ABSTRACT

Title of dissertation: DATA-AWARE SCHEDULING IN DATACENTERS

Manish Purohit, Doctor of Philosophy, 2016

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

Datacenters have emerged as the dominant form of computing infrastructure over the last two decades. The tremendous increase in the requirements of data analysis has led to a proportional increase in power consumption and datacenters are now one of the fastest growing electricity consumers in the United States. Another rising concern is the loss of throughput due to network congestion. Scheduling models that do not explicitly account for data placement may lead to a transfer of large amounts of data over the network causing unacceptable delays. In this dissertation, we study different scheduling models that are inspired by the dual objectives of minimizing energy costs and network congestion in a datacenter.

As datacenters are equipped to handle peak workloads, the average server utilization in most datacenters is very low. As a result, one can achieve

huge energy savings by selectively shutting down machines when demand is low. In this dissertation, we introduce the network-aware machine activation problem to find a schedule that simultaneously minimizes the number of machines necessary and the congestion incurred in the network. Our model significantly generalizes well-studied combinatorial optimization problems such as hard-capacitated hypergraph covering and is thus strongly NP-hard. As a result, we focus on finding good approximation algorithms.

Data-parallel computation frameworks such as MapReduce have popularized the design of applications that require a large amount of communication between different machines. Efficient scheduling of these communication demands is essential to guarantee efficient execution of the different applications. In the second part of the thesis, we study the approximability of the co-flow scheduling problem that has been recently introduced to capture these application-level demands.

Finally, we also study the question, "In *what order* should one process jobs?" Often, precedence constraints specify a partial order over the set of jobs and the objective is to find suitable schedules that satisfy the partial order. However, in the presence of hard deadline constraints, it may be impossible to find a schedule that satisfies all precedence constraints. In this

thesis we formalize different variants of job scheduling with *soft precedence constraints* and conduct the first systematic study of these problems.

DATA-AWARE SCHEDULING IN DATACENTERS

by

Manish Purohit

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

Advisory Committee:
Professor Samir Khuller, Chair/Advisor
Professor William Gasarch
Professor Bruce Golden
Professor MohammadTaghi Hajiaghayi
Professor Barna Saha

Acknowledgments

The past five years that I have spent at College Park have been some of the most fruitful and enjoyable years of my life. This acknowledgment is a small effort to thank all those people who have enriched my time here.

Fittingly, I would first like to thank Professor Samir Khuller for being the best advisor a student could ask for. Samir's incredible memory, vast repository of knowledge and his ability to explain difficult concepts intuitively have helped me a lot over the past few years. In spite of being the department chair, Samir has always made himself available for help and advice. In Samir, I have found a valuable mentor, a colleague, and a friend. I shall always look towards him for guidance even as I move on to a new career.

Over the past few years, I have had the privilege of working with some great researchers during my summer internships. I am very grateful to my mentors Barna Saha, Guy Kortsarz, Rishi Saket, Vinayaka Pandit, Sreyash Kenkre, and Emily Pitler for hosting me over various summers. Their insightful ideas and suggestions helped me develop my research interests and led to the development of several algorithms presented in this thesis. I also take this opportunity to thank Professor V.S. Subrahmanian and Professor Hal Daumé III for fruitful collaborations as I explored different research areas. I would like to thank Professors Bill Gasarch, Bruce Golden, MohammadTaghi Hajiaghayi, and Barna Saha for agreeing to be on my thesis committee. I would like to convey my warm regards to the entire staff of the Computer Science department, in particular - Jennifer Story, Fatima Bangura, and Adelaide Findlay, for their ready help.

I would like to thank Professor Rajiv Gandhi for encouraging me to pursue a PhD. Rajiv introduced me to the field of approximation algorithms and has been of great help in all stages of my doctoral studies.

I believe that graduate school is one of the most interesting chapters of one's life. I would like to thank all my friends Bhaskar, Amit, Kartik, Anshul, Neeti, Sudha, Meethu, Ninad, Rama, any many others for filling these pages with loads of fun and the best of memories. My lab mates Kanthi, Ioana, Koyel, Saba, Sheng, Saurabh, Ahmed, Eunhui have all become my treasured friends. Directly or indirectly, my thesis is a reflection of those countless hours of brainstorming and discussions that we spent together.

Finally, I want to thank my parents, Deepak and Veena, my brother, Mandar, and my fiancée, Priyanka for supporting me through the past few years. None of the work in this thesis would have been possible without their love and understanding.

Contents

List of Figures

Chapter 1

Introduction

Large scale datacenters have emerged as the dominant form of computing infrastructure over the last two decades. Modern datacenters serve not only internet giants such as Google, Amazon, and Facebook, but also an increasing number of small and medium sized organizations. The increasing popularity of cloud-computing services such as Amazon Web Services, Microsoft Azure, and many others has also contributed to the phenomenal growth of datacenters. This trend is predicted to continue and some forecasts [1] anticipate a threefold increase in datacenter traffic between 2014 and 2019 (See Figure 1.1).

Despite this tremendous growth, modern datacenters still face a number of challenges. The explosion of big data analytics and e-commerce has led to a

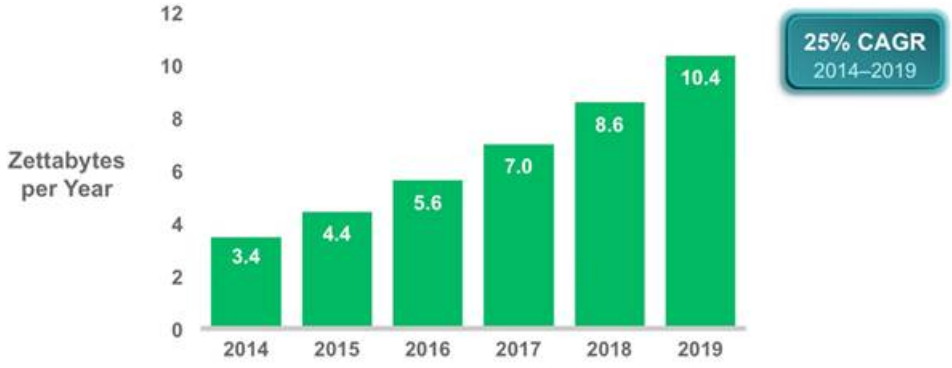

Figure 1.1: Global Datacenter IP Traffic Growth [1] with a predicted compound annual growth rate (CAGR) of 25%.

proportional increase in the power consumption by these datacenters. Indeed as observed by the NRDC in a recent issue paper [2], datacenters are one of the fastest growing electricity consumers in the United States. In 2013 alone, it is estimated that datacenters in the U.S. consumed enough electricity to power all the households in New York City twice over! Energy consumption by datacenters is projected to increase to 140 billion kilowatt-hours annually by 2020 costing over $13 billion in electricity bills [2]. Hamilton (see the SIGACT news article [3]) argues that a ten fold reduction in the power needs of datacenters may be possible if we can build systems with power management as their primary goal. Energy efficient job scheduling in datacenters is thus one of the most exciting research avenues today.

Another rising concern with the growth of big data analytics is the loss of throughput due to increasing network congestion. Datacenters now routinely process petabytes of data every day to help businesses with decision-making. Data-aware placement of jobs is essential [4, 5] to ensure efficiency for such applications as transferring large amounts of data over the network can lead to unacceptable delays. Further, applications written for popular data-parallel computation frameworks such as MapReduce [6] and Hadoop [7] often alternate between computation and communication phases. Effective scheduling of these network-intensive communication (also known as "shuffle") phases is essential to obtain good performance in a datacenter.

In this dissertation we take an algorithmic approach towards increasing datacenter efficiency by specifically targeting the issues raised above. The following sections describe the problems that we consider and give a brief overview of our contributions and techniques.

1.1 Energy Efficiency: Network-Aware Machine Activation

Before we formally introduce the problems that we study in this thesis, we note that the workload in data centers is highly non-uniform, i.e. there are sharp peaks and deep valleys in the workload. But as the systems are designed for handling peak workload, the industry has over invested in hardware to meet Service Level Agreements (SLAs) and in addition overspends in the running costs of the machines when they are on all the time. Studies indicate that average server utilization remained around 12 to 18 percent between 2006 and 2012 [2]. However, since the workload fluctuates over time, we can selectively shut down parts of the system to save energy when the demand is low. Energy savings result from not only from putting machines to a sleep state, but also from savings in cooling costs [8].

Motivated by these issues, Khuller, Li, and Saha [9] consider the problem of *which machines to shut down* and introduce the "Machine Activation" problem as follows. Given a set of jobs that need to be processed and a set of machines, the goal is to open a small subset of machines and schedule each job on an open machine while minimizing the total load on any machine. However,

all machines in a data center are not identical. In particular, every machine stores a limited amount of data and thus cannot execute every job unless the requisite data is first migrated to the machine. Unless the scheduling algorithm takes data migration into account explicitly, one may be required to transfer huge amounts of data within the data center leading to network congestion. In this thesis, we propose the network-aware machine activation problem that explicitly models data migration between machines and aims to find schedules that optimize three distinct objectives -

(i) the total activation cost of open machines,

(ii) the maximum load at any machine (makespan), and

(iii) maximum amount of data that needs to be moved to any machine.

1.1.1 The Framework

We model the data center as a star network with a central data server acting as the root as shown in Figure 1.2. Given any job j, let $\delta(j)$ denote the subset of machines that already have the data required to schedule j (shown by dotted lines in Figure 1.2). In our model, if a job j is scheduled on a machine in $\delta(j)$, then no data needs to be migrated and it requires a processing time of p_j units. On the other hand, if it is scheduled on machine $i' \notin \delta(j)$, then d_j

units of data need to be transferred from the root r to i' and it still requires a processing time of p_j units[1]. For example, if machine m_1 is the only active machine in Figure 1.2, then d_{j_2} units of data need to be transferred to machine m_1 and it incurs a total processing load of $p_{j_1} + p_{j_2}$ units. For a given schedule, we call the maximum amount of data that needs to be transferred to any machine i as the *congestion* incurred by the schedule.

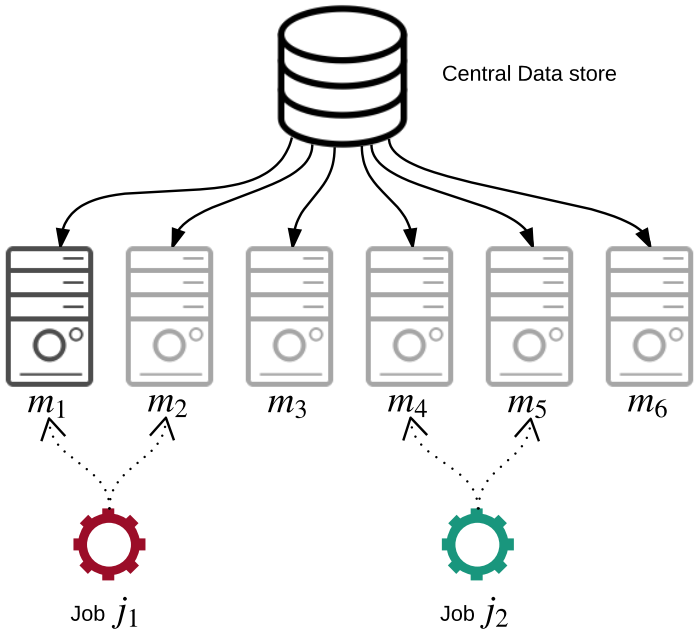

Figure 1.2: Network-Aware Machine Activation Framework

One can formulate a wide variety of interesting questions in this scheduling model. Our goal is to open a set $S \subseteq M$ of machines of minimum total cost

[1]For simplicity, here we assume that the data transfer is instantaneous.

such that we can find a schedule with makespan at most T and congestion at most B. We note that our framework directly generalizes the "machine activation" framework introduced by Khuller et al. [9] (when $\delta(j) = M$ for all jobs). We remark that while datacenters in practice do not follow the simple star layout as in Figure 1.2, we believe that it serves as a good starting point to facilitate a theoretical study of network-aware machine activation. We hope that our model can be extended to more realistic datacenter topologies and that it leads to interesting algorithmic as well as applied work in the future.

For the network-aware machine activation problem on general, unrelated parallel machines, we show that a simple randomized rounding scheme yields a schedule that violates all the three parameters by a factor of $O(\log n)$ [10]. It can be easily seen that one cannot avoid the $O(\log n)$ factor in the activation cost due to the hardness of approximating set cover [11]. On the other hand, data centers in practice often use a small replication factor. The Hadoop Distributed File System (HDFS), for instance, recommends a default replication factor of 3, i.e., every data item is replicated 3 times [12]. Consequently, for any job j, the number of machines that store the requisite data, i.e. $|\delta(j)|$, is often a small constant. For small $|\delta(j)|$ and unit machine activation costs, one

may expect an approximation factor better than $\log n$. Our main algorithmic result is a constant approximation algorithm for the network-aware machine activation problem in this case [10].

1.2 Managing Data Transfer: Co-flow Scheduling

Although processing jobs on machines with access to the relevant data can go a long way in reducing data transfer within a datacenter, it is not the only source of network congestion. Applications designed for data-parallel computation frameworks such as MapReduce usually alternate between computation and communication stages. Typically, intermediate data generated by a computation stage needs to be transferred across machines during a communication stage (called shuffle in MapReduce) for further processing.

Further, applications often need to wait before *all* of the intermediate data has been transferred over to the respective machines before continuing with the next computation phase; in MapReduce, for example, the reduce phase does not start until the shuffle phase is complete. Since a datacenter commonly serves hundreds of such applications simultaneously and yet has

limited internal bandwidth, it becomes necessary to schedule the data transfer required by these applications in order to obtain high throughput.

Chowdhury and Stoica [13] introduce co-flows as a networking abstraction to represent the collective communication requirements of a job. In order to effectively isolate the problem of scheduling data transfer to satisfy application-level objectives, the datacenter is modelled as a single $m \times m$ non-blocking switch with m input ports and m output ports. For simplicity, we assume that all ports have unit capacity, i.e., at most one unit of data can be transferred through any port at a time. A co-flow is defined as a collection of parallel flow demands that share a performance goal. For example, Figure 1.3 represents a single co-flow that is collection of four parallel flow demands - 2 data units from input 1 to output 1, 3 units from input 1 to output 2, 1 units from input 2 to output 1, and 4 units from input 2 to output 2 respectively. A co-flow j is said to be complete once *all* of its component flows have been transferred. Given a collection of such co-flows, the co-flow scheduling problem is to find a feasible schedule that minimizes the average (weighted) completion time of the schedule.

Co-flow scheduling has been a topic of active research [14, 15, 16, 17] since its introduction and is also the basis for a successful practical network scheduler

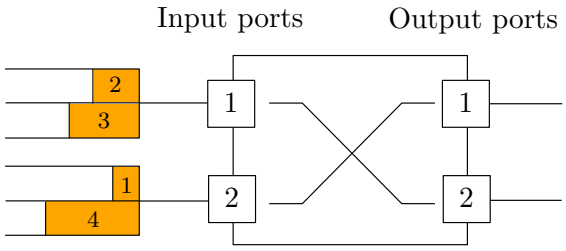

Figure 1.3: An example co-flow over a 2×2 switch.

[14]. In spite of the existence of good practical schedulers, even in the offline setting, when all jobs are known in advance, no $O(1)$ approximation algorithm was known until recently. Qiu, Stein and Zhong [16] obtain a deterministic $\frac{67}{3}$ approximation and a randomized $(9 + \frac{16\sqrt{2}}{3})$ approximation for the problem of minimizing the weighted completion time. For the special case when all release times are zero, Qiu et al. [16] demonstrate improved bounds of $\frac{64}{3}$ (deterministic) and $(8 + \frac{16\sqrt{2}}{3})$ (randomized). We highlight the connection of the co-flow scheduling problem with the well-studied concurrent open shop scheduling problem and give improved approximation guarantees for the co-flow scheduling problem [18]. We give a deterministic 12 approximation algorithm when the co-flows can have arbitrary release times. For the special case when all release times are zero, we give a deterministic 4-approximation algorithm.

1.3 Managing Data Detours: Firewall Placement

The co-flow scheduling model described above abstracts away the network structure of a datacenter and focuses on obtaining a provably good schedule that minimizes the weighted completion time of the jobs. However, in practice, machines in a datacenter are inter-connected using various hierarchical network topologies [19, 20]. In such a network, one needs to specify the path that the data being transferred between two machines needs to follow through the network. As we will see shortly, simply using the "shortest path" between two machines is not always the best way to route data.

In the Infrastructure as a Service (IaaS) cloud computing service model, clients are provided with autonomous virtual machines (VMs) as per their requirement and the service provider runs these virtual machines in its datacenter. Additionally, a client may request multiple virtual machines and require bandwidth to communicate between two machines. In this scenario, the underlying physical network of the datacenter is used to route the data between two virtual machines residing on different servers. While some traffic can take a shortest path between the two VMs, different services may use middleboxes

(e.g. load balancer, firewalls, NAT boxes) and the corresponding traffic takes a detour to pass through these middleboxes [21, 22]. Specifically for security reasons, filtering the communication between different virtual machines in a datacenter becomes a necessity. Such filtering can be accomplished by placing firewalls at strategic nodes within the datacenter and routing the traffic to pass through a firewall. For concreteness, we now use the term "firewall" to mean any middlebox that a service provider wants the data to pass through. This abstraction introduces several basic facility location problems where the firewalls correspond to facilities and communication requests between VMs correspond to demands.

Suppose some data needs to be transferred between machines s and t and let P be a path in the physical network that connects those machines. If there is no available firewall on path P, then we need to route the data first from s to some firewall f and then back from f to the destination t. Clearly, having too few firewalls would cause a large amount of traffic to be routed to a particular firewall leading to increased congestion in the links leading to the firewall. In traditional facility location problems, the objective deals with minimizing the "distance" that a demand j needs to travel to reach a facility i. However as the propagation delay within a datacenter is relatively

small, the queueing delay due to link congestion often dominates. As a result, the maximum link congestion is a better measure of the quality of a given firewall placement. Further, since the underlying network is shared between other services, we have a bandwidth constraint on the links that bounds the allowed congestion and we focus on finding a good firewall placement subject to these bandwidth constraints. Figure 1.4 illustrates our firewall placement model.

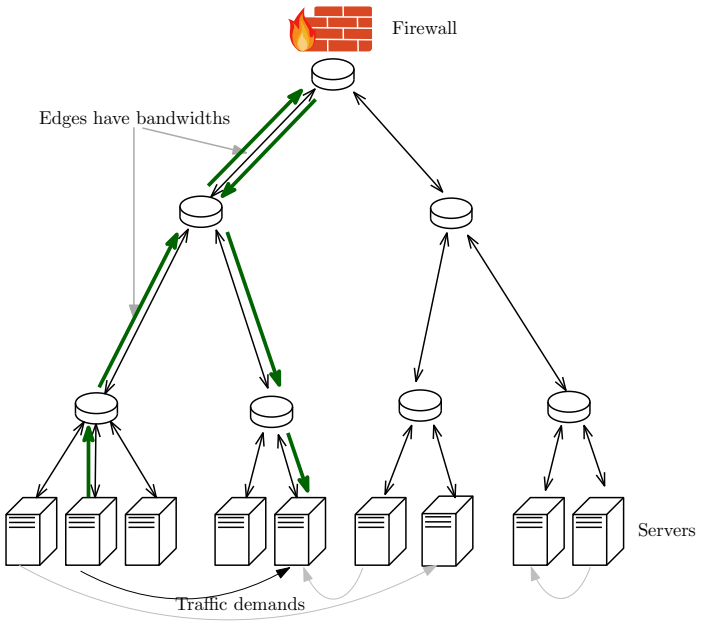

Figure 1.4: Firewall Placement Framework

The above discussion motivates the formulation of the following *Firewall Placement Problem:* Given a physical network with bandwidth constraints

on links, an assignment of virtual machines to servers, and communication demands between virtual machines, find the minimum number of firewalls necessary so that all the demands can be simultaneously satisfied while respecting the bandwidth constraints on the links. Additionally, we consider the capacitated version of the above problem where each firewall also has a capacity that restricts the maximum number of demands it can satisfy.

We study the approximability of the Firewall Placement problem on tree networks [23]. We show that the *soft capacitated firewall placement* problem with uniform capacities that allows one to place multiple firewalls at a single vertex can be solved in polynomial time via a simple greedy strategy. On the other hand, we also show that the *hard capacitated firewall placement* problem where we can place at most one firewall at a vertex is NP-hard. For this case, we design a polynomial time algorithm that requires at most the optimum number of firewalls but violates the edge bandwidths by a factor of at most 2. For the hard capacitated firewall placement problem on star networks, we give a constant approximation algorithm via a reduction to a special case of the network-aware machine activation problem.

1.4 Constraint Selection:

Scheduling with Soft Precedences

Many applications can be expressed as a set of jobs with precedence constraints that encode a dependency between two jobs. For example, a precedence constraint $j_1 \prec j_2$ indicates that the job j_1 must be completed before job j_2 can begin to process. In typical precedence constrained scheduling problems [24, 25, 26, 27, 28, 29, 30, 31], the precedence constraints are considered sacrosanct and one aims to find the "best" schedule while satisfying every constraint. Traditional optimization goals include finding a schedule that minimizes the weighted sum of completion times or one that minimizes the makespan, i.e., the earliest time when all jobs have completed.

Let us consider the simplest case of scheduling with precedence constraints. Suppose we have n unit length jobs with precedence constraints and an infinite capacity machine. The objective is to find a feasible schedule with the minimum makespan. Given such an instance, consider the associated precedence graph $G = (V, A)$ where V denotes the set of n jobs and there is an edge $e = (u, v) \in A$ if and only if $u \prec v$. It can be easily observed that the optimum makespan for the instance equals the number of vertices in the

longest directed path in G.

In practice however, it is often the case that not all precedence constraints are made equally. For instance, Lesaint et al. [32] show that in internet telephone, there are usually two types of precedence constraints - (1) Hard constraints that are *required* by the service delivery architecture, and (2) Soft constraints that are *desired* by user specifications but may be violated if it leads to better schedules. Jaskowski and Sobotka [33] argue that such a dichotomy of precedence constraints also occurs commonly in project scheduling problems. Further, they show that modelling soft constraints as distinct from and less stringent than their hard counterparts helps to significantly reduce the makespan of the schedule. In this thesis, we perform the first systematic study of scheduling with soft precedence constraints subject to a deadline [34]. We formalize the notion of "soft precedences" in a few different ways. Interestingly, these formalizations lead to natural generalizations of several well-known graph optimization problems such as maximum directed cut and maximum acyclic subgraph.

Linear Soft Precedences

In this model, we try to capture scenarios where the cost incurred due to violating a precedence constraint is proportional to the extent by which the constraint is violated. Let s_j and f_j denote the starting time and finishing time of job j respectively. We say a precedence constraint $u \prec v$ is satisfied if and only if $s_v \geq f_u$, i.e., job v is processed only once u has finished processing. However, if the constraint is not satisfied, then we incur a penalty of $(f_u - s_v)$.

In this setting, given a set of n jobs along with their processing times and precedence constraints, an infinite capacity machine and a deadline k, the *Deadline Linear Soft Precedence Scheduling* problem is to find a feasible schedule that minimizes the total penalty such that all jobs finish by the deadline. We show that the *Deadline Linear Soft Precedence Scheduling* problem can be solved optimally in polynomial time by an appropriate LP-formulation.

Discrete Soft Precedences

In this model, we assume that the cost incurred for a violating a constraint is fixed irrespective of the degree of the violation. We model the precedence constraints by a directed precedence graph $G = (V, A)$ as discussed earlier.

Further we allow every edge to have a weight $w(e)$ that signifies the importance of the associated precedence constraint. Now if a schedule does not satisfy the precedence constraint $u \prec v$ then it incurs a cost of $w(u, v)$.

