [96], and Li and Yuan [98] considered several combinatorial optimization problems including shortest paths, minimum spanning trees, where elements have weights (which are random variables), and one would like to find a solution (i.e. a subset of elements) whose expected utility is maximized. These results also apply to the stochastic versions of knapsack and bin-packing from [88] and yield bicriteria approximations. The main technique here is a clever discretization of probability distributions. However, to the best of our knowledge, such an approach is not applicable to stochastic load balancing.

Stochastic scheduling has been studied in many different contexts, by different fields (see, e.g., [111]). The work on approximation algorithms for these problems is more recent; see [106] for some early work and many references. In this chapter we consider the *(non-adaptive) fixed assignment model*, where jobs have to be assigned to machines up-front, and then the randomness is revealed. Hence, there is no element of adaptivity in these problems. This makes them suitable for settings where the decisions cannot be instantaneously implemented (e.g., for virtual circuit routing, or assigning customers to physically well-separated warehouses). A number of papers [106, 102, 79, 69] have considered scheduling problems in the *adaptive* setting, where assignments are done online and the assignment for a job may depend on the state of the system at the time of its assignment. See Section 4.2 for a comparison of adaptive and non-adaptive settings in the load balancing problem.

Very recently, Molinaro [107] obtained an $O(1)$-approximation algorithm for the stochastic q-norm problem for all $q \geq 1$, which improves over Theorem 4.1.3. In addition to the techniques in this chapter, the main idea in [107] is to use a different notion of effective size, based on the L-function method [91]. We still present our algorithm/analysis for Theorem 4.1.3 as it is conceptually simpler and may provide better constant factors for small q.

4.2 Preliminaries

The stochastic load balancing problem (StocMakespan) involves assigning n jobs to m machines. For each job $j \in [n]$ and machine $i \in [m]$, we are given a random variable X_{ij}

which denotes the processing time (size) of job j on machine i. We assume that the random variables $X_{ij}, X_{i',j'}$ are independent when $j \neq j'$ (the size of job j on different machines may be correlated). We assume access to the distribution of these random variables via some (succinct) representation. A solution is a partition $\{J_i\}_{i=1}^m$ of the jobs among the machines, such that $J_i \subseteq [n]$ is the subset of jobs assigned to machine $i \in [m]$. The expected makespan of this solution is $\mathbb{E}\left[\max_{i=1}^m \sum_{j \in J_i} X_{ij}\right]$. Our goal is to find a solution which minimizes the expected makespan.

The deterministic load balancing problem is known to be $\mathcal{NP}$-hard. However, the stochastic versions introduce considerable additional complications. For example, [88] showed that given Bernoulli trials $\{X_j\}$, it is $\#\mathcal{P}$-hard to compute the overflow probability, i.e. $\Pr[\sum_j X_j > 1]$. We now show that it is $\#\mathcal{P}$-hard even to compute the objective value of a given assignment in the identical machines setting.

Theorem 4.2.1 ($\#\mathcal{P}$-hardness)**.** *Given n jobs and m identical machines where each job j has Bernoulli random size X_j of type (q_j, s_j), it is $\#\mathcal{P}$-hard to compute the expected makespan of a given assignment.*

Proof. We prove $\#\mathcal{P}$-hardness by a reduction from the problem of counting the number of feasible solutions to the knapsack problem [53]. That is, given numbers $y_1, \ldots, y_n$ and a bound B, we want to know how many subsets of $\{y_1, \ldots, y_n\}$ add up to at most B. We consider the complementary problem of counting the number of subsets of $\{y_1, \ldots, y_n\}$ that sum to more than B.

Given $\{y_1, \ldots, y_n\}$, we create Bernoulli trials $X_1, \ldots, X_n$ such that X_j is of type $(\frac{1}{2}, y_j)$ and X_B of type $(B, 1)$ and X_{B+1} of type $(B+1, 1)$. Consider two instances of load balancing. The first instance contains two machines and jobs $X_1, \ldots, X_n$ and X_B. The second one contains two machines and jobs $X_1, \ldots, X_n$ and X_{B+1}. We want to compute the expected makespan of the following assignment for these two instance: assign jobs $X_1, \ldots, X_n$ to machine 1 and the remaining job to machine 2. We use Obj_1 and Obj_2 to denote the expected makespan of instance 1 and 2 respectively. Then we have

$$\begin{aligned} Obj_1 =& \mathbb{E}\left[\max\left\{B, \sum_{j=1}^n X_j\right\}\right] = B + \mathbb{E}\left[\max\left\{0, \sum_{j=1}^n X_j - B\right\}\right] \\ =& B + \sum_{t \geq B+1} \Pr\left(\sum_{j=1}^n X_j \geq t\right). \end{aligned}$$

Similarly,

$$Obj_2 = B + 1 + \sum_{t \geq B+2} \Pr\left(\sum_{j=1}^{n} X_j \geq t\right).$$

It follows that $\Pr\left(\sum_{j=1}^{n} X_j \geq B+1\right) = Obj_1 - Obj_2 + 1$. The theorem follows from the fact that the number of subsets of $\{y_1, \ldots, y_n\}$ that sum to more than B is equal to $\Pr\left(\sum_{i=1}^{n} X_j \geq B+1\right) \cdot 2^n$. □

Guess-and-verify approach Using a standard binary search approach, in order to obtain an $O(1)$-approximation algorithm for StocMakespan it suffices to solve the following problem $G(M)$. Given a bound $M > 0$, either find a solution with expected makespan $O(M)$ or establish that the optimal expected makespan is $\Omega(M)$. The guess-and-verify approach is

Input : Instance of StocMakespan and a constant ϵ.
Output: A guess of the optimal makespan
$LB \leftarrow \max_j \min_i \mathbb{E}[X_{ij}]$; $UB \leftarrow \sum_j \min_i \mathbb{E}[X_{ij}]$;
$M \leftarrow \frac{UB+LB}{2}$;
while $UB - LB > \epsilon LB$ **do**
  if $G(M)$ *a solution with expected makespan* $O(M)$ **then**
    $UB \leftarrow M$; $M \leftarrow \frac{UB+LB}{2}$;
  else
    $LB \leftarrow M$; $M \leftarrow \frac{UB+LB}{2}$;
  end
end
Return $M = UB$;

Algorithm 7: The guess-and-verify approach

By the definition of $G(M)$, we know that there is solution with expected makespan $O(M)$ and the optimal expected makespan is at least $\Omega(LB)$. By assuming we know optimal makespan is M, we will lose at most an additional factor of ϵ in the approximation ratio. There is a trivial upper bound on the optimal expected, which is $\sum_j \min_i \mathbb{E}[X_{ij}] \leq n \max_j \min_i \mathbb{E}[X_{ij}]$, given by assigning each job to the machine where its expected size is minimal. Hence this approach runs in $O(\log n)$ time. Moreover, by scaling down all random variables by factor M, we may assume that $M = 1$.

We now provide some definitions and background results that will be used extensively in the rest of the chapter.

4.2.1 Truncated and Exceptional Random Variables

It is convenient to divide each random variable X_{ij} into its *truncated* and *exceptional* parts, defined below:

- $X'_{ij} := X_{ij} \cdot \mathbf{I}_{(X_{ij} \leq 1)}$ (called the *truncated* part),
- $X''_{ij} := X_{ij} \cdot \mathbf{I}_{(X_{ij} > 1)}$ (called the *exceptional* part).

The reason for doing this is that these two kinds of random variables (r.v.s) behave very differently with respect to the expected makespan. It turns out that expectation is a good notion of "deterministic" size for exceptional r.v.s, whereas one needs a more nuanced notion (called effective size) for truncated r.v.s: this is discussed in detail below.

We will use the following result (which follows from [88]) to handle exceptional r.v.s.

Lemma 4.2.2 (Exceptional Items Lower Bound)**.** *Let $X_1, X_2, \ldots, X_t$ be non-negative discrete random variables each taking value zero or at least L. If $\sum_j \mathbb{E}[X_j] \geq L$ then $\mathbb{E}[\max_j X_j] \geq L/2$.*

Proof. The Bernoulli case of this lemma appears as [88, Lemma 3.3]. The extension to the general case is easy. For each X_j, introduce independent *Bernoulli* random variables $\{X_{jk}\}$ where each X_{jk} corresponds to a particular instantiation s_{jk} of X_j, i.e. $\Pr[X_{jk} = s_{jk}] = \Pr[X_j = s_{jk}]$. Note that $\max_k X_{jk}$ is stochastically dominated by X_j: so $\mathbb{E}[\max_j X_j] \geq \mathbb{E}[\max_{jk} X_{jk}]$. Moreover, $\sum_{jk} \mathbb{E}[X_{jk}] = \sum_j \mathbb{E}[X_j] \geq L$. So the lemma follows from the Bernoulli case. □

4.2.2 Effective Size and Its Properties

As is often the case for stochastic optimization problems, we want to find some deterministic quantity that is a good surrogate for each random variable, and then use this deterministic surrogate instead of the actual random variable. Here, we use the *effective size*, which is based on the logarithm of the (exponential) moment generating function [77, 85, 55].

Definition 4.2.3 (Effective Size)**.** *For any random variable X and integer $k \geq 2$, define*

$$\beta_k(X) \;:=\; \frac{1}{\log k} \cdot \log \mathbb{E}\left[e^{(\log k) \cdot X}\right]. \tag{4.2}$$

Also define $\beta_1(X) := \mathbb{E}[X]$.

Example For the following random variable X, we show some of its effective size value.

x	0	1/2	1
$\Pr[X = x]$	0.4	0.1	0.5

Table 4.1: Distribution of X

k	1	2	4	8	16	32
Effective size	0.5500	0.6243	0.6893	0.7425	0.7844	0.8169

Table 4.2: Effective size of X

Useful properties of β_k To get some intuition for effective size, consider independent r.v.s $Y_1, \ldots, Y_n$. Then if $\sum_i \beta_k(Y_i) \leq b$,

$$\Pr[\sum_i Y_i \geq c] = \Pr[e^{\log k \sum_i Y_i} \geq e^{(\log k)c}] \leq \frac{\mathbb{E}[e^{\log k \sum_i Y_i}]}{e^{(\log k)c}} = \frac{\prod_i \mathbb{E}[e^{(\log k)Y_i}]}{e^{(\log k)c}}.$$

Taking logarithms, we get

$$\log \Pr[\sum_i Y_i \geq c] \leq \log k \cdot \left[\sum_i \beta_k(Y_i) - c\right] \implies \Pr[\sum_i Y_i \geq c] \leq \frac{1}{k^{c-b}}.$$

The above calculation, very reminiscent of the standard Chernoff bound argument, can be summarized by the following lemma (shown, e.g., in [77]).