In this setting, given a set of n jobs along with their processing times and precedence constraints, an infinite capacity machine, and a deadline k the *Deadline Discrete Soft Precedence Scheduling* problem is to find a feasible schedule that maximizes the total weight of satisfied constraints such that all jobs finish by the deadline. We introduce the MAX-k-ORDERING problem and show that it is equivalent to the *Deadline Discrete Soft Precedence Scheduling* problem. We obtain a polynomial time 2-approximation algorithm via randomized rounding of the natural LP relaxation. We also provide matching lower bounds and thus prove that our algorithm is tight up to lower order terms. We also consider the minimization variant where the objective is to minimize the weight of the violated constraints.

1.5 Outline of the Dissertation

In the following chapters we formally define the different models that we have described in the previous sections and present our algorithms and analyses

on these models.

In Chapter 2, we study the network-aware machine activation problem and give approximation algorithms for different problem variants in this setting.

In Chapter 3, we introduce the Co-flow scheduling model and give improved approximation algorithms both in the setting of general release times and with zero release times.

In Chapter 4, we formalize the firewall placement model and show that the soft-capacitated firewall placement problem on trees admits an optimal polynomial time algorithm. We also show that hard-capacitated version is NP-complete and give the first approximation algorithms.

In Chapter 5, we perform a systematic study of scheduling with soft precedence constraints. We formulate the MAX-k-ORDERING problem to accurately capture the maximization variant of discrete soft precedences and give a tight approximation algorithm for it using a novel LP rounding scheme.

Chapter 2

Network-Aware Energy-Efficient Scheduling

In this chapter we formally define our framework for network-aware energy-efficient scheduling and develop approximation algorithms for the different problem variants in this framework. Section 2.1 formally defines the network-aware machine activation problem and briefly describes some prior related work in this area. In Sections 2.2- 2.5, we design approximation algorithms for different variants of the network-aware machine activation problem. Our algorithms are based on rounding the natural linear programming relaxation in multiple phases that may be of independent interest. In particular, many of

our rounding stages themselves solve auxiliary linear programs and critically rely on the sparsity of basic feasible solutions to obtain an integral solution of bounded cost.

2.1 The Framework

We model the data center as a star network with a central data server acting as the root r. In addition, we have a set M of m machines that form the leaves of the star. Each machine $i \in M$ is associated with an activation cost a_i that is the cost incurred if machine i is utilized in a schedule. Finally, we are given a set J of n jobs where each job j has a processing requirement of p_j and a data requirement of d_j units[1]. In addition, let $\delta(j) \subseteq M$ denote the set of machines that already have the data required to schedule j. In particular, if the job j is scheduled on a machine $i \in \delta(j)$, then no data needs to be migrated and it requires a processing time of p_j units. On the other hand, if job j is scheduled on machine $i' \notin \delta(j)$, then d_j units of data need to be transferred from the root r to i' and it still requires a processing time of

[1]In the unrelated machines setting, the processing (p_{ij}) and data (d_{ij}) requirement of a job depends on the machine on which it is scheduled.

p_j units[2]. For a given schedule, the congestion of the schedule is defined as the maximum amount of data that needs to be transferred to any machine $i \in M$. Figure 2.1 illustrates our framework for network-aware energy-efficient scheduling.

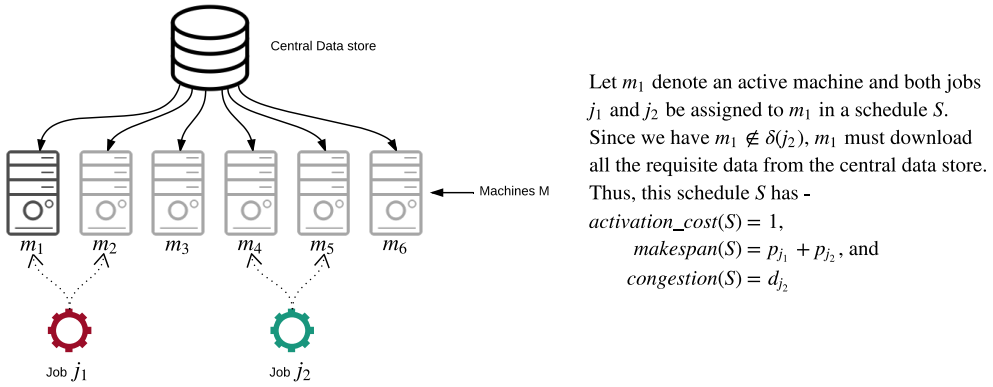

Figure 2.1: Network-Aware Machine Activation Framework

One can formulate a variety of interesting questions in this scheduling model. Our goal is to open a subset $S \subset M$ of machines of minimum total cost such that we can find a schedule with makespan at most T and congestion at most B. We note that our framework directly generalizes the "machine activation" framework introduced by Khuller et al. [9] (when $\delta(j) = M$ for all jobs).

[2]For simplicity, here we assume that the data transfer is instantaneous.

2.1.1 Related Work on Network Aware Scheduling

In a seminal paper Papadimitriou and Yannakakis [35] initiated the study of *scheduling with communication delay*. In this model there are m parallel machines and a set of precedence constrained jobs. With every arc (j, k) between two jobs in the precedence constraints, there is an accosted communication delay c_{jk}. If jobs j and k are processed on different machines, then the processing of k cannot start before c_{jk} time units have elapsed after the completion of j. However, if jobs j and k are processed on the same machine, then k can start as soon as j has been completed. Papadimitrou and Yannakakis showed even for unit jobs, and when job duplication is allowed, minimizing makespan with uniform communication delay is NP-hard. Since, their work, approximation algorithms [36, 37, 38, 39, 40, 41], and strong non-approximability results have been shown for unrelated job processing time even under simple hierarchical precedence constraints [42, 43] .

In a similar spirit, Phillips, Stein and Wein [44] introduced an interesting model of network scheduling in which before a job can be started on a machine, the job and its data need to be moved to the machine. Here there is a graph that induces the time a job requires to be moved to a machine. When moving a job's data through the network, jobs can freely share links in the network.

This essentially results in jobs having different "arrival" times depending on the machine they are scheduled on. Interestingly, except for the only recent work by Im and Moseley [45], we did not find any work in theoretical scheduling literature that considers network congestion along with scheduling. The work of [45] focuses on a tree network with unit bandwidth, and assumes all jobs are located only at the root initially. The goal is to find a dispatch time for jobs and a leaf to run the job to minimize the average flow time in an online setting. The problem has an $O(\frac{1}{\epsilon^7})$-approximation ratio with $(2+\epsilon)$ speed augmentation for unrelated machines. While their model essentially considers communication delay and not bandwidth, success of their approach is an indicative of the potential benefits of studying network-aware scheduling problems.

2.1.2 Related Work on Capacitated Covering

The early work of Wolsey [46] shows that the simple greedy algorithm gives a $\log n$ approximation for the capacitated set cover problem. The vertex cover with hard capacities problem was first studied by Chuzhoy and Naor [47] who gave a 3-approximation for the problem using randomized rounding followed by alteration. Gandhi et al.[48] later improved this result to give a

2-approximation. However, both of these approaches only work for simple graphs and fail for multigraphs where one can have multiple edges between the same set of vertices. Saha and Khuller [49] give the first constant approximation for the hard-capacitated vertex cover problem on multigraphs and hypergraphs. In a recent improvement, Cheung et al. [50] obtain a deterministic 3-approximation and a randomized 2.155-approximation algorithm for the problem on multigraphs and a $2f$-approximation algorithm on hypergraphs where f denotes the size of the largest hyperedge.

Khuller et al. [9] introduce the machine activation problem and obtain a bi-criteria approximation algorithm. In particular, they show that if there exists a schedule with makespan T and activation cost A, then one can obtain a schedule with makespan at most $(2+\epsilon)T$ and activation cost at most $2(1+1/\epsilon)(\ln \frac{n}{A}+1)A$ for any $\epsilon > 0$ using a linear programming rounding method. In the same setting, they also show that a greedy algorithm obtains a $(2, \ln n)$ bi-criteria approximation exploiting the submodularity of generalized flow [51]. For the *online* setting where jobs arrive online, Azar et al. [52] obtain a bicriteria algorithm with a poly-logarithmic competitive ratio on both makespan and activation cost.

2.1.3 Our Contributions and Techniques

We first consider the general network aware machine activation problem with arbitrary machine activation costs, processing times and data requirements. We show that a simple randomized rounding scheme finds a schedule that violates all the three parameters by a factor of $O(\log n)$. The proof of the following theorem is deferred to Section 2.5.

Theorem 1. *For the general network-aware machine activation problem on unrelated machines, if there exists a schedule with total activation cost A, makespan T, and congestion B, then there is a polynomial time algorithm that constructs a schedule of cost $O(\log n)A$, makespan $O(\log n)T$, and congestion $O(\log n)B$.*

It can be easily seen that one cannot avoid the $O(\log n)$ factor in the activation cost due to the hardness of approximating set cover [11]. On the other hand, data centers in practice often use a small replication factor. The Hadoop Distributed File System (HDFS), for instance, recommends a default replication factor of 3, i.e., every data item is replicated 3 times [12]. Consequently, for any job j, the number of machines that store the requisite data, i.e. $|\delta(j)|$, is often a small constant. For small $|\delta(j)|$ and unit activation

cost, one may expect an approximation factor better than $\log n$. The main algorithmic result of this chapter is a constant approximation algorithm for the network-aware machine activation problem in this case. In particular, we show the following theorem.

Theorem 2. *For the network-aware machine activation problem with unit activation costs, if there exists a solution with assignment cost A, makespan T, and congestion B, then there is a polynomial time algorithm that finds a schedule with assignment cost at most $(8f + 9)A$, makespan at most $5T$, and congestion at most $4B$ where $f = \max_j |\delta(j)|$.*

Our approximation algorithm is based on a novel rounding of the natural LP relaxation of the problem. The rounding proceeds over multiple stages such that in each stage we open a bounded number of machines integrally, until finally all the jobs can be satisfied by the integrally opened machines. As opposed to the traditional hypergraph cover problem, in our setting it is feasible to assign a job j to a machine $v \notin \delta(j)$ at a cost of contributing to the congestion at v. This flexibility of a job being satisfied by a machine not incident on it makes the rounding process significantly more challenging than that for the traditional hypergraph cover with hard capacities problem. Our techniques demonstrate that explicitly constructing auxiliary linear programs

that maintain feasibility of the original LP relaxation is a promising strategy for rounding complex LP relaxations. In particular, using the sparsity induced by basic feasible solutions, we can effectively bound the loss in solution quality at each step. We believe that these techniques are of independent interest and may be applicable to other problems that do not admit an easy rounding strategy.

For the case when all jobs have unit processing and data requirements, i.e. $p_j = d_j = 1, \forall j \in J$, we obtain the following improved approximation algorithm. We note that the following theorem yields a true approximation algorithm as the solution satisfies all the makespan and congestion constraints.

Theorem 3. *For the network-aware machine activation problem with unit activation costs and unit processing and data requirements for all jobs, if there exists a solution with assignment cost A, makespan T, and congestion B, then there is a polynomial time algorithm that finds a schedule with assignment cost at most $(4f + 5)A$, makespan at most T, and congestion at most B where $f = \max_j |\delta(j)|$.*

2.2 Preliminaries

Figure 2.2 shows the natural LP relaxation of the network-aware machine activation problem with unit activation costs. The variables y_i indicate whether machine i is chosen into the solution or not. If job j is assigned to a machine $i \in \delta(j)$, we set $x_{ij} = 1$. Similarly, if job j is assigned to a machine $i \notin \delta(j)$, we set $z_{ij} = 1$. For ease of notation, we also introduce variables x_{ij} where $i \notin \delta(j)$ and z_{ij} where $i \in \delta(j)$, but note that we can assume that they are set to 0 without loss of generality.

2.3 LP Rounding for Network-Aware Machine Activation

Let (x^*, y^*, z^*) denote an optimal solution to the LP in Figure 2.2. In the rest of this section, we process the optimal fractional solution (x^*, y^*, z^*) in a number of stages to obtain an integral solution. We first provide an intuitive discussion regarding the high level ideas of the proof.

$$\mathbf{LP_1}: \qquad \min \sum_{i \in M} y_i$$

subject to,

$$\forall j \in J, \qquad \sum_{i \in \delta(j)} x_{ij} + \sum_{i \notin \delta(j)} z_{ij} \geq 1 \tag{2.1}$$

$$\forall i \in M, \text{ and } j \in J, \qquad x_{ij} + z_{ij} \leq y_i, \tag{2.2}$$

$$\forall i \in M, \qquad \sum_{j \in J} p_j (x_{ij} + z_{ij}) \leq T y_i \tag{2.3}$$

$$\text{and,} \qquad \sum_{j \in J} d_j z_{ij} \leq B y_i, \tag{2.4}$$

$$\forall i \in M, \qquad 0 \leq y_i \leq 1. \tag{2.5}$$

$$\forall i \in M \text{ and } j \in J, \qquad x_{ij} \geq 0 \text{ and } z_{ij} \geq 0. \tag{2.6}$$

Figure 2.2: LP Relaxation for instance $\mathcal{I}$ of network-aware machine activation

2.3.1 High Level Ideas

Initially, we only attempt to round the machine opening variables. Our goal is to integrally open a bounded number of machines and maintain a feasible (or approximate feasible) fractional assignment. Once we have such a solution, we can obtain an integral assignment of jobs using techniques akin to those used for the Generalized Assignment Problem [53]. We now provide a high level overview of the main ideas used to obtain the set of integrally open machines.

Since in the fractional LP solution, a job j can be satisfied partially by machines $i \in \delta(j)$ and partially by machines outside $\delta(j)$, we aim to handle these two components separately. Let us call $\sum_{i \in \delta(j)} x^*_{ij}$ as the internal demand of j and $\sum_{i \notin \delta(j)} z^*_{ij}$ as the external demand of j. In the first stage, we open all machines with high y^*_i values (call this set U_1). We then note that since a job j can be assigned to any machine $i \notin \delta(j)$ as long as the vertex has enough bandwidth, it is sufficient to open a few additional vertices with high residual bandwidth (call this set U_2) to satisfy the external demand of *all* jobs j. We then observe that jobs whose internal demand is partially satisfied by $U_1 \cup U_2$ can be handled by adapting the techniques used by Cheung et al. [50] for hard-capacitated hypergraph covering.

We are now still left with jobs whose internal demand is non-zero but is satisfied outside of $U_1 \cup U_2$. Handling such jobs proves to be the most challenging aspect of the problem. Indeed, it is possible that for every job j_k, there is one machine $i_k \in \delta(j_k)$ with $x_{i_k j_k} = y_{i_k} = \epsilon$. Now, if we try to satisfy this contribution of $x_{i_k j_k}$ by integrally open machines, we may be forced to open all such machines i_k leading to a bad solution. Our approach is to reshuffle the assignments of external and internal demands in a way that we can bound the number of such jobs. We achieve this by writing two intermediate linear programs that consider the unsatisfied internal demands of these jobs as well as their already satisfied external demands. The first linear program massages the assignments using iterative rounding in an attempt to make constraints of Equation (2.2) tight. In this stage, we open a few additional vertices to satisfy the jobs that are not amenable to tight assignments. In the final stage, the tight assignment constraints allow us to drop the capacity constraints on the vertices and we formulate a linear program to obtain the final solution. Once again, we use the sparsity introduced by basic feasible solutions to bound the total number of vertices that we need to open in this step.

2.3.2 Stage 1

Given the optimal fractional solution (x^*, y^*, z^*), we partition M as follows -

- $U_1 = \{i \in M \mid y_i^* \geq \frac{1}{f+1}\}$
- $V_1 = \{i \in M \setminus U_1 \mid y_i^* > 0\}$
- Sort vertices $i \in V_1$ in non-increasing order of $\left(T - \sum_{j \in J} \frac{p_j x_{ij}^*}{y_i^*}\right)$
- $U_2 = \lceil (f+1) \sum_{i \in V_1} y_i^* \rceil$ machines from V_1 in above order.

Lemma 1. *There exists a feasible solution (x^1, y^1, z^1) to $\mathbf{LP_1}$ that satisfies the following properties -*

(i) $y_i^1 = 1,\ \forall i \in U_1 \cup U_2$ *and* $y_i^1 = y_i^*,$ *otherwise.*

(ii) $z_{ij}^1 = 0,\ \forall i \notin U_1 \cup U_2$

(iii) $\sum_{i \in M} z_{ij}^1 \geq \min(1, (f+1)out(j)),\ \forall j \in J$ *where* $out(j) = \sum_{i \in V_1} z_{ij}^*$

(iv) $x_{ij}^1 = x_{ij}^*,\ \forall i \in M$ *and* $j \in J$

Proof. For the sake of analysis, consider the following linear program where $\bar{C}_i$ denotes the capacity of machine $i \in V_1$ that is available for jobs not incident on it, i.e. $\bar{C}_i = (T - \sum_{j | i \in \delta(j)} \frac{p_j x_{ij}^*}{y_i^*})$. All external assignments that are feasible for the following LP maintain capacity and bandwidth constraints at all machines $i \in V_1$. In addition, the first constraint guarantees that the

external demand of every job j is satisfied as well (even after scaling up by $(f+1)$).

$$\text{minimize} \sum_{i \in V_1} y_i$$

subject to

$$\sum_{i \in V_1} z_{ij} \geq \min(1, (f+1)out(j)) \qquad , \forall j \in J \quad (2.7)$$

$$\sum_{j \in J} p_j z_{ij} \leq \bar{C}_i y_i \qquad , \forall i \in V_1$$

$$\sum_{j \in J} d_j z_{ij} \leq B y_i \qquad , \forall i \in V_1$$

$$z_{ij} \leq y_i \qquad , \forall i \in V_1, \forall j \in J$$

$$0 \leq z_{ij}, y_i \leq 1 \qquad , \forall i \in V_1, \forall j \in J$$

Since we have $z^*_{ij} \leq y^*_i < \frac{1}{f+1}, \forall i \in V_1$ and $\forall j \in J$, it is easy to observe that setting $z_{ij} = (f+1)z^*_{ij}$ and $y_i = (f+1)y^*_i$ is a feasible solution to the above linear program. We now claim that we can obtain a feasible solution to the above linear program such that the set of machines with non-zero y variables is exactly the set U_2 as defined earlier. Given such a feasible solution (y, z) to the above LP, we set $y^1_i = 1, \forall i \in U_2$ and $y^1_i = y^*_i, \forall i \in V_1 \setminus U_2$ and $z^1_{ij} = z_{ij}, \forall i \in V_1, \forall j \in J$. Finally since we have $x^1_{ij} = x^*_{ij}, \forall i \in M, \forall j \in J$, we

are guaranteed that (x^1, y^1, z^1) is a feasible solution to $\mathbf{LP_1}$.

Proof of the claim: Consider a feasible solution (y, z) to the above linear program such that there are two machines a and b with $0 < y_a, y_b < 1$. Without loss of generality, let $\bar{C}_a > \bar{C}_b$. We now obtain a new feasible solution (y', z') with one less fractional y variable as follows - Set $y'_a = \min(y_a + y_b, 1)$ and $y'_b = \max(0, y_a + y_b - 1)$. The z values are changed proportionately as well - $z'_{bj} = z_{bj}(\frac{y'_b}{y_b})$ and $z'_{aj} = z_{aj} + z_{bj} - z'_{bj}$ for all $j \in J$. For all other machines and jobs, (y', z') is identical to (y, z).

To show that (y', z') is feasible, we observe that $\sum_{i \in V_1} z'_{ij} = \sum_{i \in V_1} z_{ij}$ for all jobs j and hence constraints (2.7) are satisfied. Also the bandwidth and capacity constraints are satisfied at all machines other than a and b since their assignments remain unchanged. For machine b, $\sum_j p_j z'_{bj} = \sum_j p_j z_{bj} \frac{y'_b}{y_b} \leq \bar{C}_b y'_b$ and similarly $\sum_j d_j z'_{bj} = \sum_j d_j z_{bj} \frac{y'_b}{y_b} \leq B y'_b$. For machine a, we consider two cases.

Case 1: $y'_a = y_a + y_b$ and $y'_b = 0$. Now we have $\sum_j p_j z'_{aj} = \sum_j p_j (z_{aj} + z_{bj} - 0) \leq \bar{C}_a y_a + \bar{C}_b y_b \leq \bar{C}_a y'_a$ and hence the capacity constraint is satisfied.

Case 2: $y'_a = 1$ and $y'_b = y_a + y_b - 1$. We have the following.

$$\sum_j p_j z'_{aj} = \sum_j p_j (z_{aj} + z_{bj} - z'_{bj}) \leq C_a y_a + \sum_j p_j z_{bj} (1 - \frac{y_a + y_b - 1}{y_b})$$
$$\leq C_a y_a + C_b y_b (\frac{1 - y_a}{y_b}) \leq C_a = C_a y'_a$$

Analogously, we can show that the bandwidth constraints are satisfied at machine u as well. We repeat this procedure until we have at most one fractional y variable. Since we always choose to open the machine with larger residual capacity, and initially we have $\sum_{i \in V_1} y_i = (f+1) \sum_{i \in V_1} y^*_i$, the final set of machines with non-zero fractional values is exactly the set U_2 defined earlier. Recall that U_2 is defined as the set of $\lceil (f+1) \sum_{i \in V_1} y^*_i \rceil$ machines from V_1 with the highest residual capacity. □

Let $\hat{J}$ denote the set of jobs that are not completely satisfied by the open machines in U_1 and U_2, i.e., $\hat{J} = \{j \in J \mid \sum_{i \in U_1 \cup U_2} (x^1_{ij} + z^1_{ij}) < 1\}$. The following claim follows immediately from the definition of $\hat{J}$ and Lemma 1.

Claim 1. *Every job $j \in \hat{J}$ must have $out(j) < \frac{1}{f+1}$.*

The following lemma is useful to bound the total number of machines opened in this stage.

Lemma 2. $|U_1| + |U_2| \leq (f+1) \sum_{i \in V} y^*_i + 1$

Proof. Since $y_i^* \geq \frac{1}{f+1}$ for all $i \in U_1$, we have $|U_1| \leq (f+1)\sum_{i \in U_1} y_i^*$. On the other hand, by definition U_2 consists of $\lceil (f+1)\sum_{i \in V \setminus U_1} y_i^* \rceil$ machines and the desired bound follows. □

2.3.3 Stage 2

Let $J_2 = \{j \in \hat{J} \mid \sum_{i \in U_1 \cup U_2} x_{ij}^1 > 0\}$ denote the set of unsatisfied jobs that are assigned internally to at least one open machine. For convenience, also define $W = V_1 \setminus \{U_1 \cup U_2\}$ to be the set of machines with fractional y^1 values. In this stage, we aim to open a few more machines from W in order to satisfy all jobs in J_2. We follow the approach of Cheung, Goemans and Wong [50].

The main idea is to associate the deficit of every job $j \in J_2$ to the open machines in $U_1 \cup U_2$ and then write a covering LP that opens new machines in order to meet these deficits. Since for a job $j \in J_2$, $\delta(j)$ may contain multiple machines from $U_1 \cup U_2$, we distribute its deficit in proportion to its assignment. In particular, for a job $j \in J_2$ and a machine $u \in \delta(j) \cap \{U_1 \cup U_2\}$, define $\gamma_{uj} = \frac{x_{uj}^1}{\sum_{i \in \delta(j) \cap \{U_1 \cup U_2\}} x_{ij}^1}$ and associate with u a partial deficit of $p_j \gamma_{uj} \sum_{w \in \delta(j) \cap W} x_{wj}^1$. In this way, the deficit of job j is completely distributed among the machines

in $\delta(j) \cap \{U_1 \cup U_2\}$. The total deficit at a machine $u \in U_1 \cup U_2$ is thus

$$r(u) = \sum_{j \in J_2 | u \in \delta(j)} p_j \gamma_{uj} \sum_{w \in \delta(j) \cap W} x^1_{wj}$$

Next, we define the amount of coverage that a machine $w \in W$ can provide. For a job j such that $w \in \delta(j)$, and $u \in \delta(j) \cap \{U_1 \cup U_2\}$, w can offer a coverage of $p_j \gamma_{uj} x^1_{wj} / y^1_w$. We scale this coverage down by a factor of $(f-1)$ to avoid potential non-linearity in the formulation and defer the reader to [50] for further details regarding this point. Thus, machine w if opened can provide

$$M(u,w) = \sum_{j \in J_2 | \{u,w\} \subseteq \delta(j)} \frac{p_j \gamma_{uj} x^1_{wj}}{(f-1) y^1_w}$$

units of coverage to u. We now write the covering LP in Figure 2.3.

$$\mathbf{LP_2}: \qquad \min \sum_{w \in W} \tilde{y}_w$$

$$\text{subject to,} \quad \forall u \in U_1 \cup U_2, \qquad \sum_{w \in W} M(u,w) \tilde{y}_w \geq r(u).$$

$$\forall w \in W, \qquad \tilde{y}_w \geq 0.$$

Figure 2.3: Covering LP to satisfy jobs in J_2

Let $\tilde{\mathbf{y}}$ denote an optimum basic feasible solution for the above LP and

let $U_3 = \{w \in W \mid \tilde{y}_w > 0\}$ denote the machines in W that are assigned a non-zero value.

Lemma 3. *There exists a feasible solution* (x^2, y^2, z^2) *to* $\mathbf{LP_1}$ *that satisfies the following properties -*

(i) $y_i^2 = 1,\ \forall i \in U_3$ *and* $y_i^2 = y_i^1,$ *otherwise.*

(ii) $z_{ij}^2 = z_{ij}^1,\ \forall i \in M$ *and* $j \in J$

(iii) $\sum_{i \in U_1 \cup U_2 \cup U_3} (x_{ij}^2 + z_{ij}^2) \geq 1,\ \forall j \in J_2$

Proof. Let $\tilde{\mathbf{y}}$ denote an optimum basic feasible solution for the $\mathbf{LP_2}$ and let $U_3 = \{w \in W \mid \tilde{y}_w > 0\}$ denote the machines in W that are assigned a non-zero value. We use $\tilde{\mathbf{y}}$ to define (x^2, y^2, z^2) that are feasible for $\mathbf{LP_1}$ as follows. We open all the machines in U_3 and keep other opening variables unchanged. We also do not change the external assignments for any job.

$$y_i^2 = \begin{cases} 1 & , \forall i \in U_3 \\ \\ y_i^1 & , \text{otherwise} \end{cases} \tag{2.8}$$

$$z_{ij}^2 = z_{ij}^1 \qquad , \forall i \in M \text{ and } \forall j \in J$$

The internal assignments are defined using U_3 as follows.

$$x^2_{ij} = \begin{cases} x^1_{ij} & ,\forall j \notin J_2 \text{ and } \forall i \in M \\ \frac{x^1_{ij}}{(f-1)y^1_i} & ,\forall j \in J_2 \text{ and } i \in \delta(j) \cap U_3 \\ 0 & ,\forall j \in J_2 \text{ and } i \in \delta(j) \cap \{W \setminus U_3\} \\ \gamma_{ij}\left(1 - \sum_{v \notin \delta(j)} z^2_{vj} - \sum_{w \in \delta(j) \cap W} x^2_{wj}\right) & ,\forall j \in J_2 \text{ and } i \in \delta(j) \cap \{U_1 \cup U_2\} \end{cases} \tag{2.9}$$

Since we have $x^2_{ij} = x^1_{ij}$ and $z^2_{ij} = z^1_{ij}$ for any job $j \notin J_2$, constraint (2.1) is satisfied for all such jobs. On the other hand, for a job $j \in J_2$, we have

$$\begin{aligned} &\sum_{i \in \delta(j)} x^2_{ij} + \sum_{i \notin \delta(j)} z^2_{ij} = \sum_{u \in \delta(j) \cap \{U_1 \cup U_2\}} x^2_{uj} + \sum_{w \in \delta(j) \cap W} x^2_{wj} + \sum_{i \notin \delta(j)} z^2_{ij} \\ &= \sum_{u \in \delta(j) \cap \{U_1 \cup U_2\}} \gamma_{uj} \left(1 - \sum_{v \notin \delta(j)} z^2_{vj} - \sum_{w \in \delta(j) \cap W} x^2_{wj}\right) + \sum_{w \in \delta(j) \cap W} x^2_{wj} + \sum_{i \notin \delta(j)} z^2_{ij} \\ &= 1 \end{aligned}$$

where the last equation follows as $\sum_{u \in \delta(j) \cap \{U_1 \cup U_2\}} \gamma_{uj} = 1$.

It can be verified from the definition that all constraints in (2.2), (5.4), (2.6) are satisfied. Also, since we have $z^2 = z^1$ and $y^2_i \geq y^1_i, \forall i \in M$, constraints (2.4) are also satisfied for all vertices. We now show that all the capacity constraints of equation (2.3) are also satisfied and hence (x^2, y^2, z^2) is a feasible LP solution.

First, we note that $y_i^2 \geq y_i^1$ for every vertex i. Also for any vertex $i \in W \setminus U_3$, we have $x_{ij}^2 \leq x_{ij}^1$ and $z_{ij}^2 = z_{ij}^1$ for all jobs j and hence the capacity constraints are trivially satisfied since (x^1, y^1, z^1) is a feasible solution.

For a vertex $w \in U_3$, we have

$$\sum_{j \in J} p_j \left(x_{wj}^2 + z_{wj}^2\right) = \sum_{j \in J_2} p_j x_{wj}^2 + \sum_{j \in J \setminus J_2} p_j x_{wj}^2 + 0 = \sum_{j \in J_2} p_j \left(\frac{x_{wj}^1}{(f-1)y_w^1}\right) + \sum_{j \in J \setminus J_2} p_j x_{wj}^1$$
$$\leq \frac{1}{(f-1)y_w^1} \left(\sum_{j \in J} p_j x_{wj}^1\right) \leq C_w = C_w y_w^2.$$

Finally for a vertex $u \in U_1 \cup U_2$, the feasibility of the covering $\mathbf{LP_2}$ guarantees that the capacity constraints are satisfied as follows -

$$\begin{aligned}
\sum_{j \in J} p_j \left(x_{uj}^2 + z_{uj}^2\right) &= \sum_{j \in J} p_j z_{uj}^2 + \sum_{j \in J \setminus J_2} p_j x_{uj}^2 + \sum_{j \in J_2} p_j x_{uj}^2 \\
&= \sum_{j \in J} p_j z_{uj}^1 + \sum_{j \in J \setminus J_2} p_j x_{uj}^1 + \sum_{j \in J_2} p_j \gamma_{uj} \left(1 - \sum_{v \notin \delta(j)} z_{vj}^2 - \sum_{w \in \delta(j) \cap W} x_{wj}^2\right) \\
&= \sum_{j \in J} p_j z_{uj}^1 + \sum_{j \in J \setminus J_2} p_j x_{uj}^1 + \sum_{j \in J_2} p_j \gamma_{uj} \left(1 - \sum_{v \notin \delta(j)} z_{vj}^2\right) - \sum_{j \in J_2} \sum_{w \in \delta(j) \cap U_3} p_j \gamma_{uj} \frac{x_{wj}^1}{(f-1)y_w^1} \\
&= \sum_{j \in J} p_j z_{uj}^1 + \sum_{j \in J \setminus J_2} p_j x_{uj}^1 + \sum_{j \in J_2} p_j \gamma_{uj} \left(1 - \sum_{v \notin \delta(j)} z_{vj}^1\right) - \sum_{w \in U_3} M(u, w)
\end{aligned}$$

But by definition of U_3 we have, $\sum_{w \in U_3} M(u, w) \geq \sum_{w \in W} M(u, w)\tilde{y}_w \geq r(u)$.

This gives us the following.

$$\sum_{j\in J} p_j \left(x^2_{uj} + z^2_{uj}\right) \leq \sum_{j\in J} p_j z^1_{uj} + \sum_{j\in J\setminus J_2} p_j x^1_{uj} + \sum_{j\in J_2} p_j \gamma_{uj} \left(1 - \sum_{v\notin\delta(j)} z^1_{vj}\right) - r(u)$$

$$= \sum_{j\in J} p_j z^1_{uj} + \sum_{j\in J\setminus J_2} p_j x^1_{uj} + \sum_{j\in J_2} p_j \gamma_{uj} \left(1 - \sum_{v\notin\delta(j)} z^1_{vj}\right) - \sum_{j\in J_2} p_j \gamma_{uj} \sum_{w\in\delta(j)\cap W} x^1_{wj}$$

$$= \sum_{j\in J} p_j z^1_{uj} + \sum_{j\in J\setminus J_2} p_j x^1_{uj} + \sum_{j\in J_2} p_j \gamma_{uj} \left(1 - \sum_{v\notin\delta(j)} z^1_{vj} - \sum_{w\in\delta(j)\cap W} x^1_{wj}\right)$$

Since (x^1, y^1, z^1) is a feasible solution, we have $\sum_{i\in\delta(j)\cap\{U_1\cup U_2\}} x^1_{ij} + \sum_{v\notin\delta(j)} z^1_{vj} + \sum_{w\in\delta(j)\cap W} x^1_{wj} \geq 1$ and hence

$$\leq \sum_{j\in J} p_j z^1_{uj} + \sum_{j\in J\setminus J_2} p_j x^1_{uj} + \sum_{j\in J_2} p_j \gamma_{uj} \left(\sum_{i\in\delta(j)\cap\{U_1\cup U_2\}} x^1_{ij}\right)$$

$$= \sum_{j\in J} p_j z^1_{uj} + \sum_{j\in J\setminus J_2} p_j x^1_{uj} + \sum_{j\in J_2} p_j x^1_{uj} \leq C_u y^1_u = C_u y^2_u$$

Hence we have shown that the solution (x^2, y^2, z^2) constructed here is a feasible solution for $\mathbf{LP_1}$. Further from (2.8)-(2.9), it can be seen that all the three conditions of the lemma are satisfied. □

The following lemma bounds the size of the set U_3 that we have opened in this stage.

Lemma 4. $|U_3| \leq (f-1)\sum_{w\in W} y^*_w + |U_1| + |U_2|$

Proof. By definition, we have $U_3 = \{w \in W \mid \tilde{y}_w > 0\}$. Hence, we have $|U_3| \leq \sum_{w\in W} \tilde{y}_w + |\mathcal{F}_{\tilde{y}}|$ where $\mathcal{F}_{\tilde{y}}$ denotes the set of machines with fractional

$\tilde{y}$ values. Inspecting the covering LP in Figure 2.3, we observe that setting $\tilde{y}_w = (f-1)y^1_w$ yields a feasible solution and hence due to optimality we have $\sum_{w \in W} \tilde{y}_w \leq (f-1) \sum_{w \in W} y^1_w = (f-1) \sum_{w \in W} y^*_w$.

Let $\mathcal{F}_{\tilde{y}} = \{w_1, w_2, \ldots, w_r\}$ be the r fractional variables in the basic feasible solution $\tilde{\mathbf{y}}$. As shown in Theorem 5.2 of Cheung et al. [50], we know that there exist r distinct machines $u_1, u_2, \ldots, u_r \in U_1 \cup U_2$ such that $\Pi_{i=1}^{r} M(u_i, w_i) \neq 0$. Hence we have, $r \leq |U_1 \cup U_2|$ and the lemma follows. □

2.3.4 Stage 3

Let $J_3 = \{j \in J \mid \sum_{i \in U_1 \cup U_2 \cup U_3} (x^2_{ij} + z^2_{ij}) < 1\}$ denote the set of jobs that are not completely satisfied by the machines opened in the previous stages. Also, for convenience, we define $W' = V_1 \setminus \{U_1 \cup U_2 \cup U_3\}$ to be the set of machines with fractional y^2 values. By the definition of J_2 and Lemma 3, we know that for all jobs $j \in J_3$, $\sum_{i \in U_1 \cup U_2} x^1_{ij} = 0$. We now setup some additional notation. For any job $j \in J_3$, define

$$require(j) = \sum_{w \in W'} x^2_{wj} + \sum_{u \in U_1} z^2_{uj}$$

Claim 2. *For every job $j \in J_3$, $require(j) > \frac{|\delta(j) \cap W'|}{f+1}$.*

Proof. For job $j \in J_3$, recall that we have

$$\sum_{i \in M} (x^*_{ij} + z^*_{ij}) \geq 1$$

Since $\sum_{i \in U_1 \cup U_2} x^*_{ij} = 0$, we can rewrite this as

$$\sum_{w \in W'} x^*_{wj} + \sum_{u \in U_3} x^*_{uj} + \sum_{u \in U_1} z^*_{uj} + \sum_{u \in V \setminus U_1} z^*_{uj} \geq 1$$

However, Claim 1 guarantees that $\sum_{u \in V \setminus U_1} z^*_{uj} = out(j) < \frac{1}{f+1}$. Further $\forall u \in U_3$, we have $x^*_{uj} \leq y^*_u < \frac{1}{f+1}$. Substituting these values, we get

$$\sum_{w \in W'} x^*_{wj} + \sum_{u \in U_1} z^*_{uj} \geq \frac{|\delta(j) \cap W'|}{f+1}$$

Since we have $z^2_{ij} = z^*_{ij}$ for all $i \in U_1$ and $x^2_{ij} = x^*_{ij}$ for all $i \in W'$, the claim follows. □

A new LP formulation

In this section we alter the assignment variables x^2_{wj} and z^2_{uj} for jobs $j \in J_3$ and machines $w \in W'$ and $u \in U_1$ while maintaining that their sum is at least $require(j)$.

Consider a bipartite graph G_1 where on the LHS we have vertices corresponding to $U_1 \cup W'$ and on the RHS we have vertices corresponding to J_3. For every job $j \in J_3$ and $w \in \delta(j) \cap W'$, we add an edge (w, j). Similarly we

add edge (u, j) for every $u \in U_1$ and $j \in J_3$. Let E denote the edge set of graph G_1.

We now consider a new LP formulation. Let variables α_{wj} and β_{uj} represent the assignment of job j to machine $w \in W'$ and $u \in U_1$ respectively. For every machine $u \in U_1$, let $\bar{C}_u$ and $\bar{B}_u$ denote the remaining capacity and bandwidth still available at u, i.e. $\bar{C}_u = (T - \sum_{j \in J} x_{uj}^2 - \sum_{j \in J \setminus J_3} z_{uj}^2)$ and $\bar{B}_u = (B - \sum_{j \in J \setminus J_3} z_{uj}^2)$. For every machine $w \in W'$, we initially set $\bar{C}_w = T$. The following feasibility LP tries to reassign the jobs in J_3 while maintaining all makespan and congestion constraints. Note that for the purposes of this LP, the y_w^2 values are considered a constant and simply setting $\alpha_{wj} = x_{wj}^2$ and $\beta_{uj} = z_{uj}^2$ is a feasible solution. Initially we set $\widetilde{W'} = W'$ and $\widetilde{U_1} = U_1$.

The following lemma follows from the sparsity of basic feasible solutions.

Lemma 5. *For any basic feasible solution (α, β) to the LP in Figure 2.4 at least one of the following hold -*

1. $\exists\ \alpha_{wj} \in \{0, y_w^2\}$ *or* $\exists\ \beta_{uj} = 0$
2. *There exists a job $j \in J_3$ or machine $w \in \widetilde{W'}$ of degree ≤ 1 in G_1*
3. *There exists a machine $u \in \widetilde{U_1}$ of degree ≤ 3 in G_1*

Proof. Let (α, β) be a basic feasible solution to $\mathbf{LP_3}$. Let us assume that

$$\forall j \in J_3, \qquad \sum_{w \in W' | (w,j) \in E} \alpha_{wj} + \sum_{u \in U_1 | (u,j) \in E} \beta_{uj} = require(j)$$

$$\forall w \in \widetilde{W'}, \qquad \sum_{j \in J_3 | (w,j) \in E} p_j \alpha_{wj} \leq \bar{C}_w y_w^2 \tag{2.10}$$

$$\forall u \in \widetilde{U_1}, \qquad \sum_{j \in J_3 | (u,j) \in E} p_j \beta_{uj} \leq \bar{C}_u$$

$$\forall u \in \widetilde{U_1}, \qquad \sum_{j \in J_3 | (u,j) \in E} d_j \beta_{uj} \leq \bar{B}_u$$

$$\forall (w,j) \in E : w \in W', \qquad 0 \leq \alpha_{wj} \leq y_w^2 \tag{2.11}$$

$$\forall (u,j) \in E : u \in U_1, \qquad \beta_{uj} \geq 0 \tag{2.12}$$

Figure 2.4: **LP$_3$** : Feasibility LP to reassign jobs in J_3

$0 < \alpha_{wj} < y_w^2, \forall (w,j) \in E$ and $\beta_{uj} > 0, \forall (u,j) \in E$ since otherwise, we are done.

Since none of the constraints of type (2.11) and (2.12) are tight, the total number of tight constraints is at most $|J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}|$. Since (α, β) is a basic feasible solution, the number of non-zero variables is bounded by the number linearly independent tight constraints and we have the following.

$$|E| \leq |J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}| \tag{2.13}$$

Now suppose for the sake of contradiction, that every job $j \in J_3$ and machine $w \in \widetilde{W'}$ has degree at least 2 and every machine $u \in \widetilde{U_1}$ has degree at least 4 in G_1. Since G_1 is a bipartite graph with edge set E, we obtain the following.

$$|E| \geq |J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}| \tag{2.14}$$

From Equations (2.13) and (2.14), we have $|E| = |J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}|$ and every machine $i \notin \widetilde{W'} \cup \widetilde{U_1}$ has degree zero. Consequently E is the edge set of a bipartite graph where J_3 is one side of the bipartition and $\widetilde{W'} \cup \widetilde{U_1}$ forms the other side. Since all the constraints defined by vertices of a bipartite graph cannot be linearly independent, we have at most $|J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}| - 1$

linearly independent tight constraints. The basic feasibility of the solution then implies that $|E| \leq |J_3| + |\widetilde{W'}| + 2|\widetilde{U_1}| - 1$ leading to a contradiction. $\square$

Iterative Rounding

We create another bipartite graph G_2 with the same vertex set as G_1 but with no edges initially. Intuitively, while graph G_1 represents the assignments that are yet to be processed in this stage, G_2 indicates the assignments that are carried forward to the next stage.

Algorithm: While graph G_1 is not empty, apply the following steps as long as one of them applies; after each application re-solve the LP on the remaining variables and constraints to obtain a new basic feasible solution.

1. Solve $\mathbf{LP_3}$ to obtain a basic feasible solution (α, β). By Lemma 5, one of the following conditions must hold.

2. If $\alpha_{wj} = 0$ or $\beta_{uj} = 0$ for some edge $e \in E$, remove e from E. Set $x^3_{wj} = 0$ or $z^3_{uj} = 0$.

3. If $\alpha_{wj} = y^2_w$ for some edge $(w, j) \in E$, remove (w, j) from E and add it to graph G_2. Assign $x^3_{wj} = \alpha_{wj}$. For the LP, set $\bar{C_w} = \bar{C_w} - p_j$ and update $require(j) = require(j) - \alpha_{wj}$.

4. If $\exists w \in \widetilde{W'}$ of degree ≤ 1 in G_1: Let (w, j) be the incident edge. Delete w from $\widetilde{W'}$, i.e. we drop the makespan constraint (Eq (2.10)) associated with w. Since at most one additional job j can be assigned to machine w, the total load at w is at most $T + \max_j p_j$.

5. If $\exists u \in \widetilde{U_1}$ of degree ≤ 3 in G_1: For every neighbor j of u, do the following. Remove edge (u, j) from E and add it to G_2. Assign $z^3_{uj} = \beta_{uj}$. We say that the job j is accounted for by u. For the LP, set $require(j) = require(j) - \beta_{uj}$.

   Delete vertex u from G_1. As a result u can account for at most 3 jobs.

6. If there exists a vertex $j \in J_3$ of degree 1 in G_1:

   (a) Let (w, j) be the incident edge where $w \in W'$: Now, by claim 2, we know that initially $require(j) > \frac{|\delta(j) \cap W'|}{f+1}$. Since we have $\alpha_{w'j} \leq y^2_{w'} < \frac{1}{f+1}$ for all $w' \in W'$, this implies that we must have at least one edge (u, j) in G_2 where $u \in U_1$. In other words, job j has been accounted for by some vertex $u \in U_1$. We call u *responsible* for w and open w and finalize its assignments, i.e. set $y^3_w = 1$ and $x^3_{wj'} = \alpha_{wj'}$ for all edges (w, j') in G_1. Delete vertices j and w from G_1 and set $require(j') = require(j') - \alpha_{wj'}$ for all

$j' \neq j$ incident to w in G_1.

(b) Let (u, j) be the incident edge where $u \in U_1$: Move edge (u, j) to G_2 and assign $z^3_{uj} = \beta_{uj}$. For the LP, delete vertex j from G_2 and set $\bar{C}_u = \bar{C}_u - p_j\beta_{uj}$ and $\bar{B}_u = \bar{B}_u - d_j\beta_{uj}$.

7. If G_1 is not empty, return to Step 1.

Structure at the end of Stage 3

Let U_4 denote the set of vertices that we open in Step 6a. We set $y^3_u = y^2_u$ for all $u \notin U_4$. Similarly, the assignment variables for $j \notin J_3$ remain unchanged, i.e., $x^3_{ij} = x^2_{ij}$ and $z^3_{ij} = z^2_{ij}$. From the description of the algorithm, it is easy to verify that (x^3, y^3, z^3) satisfies constraints (2.1)-(2.2) and (2.4)-(2.6). Step 4 also guarantees that the makespan constraint (2.3) is satisfied approximately, i.e., we have

$$\sum_{j \in J} p_j(x^3_{ij} + z^3_{ij}) \leq Ty^3_i + \max_j p_j$$

The following claim bounds the number of vertices that we open in this step.

Claim 3. $|U_4| \leq 3|U_1|$.

Proof. Each vertex $w \in U_4$ has a vertex $u \in U_1$ responsible for it. Step 5 ensures that a vertex $u \in U_1$ can be responsible for at most 3 vertices. □

Let $\tilde{W} \subseteq W'$ denote the set of vertices not yet open with at least one edge incident on it in G_2. Also let $J_4 = \{j \in J \mid \sum_{i \in U_1 \cup U_2 \cup U_3 \cup U_4} (x^3_{ij} + z^3_{ij}) < 1\}$ denote the set of jobs that are not yet completely satisfied by the integrally open vertices. So at the end we have the following properties.

1. For every $j \in J_4$, $\delta(j) \cap \tilde{W} \neq \phi$

2. For every $j \in J_4$ and $w \in \delta(j) \cap \tilde{W}$, $x^3_{wj} > 0 \Rightarrow x^3_{wj} = y^3_{wj}$. This is because we only move an edge (w, j) to G_2 if the associated constraint is tight.

3. For every $j \in J_4$, define $require'(j) = \sum_{w \in \tilde{W}} x^3_{wj} + \sum_{u \in U_1} z^3_{uj}$. As in Claim 2, we get $require'(j) > \frac{|\delta(j) \cap \tilde{W}|}{f+1}$.

2.3.5 Stage 4

Property 2 above ensures that if we decide to open a new machine $w \in \tilde{W}$, then we can assign all jobs j with nonzero x^3_{wj} value completely to it without violating its makespan constraint (beyond the violation in (2.3.4)). Note that these jobs do not contribute to the congestion at w. We will now rewrite a

new linear program to modify the y values. We no longer need to maintain the makespan constraints on machines in $\tilde{W}$ as they are all maintained automatically.

A Final LP formulation

We define a variable ω_{uj} for every $u \in U_1$ and $j \in J_4$ such that $(u, j) \in E_2$ where E_2 is the edge set of graph G_2. We define a new variable λ_w for every $w \in \tilde{W}$ with $y^3_w > 0$ that represents whether machine w should be opened. As before, let $\bar{C}_u$ and $\bar{B}_u$ denote the remaining capacity and bandwidth of a machine $u \in U_1$. Figure 2.5 shows the complete LP formulation where we aim to minimize the number of vertices that we open from $\tilde{W}$ while maintaining that all jobs get satisfied and the other constraints remain satisfied. Note that setting $\lambda_w = y^3_w = y^*_w$ and $\omega_{uj} = z^3_{uj}$ is a feasible solution.