Lemma 4.2.4 (Upper Bound)**.** *For independent random variables $Y_1, \ldots, Y_n$, if $\sum_i \beta_k(Y_i) \leq b$ then $\Pr[\sum_i Y_i \geq c] \leq (1/k)^{c-b}$.*

The usefulness of this definition comes from a partial converse, proved in [88]:

Lemma 4.2.5 (Lower Bound)**.** *Consider independent Bernoulli random variables $Y_1, \ldots, Y_n$ where each Y_i has non-zero size s_i being an inverse power of 2 such that $1/(\log k) \leq s_i \leq 1$. If $\sum_i \beta_k(Y_i) \geq 7$ then $\Pr[\sum_i Y_i \geq 1] \geq 1/k$.*

Outline of the algorithm for identical machines In using the effective size, it is important to set the parameter k carefully. For identical machines [88] used $k = m$ the total number of machines. Using the facts discussed above, we can now outline their algorithm/analysis (assuming that all r.v.s are truncated). If the total effective size is at most (say) $20m$ then the jobs can be assigned to m machines in a way that the effective-size

load on each machine is at most 21. By Lemma 4.2.4 and union bound, it follows that the probability of some machine exceeding load 23 is at most $m \cdot (1/m)^2 = 1/m$. On the other hand, if the total effective size is more than $20m$ then even if the solution was to balance these evenly, each machine would have effective-size load at least 7. By Lemma 4.2.5 it follows that the load on each machine exceeds one with probability $1/m$, and so with m machines this gives a certificate that the makespan is $\Omega(1)$.

Challenges with unrelated machines For unrelated machines, this kind of argument breaks down even in the restricted-assignment setting where each job can go on only some subset of machines. This is because we don't know what probability of success we want to aim for. For example, even if the machines had the same speed, but there were jobs that could go only on $\sqrt{m}$ of these machines, and others could go on the remaining $m - \sqrt{m}$ of them, we would want their effective sizes to be quite different. (See the example below.) And once we go to general unrelated machines, it is not clear if any combinatorial argument would suffice. Instead, we propose an LP-based lower bound that enforces one such constraint (involving effective sizes) for every subset of machines.

Bad example for simpler effective sizes For stochastic load balancing on identical machines [88] showed that any algorithm which maps each r.v. to a single real value and performs load balancing on these (deterministic) values incurs an $\Omega(\frac{\log m}{\log \log m})$ approximation ratio. This is precisely the reason they introduced the notion of truncated and exceptional r.v.s. For truncated r.v.s, their algorithm showed that it suffices to use $\beta_m(X_j)$ as the deterministic value and perform load balancing with respect to these. Exceptional r.v.s were handled separately (in a simpler manner). For unrelated machines, we now provide an example which shows that even when all r.v.s are truncated, any algorithm which maps each r.v. to a single real value must incur approximation ratio at least $\Omega(\frac{\log m}{\log \log m})$. This suggests that more work is needed to define the "right" effective sizes in the unrelated machine setting.

There are m machines and $m + \sqrt{m}$ jobs. Each r.v. X_j takes value 1 with probability $\frac{1}{\sqrt{m}}$ (and 0 otherwise). The first $\sqrt{m}$ jobs can only be assigned to machine 1. The remaining m jobs can be assigned to any machine. Note that $OPT \approx 1 + 1/e$ which is obtained by assigning the first $\sqrt{m}$ jobs to machine 1, and each of the remaining m jobs in a one-to-one manner. Given any fixed mapping of r.v.s to reals, note that all the X_j get the same value (say θ) as they are identically distributed. So the optimal value of the corresponding (deterministic) load balancing instance is $\sqrt{m} \cdot \theta$. Hence the solution which maps $\sqrt{m}$ jobs to each of the first $1 + \sqrt{m}$ machines is an optimal solution to the deterministic instance.

However, the expected makespan of this assignment is $\Omega(\frac{\log m}{\log \log m})$.

We will use the following specific result in dealing with truncated r.v.s.

Lemma 4.2.6 (Truncated Items Lower Bound)**.** *Let $X_1, X_2, \cdots X_n$ be independent $[0,1]$ r.v.s, and $\{J_i\}_{i=1}^m$ be any partition of $[n]$. If $\sum_{j=1}^n \beta_m(X_j) \geq 17m$ then*

$$\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} X_j\right] = \Omega(1).$$

Proof. This is a slight extension of [88, Lemma 3.4], with two main differences. Firstly, we want to consider arbitrary instead of just Bernoulli r.v.s. Secondly, we use a different definition of effective size than they do. We provide the details below for completeness.

At the loss of factor two in the makespan, we may assume (by rounding down) that the only values taken by the X_j r.v.s are inverse powers of 2. For each r.v. X_j, applying [88, Lemma 3.10] yields independent Bernoulli random variables $\{Y_{jk}\}$ so that for each power-of-2 value s we have

$$\Pr[X_j = s] = \Pr[s \leq \sum_k Y_{jk} < 2s].$$

Let $\overline{X}_j = \sum_k Y_{jk}$, so $X_j \leq \overline{X}_j < 2 \cdot X_j$ and $\beta_m(\overline{X}_j) = \sum_k \beta_m(Y_{jk})$. Note also that $\beta_m(\overline{X}_j) \geq \beta_m(X_j)$. Hence $\sum_{jk} \beta_m(Y_{jk}) \geq \sum_{j=1}^n \beta_m(X_j) \geq 17m$. Now, consider the assignment of the Y_{jk} r.v.s corresponding to $\{J_i\}_{i=1}^m$, i.e. for each $i \in [m]$ and $j \in J_i$, all the $\{Y_{jk}\}$ r.v.s are assigned to part i. Then applying [88, Lemma 3.4] which works for Bernoulli r.v.s, we obtain $\mathbb{E}\left[\max_{i=1}^m \sum_{j \in J_i} \sum_k Y_{jk}\right] = \Omega(1)$. Observe that the above lemma used a different notion of effective size: $\beta'_{1/m}(X) := \min\{s, sqm^s\}$ for any Bernoulli r.v. X taking value s with probability q. However, as shown in [88, Prop 2.5], $\beta_m(X) \leq \beta'_{1/m}(X)$ which implies the version that we use here.

Finally, using $X_j > \frac{1}{2}\overline{X}_j$ we obtain

$$\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} X_j\right] \geq \frac{1}{2}\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} \overline{X}_j\right] = \frac{1}{2}\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} \sum_k Y_{jk}\right] = \Omega(1),$$

which completes the proof. □

4.2.3 Non-Adaptive and Adaptive Solutions

We note that our model involves computing an assignment that is fixed *a priori*, before observing any random instantiations. Such solutions are commonly called non-adaptive.

A different class of solutions (called adaptive) involves assigning jobs to machines sequentially, observing the random instantiation of each assigned job. Designing approximation algorithms for the adaptive and non-adaptive models are mutually incomparable. For makespan minimization on identical machines, Graham's list scheduling already gives a trivial 2-approximation algorithm in the *adaptive* case (in fact, it is 2-approximate on an per-instance basis), whereas the *non-adaptive* case is quite non-trivial and the Kleinberg et al. [88] result was the first constant-factor approximation.

We now provide an instance with identical machines where there is an $\Omega(\frac{\log m}{\log \log m})$ gap between the best non-adaptive assignment (the setting of this chapter) and the best adaptive assignment. The instance consists of m machines and $n = m^2$ jobs each of which is identically distributed taking size 1 with probability $\frac{1}{m}$ (and 0 otherwise). Recall that Graham's algorithm considers jobs in any order and places each job on the least loaded machine. It follows that the expected makespan of this adaptive policy is at most $1 + \frac{1}{m} \cdot \mathbb{E}[X_j] = 2$. On the other hand, the best static assignment has expected makespan $\Omega(\frac{\log m}{\log \log m})$, which is obtained by assigning m jobs to each machine.

4.2.4 Useful Probabilistic Inequalities

Theorem 4.2.7 (Jensen's Inequality)**.** *Let $X_1, X_2, \ldots, X_t$ be random variables and $f(x_1, \cdots, x_t)$ be any convex function. Then*

$$\mathbb{E}[f(X_1, \cdots, X_t)] \geq f(\mathbb{E}[X_1], \cdots, \mathbb{E}[X_t]).$$

Theorem 4.2.8 (Rosenthal Inequality)**.** [112, 82, 91] *Let $X_1, X_2, \ldots, X_t$ be independent non-negative random variables. Let $q \geq 1$ and $K = \Theta(q/\log q)$. Then it is the case that*

$$\mathbb{E}\left[\left(\sum_j X_j\right)^q\right] \leq K^q \cdot \max\left\{\left(\sum_j \mathbb{E}[X_j]\right)^q, \sum_j \mathbb{E}[X_j^q]\right\}.$$

4.3 Makespan Minimization

The main result of this section is:

Theorem 4.1.1 (Stochastic load balancing result)**.** *There is an $O(1)$-approximation algorithm for the problem of finding an assignment to minimize the expected makespan on unrelated machines.*

Using a binary search scheme and scaling, it suffices to find one of the following:

(i) *upper bound:* a solution with expected makespan at most $O(1)$, or

(ii) *lower bound:* a certificate that the optimal expected makespan is more than one.

Hence, we assume that the optimal solution for the instance has unit expected makespan, and try to find a solution with expected makespan $b = O(1)$; if we fail we output a lower bound certificate.

At a high level, the ideas we use are the following: first, in Section 4.3.1 we show a more involved lower bound based on the effective sizes of jobs assigned to every subset of machines. This is captured using an exponentially-sized LP which is solvable in polynomial time. Then, to show that this lower bound is a good one, we give a new rounding algorithm for this LP in Section 4.3.2 to get an expected makespan within a constant factor of the lower bound.

4.3.1 A New Lower Bound

Our starting point is a more general lower bound on the makespan. The (contrapositive of the) following lemma says that if the effective sizes are large then the expected makespan must be large too. This is much the same spirit as Lemma 4.2.5, but for the general setting of unrelated machines.

Lemma 4.3.1. *Consider any feasible solution that assigns jobs J_i to each machine $i \in [m]$. If the expected makespan $\mathbb{E}\left[\max_{i=1}^{m} \sum_{j \in J_i} X_{ij}\right] \leq 1$, then*

$$\sum_{i=1}^{m} \sum_{j \in J_i} \mathbb{E}[X''_{ij}] \leq 2, \qquad \textit{and} \tag{4.3}$$

$$\sum_{i \in K} \sum_{j \in J_i} \beta_k(X'_{ij}) \;\leq\; O(1) \cdot k, \;\; \textit{for all } K \subseteq [m], \quad \textit{where } k = |K|. \tag{4.4}$$

Proof. The first inequality (4.3) focuses on the exceptional parts, and loosely follows from the intuition that if the sum of biases of a set of independent coin flips is large (exceeds 2 in this case) then you expect one of them to come up heads. Formally, the proof follows from Lemma 4.2.2 applied to $\{X''_{ij} : j \in J_i, i \in [m]\}$.

For the second inequality (4.4), consider any subset $K \subseteq [m]$ of the machines. Then the total effective size of the jobs assigned to these machines must be small, where now the effective size β_k can be measured with parameter $k = |K|$. Formally applying Lemma 4.2.6 only to the k machines in K and the truncated random variables $\{X'_{ij} : i \in K, j \in J_i\}$ corresponding to jobs assigned to these machines, we obtain the desired inequality. □

Given these valid inequalities, our algorithm now seeks an assignment satisfying (4.3)–(4.4). If we fail, the lemma assures us that the expected makespan must be large. On the other hand, if we succeed, such a "good" assignment by itself is not sufficient. The challenge is to show the converse of Lemma 4.3.1, i.e., that any assignment satisfying (4.3)–(4.4) gives us an expected makespan of $O(1)$.

Indeed, towards this goal, we first write an LP relaxation with an exponential number of constraints, corresponding to (4.4). We can solve this LP using the ellipsoid method. Then, instead of rounding the fractional solution to satisfy all constraints (which seems very hard), we show how to satisfy only a carefully chosen subset of the constraints (4.4) so that the expected makespan can still be bounded. Let us first give the LP relaxation.

In the ILP formulation of the above lower bound, we have binary variables y_{ij} to denote the assignment of job j to machine i, and fractional variables $z_i(k)$ denote the total load on machine i in terms of the deterministic effective sizes β_k. Lemma 4.3.1 shows that the following feasibility LP is a valid relaxation:

$$\sum_{i=1}^{m} y_{ij} = 1, \ \forall j \in [n], \tag{4.5}$$

$$z_i(k) - \sum_{j=1}^{n} \beta_k(X'_{ij}) \cdot y_{ij} = 0, \ \forall i \in [m], \ \forall k = 1, 2, \cdots m, \tag{4.6}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \mathbb{E}[X''_{ij}] \cdot y_{ij} \leq 2, \tag{4.7}$$

$$\sum_{i \in K} z_i(k) \leq b \cdot k, \ \forall K \subseteq [m] \text{ with } |K| = k, \ \forall k = 1, 2, \cdots m, \tag{4.8}$$

$$y_{ij}, z_i(k) \geq 0, \quad \forall i, j, k. \tag{4.9}$$

In the above LP, $b = O(1)$ denotes the constant multiplying k in the right-hand-side of (4.4).

Although this LP has an exponential number of constraints (because of (4.8)), we can give an efficient separation oracle. Indeed, consider a candidate solution $(y_{ij}, z_i(k))$, and some integer k; suppose we want to verify (4.8) for sets K with $|K| = k$. We just need to look at the k machines with the highest $z_i(k)$ values and check that the sum of $z_i(k)$ for these machines is at most bk. So, using the Ellipsoid method we can assume that we have an optimal solution (y, z) for this LP in polynomial time. We can summarize this in the following proposition:

Proposition 4.3.2. *The linear program (4.5)–(4.9) can be solved in polynomial time. Moreover, if it is infeasible, then the optimal expected makespan is more than* 1.

4.3.2 The Rounding

Intuition In order to get some intuition about the rounding algorithm, let us first consider the case when the assignment variables y_{ij} are either 0 or 1, i.e., the LP solution assigns each job integrally to a machine. In order to bound the expected makespan of this solution, let Z_j denote the variable X_{ij}, where j is assigned to i by this solution. First consider the exceptional parts Z''_j of the random variables. Constraint (4.7) implies that $\sum_j \mathbb{E}[Z''_j]$ is at most 2. Even if the solution assigns all of these jobs to the same machine, the contribution of these jobs to the expected makespan is at most $\sum_j \mathbb{E}[Z''_j]$, and hence at most 2. Thus, we need only worry about the truncated Z'_j variables.

Now for a machine i and integer $k \in [m]$, let $z_i(k)$ denote the sum of the effective sizes $\beta_k(Z'_j)$ for the truncated r.v.s assigned to i. We can use Lemma 4.2.4 to infer that if $z_i(m) = \sum_{j \text{ assigned to } i} \beta_m(Z_j) \leq b$, then the probability that these jobs have total size at most $b + 2$ is at least $1 - 1/m^2$. Therefore, if $z_i(m) \leq b$ for all machines $i \in [m]$, then by a trivial union bound the probability that makespan is more than $b + 2$ is at most $1/m$. Unfortunately, we are not done. All we know from constraint (4.8) is that the *average* value of $z_i(m)$ is at most b (the average being taken over the m machines). However, there is a clean solution. It follows that there is at least one machine i for which $z_i(m)$ is at most b, and so the expected load on such machines stays $O(1)$ with high probability. Now we can ignore such machines, and look at the residual problem. We are left with $k < m$ machines. We recurs on this sub-problem (and use the constraint (4.8) for the remaining set of machines). The overall probability that the load exceeds $O(1)$ on any machine can then be bounded by applying a union bound.

Next, we address the fact that y_{ij} may be not be integral. It seems very difficult to round a fractional solution while respecting all the (exponentially many) constraints in (4.8). Instead, we observe that the expected makespan analysis (outlined above) only utilizes a linear number of constraints in (4.8), although this subset is not known *a priori*. Moreover, for each machine i, the above analysis only uses $z_i(k)$ for a *single* value of k (say k_i). Therefore, it suffices to find an 2 integral assignment that bounds the load of each machine i in terms of effective sizes β_{k_i}. It turns out that this problem is precisely an instance of the Generalized Assignment Problem (GAP), for which we utilize the algorithm from [117].

The rounding procedure We now describe the iterative procedure formally in Algorithm 8. Assume we have an LP solution $\{y_{ij}\}_{i\in[m],j\in[n]}, \{z_i(k)\}_{i,k\in[m]}$.

Recall that in an instance $\mathcal{I}$ of GAP, we are given a set of m machines and n jobs. For each job j and machine i, we are given two quantities: p_{ij} is the processing time of j on machine i, and c_{ij} is the cost of assigning j to i. We are also given a makespan bound b.

Input : LP solution y, z.
Output: Integer assignment of jobs to machines.

Initialize $\ell \leftarrow m$, $L \leftarrow [m]$, $c_{ij} \leftarrow \mathbb{E}[X''_{ij}]$;
while $(\ell > 0)$ **do**
  Set $L' \leftarrow \{i \in L : z_i(\ell) \leq b\}$. Machines in L' are said to be in *class* ℓ;
  Set $p_{ij} \leftarrow \beta_\ell(X'_{ij})$ for all $i \in L'$ and $j \in [n]$;
  Set $L \leftarrow L \setminus L'$ and $\ell = |L|$;
end
Define a deterministic instance $\mathcal{I}$ of the GAP as follows: the set of jobs and machines remains unchanged. For each job j and machine i, define p_{ij} and c_{ij} as above, see Figure 4.1 for an example. The makespan bound is b. Use the algorithm of Shmoys and Tardos [117] to find an assignment of jobs to machines. Output this solution.;

Algorithm 8: Stochastic load balancing rounding procedure

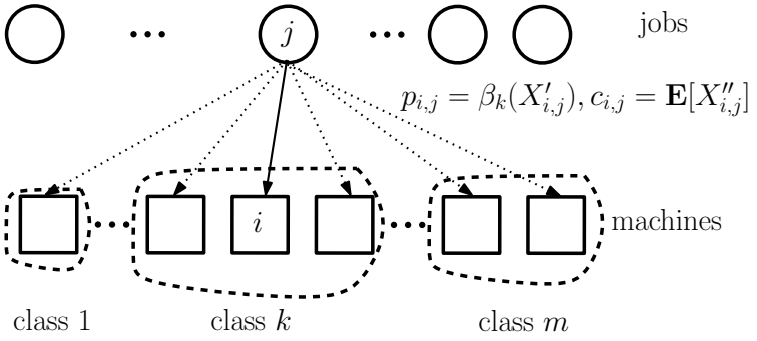

Figure 4.1: Deterministic instance example

Our goal is to assign jobs to machines to minimize the total cost of assignment, subject to the total processing time of jobs assigned to each machine being at most b. If the natural LP relaxation has optimal value $C^\star$ then the algorithm in [117] finds an assignment with cost at most $C^\star$ and makespan is at most $B + \max_{i,j} p_{ij}$ in polynomial-time.

4.3.3 The Analysis

We begin with some simple observations:

Observation 4.3.3. *The above rounding procedure terminates in at most m iterations. Furthermore, for any $1 \leq \ell \leq m$, there are at most ℓ machines of class at most ℓ.*

Proof. The first statement follows from the fact that $L' \neq \emptyset$ in each iteration. To see this, consider any iteration involving a set L of ℓ machines. The LP constraint (4.8) for L implies that $\sum_{i \in L} z_i(k) \leq b \cdot \ell$, which means there is some $i \in L$ with $z_i(\ell) \leq b$, i.e., $L' \neq \emptyset$. The second statement follows from the rounding procedure: the machine classes only decrease

over the run of the algorithm, and the class assigned to any unclassified machine equals the current number of unclassified machines. □

Observation 4.3.4. *The solution y is a feasible fractional solution to the natural LP relaxation for the* GAP *instance $\mathcal{I}$. This solution has makespan at most b and fractional cost at most 2. The rounding algorithm of Shmoys and Tardos [117] yields an assignment with makespan at most $b+1$ and cost at most 2 for the instance $\mathcal{I}$.*

Proof. Recall that the natural LP relaxation is the following:

$$\begin{aligned}
&\min \sum_{ij} c_{ij} y_{ij} \\
&\text{s.t.} \sum_{j} p_{ij} y_{ij} \leq b, && \forall i, && (4.10)\\
&\sum_{i} y_{ij} = 1, && \forall j, && (4.11)\\
&y_{ij} = 0, && \forall j \text{ s.t. } p_{ij} > 1, && (4.12)\\
&y \geq 0.
\end{aligned}$$

Firstly, note that by (4.5), y is a valid fractional assignment that assigns each job to one machine, which satisfies (4.11).

Next we show (4.10), i.e., that $\max_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \cdot y_{ij} \leq b$. This follows from the definition of the deterministic processing times p_{ij}. Indeed, consider any machine $i \in [m]$. Let ℓ be the class of machine i, and L be the subset of machines in the iteration when i is assigned class ℓ. This means that $p_{ij} = \beta_\ell(X'_{ij})$ for all $j \in [n]$. Also, because machine $i \in L'$, we have $z_i(\ell) = \sum_{j=1}^{n} \beta_\ell(X'_{ij}) \cdot y_{ij} \leq b$. So we have $\sum_{j=1}^{n} p_{ij} \cdot y_{ij} \leq b$ for each machine $i \in [m]$.

Finally, since the random variable X'_{ij} is at most 1, we get that for any parameter $k \geq 1$, $\beta_k(X'_{ij}) \leq 1$; this implies that $p_{max} := \max_{i,j} p_{ij} \leq 1$ and hence the constraints (4.12) are vacuously true. Finally, by (4.7), the objective is $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot y_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} \mathbb{E}[X''_{ij}] \cdot y_{ij} \leq 2$. Therefore the rounding algorithm [117] yields an assignment of makespan at most $b + p_{\max} \leq b + 1$, and of cost at most 2. □

In other words, if J_i be the set of jobs assigned to machine i by our algorithm, Observation 4.3.4 shows that this assignment has the following properties (let ℓ_i denote the class of machine i):

$$\sum_{i=1}^{m} \sum_{j \in J_i} \mathbb{E}[X''_{ij}] \;\leq\; 2, \quad \textit{and} \tag{4.13}$$

$$\sum_{j \in J_i} \beta_{\ell_i}(X'_{ij}) \;\leq\; b+1, \qquad \forall i \in [m]. \tag{4.14}$$

Note that we ideally wanted to give an assignment that satisfied (4.3)–(4.4), but instead of giving a bound for all sets of machines, we give just the bound on the β_{ℓ_i} values of the jobs for each machine i. The next lemma shows this is enough.