Iterative Rounding - Final LP

Let (λ, ω) be an optimal basic feasible solution for the LP in Figure 2.5. In this section, we now define (x^4, y^4, z^4) with integral y values. We initialize $(x^4, y^4, z^4) = (x^3, y^3, z^3)$ and always maintain $x^4_{wj} \in \{0, y^4_w\}$ for all $w \in \tilde{W}$. We now define some more terminology for convenience.

$\mathbf{LP_4}$: $\min \sum_{w \in \tilde{W}} \lambda_w$

subject to,

$\forall j \in J_4$, $\sum_{w \in W' | (w,j) \in E_2} \lambda_w + \sum_{u \in U_1 | (u,j) \in E_2} \omega_{uj} \geq require'(j)$

$\forall u \in \widetilde{U_1}$, $\sum_{j \in J_4 | (u,j) \in E_2} p_j \omega_{uj} \leq \bar{C}_u$

$\forall u \in \widetilde{U_1}$, $\sum_{j \in J_4 | (u,j) \in E_2} d_j \omega_{uj} \leq \bar{B}_u$

$\forall (u,j) \in E_2$ where $u \in U_1$, $\omega_{uj} \geq 0$

$\forall w \in \tilde{W}$, $0 \leq \lambda_w \leq 1$

Figure 2.5: Final LP to satisfy jobs in J_4

For any job $j \in J_4$

- If $\sum_{w \in W' | (w,j) \in E_2} \lambda_w \geq require'(j)$, we say j is *fully satisfied internally.*

- If $\sum_{u \in U_1 | (u,j) \in E_2} \omega_{uj} \geq require'(j)$, we say j is *fully satisfied externally.*

- Otherwise, j is satisfied partially internally and partially externally.

The rounding now proceeds as follows -

1. We apply each of the following three steps as long as any one of them applies and re-solve the LP after each step.

   (a) We open (set $y^4_w = 1$) all $w \in \tilde{W}$ with $\lambda_w \geq \frac{1}{(f+1)}$. We also set $x^4_{wj} = 1$ for all $j \in J_4$ such that $(w, j) \in E_2$. Now since $require'(j) > \frac{|\delta(j) \cap \tilde{W}|}{f+1}$, any job j that was fully satisfied internally, is now satisfied.

   (b) If there is a job j that is fully-satisfied externally, we simply allocate it externally according to ω_{uj}, i.e., set the $z^4_{uj} = \omega_{uj}$ and appropriately adjust the capacity $\bar{C}_u$ and bandwidth $\bar{B}_u$ of $u \in U_1$. Job j is then removed from J_4.

   (c) If there is a machine $u \in U_1$ that has at most three jobs assigned to it, i.e. $\omega_{uj} > 0$. For each such job j, do the following. Since j must be satisfied partially internally and partially externally, there

must exist a vertex $w \in \tilde{W} \cap \delta(j)$ such that $\lambda_w > 0$. We open (i.e. set $y^4_w = 1$) the vertex w and say that u is *responsible* for w. The newly opened vertex w can now satisfy all jobs j' incident on it (set $x^4_{wj'} = 1$) and thus we remove all such jobs from J_4. Remove machine w from $\tilde{W}$.

Delete vertex u from G_2. As a result u is responsible for at most 3 newly opened machines.

2. Since none of the above steps apply, each job $j \in J_4$ must have at least two incident non-zero variables and machine $u \in U_1$ must have at least four associated non-zero variables. Therefore, the number of non-zero variables is $\geq 2|J_4|$ and also $\geq 4|U_1|$. However, the number of tight linearly independent constraints is at most $|J_4| + 2|U_1|$. Since we computed a basic feasible solution, it must hold that $|J_4| = 2|U_1|$.

   Now for every job $j \in J_4$, we open at most one vertex w from $\tilde{W} \cap \delta(j)$ such that $\lambda_w > 0$. In this process we satisfy all the jobs and open at most $|J_4| = 2|U_1|$ new vertices from $\tilde{W}$. For all remaining $w' \in \tilde{W}$, set $y^4_{w'} = 0$.

Claim 4. *Let $U_5 \subset \tilde{W}$ denote the set of vertices that we opened in this step.*

Then we have

$|U_5| \leq (f+1)\sum_{w\in\tilde{W}} y^*_w + 3|U_1|$.

Proof. We add at most $3|U_1|$ vertices in step 1(c) and (2) as each new vertex w that we open in step 1(c) has a $u \in U_1$ that is responsible for it. Recall that after a vertex u is marked responsible in step 1(c), it is removed from consideration and not charged again in step 2. In Step 1(a), we add a vertex to U_5 only if $\lambda_w \geq \frac{1}{f+1}$. Thus we have, $|U_5| \leq (f+1)\sum_{w\in\tilde{W}} \lambda_w + 3|U_1|$.

However, as setting $\lambda_w = y^3_w = y^*_w$ is a feasible LP solution, we must have $\sum_{w\in\tilde{W}} \lambda_w \leq \sum_{w\in\tilde{W}} y^*_w$ and the claim follows. □

The following lemma is the result of the previous 4 stages.

Lemma 6. *There exists a polynomial time algorithm to obtain* (x, y, z) *such that*

(i) y *is integral*

(ii) (x, y, z) *satisfy constraints (2.1)-(2.2) and (2.4)-(2.6)*

(iii) For any open machine i, $\sum_{j\in J} p_j(x_{ij} + z_{ij}) \leq T + \max_j p_j$

(iv) $\sum_{i\in M} y_i \leq 8(f+1)\sum_{i\in M} y^*_i + 2$

Proof. The final solution (x^4, y^4, z^4) obtained at the end of Stage 4 satisfies the first three properties of the lemma. Recall that constraint (2.3) may be

violated by an additive factor of $\max_j p_j$ since we drop the capacity constraint in Step 4 in Stage 3. We now bound the cost of this solution as follows.

$$\sum_{i \in M} y_i^4 = |U_1| + |U_2| + |U_3| + |U_4| + |U_5|$$

Substituting from Lemma 2, Lemma 4, Claim 3, and Claim 4,

$$\leq 2|U_1| + 2|U_2| + (f-1) \sum_{w \in W} y_w^* + 3|U_1| + (f+1) \sum_{w \in \tilde{W}} y_w^* + 3|U_1|$$

$$\leq 8(f+1) \sum_{i \in M} y_i^* + 2$$

□

We next adapt the rounding algorithm for the generalized assignment problem by Shmoys and Tardos [53] to obtain an integral solution to yield Theorem 4.

Theorem 4. *For the network-aware machine activation problem with unit activation costs, if there exists a solution with assignment cost A, makespan T, and congestion B, then there is a polynomial time algorithm that finds a schedule with assignment cost at most $(8f+9)A$, makespan at most $5T$, and congestion at most $4B$ where $f = \max_j |\delta(j)|$.*

Proof. Let (x, y, z) denote the solution that satisfies the conditions of Lemma 6 and let $T' = T + \max_j p_j$. Let O denote the set of open machines, i.e.

$O = \{i \in M \mid y_i = 1\}$. From Lemma 6, we have $|O| \leq (8f+8)\sum_{i \in M} y_i^* + 2 \leq (8f+9)A$ where A is the activation cost of the optimal schedule. The last inequality follows as we can safely assume that $A \geq 2$.

We now construct an instance of the generalized assignment problem with two dimensional job sizes as follows. Let O denote the set of available machines. For each job $j \in J$, let $p_{ij} = p_j, \forall i \in O$, $d_{ij} = d_j, \forall i \in O \setminus \delta(j)$, and $d_{ij} = 0, \forall i \in O \cap \delta(j)$. Our task is to find an assignment of jobs such that for each machine i, the total processing requirement of assigned jobs is at most T' and the total data requirement of assigned jobs is at most B. The following linear program captures these constraints and relaxes the integrality condition. Initially we set $E = \{(i,j) \mid i \in O, j \in J\}$, $\tilde{O} = O$, $T'_i = T'$, and $B_i = B$.

$$\mathbf{LP_{GAP}}: \quad \sum_{i:(i,j)\in E} x'_{ij} = 1 \qquad ,\forall j \in J \qquad (2.15)$$

$$\sum_{j:(i,j)\in E} p_{ij} x'_{ij} \leq T'_i \qquad ,\forall i \in \tilde{O} \qquad (2.16)$$

$$\sum_{j:(i,j)\in E} d_{ij} x'_{ij} \leq B_i \qquad ,\forall i \in \tilde{O} \qquad (2.17)$$

$$x'_{ij} \geq 0 \qquad ,\forall (i,j) \in E$$

From Lemma 6, we observe that setting $x'_{ij} = x_{ij} + z_{ij}$ yields a feasible solution to $\mathbf{LP_{GAP}}$. We now describe an iterative rounding procedure that yields integral assignments and only violates constraints (2.16) and (2.17) by an additive factor of $3\max_j p_j$ and $3\max_j d_j$ respectively.

Let x' denote a basic feasible solution for $\mathbf{LP_{GAP}}$. We first claim that either (i) $\exists x'_{ij} \in \{0,1\}$ or (ii) $\exists i \in \tilde{O}$ such that $deg_E(i) < 4$. Suppose we have $0 < x'_{ij} < 1$ for all $(i,j) \in E$. If not, then the claim is trivially true. Hence there are at most $|J| + 2|\tilde{O}| - 1$ linearly independent tight constraints. The minus one above follows since not all vertex constraints associated with a bipartite graph can be independent. Since x' is a basic feasible solution, we must have $|E| \leq |J| + 2|\tilde{O}| - 1$. Constraint (2.15) requires that $deg_E(j) \geq 2$. Now suppose for contradiction that $deg_E(i) \geq 4$ for all $i \in \tilde{O}$. Then by a simple counting argument, we obtain $|E| \geq |J| + 2|\tilde{O}|$ leading to a contradiction.

The following iterative rounding algorithm now yields the integral assignment with the desired properties.

1. Set $F = \emptyset, T'_i = T', B_i = B, \tilde{O} = O$.

2. Solve $\mathbf{LP_{GAP}}$ to obtain a basic feasible solution x'.

3. (a) If $\exists x'_{ij} = 0$: Delete (i,j) from E.

(b) If $\exists x'_{ij} = 1$: Move (i, j) from E to F. Set $T'_i = T'_i - p_{ij}$ and $B_i = B_i - d_{ij}$. Remove j from J.

(c) Else $\exists i \in \tilde{O}$ such that $deg_E(i) < 4$: Delete i from $\tilde{O}$. Since at most 3 more jobs can be assigned to machine i, constraints (2.16) and (2.17) at machine i can only be violated by an additive factor of $3 \max_j p_j$ and $3 \max_j d_j$ respectively.

4. If $E \neq \emptyset$, return to Step 2.

The set F obtained by the above rounding procedure denotes the set of integral assignments. By the discussion above, at any machine i we have the following.

$$\sum_{j:(i,j)\in F} p_j \leq T' + 3 \max_j p_j \leq T + 4 \max_j p_j \leq 5T$$

Similarly, we also have that $\sum_{j:(i,j)\in F} d_{ij} \leq B + 3 \max_j d_j \leq 4B$ for all machines $i \in O$. □

2.4 Network-Aware Machine Activation for Unit Jobs

In this section, we show that our algorithm yields better approximation guarantees when the jobs are restricted to have unit processing and data requirements, i.e. we have $p_j = d_j = 1$ for all jobs $j \in J$. In fact, our results hold for a slightly more general version where each machine i has an integer capacity to process C_i jobs and an integral available bandwidth B_i.

Analogous to Figure 2.2, we define the linear programming relaxation of the network-aware machine activation problem with unit jobs as shown in Figure 2.6.

Let (x^*, y^*, z^*) denote an optimal feasible solution to $\mathbf{LP_{unit}}$ in Figure 2.6. We first claim that when constructing a feasible integral solution for the network-aware machine activation problem with unit jobs, we only need integrality of the y variables and not of x or z. The proof of the following lemma follows from the integrality of network flows.

Lemma 7. *If (x, y, z) is a feasible solution for $\mathbf{LP_{unit}}$ in Figure 2.6 and y is integral, then we can efficiently find integral x' and z' such that (x', y, z') is also feasible.*

$$\mathbf{LP_{unit}}: \qquad \min \sum_{i \in M} y_i$$

subject to,

$$\forall j \in J, \qquad \sum_{i \in \delta(j)} x_{ij} + \sum_{i \notin \delta(j)} z_{ij} \geq 1$$

$$\forall i \in M, \text{ and } j \in J, \qquad x_{ij} + z_{ij} \leq y_i,$$

$$\forall i \in M, \qquad \sum_{j \in J} (x_{ij} + z_{ij}) \leq C_i y_i \qquad (2.18)$$

$$\text{and,} \qquad \sum_{j \in J} z_{ij} \leq B_i y_i,$$

$$\forall i \in M, \qquad 0 \leq y_i \leq 1.$$

$$\forall i \in M \text{ and } j \in J, \qquad x_{ij} \geq 0 \text{ and } z_{ij} \geq 0.$$

Figure 2.6: LP Relaxation for instance $\mathcal{I}$ of network-aware machine activation with unit jobs.

Proof. Since the y variables are integral, we can construct the following flow network.

- The set of nodes is $M \cup J \cup \{r, s, t\}$ where s and t denote a source and sink node respectively while r denotes a dummy node whose role will be apparent soon.
- For all $i \in M$, add arcs (i, j) with capacity 1 if and only if $i \in \delta(j)$.
- For all $i \in M$, add arcs (s, i) with capacity C_i if and only if $y_i = 1$.
- For all $j \in J$, add arcs (j, t) with capacity 1.
- For all $i \in M$, add arcs (i, r) with capacity B_i if and only if $y_i = 1$.
- For all $j \in J$, add arcs (r, j) with capacity 1.

Since (x, y, z) is a feasible solution, one can construct a flow of value J by sending a flow of x_{ij} through edge (i, j) and a flow of z_{ij} through edges (i, r) and (r, j). But since all the capacities are integers, there must exist an integral flow of the same value and hence we can recover a feasible integral solution (x', y, z') using a simple maximum flow computation. □

The rest of the algorithm is similar to that in the previous section. In this

section, we only highlight the differences between the two. Stages 1 and 2 remain identical to those seen in Sections 2.3.2 and 2.3.3.

2.4.1 Stage 3

Since all jobs have unit processing and data requirements, the feasibility LP that we consider in this stage has a simpler structure than that for the general case. As in Section 2.3.4, we consider a bipartite graph G_1 where on the LHS we have vertices corresponding to $U_1 \cup W'$ and on the RHS we have vertices corresponding to J_3. Let α_{uj} be the assignment variable that denotes the fraction of job $j \in J_3$ that is assigned to machine $u \in U_1$. Since the makespan and congestion constraints must be satisfied at vertex u, the assignment variables must satisfy $\sum_{j \in J_3} \alpha_{uj} \leq C_u - \sum_{j \in J} x_{uj}^2 - \sum_{j \in J \setminus J_3} z_{uj}^2$ as well as $\sum_{j \in J_3} \alpha_{uj} \leq B_u - \sum_{j \in J \setminus J_3} z_{uj}^2$.

As a result, we define $\bar{C_u} = \min(C_u - \sum_{j \in J} x_{uj}^2 - \sum_{j \in J \setminus J_3} z_{uj}^2, B_u - \sum_{j \in J \setminus J_3} z_{uj}^2)$ to be the residual capacity / bandwidth at vertex u. Figure 2.7 denotes the new feasibility LP to capture the reassignments in this stage.

The following lemma in analogous to Lemma 5 and follows from the sparsity of basic feasible solutions.

Lemma 8. *For any basic feasible solution (α, β) to the LP in Figure 2.7 at*

$$\forall j \in J_3, \qquad \sum_{w \in W' | (w,j) \in E} \alpha_{wj} + \sum_{u \in U_1 | (u,j) \in E} \beta_{uj} = require(j)$$

$$\forall w \in \widetilde{W'}, \qquad \sum_{j \in J_3 | (w,j) \in E} p_j \alpha_{wj} \leq \bar{C}_w y_w^2$$

$$\forall u \in \widetilde{U_1}, \qquad \sum_{j \in J_3 | (u,j) \in E} p_j \beta_{uj} \leq \bar{C}_u$$

$$\forall (w,j) \in E : w \in W', \qquad 0 \leq \alpha_{wj} \leq y_w^2 \tag{2.19}$$

$$\forall (u,j) \in E : u \in U_1, \qquad \beta_{uj} \geq 0 \tag{2.20}$$

Figure 2.7: $\mathbf{LP_{unit3}}$: Feasibility LP to reassign jobs in J_3

least one of the following hold -

1. $\exists\ \alpha_{wj} \in \{0, y_w^2\}$ *or* $\exists\ \beta_{uj} = 0$

2. *There exists a job* $j \in J_3$ *or machine* $i \in \widetilde{U_1} \cup \widetilde{W'}$ *of degree* ≤ 1 *in* G_1.

Proof. Let (α, β) be a basic feasible solution to $\mathbf{LP_{unit3}}$. Let us assume that $0 < \alpha_{wj} < y_w^2, \forall (w,j) \in E$ and $\beta_{uj} > 0, \forall (u,j) \in E$ since otherwise, we are done.

Since none of the constraints of type (2.19) and (2.20) are tight, the total number of tight constraints is at most $|J_3| + |\widetilde{W'}| + |\widetilde{U_1}|$. Since (α, β) is a basic feasible solution, the number of non-zero variables is bounded by the number linearly independent tight constraints and we have the following.

$$|E| \leq |J_3| + |\widetilde{W'}| + |\widetilde{U_1}| \tag{2.21}$$

Now suppose for the sake of contradiction, that every job $j \in J_3$ and machine $i \in \widetilde{U_1} \cup \widetilde{W'}$ has degree at least 2. Since G_1 is a bipartite graph with edge set E, we obtain the following.

$$|E| \geq |J_3| + |\widetilde{W'}| + |\widetilde{U_1}| \tag{2.22}$$

From Equations (2.21) and (2.22), we have $|E| = |J_3| + |\widetilde{W'}| + |\widetilde{U_1}|$ and every machine $i \notin \widetilde{W'} \cup \widetilde{U_1}$ has degree zero. Consequently E is the edge set

of a bipartite graph where J_3 is one side of the bipartition and $\widetilde{W'} \cup \widetilde{U_1}$ forms the other side. Since all the constraints defined by vertices of a bipartite graph cannot be linearly independent, we have at most $|J_3| + |\widetilde{W'}| + |\widetilde{U_1}| - 1$ linearly independent tight constraints. The basic feasibility of the solution then implies that $|E| \leq |J_3| + |\widetilde{W'}| + |\widetilde{U_1}| - 1$ leading to a contradiction. $\square$

Iterative Rounding

The iterative rounding algorithm remains almost identical to that in Section 2.3.4. In Step 4, when we drop the capacity constraint (delete w from $\widetilde{W'}$), the integrality of the capacity $\bar{C_w}$ guarantees that assignments remain feasible for $\mathbf{LP_{unit}}$.

Similarly, in Step 5, now since there must exist a vertex $u \in \widetilde{U_1}$ of degree at most 1, the vertex u can account for at most 1 job. The following claim follows as every machine opened in this stage has a unique machine in U_1 responsible for it.

Claim 5. $|U_4| \leq |U_1|$

2.4.2 Stage 4

Once again, the final linear program that we consider in Stage 4 has a simpler structure as all jobs have unit processing and data requirements. Figure 2.8 illustrates the final linear programming relaxation adapted for unit jobs. As in the Stage 3, the makespan and congestion constraints at machine $u \in \widetilde{U_1}$ are collapsed into one constraint.

$\mathbf{LP_4}$: $\min \sum_{w \in \tilde{W}} \lambda_w$

subject to,

$\forall j \in J_4$, $\sum_{w \in W' | (w,j) \in E_2} \lambda_w + \sum_{u \in U_1 | (u,j) \in E_2} \omega_{uj} \geq require'(j)$

$\forall u \in \widetilde{U_1}$, $\sum_{j \in J_4 | (u,j) \in E_2} \omega_{uj} \leq \bar{C_u}$

$\forall (u,j) \in E_2$ where $u \in U_1$, $\omega_{uj} \geq 0$

$\forall w \in \tilde{W}$, $0 \leq \lambda_w \leq 1$

Figure 2.8: Final LP to satisfy jobs in J_4

Iterative Rounding - Final LP

Let (λ, ω) be an optimal basic feasible solution for the LP in Figure 2.8. We use notation as defined in Section 2.3.5.

The rounding now proceeds as follows -

1. We apply each of the following three steps as long as any one of them applies and re-solve the LP after each step.

   (a) We open (set $y^4_w = 1$) all $w \in \tilde{W}$ with $\lambda_w \geq \frac{1}{(f+1)}$. We also set $x^4_{wj} = 1$ for all $j \in J_4$ such that $(w, j) \in E_2$. Now since $require'(j) > \frac{|\delta(j) \cap \tilde{W}|}{f+1}$, any job j that was fully satisfied internally, is now satisfied.

   (b) If there is a job j that is fully-satisfied externally, we simply allocate it externally according to ω_{uj}, i.e., set the $z^4_{uj} = \omega_{uj}$ and appropriately adjust the capacity $\bar{C}_u$ and bandwidth $\bar{B}_u$ of $u \in U_1$. Job j is then removed from J_4.

   (c) If there is a machine $u \in U_1$ that has at most one job assigned to it, i.e. $\omega_{uj} > 0$ for only one job j. Since j must be satisfied partially internally and partially externally, there must exist a vertex $w \in \tilde{W} \cap \delta(j)$ such that $\lambda_w > 0$. We open (i.e. set $y^4_w = 1$)

the vertex w and say that u is *responsible* for w. The newly opened vertex w can now satisfy all jobs j' incident on it (set $x^4_{wj'} = 1$) and thus we remove all such jobs from J_4. Remove machine w from $\tilde{W}$.

Delete vertex u from G_2. As a result u is responsible for at most one newly opened machine.

2. Since none of the above steps apply, each job $j \in J_4$ must have at least two incident non-zero variables and machine $u \in U_1$ must have at least two associated non-zero variables. Therefore, the number of non-zero variables is $\geq 2|J_4|$ and also $\geq 2|U_1|$. However, the number of tight linearly independent constraints is at most $|J_4| + |U_1|$. Since we computed a basic feasible solution, it must hold that $|J_4| = |U_1|$.

   Now for every job $j \in J_4$, we open at most one vertex w from $\tilde{W} \cap \delta(j)$ such that $\lambda_w > 0$. In this process we satisfy all the jobs and open at most $|J_4| = |U_1|$ new vertices from $\tilde{W}$. For all remaining $w' \in \tilde{W}$, set $y^4_{w'} = 0$.

Claim 6. *Let $U_5 \subset \tilde{W}$ denote the set of vertices that we opened in this step. Then we have*

$|U_5| \leq (f+1)\sum_{w\in\tilde{W}} y^*_w + |U_1|$.

Proof. We add at most $|U_1|$ vertices in step 1(c) and (2) as each new vertex w that we open in step 1(c) has a $u \in U_1$ that is responsible for it. Recall that after a vertex u is marked responsible in step 1(c), it is removed from consideration and not charged again in step 2. In Step 1(a), we add a vertex to U_5 only if $\lambda_w \geq \frac{1}{f+1}$. Thus we have, $|U_5| \leq (f+1)\sum_{w\in\tilde{W}} \lambda_w + |U_1|$.

However, as setting $\lambda_w = y^3_w = y^*_w$ is a feasible LP solution, we must have $\sum_{w\in\tilde{W}} \lambda_w \leq \sum_{w\in\tilde{W}} y^*_w$ and the claim follows. □

The following lemma is analogous to Lemma 6 and is the result of the previous four stages.

Lemma 9. *There exists a polynomial time algorithm to obtain (x, y, z) such that*

(i) y is integral,

(ii) (x, y, z) is a feasible solution for $\mathbf{LP_{unit}}$ *in Figure 2.6, and*

(iii) $\sum_{i\in M} y_i \leq 4(f+1)\sum_{i\in M} y^*_i + 2$.

Proof. The final solution (x^4, y^4, z^4) obtained at the end of Stage 4 satisfies the first two properties of the lemma. Recall that constraint (2.18) is satisfied

even though we drop the capacity constraint in Step 4 in Stage 3 as the capacity is an integer and jobs are unit. We now bound the cost of this solution as follows.

$$\sum_{i \in M} y_i^4 = |U_1| + |U_2| + |U_3| + |U_4| + |U_5|$$

Substituting from Lemma 2, Lemma 4, Claim 5, and Claim 6,

$$\leq 2|U_1| + 2|U_2| + (f-1) \sum_{w \in W} y_w^* + |U_1| + (f+1) \sum_{w \in \tilde{W}} y_w^* + |U_1|$$

$$\leq 4(f+1) \sum_{i \in M} y_i^* + 2$$

□

Theorem 5. *For the network-aware machine activation problem with unit activation costs and unit processing and data requirements for all jobs, if there exists a solution with assignment cost A, makespan T, and congestion B, then there is a polynomial time algorithm that finds a schedule with assignment cost at most $(4f+5)A$, makespan at most T, and congestion at most B where $f = \max_j |\delta(j)|$.*

Proof. The theorem follows from Lemma 7 and Lemma 9. □

2.5 LP Rounding for General Network-Aware Machine Activation

In this section, we consider the general network aware machine activation problem with arbitrary machine activation costs, processing times and data requirements. We show that a simple randomized rounding scheme (also used by Khuller et al. [9]) yields a schedule that violates the capacity (makespan) and bandwidth constraints by a factor of $O(\log n)$ and has activation cost at most $O(\log n)$ times the optimal.

Figure 2.9 shows the natural LP relaxation of the network-aware machine activation problem. As in the previous problem, the variables y_i indicate whether machine i has been opened and variables x_{ij} and z_{ij} denote the assignment.

2.5.1 Rounding Algorithm

Let (x^*, y^*, z^*) denote the optimum solution of the above LP relaxation. The randomized rounding algorithm is as follows -

1. Independently for each (unopened) machine i, open i with probability y_i^*.

$$\min \sum_{i \in M} a_i y_i$$

subject to, $\forall j \in J, \quad \sum_{i \in \delta(j)} x_{ij} + \sum_{i \notin \delta(j)} z_{ij} \geq 1.$

$\forall i \in M,$ and $j \in J, \quad x_{ij} + z_{ij} \leq y_i,$

$\forall i \in M, \quad \sum_{j \in J : i \in \delta(j)} p_{ij} x_{ij} + \sum_{j \in J : i \notin \delta(j)} p_{ij} z_{ij} \leq C_i y_i.$

and, $\quad \sum_{j \in J : i \notin \delta(j)} d_{ij} z_{ij} \leq B_i y_i,$

$\forall i \in M, \quad y_i \geq 0.$

$\forall i \in M,$ and $j \in J, \quad x_{ij} \geq 0$ and $z_{ij} \geq 0.$

Figure 2.9: LP Relaxation for instance $\mathcal{I}$ of the network-aware machine activation problem

2. For each newly opened machine i, set $X_{ij} = \frac{x^*_{ij}}{y^*_i}$ for all incident jobs, i.e. jobs such that $(i \in j)$. Similarly, set $Z_{ij} = \frac{z^*_{ij}}{y^*_i}$ for jobs that are not incident on i. Independently for each job, assign job j to machine i with probability X_{ij} if $i \in j$ and with probability Z_{ij} otherwise.

3. If there exists an unassigned job, repeat from Step 1.

4. If a job is assigned to multiple machines, choose one arbitrarily.

2.5.2 Analysis

In each iteration of the algorithm, we open machine i with probability y^*_i. Hence, the expected activation cost of machines opened in any iteration is at most the cost of the optimal LP solution. The following lemmas bound the number of iterations (approximation factor for the activation cost) and the processing and data loads on each machine.

Lemma 10. *The rounding algorithm terminates in $O(\log n)$ iterations with high probability.*

Proof. Consider any job j and a machine $i \in j$. In a single iteration, we have $Pr(\text{job } j \text{ is not assigned to machine } i) \leq (1 - y^*_i) + y^*_i(1 - \frac{x^*_{ij}}{y^*_i}) = 1 - x^*_{ij}$.

Similarly, for a machine $i \notin j$, we have that $Pr(\text{job } j \text{ is not assigned to machine } i) \leq 1 - z^*_{ij}$.

Consequently, we have that

$Pr(\text{job } j \text{ is not assigned in an iteration}) \leq \prod_{i \in j}(1 - x^*_{ij}) \prod_{i \notin j}(1 - z^*_{ij}) \leq (1 - \frac{1}{m})^m \leq \frac{1}{e}$

It is now easy to observe that the probability that job j is not assigned after $2 \ln n$ iterations is at most $\frac{1}{n^2}$. Therefore, by union bound all the jobs are assigned after $2 \ln n$ iterations with probability at least $1 - \frac{1}{n}$. □

Lemma 11. *For any machine i, the total processing time of jobs assigned to it is $O(\log n)C_i$ with high probability. Similarly, the total data requirement of jobs assigned to i is $O(\log n)B_i$ with high probability.*

Proof. Consider any iteration h. Let X^h_{ij} denote the value of X_{ij} and Z^h_{ij} denote the value of Z_{ij} at iteration h . For each open machine i and every job j, define a random variable as follows -

$$\hat{L}^h_{ij} = \begin{cases} \frac{p_{ij}}{C_i} & \text{, if } j \text{ is assigned to } i \\ 0, & \text{, otherwise} \end{cases}$$

Now consider $L_i = \sum_{j,h} \hat{L}^h_{ij}$. By linearity of expectation and the definition of $\hat{L}^h_{ij}$, we have

$$\mathbb{E}[L_i] = \frac{\sum_h \left(\sum_{j \ni i} p_{ij} X_{ij}^h + \sum_{j \not\ni i} p_{ij} Z_{ij}^h \right)}{C_i} \leq \sum_h 1 \leq \Theta(\log n)$$

Now if P_i denotes the total processing time of jobs assigned to i, we have $\mathbb{E}[P_i] = C_i \mathbb{E}[L_i] = \Theta(C_i \log n)$. Hence, the claim follows by a standard application of the Chernoff bound.

Similarly, for each open machine i and every job j that is not incident on i, define a random variable -

$$\hat{M}_{ij}^h = \begin{cases} \frac{d_{ij}}{B_i} & \text{, if } j \text{ is assigned to } i \\ 0, & \text{, otherwise} \end{cases}$$

Now, consider $M_i = \sum_{j,h} \hat{M}_{ij}^h$. Once again, by the linearity of expectation we have

$$\mathbb{E}[M_i] = \frac{\sum_h \sum_{j \not\ni i} d_{ij} Z_{ij}^h}{B_i} \leq \sum_h 1 \leq \Theta(\log n)$$

Now, if D_i denotes the total data requirement of external jobs assigned to i, we have $\mathbb{E}[D_i] = B_i \mathbb{E}[M_i] = \Theta(B_i \log n)$. The claim follows by the Chernoff bound. □

Chapter 3

Scheduling Co-flows

Large scale data centers have emerged as the dominant form of computing infrastructure over the last decade. The success of data-parallel computing frameworks such as MapReduce [6], Hadoop [7], Google DataFlow [54], Spark [55] has led to a proliferation of applications that are designed to alternate between computation and communication stages. Typically, the intermediate data generated by a computation stage needs to be transferred across different machines during a communication stage for further processing. For example, there is a "Shuffle" phase between every consecutive "Map" and "Reduce" phases in MapReduce. With an increasing reliance on parallelization, these communication stages are responsible for a large amount of data transfer in a

datacenter. Since an application can only proceed with its next computation stage after *all* its transfer requirement is met, traditional flow-based scheduling metrics are not appropriate to measure the quality of a schedule. Chowdhury and Stoica [13] introduce co-flows as an effective networking abstraction to represent the collective communication requirements of a job. In this chapter, we consider the problem of scheduling co-flows to minimize the weighted completion time and give improved approximation algorithms.

In Section 3.1, we formally define the co-flow scheduling problem and provide an overview of prior work. Section 3.2 highlights the connection between co-flow scheduling and well-studied concurrent open shop scheduling problem. Our improved approximation guarantees in Sections 3.3 - 3.5 stem from this connection. In Section 3.6, we demonstrate the efficacy of our approximation algorithm on synthetic datasets via a preliminary experimental analysis.

3.1 Problem Setting

As in prior work [14, 16], a datacenter is modeled as a single $m \times m$ *non-blocking* switch, i.e., there are m *input ports* and m *output ports*. There are

capacity constraints on all ports. For simplicity, we assume that all ports have unit capacity - i.e., at most one unit of data can be transferred through any port at a time.

A co-flow is defined as a collection of parallel flow demands that share a performance goal. Each co-flow j has weight w_j, release time r_j, and is represented as a $m \times m$ integer matrix $D^j = [d^j_{io}]$ where the entry d^j_{io} represents the number of data units that must be transferred from input port i to output port o for co-flow j. Figure 3.1 shows a single co-flow over a 2×2 switch. For instance, the co-flow depicted needs to transfer 2 units of data from input 1 to output 1 and 3 units of data from input 1 to output 2.

A co-flow j is available to be scheduled at its release time r_j and is said to be completed when all the flows in the matrix D^j have been scheduled. More formally, the completion time C_j is defined as the earliest time by which d^j_{io} units of its data have been transferred from input port i to output port o for every i and o. We assume that time is slotted and data transfer within the switch is instantaneous. Since each input port i can transmit at most one unit of data and each output port o can receive at most one unit of data in each time slot, a feasible schedule for a single time slot is described by a matching. Our goal is to find a feasible, integral schedule that minimizes the

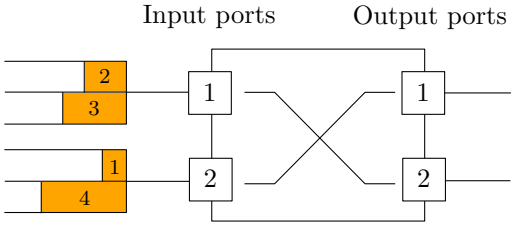

Figure 3.1: An example co-flow over a 2×2 switch.

total, weighted completion time of the co-flows, i.e. minimize $\sum_j w_j C_j$.

3.1.1 Related Work

Co-flow scheduling has been a topic of active research [14, 15, 16, 17] since its introduction and is also the basis for a successful practical network scheduler [14]. In the online setting, jobs arrive online and the scheduler needs to make decisions without knowledge of future arrivals. Chowdhury and Stoica [15] design *Aalo*, a system for scheduling co-flows without prior information. Although their algorithms do not admit provable guarantees, it is shown to perform very well in practice. Even in the offline setting, when all jobs are known in advance, no $O(1)$ approximation algorithm was known until recently. Qiu, Stein and Zhong [16] obtain a deterministic $\frac{67}{3}$ approximation and a randomized $(9 + \frac{16\sqrt{2}}{3})$ approximation for the problem of minimizing the weighted completion time. For the special case when all release times are

zero, Qiu et al. [16] demonstrate improved bounds of $\frac{64}{3}$ (deterministic) and $(8 + \frac{16\sqrt{2}}{3})$ (randomized).

3.2 Connection to Concurrent Open Shop

The co-flow scheduling problem as described above generalizes the well-studied concurrent open shop problem [56, 57, 58, 59, 60]. In the concurrent open shop problem, we have a set of m machines and each job j with weight w_j is composed of m tasks $\{t_i^j\}_{i=1}^m$, one on each machine. Let p_i^j denote the processing requirement of task t_i^j. A job j is said to be completed once all its tasks have completed. Any machine can perform at most one unit of processing at a time. The objective is to find a feasible schedule that minimizes the total weighted completion time of jobs. An LP-relaxation using completion time variables yields a 2-approximation algorithm for concurrent open shop scheduling when all release times are zero [57, 58, 59] and a 3-approximation algorithm for arbitrary release times [58, 59]. Mastrolilli et al. [56] show that a simple greedy algorithm also yields a 2-approximation for concurrent open shop without release times using primal-dual techniques.

It can be seen that the concurrent open shop problem is a special case

of co-flow scheduling when the demand matrices D^j for all co-flows j are diagonal [14, 16]. At first glance, it appears that co-flow scheduling is much harder than concurrent open shop: For example, while concurrent open shop always admits an optimal permutation schedule [56], such a property is not true for co-flows [14]. Indeed, even without release times, the best known approximation algorithm for scheduling co-flows has an approximation factor of $\frac{64}{3}$ [16], in contrast to the many 2-approximations known for the concurrent open shop problem.

3.3 Our Contribution and Techniques

The main algorithmic contribution of this paper is the following improved approximation guarantee for the offline co-flow scheduling problem. Our results significantly improve upon the approximation ratios obtained by Qiu et al. [16].

Theorem 6. *There exists a deterministic, polynomial time 4-approximation algorithm for co-flow scheduling without release times.*

Theorem 7. *There exists a deterministic, polynomial time 12-approximation algorithm for co-flow scheduling with release times.*

Qiu et al. [16] suggest a two-staged approach to finding good co-flow schedules. In the first stage, they find a good permutation of the co-flows by rounding an interval-indexed LP relaxation similar to that of Wang and Cheng [60]. The second stage groups together co-flows based on this permutation and obtains a feasible co-flow schedule.

In this paper, we show that the first stage in the above approach can be replaced by a reduction to the concurrent open shop scheduling problem. This explicit reduction allows us to use a 2-approximation algorithm for the concurrent open shop scheduling problem (3-approximation for the case with release times). We then show that greedily scheduling co-flows in order yields a feasible co-flow schedule such that the completion time of co-flow j is at most twice the completion time of the corresponding job in the concurrent open shop instance.

3.4 Preemptive Concurrent Open Shop with Release Times

Before we discuss our approximation algorithms for the co-flow scheduling problem, we first focus on the concurrent open shop scheduling problem

when job preemptions are allowed, i.e., a machine is allowed to interrupt the processing of any job j, start processing some other job j' and then return to the processing j without any penalty. When all jobs have release time zero, one can easily convert any optimal preemptive schedule to a non-preemptive schedule of the same cost by a simple swapping argument. Indeed, it can be shown that not only is the optimal schedule non-preemptive, but there is an optimal schedule in which every machine processes the jobs in the same permutation [56]. However, when jobs have different release times, the optimal preemptive schedule may be better than the optimal non-preemptive schedule. In this section, we show that the schedule obtained by the algorithm of Garg et al. [58] and Leung et al. [59] is a 3-approximation even with respect to the optimal preemptive schedule.

Consider the following linear programming relaxation for the concurrent open shop problem with release times and job preemptions are allowed.

$$\mathbf{LP_1} : \text{minimize } \sum_j w_j C_j$$

subject to

$$C_{i,j} \geq r_j + p_i^j \qquad ,\forall i \in M, \forall j \in J$$

$$C_j \geq C_{i,j} \qquad ,\forall i \in M, \forall j \in J$$

$$\sum_{j \in S} p_i^j C_{i,j} \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_i^j\right)^2 + \sum_{j \in S}(p_i^j)^2\right] \qquad ,\forall i \in M, \forall S \subseteq J$$

We observe that $\mathbf{LP_1}$ defined above is a valid relaxation for the concurrent open shop problem even when preemption of jobs is allowed. The first two sets of constraints ensure that the completion time of a job is at least its release time plus its processing requirement on any machine. The third set of constraints are standard in the scheduling literature [61] and only require that the completion time of a job j on machine i is at least the sum of processing times (on machine i) of all jobs k that complete before j in the schedule. Since this constraint too must be satisfied by any optimal (preemptive) schedule, $\mathbf{LP_1}$ is a valid linear programming relaxation.

The following lemma by Garg et al. [58] and Leung et al. [59] thus implies a 3-approximation for concurrent open shop scheduling with release times even with respect to an optimal preemptive schedule.

Lemma 12. *Let $\bar{C}_1, \bar{C}_2, \ldots, \bar{C}_n$ denote an optimal solution for* $\mathbf{LP_1}$*. Then there exists a polynomial time algorithm that obtains a feasible concurrent open shop schedule such that $\tilde{C}_j \leq 3\bar{C}_j$, where $\tilde{C}_j$ denotes the completion time of job j in the schedule.*

3.5 Improved Algorithms for Scheduling Co-flows

We first introduce some notation to facilitate the following discussion. For every co-flow j and input port i, we define the load $L_i^j = \sum_{o=1}^{m} d_{io}^j$ to be the total amount of data that co-flow j needs to transmit through port i. Similarly, we define $L_o^j = \sum_{i=1}^{m} d_{io}^j$ for every co-flow j and output port o.

Our algorithm consists of two stages. In the first stage, we obtain a permutation of the co-flows via a reduction to the concurrent open shop scheduling problem. In the second stage, we show how one can obtain provably good co-flow schedules using the permutation obtained earlier. We now describe the two stages separately in the following sections.

3.5.1 Reduction to Concurrent Open Shop:

Let $\mathcal{I}$ denote an instance of the co-flow scheduling problem. We now construct an instance $\mathcal{I}'$ of the concurrent open shop scheduling problem on $2m$ machines (one for each port) and n jobs (one for each co-flow). For a job j, set $p_s^j = L_s^j$, i.e., the processing requirement of job j on a machine s is set to be the load of the co-flow j on the corresponding port. Let $OPT(\mathcal{I})$ denote the cost of an optimal co-flow schedule for instance $\mathcal{I}$ and $OPT(\mathcal{I}')$ denote the cost of an optimal concurrent open shop schedule for the instance $\mathcal{I}'$.

Lemma 13. $OPT(\mathcal{I}') \leq OPT(\mathcal{I})$

Proof. Let S^* denote an optimal co-flow schedule for instance $\mathcal{I}$. For a co-flow j and port s, let T_s^j denote the set of time slots when data corresponding to co-flow j is being processed (either input or output) at port s as per schedule S^*. Now note that processing one unit of the corresponding job j on machine s in the concurrent open shop instance $\mathcal{I}'$ at all times in T_s^j leads to a feasible schedule. We remark that the concurrent open shop schedule obtained here is preemptive. As we show in Section 3.4, the LP-based approximation algorithms [58, 59] for concurrent open shop scheduling also yield guarantees with respect to the optimal *preemptive* schedule. □

Let $\bar{C}_j$ denote the completion time of job j in an α-approximate schedule for the concurrent open shop instance $\mathcal{I}'$. Further, let us assume without loss of generality that the co-flows are ordered so that the following holds.

$$\bar{C}_1 \leq \bar{C}_2 \leq \ldots \leq \bar{C}_n \tag{3.1}$$

Note that since a job j can only be processed after it is released the following holds for every job (co-flow) j.

$$\bar{C}_j \geq r_j + \max_s p_s^j \tag{3.2}$$

Similarly, Equation (3.1) guarantees that for any job k, all jobs $j \leq k$ must be processed completely before time $\bar{C}_k$. Consequently, the following holds for every job k. If all the jobs have zero release times, then a job k can be processed on a machine s as soon as the previous jobs have finished processing and the following holds with equality.

$$\bar{C}_k \geq \max_s \sum_{j \leq k} p_s^j \tag{3.3}$$

Corollary 1. $\sum_j w_j \bar{C}_j \leq 3 \times OPT(\mathcal{I})$. *Further if all release times are zero, then* $\sum_j w_j \bar{C}_j \leq 2 \times OPT(\mathcal{I})$

Proof. The concurrent open shop scheduling problem with release times has well-known 3-approximation algorithms [59, 58] that also yield a 2-

approximation when all release times are zero. Combining any of these algorithms with Lemma 13 yields the corollary. □

3.5.2 Scheduling Co-Flows Without Release Times

In the rest of this section, we assume that all the co-flows are released at time zero, i.e. $r_j = 0, \forall j$. Further, without loss of generality, we assume that the co-flows are ordered as per Equation (3.1). We now introduce some notation to facilitate the discussion in this section.

Change of Notation

In Section 3.1, we represent a co-flow j as an $m \times m$ integer matrix D^j. Equivalently, a co-flow j can be represented by a weighted, bipartite graph $G_j = (I, O, E_j)$ where the set of input ports (I) and the set of output ports (O) form the two sides of the bipartition and an edge (i, o) with a weight of d^j_{io} represents that the co-flow j requires d^j_{io} units of data to be transferred from input port i to output port o. Looking at a co-flow from such a graphical perspective guides the algorithm developed in this section.

Representing a co-flow as a bipartite graph simplifies some of the notation that we've seen previously. For instance, for any co-flow j, the load of j on

port i is simply the weighted degree of vertex i in graph G_j, i.e., we have the following.

$$L_i^j = deg_{G_j}(i)$$

For any co-flow k, let $\widetilde{G_k} = (I, O, \bigcup_{j=1}^{k} E_j)$ denote the bipartite graph obtained by the union of the first k co-flows. For any vertex $s \in I \cup O$, $deg_{\widetilde{G_k}}(s)$ denotes the total processing requirement of the first k jobs on machine s in the concurrent open shop instance. Thus, from Equation (3.3) (with equality as release times are all zero), we have the following.

$$\bar{C_k} = \Delta(\widetilde{G_k}) = \max_{s \in I \cup O} deg_{\widetilde{G_k}}(i)$$

We now restate Corollary 1 in terms of the weighted degree of the cumulative graphs $\widetilde{G_k}$.

Corollary 2. $\sum_j w_j \Delta(\widetilde{G_j}) \leq 2 \times OPT(\mathcal{I})$.

Finding Matching Schedules

Recall that in any feasible co-flow schedule, every input port and every output port can process at most one unit of data at any time, i.e., the data processed

in any time slot can be described by a matching. The following lemma by Qiu et al. [16] follows from repeated applications of Hall's theorem and bounds the number of time steps it takes to schedule a single co-flow.

Lemma 14. *There exists a polynomial time algorithm that schedules co-flow* j *in* $\max\{\max_i L_i^j, \max_o L_o^j\} = \max_s p_s^j$ *time slots, were it to be scheduled alone, i.e. the edges of the bipartite graph* G_j *can be partitioned into* $\Delta(G_j)$ *matchings.*

We now show that a simple greedy algorithm that processes all co-flows sequentially in the permutation given by Equation (3.1) yields a provably good co-flow schedule. By Lemma 14, if we naïvely process each co-flow sequentially and find a matching schedule for each co-flow separately, then co-flow k completes by time $\sum_{j=1}^{k} \Delta(G_j) >> \Delta(\widetilde{G_k})$. The key idea behind our algorithm is that we can safely "move" edges back from a graph G_k to graph G_j where $k > j$ as long as the maximum degree of G_j does not increase. Effectively, we can now find a matching schedule that not only processes the co-flow j in time $\Delta(G_j)$ but also processes some edges of the co-flow k. Algorithm 1 describes the complete pseudocode.

In Algorithm 1, we first initialize all $G'_j = G_j$ for all co-flows j. For all $k > j$, we then move edges from graph G'_k to G'_j as long as the maximum

Input : Bipartite graphs $G_1, G_2, \ldots, G_n$ representing n co-flows

Output : Feasible co-flow schedule

// Initialization;

for $j = 1, 2, \ldots, n$ **do**
  $G'_j = G_j$

// Moving Back Edges;

for $j = 1, 2, \ldots, n-1$ **do**
  for $k = j+1, j+2, \ldots, n$ **do**
    for $e \in G'_k$ **do**
      if $\Delta(G'_j + e) = \Delta(G'_j)$ **then**
        $G'_j = G'_j + e$;
        $G'_k = G'_k - e$;

// Scheduling;

for $j = 1, 2, \ldots, n$ **do**
  Schedule G'_j using Lemma 14 in $\Delta(G'_j)$ time steps;

Algorithm 1: Greedily Scheduling Co-Flows

degree of G'_j does not increase. Intuitively, in doing so we can process parts of co-flow k while we schedule co-flow j in $\Delta(G'_j)$ time. The following lemma bounds the sum of degrees of the graphs $G'_j, \forall j \leq k$ in terms of the degree of the union of the k graphs.

Lemma 15. *For all* $k \in \{1, 2, \ldots n\}$, $\sum_{j \leq k} \Delta(G'_j) \leq 2\Delta(\widetilde{G_k})$.