Lemma 4.3.5. *The expected makespan of the assignment $\{J_i\}_{i \in [m]}$ is at most $4b + 10$.*

Proof. Let I^{hi} denote the index set of machines of class 3 or higher. Observation 4.3.3 shows that there are at most 3 machines which are not in I^{hi}. For a machine i, let $T_i = \sum_{j \in J_i} X'_{ij}$ denote the total load due to truncated sizes of jobs assigned to it. Clearly, the makespan is bounded by

$$\max_{i \in I^{\text{hi}}} T_i + \sum_{i \notin I^{\text{hi}}} T_i + \sum_{i=1}^{m} \sum_{j \in J_i} X''_{ij}.$$

The expectation of third term is at most two, using (4.13). We now bound the expectation of the second term above. A direct application of Jensen's inequality (Theorem 4.2.7) for concave functions shows that $\beta_k(X) \geq \mathbb{E}[X]$ for any random variable X and any $k \geq 1$. Then applying inequality (4.14) shows that $\mathbb{E}[T_i] \leq b+1$ for any machine i. Therefore, the expected makespan of our solution is at most

$$\mathbb{E}\left[\max_{i \in I^{\text{hi}}} T_i\right] + 3(b+1) + 2. \tag{4.15}$$

It remains to bound the first term above.

Observation 4.3.6. *For any machine i, $\Pr\left[\sum_{j \in J_i} X'_{ij} > b + 1 + \alpha\right] \leq \ell_i^{-\alpha}$ for all $\alpha \geq 0$.*

Proof. Inequality (4.14) for machine i shows that $\sum_{j \in J_i} \beta_{\ell_i}(X'_{ij}) \leq b+1$. But recalling the definition of the effective size (Definition 4.2.3), the result follows from Lemma 4.2.4. □

Now we can bound the probability of any machine in I^{hi} having a high makespan.

Lemma 4.3.7. *For any $\alpha > 2$, $\Pr\left[\max_{i \in I^{\text{hi}}} T_i > b + 1 + \alpha\right] \leq 2^{2-\alpha}/(\alpha - 2)$.*

Proof. Using a union bound, we get

$$\begin{aligned}\Pr\left[\max_{i\in I^{\text{hi}}} T_i > b+1+\alpha\right] &\le \sum_{\ell=3}^{m} \sum_{i:\ell_i=\ell} \Pr\left[T_i > b+1+\alpha\right] \\ &\le \sum_{\ell=3}^{m} \ell^{-\alpha} \cdot (\text{\# of class } \ell \text{ machines}) \\ &\le \sum_{\ell=3}^{m} \ell^{-\alpha+1} \;\le\; \int_{x=2}^{\infty} x^{-\alpha+1} dx \;=\; \frac{2^{-\alpha+2}}{\alpha-2}.\end{aligned}$$

The first inequality uses a trivial union bound, the second uses Observation 4.3.6 above, and the third inequality is by Observation 4.3.3. □

Using the above lemma, we get

$$\begin{aligned}\mathbb{E}[\max_{i\in I^{\text{hi}}} T_i] &= (b+4) + \int_{\alpha=3}^{\infty} \Pr[\max_{i\in I^{\text{hi}}} T_i > b+1+\alpha] d\alpha \\ &\le\; (b+4) + \int_{\alpha=3}^{\infty} 2^{2-\alpha} d\alpha \;\le\; b+5.\end{aligned}$$

Inequality (4.15) now shows that the expected makespan is at most $(b+5)+3(b+1)+2$. □

This completes the proof of Theorem 3.1.2.

4.4 Budgeted Makespan Minimization

We now consider a generalization of the STOCMAKESPAN problem, called BUDGETSTOCMAKESPAN, where each job j also has *reward* $r_j \ge 0$. We are required to schedule some subset of jobs whose total reward is at least some target value R. The objective, again, is to minimize the expected makespan. If the target $R = \sum_{j\in[n]} r_j$ then we recover the STOCMAKESPAN problem. We show:

Theorem 4.1.2 (Budgeted makespan minimization result)**.** *There is an $O(1)$-approximation algorithm for the budgeted makespan minimization problem on unrelated machines.*

Naturally, our algorithm/analysis will build on the ideas developed in Section 4.3, but we will need some new ideas to handle the fact that only a subset of jobs need to be scheduled. As in the case of STOCMAKESPAN problem, we can formulate a suitable LP relaxation. A similar rounding procedure reduces the stochastic problem to a deterministic problem, which we call BUDGETED GAP. An instance of BUDGETED GAP is similar to that of GAP, besides the additional requirement that jobs have rewards and we are required

to assign jobs of total reward at least some target R. Rounding the natural LP relaxation for BUDGETED GAP turns out to be non-trivial. Indeed, using ideas from [117], we reduce this rounding problem to rounding a fractional matching solution with additional constraints, and solve the latter using polyhedral properties of bipartite matching polyhedra.

As before, using a binary-search scheme (and by scaling down the sizes), we can assume that we need to either (i) find a solution of expected makespan $O(1)$, or (ii) prove that the optimal value is more than 1. We use a natural LP relaxation which has variables y_{ij} for each job j and machine i. The LP includes the constraints (4.6)-(4.9) for the base problem, and in addition it has the following two constraints:

$$\sum_{i=1}^{m} y_{ij} \leq 1, \qquad \forall j = 1, \cdots n, \tag{4.16}$$

$$\sum_{j=1}^{n} r_j \cdot \sum_{i=1}^{m} y_{ij} \geq R. \tag{4.17}$$

The first constraint (4.16) replaces constraint (4.5) and says that not all jobs need to be assigned. The second constraint (4.17) ensures that the assigned jobs have total reward at least the target R. For technical reasons that will be clear later, we also perform a preprocessing step: for i, j pairs where $\mathbb{E}[X''_{ij}] > 2$, we force the associated y_{ij} variable to zero. Note that by Lemma 4.3.1, this variable fixing is valid for any integral assignment that has expected makespan at most one (in fact, we have $\sum_i \sum_j \mathbb{E}[X''_{ij}] \cdot y_{ij} \leq 2$ for such an assignment). As in Section 4.3.1 this LP can be solved in polynomial time via the ellipsoid method. If the LP is infeasible we get a proof (using Lemma 4.3.1) that the optimal expected makespan is more than one. Hence we may assume the LP is feasible, and proceed to round the solution along the lines of Section 4.3.2.

Recall that the rounding algorithm in Section 4.3.2 reduces the fractional LP solution to an instance of the generalized assignment problem (GAP). Here, we will use a further generalization of GAP, which we call BUDGETED GAP. An instance of this problem is similar to an instance of GAP. We are given m machines and n jobs, and for each job j and machine i, we are given the processing time p_{ij} and the associated assignment cost c_{ij}. Now each job j has a reward r_j, and there are two "target" parameters: the reward target R and the makespan target B. We let p_{max} and c_{max} denote the maximum values of processing time and cost respectively. A solution must assign a subset of jobs to machines such that the total reward of assigned jobs is at least R. Moreover, as in the case of GAP, the goal is to minimize the total assignment cost subject to the condition that the makespan is at most B. Our main technical theorem of this section shows how to round an LP relaxation of this

BUDGETED GAP problem.

Theorem 4.4.1. *There is a polynomial-time rounding algorithm for* BUDGETED GAP *that given any fractional solution to the natural LP relaxation of cost C^*, produces an integer solution having total cost at most $C^* + c_{max}$ and makespan at most $B + 2p_{max}$.*

Before we prove this theorem, let us use it to solve the BUDGETSTOCMAKESPAN, and prove Theorem 4.1.2. Proceeding as in Section 4.3.2, we perform Steps 1-2 from the rounding procedure. This rounding gives us values p_{ij} and c_{ij} for each job/machine pair. Now, instead of reducing to an instance of GAP, we reduce to an instance $\mathcal{I}'$ of BUDGETED GAP. The instance $\mathcal{I}'$ has the same set of jobs and machines as in the original BUDGETSTOCMAKESPAN instance $\mathcal{I}$. For each job j and machine i, the processing time and the assignment cost are given by p_{ij} and c_{ij} respectively. Furthermore, the reward r_j for job j, and the reward target R are same as those in $\mathcal{I}$. The makespan bound $b = O(1)$ (as in (4.8)). It is easy to check that the fractional solution y_{ij} is a feasible fractional solution to the natural LP relaxation for $\mathcal{I}'$ (given below), and the assignment cost of this fractional solution is at most 2. Applying Theorem 4.4.1 yields an assignment $\{J_i\}_{i=1}^m$, which has the following properties:

- The makespan is at most $b + 2 = O(1)$; i.e., $\sum_{j \in J_i} p_{ij} \leq b + 2p_{max} \leq b + 2$ for each machine i. Here we used the fact that $p_{max} \leq 1$.
- The cost of the solution, $\sum_{i=1}^m \sum_{j \in J_i} c_{ij}$, is at most 4. This uses the fact that the LP cost $C^* = \sum c_{ij} \cdot y_{ij} \leq 2$ and $c_{max} \leq 2$ by the preprocessing on the $\mathbb{E}[X''_{ij}]$ values.
- The total reward for the assigned jobs, $\sum_{j \in \cup_i J_i} r_j$, is at least R.

Now arguing exactly as in Section 4.3.2, the first two properties imply that the expected makespan is $O(1)$. The third property implies the total reward of assigned jobs is at least R, and completes the proof of Theorem 4.1.2.