Proof. Since the graphs G'_j keep changing during the course of the algorithm, for the sake of analysis, let $G_{j|k}$ where $j < k$ be the state of the graph G'_j immediately after we have transferred all possible edges from G'_k to G'_j. Let $G_{k|k}$ denote the graph G'_k after all possible edges have been moved to G'_{k-1}. Since, we move edges back to a graph G'_j only if it doesn't increase the maximum degree, we have the following.

$$\Delta(G'_j) = \Delta(G_{j|k}) \text{ for all } j \leq k. \tag{3.4}$$

For any $k \in \{1, 2, \ldots, n\}$, consider the set S of graphs $G_{1|k}, G_{2|k}, \ldots G_{k|k}$. Let u be a vertex of maximum degree in $G_{k|k}$, i.e. $deg_{G_{k|k}}(u) = \Delta(G_{k|k})$ and consider any edge $e = (u, v)$ incident on u in $G_{k|k}$. Since edge (u, v) was not moved to any of the graphs $G_{j|k}$ for $j < k$, we must have that either u or v had maximum degree in $G_{j|k}$. Let $S_u = \{G_{j|k} \mid deg_{G_{j|k}}(u) = \Delta(G_{j|k})\}$ and

$S_v = \{G_{j|k} \mid deg_{G_{j|k}}(v) = \Delta(G_{j|k})\}$ denote the subsets of the graph where vertex u or v has the maximum degree respectively.

Now, let $\hat{G}_k = \bigcup_{j=1}^{k} G_{j|k}$ be the union of the graphs $G_{j|k}$. Since $\hat{G}_k$ contains all edges from the graphs $G_1, \ldots, G_k$ and no edges from graphs G_l for $l > k$, $\hat{G}_k$ is identical to the cumulative graph of the first k co-flows. In particular, we have the following.

$$\Delta(\hat{G}_k) = \Delta(\widetilde{G_k}) \tag{3.5}$$

Let us now consider the maximum degree of the graph $\hat{G}_k$.

$$\begin{aligned}
\Delta(\hat{G}_k) &\geq \max\left\{deg_{\hat{G}_k}(u), deg_{\hat{G}_k}(v)\right\} \\
&\geq \max\left\{\sum_{G\in S_u} deg_G(u), \sum_{G\in S_v} deg_G(v)\right\} \\
&= \max\left\{\sum_{G\in S_u} \Delta(G), \sum_{G\in S_v} \Delta(G)\right\}
\end{aligned}$$

From Equation (3.4), we have the following.

$$\sum_{j\leq k} \Delta(G'_j) = \sum_{j\leq k} \Delta(G_{j|k}) = \sum_{G\in S} \Delta(G)$$

However, since $S_u \cup S_v = S$ as either u or v has maximum degree in every graph in S, we get the following.

$$\begin{aligned}
\sum_{j\leq k} \Delta(G'_j) &\leq 2\max\left\{\sum_{G\in S_u} \Delta(G), \sum_{G\in S_v} \Delta(G)\right\} \\
&\leq 2\Delta(\hat{G}_k) = 2\Delta(\widetilde{G_k})
\end{aligned}$$

where the last equality follows from Equation (3.5). □

The following theorem shows that Algorithm 1 combined with the permutation of co-flows obtained via the reduction to the concurrent open-shop problem yields a 4-approximation algorithm for co-flow scheduling without release times and thus proves Theorem 6.

Theorem 8. *For any co-flow k, let $C_k(alg)$ denote the completion time of co-flow k when scheduled as per Algorithm 1. Then $\sum_k w_k C_k(alg) \leq 4OPT(\mathcal{I})$.*

Proof. Algorithm 1 sequentially schedules the graphs G'_j obtained after moving the edges using Lemma 14. The completion time of co-flow k is thus at most the sum of the degrees of graphs G'_j where $j < k$.

$$C_k(alg) \leq \sum_{j \leq k} \Delta(G'_j) \leq 2\Delta(\widetilde{G_k})$$

where the second inequality is due to Lemma 15. Corollary 2 now yields the desired theorem.

$$\sum_k w_k C_k(alg) \leq \sum_k w_k 2\Delta(\widetilde{G_k}) \leq 4OPT(\mathcal{I}).$$

□

3.5.3 Scheduling Co-flows With Release Times

In this section, we consider co-flow scheduling where each co-flow j has an arbitrary release time r_j. Following the strategy by Qiu et al. [16], we obtain a feasible schedule for all co-flows by grouping together the co-flows j that have similar $\bar{C}_j$ values. We divide the time slots into geometrically increasing intervals - let $I_1 = [1]$ and $I_l = (2^{l-2}, 2^{l-1}]$ for $l > 1$ denote these intervals. Let S_l denote the set of co-flows j whose $\bar{C}_j$ lies in the interval I_l.

$$S_l = \{j \in J | \bar{C}_j \in I_l\}$$

We then group together all the co-flows that lie in a group and then schedule the groups sequentially as shown in Algorithm 2.

for $l = 1, 2, \ldots$ **do**

Wait until the last co-flow in S_l is released AND all co-flows in S_{l-1} have finished (whichever is later).

Group all co-flows in S_l and schedule as per Lemma 14.

Algorithm 2: Grouping Algorithm to Schedule Co-Flows

Analysis: Let $\hat{C}_l$ denote the time by which all co-flows in S_l have been scheduled by the above algorithm.

Claim 7. $\hat{C}_l \leq 2 \times 2^{l-1} = 2^l$ *for every group* S_l.

Proof. We prove by induction. For group S_1, we start executing the schedule at $\max_{j \in S_1} r_j \leq \max_{j \in S_1} \bar{C}_j \leq 2^{1-1} = 1$ and the schedule takes time at most $\bar{C}_k \leq 2^{1-1} = 1$ where k is the last co-flow in the group. So the base case is true.

Now assume that the claim is true for some group S_l. As per the algorithm, the co-flows in group S_{l+1} start executing at $\max\{\hat{C}_l, \max_{j \in S_{l+1}} r_j\}$ whichever is later. By induction hypothesis, we are guaranteed that $\hat{C}_l \leq 2^l$. Also $\max_{j \in S_{l+1}} r_j \leq \max_{j \in S_{l+1}} \bar{C}_j \leq 2^l$ where the first inequality follows from Equation (3.2). Thus the co-flows in group S_{l+1} start executing latest at time 2^l. By Lemma 14, all these co-flows require at most $\max_s \sum_{j \in S_{l+1}} p_s^j \leq \bar{C}_k \leq 2^l$ time units to complete where k in the last co-flow in S_{l+1}. The first inequality above follows from Equation (3.3). As a result, all the co-flows in this group are scheduled by time $2^l + 2^l = 2^{l+1}$. And thus the claim follows by induction. □

Claim 8. *For any co-flow j, let $C_j(alg)$ denote the completion time of co-flow j as per the algorithm. Then $C_j(alg) < 4\bar{C}_j$.*

Proof. Consider any co-flow j, and let l be such that $j \in S_l$. Hence we have $\bar{C}_j > 2^{l-2}$. By Claim 7, we have

$$C_j(alg) \leq \hat{C}_l \leq 2^l = 4 \times 2^{l-2} < 4\bar{C}_j$$ □

Claim 8 and Corollary 1 then imply the following theorem.

Theorem 9. *There exists a deterministic, polynomial time 12-approximation algorithm for co-flow scheduling with release times.*

3.6 Experimental Analysis

In this section we perform a preliminary experimental analysis to evaluate the performance of the approximation algorithm developed in Section 3.5.

As we discussed in the previous section, algorithms for co-flow scheduling consist of two stages - first an *ordering stage* that finds a permutation of the co-flows, and then a *scheduling stage* that processes the co-flows in that permutation to find a feasible schedule. In our experiments, we consider different algorithms and heuristics for both the stages to study the effect of our proposed algorithm on the two stages separately.

In our experiments, we consider different ordering heuristics for the first stage including the one obtained via a reduction to the concurrent open shop scheduling problem. For the scheduling stage, we use both Algorithm 1 as well as the grouping strategy proposed by Qiu et al. [16].

3.6.1 Datasets

A recent preprint by Qiu, Stein and Zhong [62] proposed a distribution over random co-flow instances to evaluate the performance of different co-flow scheduling algorithms. We generate synthetic datasets by following the methodology described therein. We create 30 instances, each having $n = 160$ co-flows over a network with $m = 16$ inputs and outputs. The first 5 instances represent workloads with sparse co-flows and each co-flow only has m flows, i.e., the matrix D^j for any co-flow j only has m non-zero entries. Instances $6 - 10$ represent workloads with dense co-flows and every co-flow here consists of m^2 flows. For instances $11 - 30$, each co-flow consists of some u flows where u is an integer chosen uniformly at random from the range $\{m, m + 1, \ldots, m^2\}$. Given the number k of flows in any co-flow j, k pairs of input and output ports are chosen at random, i.e., k entries in the $m \times m$ matrix D^j are chosen uniformly at random. For each pair (i, o) that is selected, an integer processing requirement $d^j_{i,o}$ is randomly selected from the uniform distribution on $\{1, 2, \ldots, 100\}$. For the purposes of these experiments, we only consider co-flows with zero release times.

3.6.2 Ordering Heuristics

In addition to ordering of co-flows obtained via a reduction to the concurrent open-shop scheduling problem (Section 3.5.1), we consider different simple greedy ordering heuristics.

1. **Random (RAND):** The co-flows are ordered via a random permutation.

2. **Shortest Total Processing Time First (STPT):** The co-flows are arranged in non-decreasing order of their total processing requirement, i.e., $\sum_i \sum_o d_{io}^j$.

3. **Shortest Maximum Processing Time First (SMPT):** The co-flows are arranged in non-decreasing order of their maximum processing requirement at any port, i.e. $\max\left\{\max_i L_i^j, \max_o L_o^j\right\}$

4. **Shortest Maximum Completion Time First (SMCT):** For this heuristic, we treat every input and output port as an independent machine and solve the single machine scheduling problem at each machine separately. For a machine s, we order the co-flows (jobs) in order of non-decreasing L_s^j where L_s^j is the load of co-flow j on port s and compute the completion time of co-flow j on machine s as $C_s(j)$. The

SMCT heuristic then orders the co-flows in non-decreasing order of their maximum completion times, i.e. $\max_s C_s(j)$.

5. **Concurrent Open-Shop Scheduling (COSS):** We use the reduction from Section 3.5.1 to obtain an instance of the concurrent open shop scheduling problem. We use a primal-dual 2-approximation algorithm by Mastrolilli et al. [56] to solve the concurrent open shop instance. The co-flows are then ordered in non-decreasing order of their completion times in the obtained schedule.

3.6.3 Scheduling Strategies

Given an ordering of the co-flows, we have multiple strategies to obtain an actual feasible schedule of the co-flows. For instance, Qiu et al. [16] propose grouping together co-flows that have similar aggregate demand and scheduling the group together using Lemma 14. In contrast, in Section 3.5.2, we present an algorithm that consolidates the co-flows by moving back edges between co-flows as long as the maximum load of a co-flow does not increase and then schedules the consolidated co-flows using Lemma 14. In our experiments, we consider the following different scheduling strategies.

1. **Greedy Scheduling:** The co-flows are scheduled using Algorithm 1.

2. **Grouping with backfilling:** Co-flows with similar aggregate loads are grouped together and scheduled sequentially using Algorithm 2. When the grouped co-flows are scheduled using Lemma 14, the decomposition into matchings may introduce unforced idle time since all machines in the group may not have identical loads (i.e. the graph may not be regular). If while processing group S_l, the schedule matches input i and output o even though S_l has no demand between those ports, we *move back* the edge (i, o) from other co-flow groups $S_{l'}$ $(l' > l)$. It is clear that such backfilling can only help to improve the average completion time.

3.6.4 Experimental Results

We consider 10 different co-flow scheduling algorithms - the 5 ordering heuristics described in Section 3.6.2, followed by the 2 scheduling strategies described in Section 3.6.3. Our 4-approximation algorithm in Section 3.5 corresponds to COSS ordering followed by the Greedy scheduling strategy. We compare the performance of all the 10 algorithms on each of the 30 randomly generated synthetic instances.

Tables 3.1 and 3.2 show the average completion time of the co-flows for

each of the 30 instances. All the values are normalized with respect to the COSS Ordering and Greedy scheduling strategy. Table 3.1 compares the performance of the five different ordering strategies when the co-flows are scheduled greedily as per Algorithm 1. We observe that the Concurrent Open-Shop Scheduling based ordering yields gains of up to 8% over the other greedy ordering heuristics and up to 17% improvement over random ordering. Although the STPT ordering sometimes performs better, the loss in quality is never over 2%. As shown in Table 3.2, when co-flows are scheduled using the grouping strategy of Qiu et al. [16], the average completion time of jobs is up to 66% higher when using the COSS ordering.

3.6.5 Conclusions

Our preliminary experimental analysis demonstrates the important of intelligent ordering schemes to obtain good co-flow schedules. We observe that especially for sparse co-flows, a good ordering of co-flows can lead to substantial savings in the average completion times. We also show that scheduling the co-flows sequentially using Algorithm 1 leads to significantly better schedules as compared to the grouping strategy of Qiu et al. [16]. Thus our algorithm not only yields an improved approximation ratio but also yields substantial

Instance	# of flows in each co-flow	RAND	STPT	SMPT	SMCT	COSS
1	m	1.176	1.045	1.058	1.014	1
2	m	1.132	1.026	1.089	1.003	1
3	m	1.110	1.018	1.058	1.030	1
4	m	1.129	1.054	1.056	1.018	1
5	m	1.180	1.061	1.080	1.024	1
6	m^2	1.026	1.000	1.005	1.005	1
7	m^2	1.018	0.991	1.003	0.997	1
8	m^2	1.023	0.996	1.008	1.005	1
9	m^2	1.012	0.986	1.001	1.001	1
10	m^2	1.016	0.998	1.003	1.002	1
11	$Unif(m, m^2)$	1.006	0.986	1.000	0.993	1
12	$Unif(m, m^2)$	1.015	0.992	1.003	0.998	1
13	$Unif(m, m^2)$	1.191	1.059	1.082	1.021	1
14	$Unif(m, m^2)$	1.018	0.990	0.998	0.993	1
15	$Unif(m, m^2)$	1.015	0.983	1.001	0.989	1
16	$Unif(m, m^2)$	1.046	0.994	1.020	1.002	1
17	$Unif(m, m^2)$	1.007	0.997	1.000	0.995	1
18	$Unif(m, m^2)$	1.009	0.990	1.000	0.999	1
19	$Unif(m, m^2)$	1.042	1.017	1.024	1.003	1
20	$Unif(m, m^2)$	1.014	0.996	1.011	0.997	1
21	$Unif(m, m^2)$	1.049	1.006	1.032	0.998	1
22	$Unif(m, m^2)$	1.008	0.991	1.003	0.995	1

Instance	# of flows in each co-flow	RAND	STPT	SMPT	SMCT	COSS
1	m	1.808	1.540	1.527	1.552	1.613
2	m	1.651	1.522	1.580	1.594	1.664
3	m	1.627	1.633	1.584	1.561	1.654
4	m	1.807	1.564	1.587	1.545	1.643
5	m	1.832	1.514	1.509	1.526	1.622
6	m^2	1.356	1.336	1.333	1.340	1.385
7	m^2	1.360	1.330	1.332	1.334	1.386
8	m^2	1.348	1.333	1.336	1.335	1.394
9	m^2	1.359	1.342	1.328	1.335	1.376
10	m^2	1.360	1.342	1.344	1.331	1.380
11	$Unif(m, m^2)$	1.325	1.301	1.318	1.300	1.371
12	$Unif(m, m^2)$	1.370	1.326	1.324	1.326	1.397
13	$Unif(m, m^2)$	1.625	1.473	1.552	1.590	1.727
14	$Unif(m, m^2)$	1.397	1.363	1.377	1.360	1.502
15	$Unif(m, m^2)$	1.391	1.356	1.355	1.348	1.423
16	$Unif(m, m^2)$	1.512	1.482	1.435	1.385	1.611
17	$Unif(m, m^2)$	1.313	1.308	1.318	1.304	1.350
18	$Unif(m, m^2)$	1.352	1.331	1.323	1.327	1.387
19	$Unif(m, m^2)$	1.425	1.416	1.427	1.441	1.576
20	$Unif(m, m^2)$	1.316	1.324	1.324	1.300	1.347
21	$Unif(m, m^2)$	1.389	1.389	1.350	1.322	1.451
22	$Unif(m, m^2)$	1.331	1.297	1.287	1.303	1.368

gains in practice. Finally, we note that although the greedy ordering heuristics such as shortest processing time first (STPT), shortest maximum processing time first (SMPT) and shortest maximum completion time first (SMCT) seem to perform very well on the random instances, none of these heuristics are guaranteed to perform well on all instances. Indeed, Leung, Li and Pinedo [59] showed that these heuristics are only m-approximation algorithms for the related concurrent open-shop scheduling problem.

Chapter 4

Firewall Placement

This chapter considers the Firewall Placement problem in datacenters. A cloud service provider often needs to place firewalls at strategic nodes in its network in order to filter the communication between different client-owned virtual machines. Given a datacenter network, and the communication demands between virtual machines, the firewall placement problem is to determine the minimum number of firewalls necessary so that all the demands can be routed via a firewall and all bandwidth constraints on the network links are respected.

In Section 4.1, we describe the problem setting and formally define the different problems we consider. Section 4.2 reviews prior work on related

problems and we list our results in Section 4.3. In Sections 4.4-4.8, we present our algorithms for the soft and hard capacitated versions of the firewall placement problem and associated hardness results.

4.1 Setting and Problem Definitions

We restrict our attention to datacenters that are arranged in a tree layout. Here, the physical network is a rooted tree $T = (V, E)$ where the internal vertices denote various switches and routers while the leaf vertices denote the physical servers that are available to host the virtual machines. Every edge $e \in E$ in the tree has a bandwidth $b(e)$ that is simultaneously available both in the uplink and downlink directions. Each physical server (leaf of the tree) accommodates a number of virtual machines and we assume that this assignment of virtual machines to the servers is fixed. Finally, we are given communication requests between pairs of virtual machines. We model these requests as a demand graph $H = (L, D)$ where $L \subset V$ denotes the set of leaves of the tree T. An edge $(s, t) \in D$ indicates that one must route unit flow from s to t. We allow D to contain self-loops and multi-edges.

Our objective is to place the minimum number of firewalls on the nodes

of T so that all the demands can be simultaneously satisfied where a demand (s, t) is satisfied if we can route unit flow from s to some firewall f and also from f to the sink t. The bandwidth $b(e)$ of an edge upper bounds the number of demands that may be routed through that edge. In addition, each firewall has a capacity C_v that represents the maximum number of demands it can satisfy. Formally, the capacitated firewall placement is defined as follows.

Definition 1. Firewall Placement Problem*: We are given a tree $T = (V, E)$ rooted at $r \in V$ with bidirectional edges having bandwidth $b(e) = b(u, v) = b(v, u)$. Let $H = (L, D)$ be a directed multi-graph that denotes the communication demands where $L \subset V$ is the set of leaves of T. Let C_v denote the capacity of a firewall at node $v \in V$. The goal is to find the minimum number of firewalls to open such that all demands can be feasibly routed via a firewall while respecting both edge bandwidths and firewall capacities.*

We study two variants of the above problem, namely, with soft capacities and hard capacities. In the soft-capacitated firewall placement problem, one can place multiple copies of the firewall on the same vertex. In the hard-capacitated version, however, one can place at most one firewall at any vertex.

For our algorithm for the hard-capacitated firewall placement problem,

we use an algorithm for the Simultaneous Source Location problem on trees (SSL) as a subroutine. We now define the SSL problem and mention relevant results. In the simultaneous source location problem on trees, each vertex of the tree has a demand d_v, and the task is to open the minimum number of sources, so that a flow of d_v units can be feasibly routed to each vertex v. More formally.

Definition 2. Capacitated Simultaneous Source Location on Trees: *Given a tree $T = (V, E)$, with edge bandwidths $b' : E \to \mathbb{R}^+$, potential source capacities $c : V \to \mathbb{R}^+$, and a set of vertex demands $d : V \to \mathbb{R}^+$, what is the minimum number of sources that need to be opened so that all demands can be satisfied simultaneously?*

We note that on general graphs, the firewall placement problem is hard to approximate within any non-trivial factor. The feasibility problem of given a general graph with edge bandwidths and communication demands between pairs of vertices, can we route a unit flow between all the demand pairs while satisfying the bandwidth constraints generalizes the well-studied *Edge Disjoint Paths* problem and is thus NP-hard. We note that even when the underlying graph is a star, in the most general version where a vertex v has capacity C_v and one can place at most 1 firewall at any vertex, the capacitated firewall

placement problem generalizes the well-studied hard capacitated vertex cover problem. Recall that given an undirected graph $G = (V, E)$ where vertex $v \in V$ has a capacity k_v, the hard-capacitated vertex cover problem is to find the smallest set of vertices such that each edge is covered by one of its end points and a vertex v covers at most k_v incident edges. The reduction follows by creating a star network where the leaves of the star correspond to the vertices of G and the communication demands correspond to the edges of G.

4.2 Related Work

In recent years, minimizing congestion in datacenters has been an important research topic. A number of papers [63, 64, 65] consider the problem of assigning virtual machines to the physical servers, so that the congestion due to communication is minimized. Bansal et. al. [63] consider the problem of assigning virtual machines to the physical servers, so that the congestion due to both communication on links as well as processing on nodes is minimized. Dutta et. al. [64] consider a similar assignment problem but in the special case of rooted path requests, where each "request" is in the form of a chain of VMs with an uplink bandwidth to a gateway node. Wen et al. [65] consider

the problem of migrating virtual machines so as to relieve network congestion when required and provide heuristics for the same. Closer in spirit to our work is the Simultaneous Source Location (SSL) problem introduced by Andreev et al. [66]. SSL is a special case of our problem where each pair of communicating VMs resides on the same physical server and firewalls are uncapacitated.

The hard-capacitated vertex cover problem was first studied by Chuzhoy and Naor [47] who gave a 3-approximation for the problem using randomized rounding followed by alteration. Gandhi et al. [48] later improved this result to give a 2-approximation. However, both of these approaches work only for simple graphs and fail for multigraphs where one can have multiple edges between the same pair of vertices. Saha and Khuller [49] give the first constant approximation algorithm for hard-capacitated vertex cover on multigraphs and hypergraphs. In a recent improvement, Cheung et al. [50] obtain a deterministic 3-approximation and a randomized 2.155-approximation algorithm for hard-capacitated vertex cover on multigraphs.

The edge disjoint paths problem (EDP) and related problems on directed and undirected graphs have been an area of very active research over the past decade. On general directed graphs, the best known approximation algorithm [67, 68, 69] achieves a factor of $\tilde{O}(\min(n^{2/3}, \sqrt{m}))$ while Guruswami et al.

[70] give a matching $\Omega(m^{1/2-\epsilon})$ hardness of approximation. Chuzhoy and Li [71] give a polylogarithmic approximation on the number of demand pairs satisfied but allow violating the edge bandwidths by a factor of 2.

4.3 Our Contribution

- We show that the soft-capacitated firewall placement problem with uniform capacities can be solved optimally in polynomial time using a greedy algorithm.
- In stark contrast, we show that the hard-capacitated firewall placement problem is NP-hard.
- We extend the dynamic programming algorithm for Andreev et al. [66] to solve the Capacitated Simultaneous Source Location problem on a tree.
- For the hard-capacitated firewall placement problem, we design an algorithm that uses at most the optimum number of firewalls but violates the edge bandwidths by a factor of at most 2.
- For the hard-capacitated firewall placement problem on a star, we give

a 13-approximation algorithm without any bandwidth violation.

4.4 Preprocessing

We first perform a simple preprocessing step that reduces the Firewall Placement Problem to a generalization of the SSL problem that we call Path-SSL.

Consider any feasible solution $\mathcal{S}$ to the Firewall Placement Problem. For any demand $d = (s, t)$, $\mathcal{S}$ sends unit flow from s to t via some firewall f. Since the underlying network is a tree, $\mathcal{S}$ must always send unit flow from s to t via the unique path (p_{st}) between them in the tree and further send unit flow from a vertex $v \in p_{st}$ to a firewall f and back. Hence, without loss of generality, we can always send unit flow on the unique paths between the end points of each demand and then try to find the minimum number of firewalls required so that we can send and receive a unit flow from some vertex on every such path. More formally, we take the following steps - for every demand (s, t), decrement the bandwidth of every edge on the unique path p_{st} between s and t in the tree T. We then replace every bidirectional edge by an undirected edge having minimum of the two bandwidths,i.e.