4.4.1 Proof of Theorem 4.4.1

Proof of Theorem 4.4.1. Let $\mathcal{I}$ be an instance of BUDGETED GAP as described above. The natural LP relaxation for this problem is as follows:

$$\begin{aligned} \min \quad & \textstyle\sum_{ij} c_{ij} y_{ij} \\ & \textstyle\sum_j p_{ij} y_{ij} \leq B, \qquad \forall i, \end{aligned} \tag{4.18}$$

$$\sum_i y_{ij} \leq 1, \qquad \forall j, \tag{4.19}$$

$$\sum_{i,j} y_{ij} r_j \geq R, \tag{4.20}$$

$$y_{ij} = 0, \qquad \forall j \text{ s.t. } p_{ij} > b, \tag{4.21}$$

$$y \geq 0.$$

Let $\{y_{ij}\}$ denote an optimal fractional solution to this LP. For each machine i, let $t_i := \lceil \sum_j y_{ij} \rceil$ be the (rounded) fractional assignment to machine i. Using the algorithm in Theorem 2.1 of [117], we obtain a bipartite graph $G = (V_1 \cup V_2, E)$ and a fractional matching y' in G, where:

- $V_1 = [n]$ is the set of jobs and V_2 (indexed by $i' = 1, \ldots, m'$) consists of t_i copies for each machine $i \in [m]$. The cost $c_{i'j} = c_{ij}$ for any job $j \in [n]$ and any machine-copy i' of machine $i \in [m]$.
- for each job $j \in [n]$ we have $\sum_{i=1}^{m'} y'_{ij} = \sum_{i=1}^{m} y_{ij} \leq 1$ for all $j \in [n]$.
- the reward $\sum_{j=1}^{n} r_j \sum_{i=1}^{m'} y'_{ij} \geq R$ and the cost $\sum_{i'=1}^{m'} \sum_{j=1}^{n} c_{i'j} y'_{ij} = C^*$ are same as for y.
- the jobs of V_1 incident to copies of any machine $i \in [m]$ can be divided into (possibly overlapping) groups $H_{i,1}, \cdots H_{i,t_i}$ where

$$\sum_{j \in H_{i,g}} y'_{ij} = 1 \text{ for all } 1 \leq g \leq t_i - 1, \quad \text{and} \quad \sum_{j \in H_{i,t_i}} y'_{ij} \leq 1,$$

  and for any two consecutive groups $H_{i,g}$ and $H_{i,g+1}$ we have $p_{ij} \geq p_{ij'}$ for all $j \in H_{i,g}$ and $j' \in H_{i,g+1}$. Informally, this is achieved by sorting the jobs in non-increasing order of p_{ij}, and assigning the k^{th} unit of $\sum_j y_{ij}$ to the k^{th} machine-copy for each $1 \leq k \leq t_i$.

A crucial property of this construction shown in [117] is that any assignment that places at most one job on each machine-copy has makespan at most $B + p_{max}$ in the original instance $\mathcal{I}$ (where for every machine i, we assign to it all the jobs which are assigned to a copy of i in this integral assignment). We will use the following simple extension of this property: if the assignment places *two* jobs on one machine-copy and at most one job on all other machine-copies, then it has makespan at most $B + 2p_{max}$ in the instance $\mathcal{I}$.

Observe that the solution y' is a feasible solution to the following LP with variables $\{z_{ij}\}_{(i,j) \in E}$.

$$\min \quad \sum_{ij} c_{ij} z_{ij} \tag{4.22}$$

$$\sum_{i\in[m']:(ij)\in E} z_{ij} \leq 1, \qquad \forall j \in [n], \tag{4.23}$$

$$\sum_{j\in[n]:(ij)\in E} z_{ij} \leq 1, \qquad \forall i \in [m'], \tag{4.24}$$

$$\sum_{(ij)\in E} r_j \cdot z_{ij} \geq R, \tag{4.25}$$

$$z \geq 0. \tag{4.26}$$

So the optimal value of this auxiliary LP is at most C^*. We note that its integrality gap is unbounded even when c_{max} is small; see the example below. So this differs from [117] for the usual GAP where the corresponding LP (without (4.25)) is actually integral. However, we show below how to obtain a good integral solution that violates the matching constraint for just a *single* machine-copy in V_2.

Indeed, let z be an optimal solution to this LP: so $c^T z \leq C^*$. Note that the feasible region of this LP is just the bipartite-matching polytope on G intersected with one extra linear constraint (4.25) that corresponds to the total reward being at least R. So z must be a convex combination of two adjacent extreme points of the bipartite-matching polytope. Using the integrality and adjacency properties (see [15]) of the bipartite-matching polytope, it follows that $z = \lambda_1 \cdot \mathbf{1}_{M_1} + \lambda_2 \cdot \mathbf{1}_{M_2}$ where:

- $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$.
- M_1 and M_2 are integral matchings in G.
- The symmetric difference $M_1 \oplus M_2$ is a single cycle or path.

For any matching M let $c(M)$ and $r(M)$ denote its total cost and reward respectively. Without loss of generality, we assume that $r(M_1) \geq r(M_2)$. If $c(M_1) \leq c(M_2)$ then M_1 is itself a solution with reward at least R and cost at most C^*. So we assume $c(M_1) > c(M_2)$ below.

If $M_1 \oplus M_2$ is a cycle then we output M_2 as the solution. Note that the cycle must be an even cycle: so the set of jobs assigned by M_1 and M_2 is identical. As the reward function is only dependent on the assigned jobs (and not the machines used in the assignment) it follows that $r(M_2) = r(M_1) \geq R$. So M_2 is indeed a feasible solution and has cost $c(M_2) \leq c^T z \leq C^*$.

Now consider the case that $M_1 \oplus M_2$ is a path. If the set of jobs assigned by M_1 and M_2 are the same then M_2 is an optimal integral solution (as above). The only remaining case is that M_1 assigns one additional job (say j^* to i^*) over the jobs in M_2. Then we return the solution $M_2 \cup \{(j^*, i^*)\}$. Note that this is not a feasible matching. But the only infeasibility is at machine-copy i^* which may have two jobs assigned; all other machine-copies have

at most one job. The reward of this solution is $r(M_1) \geq R$. Moreover, its cost is at most $c(M_2) + c_{i^*j^*} \leq C^* + c_{max}$.

Now using this (near-feasible) assignment gives us the desired cost and makespan bounds, and completes the proof of Theorem 4.4.1. □

Integrality Gap for Budgeted Matching LP. Here we show that the LP (4.22)–(4.26) used in the algorithm for budgeted GAP has an unbounded integrality gap, even if we assume that $c_{max} \ll OPT$. The instance consists of n jobs and $m = n - 1$ machines. For each machine $i \in [m]$, there are two incident edges in E: one to job i (with cost 1) and the other to job $i+1$ (with cost n). So E is the disjoint union of two machine-perfect matchings M_1 (of total cost m) and M_2 (of total cost mn). (See also Figure 4.4.1 as an illustration) The rewards are

$$r_j = \begin{cases} 1, & \text{if } j = 1, \\ 4, & \text{if } 2 \leq j \leq n-1, \\ 2, & \text{if } j = n. \end{cases}$$

and the target $R = 4(n-2) + 1 + \epsilon$ where $\epsilon \to 0$. Note that the only (minimal) integral solution involves assigning the jobs $\{2, 3, \cdots, n\}$ which has total reward $4(n-2) + 2$. This solution has cost $OPT = mn$ and corresponds to matching M_2. On the other hand, consider the fractional solution $z = \epsilon \mathbf{1}_{M_2} + (1-\epsilon)\mathbf{1}_{M_1}$. This is clearly feasible for the matching constraints, and its reward is $\epsilon(4(n-2)+2) + (1-\epsilon)(4(n-2)+1) = R$. So z is a feasible fractional solution. The cost of solution z is at most $m + \epsilon(mn) \ll OPT$.

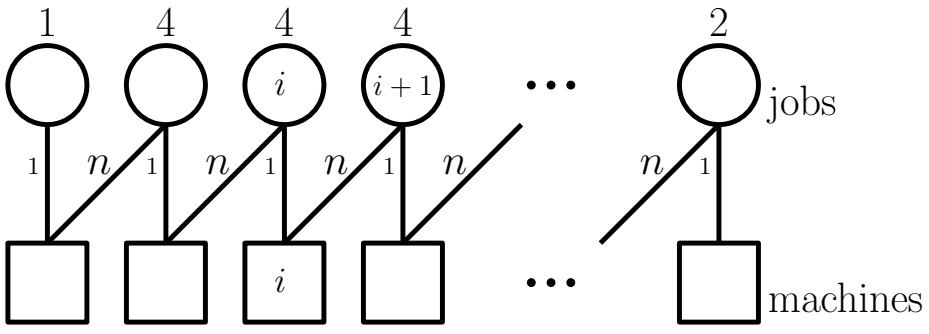

Figure 4.2: Example for the matching instance

4.5 ℓ_q-norm Objectives

In this section, we prove Theorem 4.1.3. Given an assignment $\{J_i\}_{i=1}^m$, the load L_i on machine i is the r.v. $L_i := \sum_{j \in J_i} X_{ij}$. Our goal is to find an assignment to minimize the expected q-norm of the load vector $\mathbf{L} := (L_1, L_2, \ldots, L_m)$. Recall that the makespan is $\|\mathbf{L}\|_\infty$ which is approximated within constant factors by $\|\mathbf{L}\|_{\log m}$. So the q-norm problem is a generalization of STOCMAKESPAN. Our main result here is:

Theorem 4.1.3 (Stochastic q-norm minimization result)**.** *There is an $O(\frac{q}{\log q})$-approximation algorithm for the stochastic q-norm minimization problem on unrelated machines.*

We begin by assuming that we know the optimal value M of the q-norm. Our approach parallels that for the case of minimizing the expected makespan, with some changes. In particular, the main steps are: (1) find valid inequalities satisfied by any assignment for which $\mathbb{E}[\|\mathbf{L}\|_q] \leq M$, (2) reduce the problem to a deterministic assignment problem for which any feasible solution satisfies the valid inequalities above, (3) solve the deterministic problem by writing a convex programming relaxation, and give a rounding procedure for a fractional solution to this convex program, and (4) prove that the resulting assignment of jobs to machines has small q-norm of the load vector.

4.5.1 Useful Bounds

We start with stating some valid inequalities satisfied by any assignment $\{J_i\}_{i=1}^m$. For each $j \in [n]$ define $Y_j = X_{ij}$ where $j \in J_i$. By definition of M, we know that

$$\mathbb{E}\left[\left(\sum_{i=1}^{m}(\sum_{j\in J_i} Y_j)^q\right)^{1/q}\right] \leq M. \tag{4.27}$$

As in Section 4.3, we split each random variable Y_j into two parts: truncated $Y'_j = Y_j \cdot \mathbf{I}_{Y_j \leq M}$ and exceptional $Y''_j = Y_j \cdot \mathbf{I}_{Y_j > M}$. The claim below is analogous to (4.3), and states that the total expected size of the exceptional parts cannot be too large.

Claim 4.5.1. *For any schedule satisfying* (4.27)*, we have $\sum_{j=1}^n \mathbb{E}[Y''_j] \leq 2M$.*

Proof. Suppose for a contradiction that $\sum_{j=1}^n \mathbb{E}[Y''_j] > 2M$. We have Lemma 4.2.2 implies $\mathbb{E}[\max_{j=1}^n Y''_j] > M$. Now using the monotonicity of norms and the fact that $Y''_j \leq Y_j$,

$$\max_{j=1}^{n} Y''_j \leq \|(Y''_1, \cdots, Y''_n)\|_q \leq \left(\sum_{i=1}^{m}(\sum_{j\in J_i} Y_j)^q\right)^{1/q},$$

which contradicts (4.27). □

Our next two bounds deal with the truncated r.v.s Y'_j. The first one states that if we replace Y'_j by its expectation $\mathbb{E}[Y'_j]$, the q-norm of this load vector of expectations cannot exceed M. The second bound states that the expected q^{th} moment of the vector $(Y'_j)_{j=1}^n$ is bounded by a constant times M^q.