$b(\{u,v\}) = \min(b((u,v)), b((v,u)))$. Now, a demand $d = (s,t)$ is satisfied if one can route unit flow from a firewall f to any vertex on the path $p_d = p_{st}$.

As a result, we have reduced the Firewall Placement Problem to the following generalization of the SSL problem.

Definition 3. Path-SSL: *Given an undirected, rooted tree $T = (V, E)$, with edge bandwidths $b : E \to \mathbb{Z}^+ \cup \{0\}$, potential firewall capacities $C : V \to \mathbb{Z}^+$, and a set of m demand paths $\{p_d\}_{d=1}^m$, find the minimum number of firewalls to place (on vertices) such that one can route a unit flow from a firewall to some vertex $v_d \in p_d$ for every demand path p_d.*

4.5 Firewall Placement with Soft Capacities

We now consider the Firewall Placement Problem with *Soft Capacities* where all firewalls have the same capacity denoted by C. In this setting, one is allowed to place multiple firewalls at the same node in the network. We show that a simple greedy algorithm yields an optimal solution for this problem. We first apply the preprocessing steps described in the previous section and are thus interested in solving the Path-SSL problem with soft-capacities.

We now setup some notation that will be useful for the description of the

algorithm. For any vertex $v \in T$, let T_v denote the subtree rooted at v and $V(T_v)$ be the vertices in T_v. Let D_v denote the set of all demand paths fully contained in T_v. For convenience, we refer to $b(v) = b(parent(v), v)$, i.e., $b(v)$ denotes the bandwidth of the edge between v and its parent.

$F \leftarrow \phi$; `// F is the multiset of firewalls`

foreach *vertex* $v \in V$ *in leaf to root order* **do**

  $F_v \leftarrow F \cap V(T_v)$;

  Compute the maximum number of demands in D_v that can be satisfied by F_v. This is a flow problem in an auxiliary graph. Let D denote number of unsatisfied demands in D_v;

  if $b(v) < D$ **then**

    Add $\left\lceil \frac{D-b(v)}{C} \right\rceil$ copies of v to F ; `// Place firewalls at` v

Algorithm 3: Algorithm for Firewall Placement with Soft Capacities

Algorithm 3 describes our complete algorithm. We process vertices in leaf to root order. At every vertex v, we first try to satisfy as many demands fully contained in T_v using the firewalls that have already been placed inside T_v earlier. Note that given a set of firewalls, computing the maximum number of demands that can be satisfied is a maximum flow problem in an auxiliary

network obtained from the tree by adding a source for every demand that is connected to every vertex on the demand path, and adding a single sink node that is connected to all the firewalls. Let D denote the number of unsatisfied demands. Now, intuitively, $D - b(v)$ denotes the number of demands that must be satisfied within T_v itself. This is accomplished by placing $\left\lceil \frac{D-b(v)}{C} \right\rceil$ firewalls at v.

Claim 9. *There exists an optimal solution such that for any vertex v, there do not exist demands $d \in D_v$ and $d' \notin D_v$ such that d is assigned to facility $f' \notin T_v$ and d' is assigned to facility $f \in T_v$.*

Proof. Let OPT be an optimal solution such that there is a vertex v with d assigned to f' and d' assigned to f. Let n be node on demand d that f' supplies to, and similarly n' be the node on d'. Hence, we have the available paths $n \to v \to f'$ and $n' \to v \to f$ to satisfy both the demands. Now consider an alternative solution OPT' which is same as OPT except that d is assigned to f and d' is assigned to f'. This assignment is feasible as there exist paths $n \to v \to f$ and $n' \to v \to f'$ (edges are undirected). Thus, we have an optimal solution satisfying the required properties. □

Lemma 16. *Algorithm 3 returns an optimal solution.*

Proof. We claim that we maintain the following invariant after processing each vertex - Let $V' \subseteq V$ denote the set of visited vertices, let S' denote the multi-set of facilities that have been opened so far, and $R' = V' \setminus S'$[1] denote the visited vertices where we do not place a facility. Then there exists an optimal solution OPT such that $S' \subseteq OPT$ and further $OPT \cap R' = \phi$. Clearly, if this invariant is maintained, then the algorithm terminates with an optimal solution.

We prove the claim by induction. Initially, both the statements are trivially true. Consider one iteration and let v be the vertex under consideration. Consider a demand $d \in D_v$. By Claim 9, without loss of generality, one can assign d to a reachable facility in F_v rather than a facility outside T_v. Also as $OPT \cap R' = \phi$, we know that any demand $d \in D_v$ that is not satisfied by F_v can be satisfied by a facility at v, and can hence be replaced by a (v, v) demand. Let D denote the number of demands that cannot be satisfied by the facilities in F_v. Now, if $b(v) < D$, then OPT must open at least one facility in T_v to satisfy the $D - b(v)$ extra demands. By induction hypothesis, no vertex in $(T_v - \{v\}) \setminus S'$ can be in OPT. Hence, OPT must contain $\lceil \frac{D-b(v)}{C} \rceil$ facilities at v and we maintain the invariant that

[1] Abusing notation slightly to incorporate multisets

$S'_{\text{new}} = S' \cup \{v\} \subseteq OPT$. If on the other hand, $u(v) \geq D$, then a facility at v can be replaced by a facility at $parent(v)$ without violating edge capacities. And hence $\{R'_{\text{new}} = \{R' \cup \{v\}\} \cap OPT = \phi$. □

4.6 Firewall Placement with Hard Capacities

In this section, we consider firewall placement with hard capacity constraints. In contrast with the previous section, we do not allow multiple facilities to be placed at a node. We now describe an algorithm that provides a placement using at most the minimum number of firewalls, but violates the edge bandwidths by a factor of at most 2. Note that in this case we do not apply the preprocessing steps described in Section 4.4.

We reduce the firewall placement problem with hard capacities to the capacitated simultaneous source location problem as follows. We create an instance I' of the capacitated simultaneous source location on the same tree T and edge bandwidths $\tilde{b}$ as follows - For every internal vertex v, set the demand $d(v) = 0$, and for every leaf v set $d(v) = |\{d \in D | d = (v, w)\}|$, i.e., we set $d(v)$ to be the number of demand paths having v as the start vertex. The potential capacities for all vertices are retained as $c(v)$. Further, we

assume that every tree edge $e = \{u, v\}$ is undirected and has bandwidth $\tilde{b}(e) = b(u, v) = b(v, u)$. We then solve the capacitated simultaneous source location problem on a tree using dynamic programming.

A feasible solution to the instance I' of the capacitated simultaneous source location problem guarantees that there exists a feasible flow that satisfies all bandwidth and capacity constraints so that a unit flow can be routed from source vertex s to a facility f and back to s. Given such a solution, we further send unit flow from s to t along the unique simple path between the two leaves in the tree for each demand pair $d = (s, t)$ in order to obtain a feasible firewall placement solution. For any edge e, $F_2(e) = |\{d \in D | e \in p_d\}| \leq b(e)$ denotes the additional flow that we send in this stage. As a result, the total flow through any edge e is at most twice the bandwidth $b(e)$. The following claim now guarantees that we open at most the optimum number of firewalls and thus proves the theorem.

Lemma 17. *Let k denote the optimal number of firewalls for the given instance of the firewall placement problem. Then there exists a solution to the instance I' of the capacitated simultaneous source location problem using k sources.*

Proof. Let OPT_{FP} denote an optimal solution to the firewall placement

problem that uses k firewalls $S \subseteq V$ ($|S| = k$) be the set of open facilities in OPT_{FP}. For a demand j (between leaves s_j and t_j), let $f(j) \in S$ be the firewall that is assigned to j in OPT_{FP}. Further, let $F(e)$ denote the flow through edge $e = (u, v)$ in OPT_{FP}. Since OPT_{FP} must send a unit flow from s_j to $f(j)$ (and from $f(j)$ to t_j), we have $F(e) \geq |\{j \in D | e \in P_{(s_j, f(j))}\}|$ where $P_{(s_j, f(j))}$ denotes the unique simple path in the tree between vertices s_j and $f(j)$.

We now claim that the set S is also a feasible solution to the instance I' of the capacitated simultaneous source location problem. Let $f'(j) \in S$ denote the facility to be assigned to demand j for the instance I'. Initially suppose $f'(j) = f(j)$ for all demands j. We note that any feasible solution to the capacitated simultaneous source location problem must be able to route unit flow from s_j to a facility ($f'(j)$) and *back to* s_j and hence this facility assignment may not lead to a feasible flow. We now successively update the assignments and demonstrate that a feasible flow exists.

Consider any pair of adjacent vertices u and v in the tree. The edge $e = \{u, v\}$ divides the tree into two parts. Let $T(u)(T(v))$ denote the subtree containing $u(v)$. Now suppose there exists demands j and k such that $s_j \in T(u)$ and $s_k \in T(v)$ while $f'(j) \in T(v)$ and $f'(k) \in T(u)$. In

this case, we update the assignments so that demand $f'_{new}(j) = f'(k)$ and $f'_{new}(k) = f'(j)$. It can be observed that such a swap only reduces the flow through edge e and e^r while maintaining the flow through all other edges.

We apply the above swapping procedure to every edge in the tree and let $\tilde{f}(j)$ denote the final facility assigned to demand j and $F'(e) = |\{j \in D | e \in P_{(s_j,\tilde{f}(j))}\}|$ denote the flow from through any edge e in the tree. Due to the previous swaps, we are now guaranteed that for any pair of adjacent vertices $\{u, v\}$, either $F'(u, v) = 0$ and/or $F'(v, u) = 0$. Now, let $\tilde{F}(e) = |\{j \in D | e \in P_{(s_j,\tilde{f}(j))} \cup P_{(\tilde{f}(j),s_j)}\}|$ denote the flow through edge e as required by the instance I'. Hence, we have $\tilde{F}(e) = F'(e) + F'(e^r) \leq b(e)$ and hence, S is a feasible solution to the capacitated simultaneous source location problem of size k. □

4.6.1 Capacitated Simultaneous Source Location

We now show that the capacitated simultaneous source location problem can be solved in polynomial time on trees using an extension of the dynamic programming algorithm given by Andreev et al. [66]. We assume that the tree is binary. Note that this is without loss of generality, as any non-binary tree can be converted to a binary tree by adding dummy nodes (with $c_v = 0$). Let

$f(v, i)$ denote the amount of flow that needs to be sent into T_v by $parent(v)$ in order to satisfy all demands assuming that T_v has i sources. It is easy to observe that the minimum k such that $f(root, k) \leq 0$ is the desired solution. We compute f using dynamic programming as follows.

1. For every leaf v (Base Case):
   (a) $f(v, 0) = d_v$ (deficit)
   (b) If $c_v >= d_v$, $f(v, i) = -\min\{u(v), (c_v - d_v)\}$ (surplus)
   (c) Else, $f(v, i) = d_v - c_v$ (deficit)
2. For every vertex v such that f is defined for both children:
   (a) $f(v, i) = \min(A, B)$ where
   (b) $A = \min_{i_1+i_2=i}(\max\{f(v_1, i_1) + f(v_2, i_2) + d_v, -u(v)\})$
   (c) $B = \min_{i_1+i_2=i-1}(C(i_1, i_2))$ where
   (d) If $c_v >= f(v_1, i_1)+f(v_2, i_2)+d_v$, then $C(i_1, i_2) = -\min\{u(v), (c_v - f(v_1, i_1) - f(v_2, i_2) - d_v)\}$ (surplus)
   (e) Else $C(i_1, i_2) = f(v_1, i_1) + f(v_2, i_2) + d_v - c_v$ (deficit)

The base cases are easy. If v does not have a source, then it needs d_v flow from its parent. On the other hand, if v does have a source, it can send flow

to its parent (upper bounded by $u(v)$) depending upon whether the capacity is sufficient to meet all of v's demand.

For the inductive case, to compute $f(v, i)$, we consider two cases namely - (A) : There is no source at v, and (B) : There is a source at v. To compute A, we find the best way to divide the i sources among the two subtrees. The total demand now is d_v along with the flows needed by the two subtrees, i.e. $f(v_1, i_1) + f(v_2, i_2) + d_v$. We have a max() there to ensure that v can send more surplus than $u(v)$. Similarly to compute B, we find the best way to divide the remaining $i - 1$ sources among the two subtrees. $f(v, i)$ is then computed depending upon whether c_v is sufficient to meet all the demand or not as in the base case.

4.7 Firewall Placement with Hard Capacities and No Bandwidth Violation

We now consider the firewall placement problem with hard capacity constraints (and no bandwidth violation) when the underlying tree T is a star. In this setting, we are given a star $T = (\{r\} \cup V, E)$ and a demand graph $H = (V, D)$. We first apply the preprocessing step described in Section 4.4.

The following lemma shows that we can assume without loss of generality that firewalls can only be placed on leaves of the star.

Lemma 18. *Given an instance $\mathcal{I}$ of the firewall placement problem on a star, we can obtain an instance $\mathcal{I}'$ such that the root of the star in instance $\mathcal{I}'$ has capacity zero, i.e. $C_{r'} = 0$, and $OPT(\mathcal{I}) = OPT(\mathcal{I}')$ where $OPT(\mathcal{I})$ denotes the cost of the optimal solution for instance $\mathcal{I}$.*

Proof. Let instance $\mathcal{I}$ be defined on a star $T = (\{r\} \cup V, E)$ and the demand graph $H = (V, D)$. We create a new star graph $T' = (\{r'\} \cup V', E')$ where $V' = \{r\} \cup V$ forms the set of leaves and r' is the new root. For every edge $(r', v) \in E'$ where $v \in V$, we set $b(r', v) = b(r, v)$, i.e., we leave the bandwidth of every vertex $v \in V$ unchanged. Finally, we set $b(r', r) = |D|$. This allows any demand $d = (s, t)$ to be assigned to r in the instance $\mathcal{I}'$ as long as r has enough capacity to handle it. We set $C_{r'} = 0$ and the capacities of all other vertices remain unaltered. The new instance $\mathcal{I}'$ is defined by the star T' and the original demand graph H.

Now consider any optimal solution S to instance $\mathcal{I}$. If $r \notin S$, then it is easy to observe that S is also a feasible solution to instance $\mathcal{I}'$. This is because both the bandwidths and capacities of vertices other than the root remain unchanged. On the other hand if $r \in S$, then S is still a feasible

solution for $\mathcal{I}'$ as r now has enough bandwidth to satisfy all demands and its capacity is unchanged. Similarly, given any optimal solution S' to instance $\mathcal{I}'$, it is easy to observe that S' is feasible of $\mathcal{I}$ as well and the lemma follows. $\square$

For the rest of this section, we now assume that capacity of the root r of the star is zero. The problem now is to select the smallest set $U \subset V$ of vertices such that one can feasibly route unit flow from a vertex $u \in U$ to one of $\{r, s, t\}$ for every demand (s, t).

Consider any demand $d = (s, t)$. If d is satisfied by a vertex $u \notin \{s, t\}$, then unit flow must be routed from u to r; in other words vertex u can satisfy at most $b(u)$ demands that are not incident on it where $b(u)$ denotes the bandwidth of the edge (r, u). On the other hand, due to capacity constraints, a vertex u can satisfy a total of at most C_u demands. The firewall placement problem with hard capacities on a star network thus naturally reduces to an instance of the network-aware machine activation problem with unit jobs as considered in Section 2.4. In this reduction, we introduce a job j such that $\delta(j) = \{s, t\}$ for every demand $d = (s, t)$. Theorem 5 thus applies and we obtain the following constant approximation for the firewall placement problem on a star.

Theorem 10. *There exists a polynomial time 13-approximation algorithm*

for the hard-capacitated firewall placement problem on a star.

Proof. Theorem 5 yields a $(4f + 5)$-approximation algorithm where $f = \max_j |\delta(j)|$. Since we have $\delta(j) = \{s, t\}$ for every demand $d = (s, t)$ in the reduction, we have $f = 2$ and the theorem follows. □

4.8 Lower Bounds

We note that the hard and soft capacitated versions of the firewall placement problem with non-uniform firewall capacities generalize the respective versions of the capacitated vertex cover problem. Let $G = (V, E)$ be the graph in an instance of capacitated vertex cover. One can now create an equivalent instance of the firewall placement problem by creating a star network as follows - Let V be the leaves of a star rooted at r. We set the capacity of r to be zero and the capacities of the other vertices remain as they were in the vertex cover instance. The graph $G = (V, E)$ forms the demand graph by considering an arbitrary orientation of the edges. Finally the edge bandwidths are set so that no edge has any surplus bandwidth, i.e., we have $b(v, r) = b(r, v) = \max(deg^{out}(v), deg^{in}(v))$. As a result, demand edge (u, v) can only be satisfied be satisfied by either u, v or the root r. As the root

has zero capacity, we have exactly the vertex cover problem. As a result both the hard and soft capacitated firewall placement problems with non-uniform firewall capacities are as hard as vertex cover and hence cannot be approximated with a factor better than 2 assuming the Unique Games Conjecture [72].

We now show that the hard-capacitated firewall placement problem remains NP-complete even with uniform firewall capacities by a reduction from hard-capacitated vertex cover with uniform capacities.

Theorem 11. *The hard-capacitated firewall placement problems with uniform capacities is NP-complete.*

Proof. An instance of hard-capacitated vertex cover with uniform capacities consists of a graph $G = (V, E)$, an integer $C \leq n$ and an integer $k \leq n$ where $n = |V|$ is the number of vertices of G. Given such an instance the hard-capacitated vertex cover problem is to decide if G contains a vertex cover of size $\leq k$ such that any selected vertex covers at most C incident edges.

We perform the same reduction as described in the case of the non-uniform capacities with the exception that now the root r of the star also has capacity C. Figure 4.1 shows an example of the reduction.

We first observe that if G has a hard-capacitated vertex cover of size k, then placing firewalls on the same k vertices is a feasible solution for the firewall placement instance. On the other hand, let us suppose that the firewall placement instance has a solution with k firewalls. Now, in this solution if the root r does not have a firewall, then the reduction ensures that the k selected vertices also form a vertex cover of G. But if r does have a firewall, then it may satisfy up to C demands. Consider an edge $e = (v_1, v_2)$ that is satisfied by r in the firewall placement instance. Further suppose there exists a path p in G from v_1 (or v_2) to some vertex v' such that either (a) there is no firewall that v' or (b) v' has spare capacity. In this case, one can assign the edge e to v_1 and propagate the assignments along the path p by placing a new firewall at v' if necessary. On the other hand, if no such path exists, then edge e belongs to a connected component with edge density $> C$ and hence no feasible vertex cover solution exists. As a result, given a firewall placement solution with k firewalls, one can either determine that the vertex cover instance is infeasible or find a vertex cover of size $k + C$.

We have shown that if the hard-capacitated firewall placement problem with uniform capacities can be solved in polynomial time, then one can obtain an additive C approximation to the hard-capacitated vertex cover problem.

However, the existence of such an additive C approximation algorithm actually implies that the hard-capacitated vertex cover with uniform capacities problem can be solved in polynomial time. Given a graph $G = (V, E)$ and a capacity C, we create $C + 1$ disjoint copies of the graph to obtain a new instance $\mathcal{G}'$. Now, if G has a vertex cover of size k then G' has a vertex cover of size $k(C + 1)$. We then use the above additive C approximation algorithm to obtain a solution with at most $k(C + 1) + C$ vertices. Since G' consists of $C + 1$ disjoint copies of G, at least one of the copies uses at most $k + \frac{C}{C+1}$ vertices. As the solution size must be an integer, we obtain a solution of size at most k. □ □

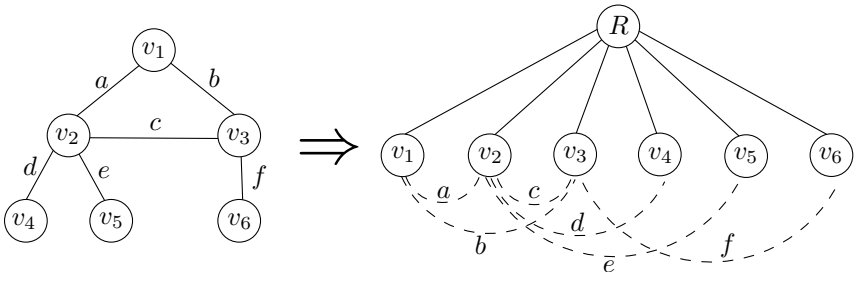

Figure 4.1: Capacitated Vertex Cover to Firewall Placement reduction

4.9 Future Directions

In this section we summarize some of the prominent open problems related to firewall placement and future research directions.

No Bandwidth Violation

In Section 4.6, we gave a polynomial time algorithm for the hard-capacitated firewall placement problem that places the minimum number of firewalls but violates edge bandwidths by a factor of at most 2. Our first open problem is to obtain an approximation algorithm for the firewall placement problem with hard capacities that respects all edge bandwidths.

In Section 4.7, we make partial progress on this problem by designing a constant approximation algorithm for hard-capacitated firewall placement when the underlying physical network is a star.

Non Uniform Soft Capacities

As discussed in Section 4.5, the soft-capacitated firewall placement problem with uniform firewall capacities can be solved in polynomial time. However, if firewall capacities are allowed to be non-uniform, then it generalizes the

soft-capacitated vertex cover problem and is thus NP-hard. Our next open problem is to obtain an approximation algorithm soft capacitated firewall placement on a tree with non uniform capacities.

Chapter 5

Scheduling with Soft Precedences

We now consider the soft-precedence constrained scheduling problem. We model the precedence constraints between jobs as a directed graph where an edge $e = (u, v)$ denotes that u must be executed before v in the schedule. We then claim that different soft-precedence constrained scheduling problems can be treated as variants of a certain vertex ordering problem on this associated directed graph. We first tackle the problem of *Discrete Soft Precedence Scheduling*. In Section 5.1, we formally define the Max-k-Ordering and OffsetRMAS problems that accurately model the scheduling problem

with discrete soft precedences. In Sections 5.4- 5.6, we provide randomized approximation algorithms for the different problems considered. Section 5.7 proves a matching integrality gap that is retained even after almost polynomial rounds of the Sherali-Adams hierarchy. In Section 5.8, we consider the minimization version of the *Discrete Soft Precedence Scheduling* problem. Finally, in Section 5.9, we study the *Linear Soft Precedence Scheduling* problem and show that it can be solved optimally in polynomial time using linear programming.

5.1 Motivation and Problem Definitions

We first consider the following simple case of discrete time scheduling: given n unit length jobs with precedence constraints and an infinite capacity machine, find a schedule so that all the jobs are completed by timestep k. Since it may not be feasible to satisfy all the precedence constraints, the goal is to satisfy the maximum number.

One can now associate the following directed, precedence graph $G = (V, A)$ with such an instance where V denotes the set of all jobs and $e = (u, v) \in A$ if and only if u needs to be scheduled before v as per the precedence constraint.

Note that if we are forced to satisfy all the precedence constraints, then the precedence graph must be acyclic and the makespan[1] of the optimal schedule is exactly equal to one plus the length of the longest path in G. As a result, we can reformulate the soft-precedence constrained scheduling problem on unit jobs as - "Given a directed graph $G = (V, A)$ and an integer deadline k, find an acyclic subgraph $H = (V, A')$ of G having maximum number of edges such that H does not have any directed paths of length k." Further, we allow each edge e (precedence constraint) to have a weight $w(e)$ and the objective is then to maximize the total weight of edges in H.

We now formally define the Max-k-Ordering problem that we consider in this work and prove its equivalence to soft-precedence constrained scheduling on unit jobs.