Claim 4.5.2. *For any schedule satisfying* (4.27) *we have*

$$\sum_{i=1}^{m} \left(\sum_{j \in J_i} \mathbb{E}[Y'_j] \right)^q \leq M^q.$$

Proof. Since the function

$$f(Y'_1, \cdots Y'_n) := \left(\sum_{i=1}^{m} (\sum_{j \in J_i} Y'_j)^q \right)^{1/q}$$

is a norm and hence convex, Jensen's inequality (Theorem 4.2.7) implies $\mathbb{E}[f(Y'_1, \cdots Y'_n)] \geq f(\mathbb{E}[Y'_1], \cdots \mathbb{E}[Y'_n])$. Raising both sides to the q^{th} power and using (4.27), the claim follows. □

Claim 4.5.3. *Let* $\alpha = 2^{q+1} + 8$. *For any schedule satisfying* (4.27) *we have*

$$\sum_{j=1}^{n} \mathbb{E}[(Y'_j)^q] \leq \alpha \cdot M^q.$$

Proof. Define $Z := \sum_{j=1}^{n} (Y'_j)^q$ as the quantity of interest. Observe that it is the sum of independent $[0, M^q]$ bounded random variables. Since $q \geq 1$ and the r.v.s are non-negative, $Z \leq \sum_{i=1}^{m} (\sum_{j \in J_i} Y'_j)^q$. Thus (4.27) implies $\mathbb{E}[Z^{1/q}] \leq M$. However, now Jensen's inequality cannot help upper-bound $\mathbb{E}[Z]$.

Instead we use a second-moment calculation. To reduce notation let $Z_j := (Y'_j)^q$, so $Z = \sum_{j=1}^{n} Z_j$. The variance of Z is $var(Z) = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 \leq \sum_{j=1}^{n} \mathbb{E}[Z_j^2] \leq M^q \cdot \mathbb{E}[Z]$ as each Z_j is $[0, M^q]$ bounded. By Chebyshev's inequality,

$$\Pr\left[Z < \frac{\mathbb{E}[Z]}{2} - 4M^q \right] \leq \frac{var(Z)}{(\mathbb{E}[Z]/2 + 4M^q)^2} \leq \frac{var(Z)}{(\mathbb{E}[Z]/2) \cdot 4M^q} \leq \frac{2M^q \cdot E[Z]}{\mathbb{E}[Z] \cdot 4M^q} \leq \frac{1}{2}.$$

This implies

$$E[Z^{1/q}] \geq \frac{1}{2} \left(\frac{\mathbb{E}[Z]}{2} - 4M^q \right)^{1/q}.$$

Using the bound $\mathbb{E}[Z^{1/q}] \leq M$ from above, we now obtain $\mathbb{E}[Z] \leq 2 \cdot ((2M)^q + 4M^q)$ as desired. □

In the next sections, we show that the three bounds above are enough to get a meaningful lower bound on the optimal q-norm of load.

4.5.2 Reduction to a Deterministic Scheduling Problem

We now formulate a surrogate deterministic scheduling problem, which we call q-DETSCHED. An instance of this problem has n jobs and m machines. For each job j and machine i, there is a processing time p_{ij} and two costs c_{ij} and d_{ij}. There are also bounds C and D on the two cost functions respectively. The goal is to find an assignment of jobs to machines that minimizes the q-norm of the machine loads subject to the constraint that the total c-cost and d-cost of the assignments are at most C and D respectively. We now show how to convert an instance $\mathcal{I}_{stoc}$ of the (stochastic) expected q-norm minimization problem to an instance $\mathcal{I}_{det}$ of the (deterministic) q-DETSCHED problem.

Suppose $\mathcal{I}_{stoc}$ has m machines and n jobs, with random variables X_{ij} for each machine i and job j. As before, let $X'_{ij} = X_{ij} \cdot \mathbf{I}_{X_{ij} \leq M}$ and $X''_{ij} = X_{ij} \cdot \mathbf{I}_{X_{ij} > M}$ denote the truncated and exceptional parts of each random variable X_{ij} respectively. Then instance $\mathcal{I}_{det}$ has the same set of jobs and machines as those in $\mathcal{I}$. Furthermore, define

- the processing time $p_{ij} = \mathbb{E}[X'_{ij}]$,
- the c-cost $c_{ij} = \mathbb{E}[X''_{ij}]$ with bound $C = 2M$,
- the d-cost $d_{ij} = \mathbb{E}[(X'_{ij})^q]$ with bound $D = \alpha \cdot M^q$.

Observation 4.5.4. *If there is any schedule of expected q-norm at most M in the instance $\mathcal{I}_{stoc}$, then optimal value of the instance $\mathcal{I}_{det}$ is at most M.*

Proof. This follows directly from Claims 4.5.1, 4.5.2 and 4.5.3. □

4.5.3 Approximation Algorithm for q-DETSCHED

Our approximation algorithm for the q-DETSCHED problem is closely based on the algorithm for unrelated machine scheduling to minimize ℓ_q-norms [14]. We show:

Theorem 4.5.5. *There is a polynomial-time algorithm that given any instance $\mathcal{I}_{det}$ of q-DETSCHED, finds a schedule with (i) q-norm of processing times at most $2^{1+2/q} \cdot OPT(\mathcal{I}_{det})$, (ii) c-cost at most $3C$ and (iii) d-cost at most $3D$.*

Proof. We only provide a sketch as many of these ideas parallel those from [14]. Start with a convex programming "relaxation" with variables x_{ij} (for assigning job j to machine i).

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m} \ell_i^q + \sum_{ij} p_{ij}^q \cdot x_{ij} \\
\text{s.t.} \quad & \ell_i = \sum_j p_{ij} \cdot x_{ij}, \quad \forall i, \\
& \sum_i x_{ij} = 1, \quad \forall j, \\
& \sum_{ij} c_{ij} \cdot x_{ij} \leq C, \\
& \sum_{ij} d_{ij} \cdot x_{ij} \leq D.
\end{aligned}$$

This convex program can be solved to arbitrary accuracy and its optimal objective value is $V \leq 2 \cdot OPT(\mathcal{I}_{det})^q$. Let (x, ℓ) denote the optimal fractional solution below.

We now further reduce this q-norm problem to GAP. The GAP instance $\mathcal{I}_{gap}$ has the same set of jobs and machines as those in $\mathcal{I}_{det}$. For a job j and machine i, the processing time remains p_{ij}. However, the cost of assigning j to i is now $\gamma_{ij} := \frac{c_{ij}}{C} + \frac{d_{ij}}{D} + \frac{p_{ij}^q}{V}$. Furthermore, we impose a bound of ℓ_i on the total processing time of jobs assigned to each machine i (i.e., the makespan on i is constrained to be at most ℓ_i). Note that the solution x to the convex program is also a feasible fractional solution to the natural LP-relaxation for GAP with an objective function value of $\sum_{ij} \gamma_{ij} \cdot x_{ij} \leq 3$. The rounding algorithm in [117] can now be used to round x into an integral assignment $\{A_{ij}\}$ with γ-cost also at most 3, and load on each machine i being $L_i \leq \ell_i + m_i$, where m_i denotes the maximum processing time of any job assigned to machine i by this algorithm. The definition of γ and the bound on the γ-cost implies that the c-cost and d-cost of this assignment are at most $3C$ and $3D$ respectively. To bound the q-norm of processing times,

$$\begin{aligned}
\sum_{i=1}^{m} L_i^q &\leq 2^{q-1} \left(\sum_i \ell_i^q + \sum_i m_i^q \right) \leq 2^{q-1} \left(V + \sum_{ij} p_{ij}^q \cdot A_{ij} \right) \\
&\leq 2^{q-1}(V + 3V) = 2^{q+1} \cdot V.
\end{aligned}$$

Above, the first inequality uses $(a+b)^q \leq 2^{q-1}(a^q + b^q)$, and the third inequality uses the fact that $p_{ij}^q A_{ij} \leq V \cdot \gamma_{ij} A_{ij} \leq 3V$ by the bound on the γ cost. The proof is now completed by using $V \leq 2 \cdot OPT(\mathcal{I}_{det})^q$. □

4.5.4 Interpreting the Rounded Solution

Starting from an instance $\mathcal{I}_{stoc}$ of expected q-norm minimization problem, we first constructed an instance $\mathcal{I}_{det}$ of q-DETSCHED. Let $\mathcal{J} = (J_1, \ldots, J_m)$ denote the solution found by applying Theorem 4.5.5 to the instance $\mathcal{I}_{det}$. If the q-norm of processing times of this assignment (as a solution for $\mathcal{I}_{det}$) is more than $2^{1+2/q}M$ then using Observation 4.5.4 and Theorem 4.5.5, we obtain a certificate that the optimal value of $\mathcal{I}_{stoc}$ is more than M. So we assume that $\mathcal{J}$ has objective at most $2^{1+2/q}M$ (as a solution to $\mathcal{I}_{det}$). We use exactly this assignment as a solution for the stochastic problem as well. It remains to bound the expected q-norm of this assignment.

By the reduction from $\mathcal{I}_{stoc}$ to $\mathcal{I}_{det}$, and the statement of Theorem 4.5.5, we know that

$$\sum_{i=1}^{m}\sum_{j\in J_i}\mathbb{E}[X''_{ij}] = \sum_{i=1}^{m}\sum_{j\in J_i} c_{ij} \leq 6M, \tag{4.28}$$

$$\sum_{i=1}^{m}\left(\sum_{j\in J_i}\mathbb{E}[X'_{ij}]\right)^q = \sum_{i=1}^{m}(\sum_{j\in J_i} p_{ij})^q \leq 2^{q+2}\cdot M^q, \tag{4.29}$$

$$\sum_{i=1}^{m}\sum_{j\in J_i}\mathbb{E}[(X'_{ij})^q] = \sum_{i=1}^{m}\sum_{j\in J_i} d_{ij} \leq 3\alpha M^q. \tag{4.30}$$

We now derive properties of this assignment as a solution for $\mathcal{I}_{stoc}$.

Claim 4.5.6. *The expected q-norm of exceptional jobs* $\mathbb{E}[(\sum_i(\sum_{j\in J_i} X''_{ij})^q)^{1/q}] \leq 6M$.

Proof. This follows from (4.28), since the ℓ_q-norm of a vector is at most its ℓ_1-norm. □

Claim 4.5.7. *The expected q-norm of truncated jobs* $\mathbb{E}[(\sum_i(\sum_{j\in J_i} X'_{ij})^q)^{1/q}] \leq O(\frac{q}{\log q})M$.

Proof. Define random variables $Q_i := (\sum_{j\in J_i} X'_{ij})^q$, so that the q-norm of the loads is

$$Q := (\textstyle\sum_{i=1}^m Q_i)^{1/q} = (\sum_{i=1}^m(\sum_{j\in J_i} X'_{ij})^q)^{1/q}.$$

Since $f(Q_1, \cdots Q_m) = (\sum_{i=1}^m Q_i)^{1/q}$ is a concave function for $q \geq 1$, using Jensen's inequality (Theorem 4.2.7) again,

$$\mathbb{E}[Q] \leq \left(\sum_{i=1}^{m}\mathbb{E}[Q_i]\right)^{1/q}. \tag{4.31}$$

We can bound each $\mathbb{E}[Q_i]$ separately using Rosenthal's inequality (Theorem 4.2.8):

$$\mathbb{E}[Q_i] = \mathbb{E}\left[(\sum_{j\in J_i} X'_{ij})^q\right] \leq K^q\cdot\left((\sum_{j\in J_i}\mathbb{E}[X'_{ij}])^q + \sum_{j\in J_i}\mathbb{E}[(X'_{ij})^q]\right),$$

where $K = O(q/\log q)$. Summing this over all $i = 1, \ldots, m$ and using (4.29) and (4.30), we get

$$\sum_{i=1}^{m} \mathbb{E}[Q_i] \leq K^q \cdot (2^{q+2} + 3\alpha) M^q. \tag{4.32}$$

Recall from Claim 4.5.3 that $\alpha = 2^{q+1} + 8$. Now plugging this into (4.31) we obtain $\mathbb{E}[Q] \leq O(K) \cdot M$. □

Finally, using Claims 4.5.6 and Claim 4.5.7 and the triangle inequality, the expected q-norm of solution $\mathcal{J}$ is $O(\frac{q}{\log q}) \cdot M$, which completes the proof of Theorem 4.1.3.

Explicit approximation ratio Here we show the approximation ratio explicitly. By equations (4.31) and (4.32), we have the expected q-norm of truncated jobs is

$$\mathbb{E}[Q] \leq (K^q \cdot (2^{q+2} + 3\alpha) M^q)^{1/q} = (K^q \cdot (2^{q+2} + 3(2^{q+1} + 8)) M^q)^{1/q} = (10 + 3 \cdot 2^{3-q})^{1/q} 2KM.$$

And the expected q-norm of exceptional jobs is at most $6M$ by Claim 4.5.6. By the triangle inequality, the expected q-norm of solution $\mathcal{J}$ is at most $(6 + (10 + 3 \cdot 2^{3-q})^{1/q} 2K)M$. Note that for any constant $\epsilon > 0$, we can get M to ϵ close to the optimal objective value by the binary search approach. Hence the overall approximation ratio for q-norm is $(6 + (10 + 3 \cdot 2^{3-q})^{1/q} 2K)(1 + \epsilon)$, for any $\epsilon > 0$, where K is the parameter in Theorem 4.2.8.

The following known result provides a bound of the parameter K^q in Theorem 4.2.8.

Theorem 4.5.8 ([78]). *Let Z denote a random variable with Poisson distribution with parameter 1, i.e., $\Pr[Z = k] = e^{-1}/k!, k = 0, 1, 2, \ldots$. The best constant K^q in Theorem 4.2.8 is $\mathbb{E}Z^q$.*

Example For ℓ_2-norm, $K^2 \leq \mathbb{E}Z^2 = 2 \Rightarrow K = \sqrt{2}$. The overall approximation ratio for ℓ_2-norm minimization is $(6 + (10 + 3 \cdot 2^1)^{1/2} 2\sqrt{2})(1 + \epsilon) = (6 + 8\sqrt{2})(1 + \epsilon) = 17.31(1 + \epsilon)$. For ℓ_3-norm, $K^3 = 5 \Rightarrow K = \sqrt[3]{5}$. The overall approximation ratio for ℓ_3-norm minimization is $(6 + (10 + 3)^{1/3} 2\sqrt[3]{5})(1 + \epsilon) = 14.04(1 + \epsilon)$.

4.6 Conclusion

In this chapter, we obtained the first constant-factor approximation for the general case of unrelated machines. We also consider two generalizations. The first is the budgeted makespan minimization problem and the second problem involves q-norm objectives. An

interesting open problem is to extend the techniques to deal with other discrete optimization problems in stochastic setting, such as routing [33].

Credit: The results in this chapter are from "Stochastic load balancing on unrelated machines" [64], obtained jointly with Anupam Gupta, Amit Kumar, and Viswanath Nagarajan.

BIBLIOGRAPHY

[1] Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science **234**(1-2), 203–218 (2000)

[2] Adler, I., Grohe, M., Kreutzer, S.: Computing excluded minors. In: SODA, pp. 641–650 (2008)

[3] Ageev, A.A., Hassin, R., Sviridenko, M.: A 0.5-approximation algorithm for MAX DICUT with given sizes of parts. SIAM Journal on Discrete Mathematics **14**(2), 246–255 (2001)

[4] Ageev, A.A., Sviridenko, M.: Approximation algorithms for maximum coverage and max cut with given sizes of parts. In: IPCO, vol. 1610, pp. 17–30 (1999)

[5] Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: A general approach to online network optimization problems. ACM Transactions on Algorithms **2**(4), 640–660 (2006)

[6] Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem. SIAM Journal on Computing **39**(2), 361–370 (2009)

[7] Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: Constant integrality gap lp formulations of unsplittable flow on a path. In: IPCO, pp. 25–36 (2013)

[8] Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. Journal of the ACM **44**(3), 486–504 (1997)

[9] Awerbuch, B., Azar, Y., Grove, E.F., Kao, M., Krishnan, P., Vitter, J.S.: Load balancing in the l_p norm. In: FOCS, pp. 383–391 (1995)

[10] Awerbuch, B., Azar, Y., Plotkin, S.: Throughput-competitive on-line routing. In: FOCS, pp. 32–40 (1993)

[11] Azar, Y., Bhaskar, U., Fleischer, L.K., Panigrahi, D.: Online mixed packing and covering. In: SODA (2013)

[12] Azar, Y., Buchbinder, N., Chan, T.H., Chen, S., Cohen, I.R., Gupta, A., Huang, Z., Kang, N., Nagarajan, V., Naor, J., Panigrahi, D.: Online algorithms for covering and packing problems with convex objectives. In: FOCS, pp. 148–157 (2016)

[13] Azar, Y., Cohen, I.R., Panigrahi, D.: Online covering with convex objectives and applications. CoRR **abs/1412.3507** (2014)

[14] Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. In: STOC, pp. 331–337 (2005)

[15] Balinski, M.L., Russakoff, A.: On the assignment polytope. SIAM Review **16**(4), 516 – 525 (1974)

[16] Bansal, N., Buchbinder, N., Naor, J.: Randomized competitive algorithms for generalized caching. SIAM Journal on Computing **41**(2), 391–414 (2012)

[17] Bansal, N., Pruhs, K.: Server scheduling to balance priorities, fairness, and average quality of service. SIAM Journal on Computing **39**(7), 3311–3335 (2010)

[18] Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Operations Research **36**(3), 493–513 (1988)

[19] Bartal, Y., Fiat, A., Leonardi, S.: Lower bounds for on-line graph problems with application to on-line circuit and optical routing. SIAM Journal on Computing **36**(2), 354–393 (2006)

[20] Bateni, M., Charikar, M., Guruswami, V.: MaxMin allocation via degree lower-bounded arborescences. In: STOC, pp. 543–552 (2009)

[21] Ben-David, S., Borodin, A., Karp, R., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica **11**(1), 2–14 (1994)

[22] Bienstock, D., Özbay, N.: Tree-width and the Sherali-Adams operator. Discrete Optimization **1**(1), 13–21 (2004)

[23] Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media (2011)

[24] Blum, A., Gupta, A., Mansour, Y., Sharma, A.: Welfare and profit maximization with production costs. In: FOCS, pp. 77–86 (2011)

[25] Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: WG, vol. 344, pp. 1–10 (1988)

[26] Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. cambridge university press (2005)

[27] Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)

[28] Buchbinder, N., Chen, S., Gupta, A., Nagarajan, V., Naor, J.: Online packing and covering framework with convex objectives. CoRR **abs/1412.8347** (2014)

[29] Buchbinder, N., Jain, K., Naor, J.: Online primal-dual algorithms for maximizing ad-auctions revenue. In: ESA, pp. 253–264 (2007)

[30] Buchbinder, N., Naor, J.: Online primal-dual algorithms for covering and packing. Mathematics of Operations Research **34**(2), 270–286 (2009)

[31] Buchbinder, N., Naor, J.S.: The design of competitive online algorithms via a primal-dual approach. Found. Trends Theor. Comput. Sci. **3**(2-3), 93–263 (2007)

[32] Călinescu, G., Chekuri, C., Pál, M., Vondrák, J.: Maximizing a monotone submodular function subject to a matroid constraint. SIAM Journal on Computing **40**(6), 1740–1766 (2011)

[33] Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. Algorithmica **47**(1), 53–78 (2007)

[34] Chakrabarty, D., Ene, A., Krishnaswamy, R., Panigrahi, D.: Online buy-at-bulk network design. SIAM Journal on Computing **47**(4), 1505–1528 (2018)

[35] Chan, S.O., Lee, J.R., Raghavendra, P., Steurer, D.: Approximate constraint satisfaction requires large LP relaxations. Journal of the ACM **63**(4), 34:1–34:22 (2016)

[36] Chan, T.H., Huang, Z., Kang, N.: Online convex covering and packing problems. CoRR **abs/1502.01802** (2015)

[37] Chang, K.C., Du, D.H.: Efficient algorithms for layer assignment problem. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **6**(1), 67–78 (1987)

[38] Charikar, M., Chekuri, C., Cheung, T.y., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed steiner problems. Journal of Algorithms **33**(1), 73–91 (1999)

[39] Charikar, M., Karagiozova, A.: On non-uniform multicommodity buy-at-bulk network design. In: STOC, pp. 176–182 (2005)

[40] Chekuri, C., Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.R.: Approximation algorithms for nonuniform buy-at-bulk network design. SIAM Journal on Computing **39**(5), 1772–1798 (2010)

[41] Chekuri, C., Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.R.: Approximation algorithms for nonuniform buy-at-bulk network design. SIAM Journal on Computing **39**(5), 1772–1798 (2010)

[42] Chekuri, C., Khanna, S., Naor, J.: A deterministic algorithm for the cost-distance problem. In: SODA, pp. 232–233 (2001)

[43] Chekuri, C., Vondrák, J., Zenklusen, R.: Submodular function maximization via the multilinear relaxation and contention resolution schemes. SIAM Journal on Computing **43**(6), 1831–1879 (2014)

[44] Courcelle, B.: On the expression of graph properties in some fragments of monadic second-order logic. In: Descriptive complexity and finite models, vol. 31, pp. 33–62 (1997)

[45] Dantzig, G.B.: Linear programming under uncertainty. Management Science **1** (1955)

[46] Dean, B.C., Goemans, M.X., Vondrdk, J.: Approximating the stochastic knapsack problem: The benefit of adaptivity. In: FOCS, pp. 208–217 (2004)

[47] Demaine, E.D., Hajiaghayi, M., Kawarabayashi, K.: Contraction decomposition in h-minor-free graphs and algorithmic applications. In: STOC, pp. 441–450 (2011)

[48] Demaine, E.D., Hajiaghayi, M.T., Kawarabayashi, K.: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: FOCS, pp. 637–646 (2005)

[49] Devanur, N.R., Huang, Z.: Primal dual gives almost optimal energy efficient online algorithms. In: SODA, pp. 1123–1140 (2014)

[50] Devanur, N.R., Jain, K.: Online matching with concave returns. In: STOC, pp. 137–144 (2012)

[51] Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer Science & Business Media (2012)

[52] Dye, S., Stougie, L., Tomasgard, A.: The stochastic single resource service-provision problem. Naval Research Logistics (NRL) **50**(8), 869–887 (2003)

[53] Dyer, M., Frieze, A., Kannan, R., Kapoor, A., Perkovic, L., Vazirani, U.: A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. Combinatorics, Probability and Computing **2**(3), 271–284 (1993)

[54] Elad, N., Kale, S., Naor, J.S.: Online semidefinite programming. In: ICALP, pp. 40:1–40:13 (2016)

[55] Elwalid, A.I., Mitra, D.: Effective bandwidth of general markovian traffic sources and admission control of high speed networks. IEEE/ACM Transactions on Networking **1**(3), 329–343 (1993)

[56] Ene, A., Chakrabarty, D., Krishnaswamy, R., Panigrahi, D.: Online buy-at-bulk network design. In: FOCS, pp. 545–562 (2015)

[57] Feldman, M., Naor, J., Schwartz, R.: A unified continuous greedy algorithm for submodular maximization. In: FOCS, pp. 570–579 (2011)

[58] Fortunato, S.: Community detection in graphs. Physics reports **486**(3), 75–174 (2010)

[59] Friggstad, Z., Könemann, J., Kun-Ko, Y., Louis, A., Shadravan, M., Tulsiani, M.: Linear programming hierarchies suffice for directed Steiner tree. In: IPCO, vol. 8494, pp. 285–296 (2014)

[60] Goel, A., Indyk, P.: Stochastic load balancing and related problems. In: FOCS, pp. 579–586 (1999)

[61] Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM **42**(6), 1115–1145 (1995)

[62] Grinold, R.C.: Infinite horizon stochastic programs. SIAM journal on control and optimization **24**(6), 1246–1260 (1986)

[63] Gupta, A., Krishnaswamy, R., Pruhs, K.: Online primal-dual for non-linear optimization with applications to speed scaling. In: WAOA, pp. 173–186 (2012)

[64] Gupta, A., Kumar, A., Nagarajan, V., Shen, X.: Stochastic load balancing on unrelated machines. In: SODA, pp. 1274–1285 (2018)

[65] Gupta, A., Nagarajan, V.: Approximating sparse covering integer programs online. Mathematics of Operations Research **39**(4), 998–1011 (2014)

[66] Gupta, A., Nagarajan, V., Singla, S.: Algorithms and adaptivity gaps for stochastic probing. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms, pp. 1731–1747 (2016)

[67] Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: STOC, pp. 417–426 (2004)

[68] Gupta, A., Talwar, K., Witmer, D.: Sparsest cut on bounded treewidth graphs: algorithms and hardness results. In: STOC, pp. 281–290 (2013)

[69] Gupta, V., Moseley, B., Uetz, M., Xie, Q.: Stochastic online scheduling on unrelated machines. In: IPCO, pp. 228–240 (2017)

[70] Hajiaghayi, M.T., Kortsarz, G., MacDavid, R., Purohit, M., Sarpatwar, K.K.: Approximation algorithms for connected maximum cut and related problems. In: ESA, vol. 9294, pp. 693–704 (2015)

[71] Håstad, J.: Clique is hard to approximate withinn1- ε. Acta Mathematica **182**(1), 105–142 (1999)

[72] Heitsch, H., Römisch, W.: Scenario tree modeling for multistage stochastic programs. Mathematical Programming **118**(2), 371–406 (2009)

[73] Helvig, C.S., Robins, G., Zelikovsky, A.: An improved approximation scheme for the group steiner problem. Networks **37**(1), 8–20 (2001)

[74] Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM **34**(1), 144–162 (1987)

[75] Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM Journal on Computing **17**(3), 539–551 (1988)

[76] Huang, Z., Kim, A.: Welfare maximization with production costs: A primal dual approach. In: SODA, pp. 59–72 (2015)

[77] Hui, J.Y.: Resource allocation for broadband networks. In: IEEE International Conference on Communications,-Spanning the Universe., pp. 1004–1011. IEEE (1988)

[78] Ibragimov, R., Sharakhmetov, S.: The best constant in the rosenthal inequality for nonnegative random variables. Statistics & probability letters **55**(4), 367–376 (2001)

[79] Im, S., Moseley, B., Pruhs, K.: Stochastic scheduling of heavy-tailed jobs. In: STACS, vol. 30, pp. 474–486 (2015)

[80] Immorlica, N., Karger, D., Minkoff, M., Mirrokni, V.S.: On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In: SODA, pp. 691–700 (2004)

[81] Jalali, A., Srebro, N.: Clustering using max-norm constrained optimization. In: ICML (2012)

[82] Johnson, W.B., Schechtman, G., Zinn, J.: Best constants in moment inequalities for linear combinations of independent and exchangeable random variables. The Annals of Probability **13**(1), 234–253 (1985)

[83] Kall, P.: Computational methods for solving two-stage stochastic linear programming problems. pp. 261–271 (1979)

[84] Kall, P., Mayer, J.: Slp-ior: An interactive model management system for stochastic linear programs. Mathematical Programming **75**(2), 221–240 (1996)

[85] Kelly, F.P.: Notes on effective bandwidths. In: Stochastic Networks: Theory and Applications, pp. 141–168 (1996)

[86] Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? SIAM Journal on Computing **37**(1), 319–357 (2007)

[87] Kim, J.H., Vu, V.H.: Concentration of multivariate polynomials and its applications. Combinatorica **20**(3), 417–434 (2000)

[88] Kleinberg, J., Rabani, Y., Tardos, E.: Allocating bandwidth for bursty connections. SIAM Journal on Computing **30**(1), 191–217 (2000)

[89] Knop, D., Koutecký, M., Masařík, T., Toufar, T.: Simplified algorithmic metatheorems beyond MSO: Treewidth and neighborhood diversity. In: WG, vol. 10520, pp. 344–357 (2017)

[90] Kouteckỳ, M., Lee, J., Nagarajan, V., Shen, X.: Approximating max-cut under graph-mso constraints. Operations Research Letters **46**(6), 592–598 (2018)

[91] Latała, R.: Estimation of moments of sums of independent real random variables. The Annals of Probability **25**(3), 1502–1513 (1997)

[92] Lee, J., Mirrokni, V.S., Nagarajan, V., Sviridenko, M.: Maximizing nonmonotone submodular functions under matroid or knapsack constraints. SIAM Journal on Discrete Mathematics **23**(4), 2053–2078 (2010)

[93] Lee, J., Sviridenko, M., Vondrák, J.: Submodular maximization over multiple matroids via generalized exchange properties. Mathematics of Operations Research **35**(4), 795–806 (2010)

[94] Lee, J.D., Recht, B., Salakhutdinov, R., Srebro, N., Tropp, J.A.: Practical large-scale optimization for max-norm regularization. In: NIPS, pp. 1297–1305 (2010)

[95] Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming **46**, 259–271 (1990)

[96] Li, J., Deshpande, A.: Maximizing expected utility for stochastic combinatorial optimization problems. In: FOCS, pp. 797–806 (2011)

[97] Li, J., Liu, Y.: Approximation algorithms for stochastic combinatorial optimization problems. Journal of the Operations Research Society of China **4**(1), 1–47 (2016)

[98] Li, J., Yuan, W.: Stochastic combinatorial optimization via poisson approximation. In: STOC, pp. 971–980 (2013)

[99] Magen, A., Moharrami, M.: Robust algorithms for on minor-free graphs based on the Sherali-Adams hierarchy. In: APPROX and RANDOM, vol. 5687, pp. 258–271 (2009)

[100] Mangasarian, O., Rosen, J.: Inequalities for stochastic nonlinear programming problems. Operations Research **12**(1), 143–154 (1964)

[101] Mathieu, C., Sinclair, A.: Sherali-adams relaxations of the matching polytope. In: STOC, pp. 293–302 (2009)

[102] Megow, N., Uetz, M., Vredeveld, T.: Models and algorithms for stochastic online scheduling. Mathematics of Operations Research **31**(3), 513–525 (2006)

[103] Mehlhorn, K., Ziegelmann, M.: Resource constrained shortest paths. In: ESA, pp. 326–337 (2000)

[104] Menache, I., Singh, M.: Online caching with convex costs: Extended abstract. In: SPAA, pp. 46–54 (2015)

[105] Meyerson, A.: Online algorithms for network design. In: SPAA, vol. 4, pp. 275–280 (2004)

[106] Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling: the power of LP-based priority policies. Journal of the ACM **46**(6), 924–942 (1999)

[107] Molinaro, M.: Stochastic l_p load balancing and moment problems via the l-function method. CoRR **abs/1810.05245** (2018)

[108] Moreau, J.J.: Fonctionnelles convexes. Séminaire Jean Leray (2), 1–108 (1967)

[109] Nagarajan, V., Shen, X.: Online covering with sum of ℓ_q-norm objectives. In: ICALP, pp. 12:1–12:12 (2017)

[110] Philpott, A.B., De Matos, V.L.: Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion. European Journal of Operational Research **218**(2), 470–483 (2012)

[111] Pinedo, M.: Offline deterministic scheduling, stochastic scheduling, and online deterministic scheduling. In: Handbook of Scheduling - Algorithms, Models, and Performance Analysis. (2004)

[112] Rosenthal, H.P.: On the subspaces of L^p $(p > 2)$ spanned by sequences of independent random variables. Israel J. Math. **8**, 273–303 (1970)

[113] Rozenberg, G., Salomaa, A.: Handbook of Formal Languages: Volume 3 Beyond Words. Springer Science & Business Media (2012)

[114] Schudy, W., Sviridenko, M.: Concentration and moment inequalities for polynomials of independent random variables. In: SODA, pp. 437–446 (2012)

[115] Shen, X., Lee, J., Nagarajan, V.: Approximating graph-constrained max-cut. Mathematical Programming **172**(1-2), 35–58 (2018)

[116] Sherali, H.D., Adams, W.P.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal on Discrete Mathematics **3**(3), 411–430 (1990)

[117] Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Mathematical programming **62**(1-3), 461–474 (1993)

[118] Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2), 202–208 (1985)

[119] Strömberg, T.: The operation of infimal convolution. Instytut Matematyczny Polskiej Akademi Nauk (1996)

[120] Vicente, S., Kolmogorov, V., Rother, C.: Graph cut based image segmentation with connectivity priors. In: CVPR (2008)

[121] Zosin, L., Khuller, S.: On directed steiner trees. In: SODA, pp. 59–63 (2002)