Definition 4. Max-k-Ordering*: Given an n-vertex digraph $D = (V, A)$ with a non-negative weight function $w : A \to \mathbb{R}^+$, and an integer $2 \leq k \leq n$, find a labeling to the vertices $\ell : V \to [k]$ that maximizes the weighted fraction of edges $e = (u, v) \in A$ such that $\ell(u) < \ell(v)$, i.e. forward edges.*

It can be seen that Max-k-Ordering is equivalent to the problem of

[1]Makespan of the schedule refers to the earliest timestep by which all the jobs have been completed.

computing the maximum weighted subgraph of D which is acyclic and does not contain any directed path of length k. The following lemma implies this equivalence.

Lemma 19. *Given a digraph $D = (V, A)$, there exists a labeling $\ell : V \to [k]$ with each edge $e = (u, v) \in A$ satisfying $\ell(u) < \ell(v)$, if and only if D is acyclic and does not contain any directed path of length k.*

Proof. If such a labeling ℓ exists then every edge is directed from a lower labeled vertex to a higher labeled one. Thus, there are no directed cycles in D. Furthermore, any directed path in D has at most k vertices on it, and is of length at most $k-1$. On the other hand, if D satisfies the second condition in the lemma, then choose $\ell(v)$ for any vertex v to be $t_v + 1$, where t_v is the length of the longest path from any source to v. It is easy to see that $\ell(v) \in [k]$ and for each edge (u, v), $\ell(u) < \ell(v)$. □

We now generalize the Max-k-Ordering problem to model more complex scheduling scenarios. In particular, we now allow jobs to have arbitrary lengths (processing times). The arbitrary processing times can be modelled by "offsets" for every edge - an edge $e = (u, v)$ with offset o_e is satisfied by a labelling ℓ if and only if $\ell(u) + o_e \leq \ell(v)$. In particular, setting $o_e = p(u)$ where $p(u)$ is the

processing time of job u is sufficient to ensure that the precedence constraint is satisfied only if job v starts after job u has finished execution. In addition, each job can have a set of allowable timesteps when it can be scheduled. Such a setting corresponds to the availability of certain resources only at particular timesteps. We model this requirement by associating a finite set $S_v \subseteq \mathbb{Z}^+$ with every job v that restricts the set of labels that v can be assigned.

The following generalizations of Max-k-Ordering studied in this work, viz. RMAS and OffsetRMAS, model these scheduling scenarios.

Definition 5. OffsetRMAS*: The input is a digraph $D = (V, A)$ with a finite subset $S_v \subseteq \mathbb{Z}^+$ of labels for each vertex $v \in V$, a non-negative weight function $w : A \to \mathbb{R}^+$, and offsets $o_e \in \mathbb{Z}^+$ for each edge $e \in A$. A labeling ℓ to V s.t. $\ell(v) \in S_v, \forall v \in V$ satisfies an edge $e = (u, v)$ if $\ell(u) + o_e \leq \ell(v)$. The goal is to compute a labeling that maximizes the weighted fraction of satisfied edges.* RMAS *is the special case when each offset is unit.*

Also of interest is the minimization version of Max-k-Ordering on directed *acyclic* graphs (DAGs). We refer to it as *DAG edge deletion* or DED(k) where the goal is to delete the minimum number of directed edges from a DAG so that the remaining digraph does not contain any path of length k. Note that the problem for arbitrary k does not admit any approximation

factor on general digraphs since even detecting whether a digraph has a path of length k is NP-hard.

Definition 6. DED(k)*: Given a DAG $D = (V, A)$ with a non-negative weight function $w : A \to \mathbb{R}^+$, and an integer $2 \leq k \leq n-1$, find a minimum weight set of edges $F \subseteq A$ such that $(V, A \setminus F)$ does not contain any path of length k.*

5.2 Related Work

The Max-k-Ordering problem naturally generalizes several well-known NP-hard optimization problems on directed graphs. When $k = n$, the Max-k-Ordering problem reduces to the *Maximum Acyclic Subgraph* problem (MAS): Given a directed graph, find a subgraph of maximum number of edges that is acyclic. It is easy to see that MAS admits a trivial 2-approximation, by taking any linear ordering or its reverse, and this is also obtained by a random ordering. For Max-k-Ordering the random k-ordering yields a $2k/(k-1)$-approximation for any $k \in \{2, \ldots, n\}$.

For $k = 2$, the Max-k-Ordering problem reduces exactly to the well known Max-DiCut problem : Given a directed graph $G = (V, A)$, find a

directed cut of maximum weight, i.e., find a subset $S \subset V$ that maximizes $|\delta(S, V \setminus S)|$. For MAX-DICUT the semidefinite programming (SDP) relaxation is shown to yield a ≈ 1.144-approximation in [73], improving upon previous analyses of [74], [75], and [76]. As mentioned above, RMAS is a generalization of MAX-k-ORDERING, and a $2\sqrt{2}$-approximation for it based on linear programming (LP) rounding was shown recently by Grandoni et al. [77] which is also the best known approximation for MAX-k-ORDERING for $k = 3$. For $4 \leq k \leq n-1$, to the best of our knowledge the proven approximation factor for MAX-k-ORDERING remains $2k/(k-1)$.

On the hardness side, Newman [78] showed that MAS is NP-hard to approximate within a factor of $66/65$. Assuming Khot's [79] Unique Games Conjecture (UGC), Guruswami et al. [80] gave a $(2-\varepsilon)$-inapproximability for any $\varepsilon > 0$. Note that MAX-DICUT is at least as hard as MAX-CUT. Thus, for $k = 2$, MAX-k-ORDERING is NP-hard to approximate within factor $(13/12 - \varepsilon)$ [81], and within factor 1.1382 assuming the UGC [82]. For larger constants k, the result of Guruswami et al. [80] implicitly shows a $(2 - o_k(1))$-inapproximability for MAX-k-ORDERING, assuming the UGC.

For the vertex deletion version of DED(k), Paik et al. [83] gave linear time and quadratic time algorithms for rooted trees and series-parallel graphs

respectively. The problem reduces to vertex cover on k-uniform hypergraphs for any constant k thereby admitting a k-approximation, and a matching $(k-\varepsilon)$-inapproximability assuming the UGC was obtained by Svensson [84].

5.3 Our Results

The main algorithmic result of this paper is the following improved approximation guarantee for Max-k-Ordering.

Theorem 12. *There exists a polynomial time 2-approximation algorithm for* Max-k-Ordering *on n-vertex weighted digraphs for any $k \in \{2, \ldots, n\}$.*

The above approximation is obtained by appropriately rounding the standard LP relaxation of the CSP formulation of Max-k-Ordering and improves on the previously known approximation factors of $2\sqrt{2}$ for $k = 3$ (implicit in [77]), and $2k/(k-1)$ for $4 \leq k \leq n-1$. The details are given in Section 5.5.

Using an LP rounding approach similar to Theorem 12, in Section 5.6 we show the following improved approximation for OffsetRMAS which implies the same for RMAS. Our result improves the previous $2\sqrt{2} \approx 2.828$-approximation for RMAS obtained by Grandoni et al. [77].

Theorem 13. *There exists a polynomial time* $4\sqrt{2}/(\sqrt{2}+1) \approx 2.344$ *approximation algorithm for* OffsetRMAS *on weighted digraphs.*

Our next result shows a lower bound – matching the approximation obtained in Theorem 12 – for the LP relaxation of Max-k-Ordering augmented with nearly polynomial rounds of the Sherali-Adams hierarchy of constraints. We prove the following theorem in Section 5.7.

Theorem 14. *For any small enough constant* $\varepsilon > 0$*, there exists* $\gamma > 0$ *such that for* Max-k-Ordering *on* n*-vertex weighted digraphs and any* $k \in \{2, \ldots, n\}$*, the LP relaxation with* $n^{(\gamma/\log\log k)}$ *rounds of Sherali-Adams constraints has a* $(2-\varepsilon)$ *integrality gap.*

For DED(k) on DAGs we prove in Section 5.8 the following approximation for any k, not necessarily a constant.

Theorem 15. *The standard LP relaxation for* DED(k) *on* n*-vertex DAGs can be solved in polynomial time for* $k = \{2, \ldots, n-1\}$ *and yields a* k*-approximation. The same approximation factor is also obtained by a combinatorial algorithm.*

We complement the above by showing in Section 5.8.3 a $(\lfloor k/2 \rfloor - \varepsilon)$ hardness factor for DED(k) via a simple gadget reduction from Svensson's [84]

$(k - \varepsilon)$-inapproximability for the vertex deletion version for constant k, assuming the UGC.

5.3.1 Overview of Techniques

The approximation algorithms we obtain for Max-k-Ordering and its generalizations are based on rounding the standard LP relaxation for the instance. Max-k-Ordering is viewed as a constraint satisfaction problem (CSP) over alphabet $[k]$, and the corresponding LP relaxation has $[0, 1]$-valued variables x_i^v for each vertex v and label $i \in [k]$, and y_{ij}^e for each edge (u, v) and pairs of labels i and j to u and v respectively. We show that a generalization of the rounding algorithm used by Trevisan [85] for approximating q-ary boolean CSPs yields a 2-approximation in our setting. The key ingredient in the analysis is a lower bound on a certain product of the $\{x_i^u\}, \{x_i^v\}$ variables corresponding to the end points of an edge $e = (u, v)$ in terms of the $\{y_{ij}^e\}$ variables for that edge. This improves a weaker bound shown by Grandoni et al. [77]. For OffsetRMAS, a modification of this rounding algorithm yields the improved approximation.

The construction of the Sherali-Adams LP integrality gap for Max-k-Ordering begins with a simple integrality gap instance for the basic LP

relaxation. This instance is appropriately sparsified to ensure that subgraphs of polynomially large (but bounded) size are *tree-like*. On trees, it is easy to construct a distribution over labelings from $[k]$ to the vertices (thought of as k-orderings), such that the marginal distribution on each vertex is uniform over $[k]$ and a large fraction of edges are satisfied in expectation. Using this along with the sparsification allows us to construct distributions for each bounded subgraph, i.e. good local distributions. Combining this with a geometric *embedding* of the marginals of these distributions followed by Gaussian rounding yields modified local distributions which are *consistent* on the common vertex sets. These correspond to an LP solution with a high objective value, for large number of rounds of Sherali-Adams constraints. Our construction follows the approach in a recent work of Lee [86] which is based on earlier works of Arora et al. [87] and Charikar et al. [88].

For the DED(k) problem, the approximation algorithms stated in Theorem 15 are obtained using the acyclicity of the input DAG. In particular, we show that both, the LP rounding and the local ratio approach, can be implemented in polynomial time on DAGs yielding k-approximate solutions.

5.4 Preliminaries

This section describes the LP relaxations for the Max-k-Ordering, Offsetrmas, and the DED(k) problems. All of our approximation algorithms are obtained by appropriately rounding an optimal solution to the corresponding LP relaxation. Finally, we also describe the LP relaxation after augmentation by r rounds of Sherali-Adams constraints.

5.4.1 LP Relaxation for Max-k-Ordering

From Definition 4, an instance $\mathcal{I}$ of Max-k-Ordering is given by $D = (V, A)$, k, and w. Viewing it as a CSP over label set $[k]$, the standard LP relaxation given in Figure 5.1 is defined over variables x_i^v for each vertex v and label i, and y_{ij}^e for each edge $e = (u, v)$ and labels i to u and j to v.

The x_i^v variables denote if vertex v is assigned label i. The constraint (5.1) guarantees that every vertex is assigned a color (fractionally). The variable $y_{ij}^{(u,v)}$ is intended to represent the event that u is assigned to label i and v is assigned to label j. Such a variable denoting the joint probability distribution is necessary so that the objective can be expressed as a linear function of the variables. The set of constraints (5.2) and (5.3) ensure that the marginals induced by the y variables are consistent with the x variables.

$$\max \sum_{e \in A} w(e) \cdot \sum_{\substack{i,j \in [k] \\ i<j}} y_{ij}^{e}$$

subject to,

$$\forall v \in V, \qquad \sum_{i \in [k]} x_i^v = 1. \quad (5.1)$$

$$\forall e = (u, v) \in A, \text{ and } i, j \in [k], \qquad \sum_{\ell \in [k]} y_{i\ell}^{e} = x_i^u, \quad (5.2)$$

$$\text{and,} \qquad \sum_{\ell \in [k]} y_{\ell j}^{e} = x_j^v. \quad (5.3)$$

$$\forall v \in V, \text{ and } i \in [k], \qquad x_i^v \geq 0.$$

$$\forall e \in A, \text{ and } i, j \in [k], \qquad y_{ij}^{e} \geq 0. \quad (5.4)$$

Figure 5.1: LP Relaxation for instance $\mathcal{I}$ of Max-k-Ordering.

Sherali-Adams Constraints

Let $z_\sigma^S \in [0,1]$ be a variable corresponding to a subset S of vertices, and a labeling $\sigma : S \to [k]$. The LP relaxation in Figure 5.1 can be augmented with r rounds of Sherali-Adams constraints which are defined over the variables $\{z_\sigma^S \mid 1 \leq |S| \leq r+1\}$. The additional constraints are given in Figure 5.2. The Sherali-Adams variables define, for each subset S of at most $(r+1)$ vertices, a distribution over the possible labelings from $[k]$ to the vertices in S. The constraints given by Equation (5.5) ensure that these distributions are consistent across subsets. Additionally, Equations (5.6) and (5.7) ensure the consistency of these distributions with the variables of the standard LP relaxation given in Figure 5.1.

5.4.2 LP Relaxation for RMAS and OffsetRMAS

The LP relaxation for RMAS is a generalization of the one in Figure 5.1 for Max-k-Ordering and we omit a detailed definition. Let $\mathcal{S} = \cup_{v \in V} S_v$ denote the set of all labels. For convenience, we define variables $\{x_i^v \mid v \in V, i \in \mathcal{S}\}$ and $\{y_{ij}^e \mid e = (u,v) \in A, i,j \in \mathcal{S}\}$ and force the infeasible assignments to be zero, i.e. $x_i^v = 0$ for $i \notin S_v$. The other constraints are modified accordingly. For OffsetRMAS, an additional change is that the contribution to the

$\forall S \subseteq T \subseteq V,$

$1 \leq |S|, |T| \leq r+1,$

and $\sigma : S \to [k]$, $$z_\sigma^S = \sum_{\substack{\rho: T \to [k] \\ \rho|_S = \sigma}} z_\rho^T. \quad (5.5)$$

$\forall S \subseteq V, 1 \leq |S| \leq r+1,$

and $\sigma : S \to [k]$, $$0 \leq z_\sigma^S \leq 1.$$

$\forall v \in V,$ and $\sigma : \{v\} \to [k],$

s.t. $\sigma(v) = i$, $$x_i^v = z_\sigma^{\{v\}}. \quad (5.6)$$

$\forall e = (u, v) \in A,$ and,

$\sigma : \{u, v\} \to [k],$

s.t. $(\sigma(u), \sigma(v)) = (i, j)$, $$y_{ij}^e = z_\sigma^{\{u,v\}}. \quad (5.7)$$

Figure 5.2: r-round Sherali-Adams constraints for LP relaxation in Figure 5.1.

objective from each edge $e = (u, v)$ is $\sum_{i \in S_u, j \in S_v, i + o_e \leq j} y_{ij}^e$.

5.4.3 LP Relaxation for DED(k)

The natural LP relaxation for DED(k) on an n-vertex DAG $D = (V, E)$ is given in Figure 5.3. The variable x_e indicates whether edge e has been deleted or not. Constraint (5.8) ensures that for every directed path of length k, at least one edge has been deleted.

$$\min \sum_{e \in E} w(e) x_e$$

subject to,

$$\forall \text{ paths } P \text{ of length } k, \qquad \sum_{e \in P} x_e \geq 1. \tag{5.8}$$

$$\forall e \in E, \qquad x_e \geq 0.$$

Figure 5.3: LP Relaxation for instance $\mathcal{I}$ of DED(k).

5.5 A 2-Approximation for Max-k-Ordering

This section proves the following theorem that implies Theorem 12.

Theorem 16. *Let $\{x_i^v\}, \{y_{ij}^e\}$ denote an optimal solution to the LP in Figure 5.1. Let $\ell : V \to [k]$ be a randomized labeling obtained by independently assigning to each vertex v label i with probability $\left(\frac{1}{2k} + \frac{x_i^v}{2}\right)$. Then, for any edge $e = (u, v)$,*

$$\Pr[\ell(u) < \ell(v)] \geq \frac{1}{2} \left(\sum_{\substack{i,j \in [k] \\ i<j}} y_{ij}^e \right).$$

To analyze the rounding given above, we need the following key lemma that bounds the sum of product of row and column sums of a matrix in terms of the matrix entries. It improves a weaker bound shown by Grandoni et al. [77] and also generalizes to arbitrary offsets.

Lemma 20. *Let $\mathbb{A} = [a_{ij}]$ be a $k \times k$ matrix with non-negative entries. Let $r_i = \sum_j a_{ij}$ and $c_j = \sum_i a_{ij}$ denote the sum of entries in the i^{th} row and j^{th} column respectively, and let $1 \leq \theta \leq k-1$ be an integer offset . Then,*

$$\sum_{\substack{i+\theta \leq j \\ i,j \in [k]}} r_i c_j \geq \frac{k-\theta+1}{2(k-\theta)} \left(\sum_{\substack{i+\theta \leq j \\ i,j \in [k]}} a_{ij} \right)^2.$$

Proof. The LHS of the above is simplified as,

$$\sum_{i+\theta\le j} r_i c_j = \sum_{i+\theta\le j}\left[\left(\sum_{j'} a_{ij'}\right)\left(\sum_{i'} a_{i'j}\right)\right] \tag{5.9}$$

$$\ge \sum_{x+\theta\le y} a_{xy}^2 + 2\cdot \sum_{\substack{x+\theta\le y\\ x+\theta\le y'\\ y<y'}} a_{xy}a_{xy'} + \sum_{\substack{x+\theta\le y\\ x'+\theta\le y'\\ x<x'}} a_{xy}a_{x'y'}, \tag{5.10}$$

where all the indices above are in $[k]$. Note that (5.10) follows from (5.9) because:

(i) For any $x+\theta\le y$, a_{xy}^2 appears in the RHS of (5.9) when $i=x$ and $j=y$.

(ii) For $x+\theta\le y$ and $x+\theta\le y'$, $a_{xy}a_{xy'}$ appears in the RHS of (5.9) both, when $i=x, j=y$, and when $i=x, j=y'$.

(iii) For any $x+\theta\le y$ and $x'+\theta\le y'$ (say $x<x'$), it must be that $x+\theta\le y'$, and hence $a_{xy}a_{x'y'}$ appears in the RHS of (5.9) when $i=x$ and $j=y'$.

Thus, we obtain,

$$\sum_{i+\theta\le j} r_i c_j \ge \left(\sum_{x+\theta\le y} a_{xy}\right)^2 - \left(\sum_{\substack{x+\theta\le y\\ x'+\theta\le y'\\ x<x'}} a_{xy}a_{x'y'}\right).$$

Therefore, it is sufficient to show that

$$\sum_{\substack{x+\theta\le y\\ x'+\theta\le y'\\ x<x'}} a_{xy}a_{x'y'} \le \frac{k-\theta-1}{2(k-\theta)}\left(\sum_{x+\theta\le y} a_{xy}\right)^2. \tag{5.11}$$

Substituting,

$$\left(\sum_{x+\theta\leq y} a_{xy}\right)^2 = \sum_{x+\theta\leq y} a_{xy}^2 + 2\cdot \sum_{\substack{x+\theta\leq y\\ x+\theta\leq y'\\ y<y'}} a_{xy}a_{xy'} + 2\cdot \sum_{\substack{x+\theta\leq y\\ x'+\theta\leq y'\\ x<x'}} a_{xy}a_{x'y'},$$

and simplifying, inequality (5.11) can be rewritten as,

$$\sum_{x+\theta\leq y} a_{xy}^2 + 2\sum_{\substack{x+\theta\leq y\\ x+\theta\leq y'\\ y<y'}} a_{xy}a_{xy'} - \left(\frac{2}{k-\theta-1}\right)\cdot \sum_{\substack{x+\theta\leq y\\ x'+\theta\leq y'\\ x<x'}} a_{xy}a_{x'y'} \geq 0,$$

$$\Leftrightarrow\ \overline{a}^{\mathsf{T}} M \overline{a} \geq 0,$$

where $\overline{a} \in \mathbb{R}^{\mathcal{Z}}$, $\mathcal{Z} := \{(x,y) \mid x+\theta \leq y \text{ and } x,y \in [k]\}$ with $\overline{a}_{(x,y)} := a_{xy}$, and $M \in \mathbb{R}^{\mathcal{Z}\times\mathcal{Z}}$ is a symmetric matrix defined as follows:

$$M_{(x,y)(x',y')} = \begin{cases} 1 & \text{if } (x,y) = (x',y'),\\ 1 & \text{if } x' = x, \text{ and } y \neq y',\\ -1/(k-\theta-1) & \text{if } x \neq x'. \end{cases}$$

To complete the proof of the lemma we show that M is positive semidefinite. Consider the set of unit vectors $\{v_x \mid 1 \leq x \leq k-\theta\}$ given by the normalized corner points of the $(k-\theta-1)$-dimensional simplex centered at the origin. It is easy to see (for e.g. in Lemma 3 of [89]) that, $\langle v_x, v_{x'}\rangle = -1/(k-\theta-1)$ if $x \neq x'$. Thus, M is the Gram Matrix obtained by associating the vector v_x with the row (column) indexed by (x,y) for $1 \leq x \leq k-\theta$. □

Proof of Theorem 16. For brevity, let $z_e = \sum_{i<j} y^e_{ij}$ denote the contribution of the edge e to the LP objective. From the definition of the rounding procedure we have,

$$\begin{aligned}
\Pr[\ell(u) < \ell(v)] &= \sum_{i<j} \Pr[\ell(u) = i] \Pr[\ell(v) = j] \\
&= \sum_{i<j} \left(\frac{1}{2k} + \frac{x^u_i}{2}\right)\left(\frac{1}{2k} + \frac{x^v_j}{2}\right) \\
&= \frac{1}{4}\left(\frac{(k-1)}{2k} + \frac{1}{k}\sum_{i<j}(x^u_i + x^v_j) + \sum_{i<j} x^u_i x^v_j\right)
\end{aligned}$$

We can now apply Lemma 20 to the $k \times k$ matrix $[y^e_{ij}]$. The LP constraints guarantee that $r_i = x^u_i$ and $c_j = x^v_j$ are equal to the row and column sums respectively. Further, substituting offset $\theta = 1$, we obtain

$$\Pr[\ell(u) < \ell(v)] \geq \frac{1}{4}\left(\frac{(k-1)}{2k} + \frac{1}{k}\sum_{i<j}(x^u_i + x^v_j) + \frac{k}{2(k-1)} z_e^2\right). \quad (5.12)$$

On the other hand,

$$\begin{aligned}
\sum_{i<j}(x^u_i + x^v_j) &= \sum_{i=1}^{k-1}(k-i)x^u_i + \sum_{j=2}^{k}(j-1)x^v_j \\
&\geq \sum_{i=1}^{k-1}\left[(k-i)\sum_{j'>i} y^e_{ij'}\right] + \sum_{j=2}^{k}\left[(j-1)\sum_{i'<j} y^e_{i'j}\right]. \quad (5.13)
\end{aligned}$$

For $a < b$, y^e_{ab} appears $(k-a)$ times in the RHS of the above inequality when $i = a$, and $(b-1)$ times when $j = b$. Since $k - a + b - 1 \geq k$, we obtain that RHS of Equation (5.13) is lower bounded by $k\sum_{a<b} y^e_{ab} = kz_e$. Substituting

back into Equation (5.12) and simplifying gives us that $\Pr[\ell(u) < \ell(v)]$ is at least,

$$\frac{z_e}{4}\left[1 + \frac{1}{2}\left(\frac{(k-1)}{kz_e} + \frac{kz_e}{(k-1)}\right)\right] \geq \frac{z_e}{4}(1+1) = \frac{z_e}{2},$$

where we use $t + 1/t \geq 2$ for $t > 0$. □

5.6 Approximation for OffsetRMAS

Let $D = (V, A)$, $\{S_v\}_{v \in V}$, w, and $\{o_e\}_{e \in A}$ constitute an instance of OffsetRMAS as given in Definition 5. Without loss of generality, one can assume that for each edge $e = (u, v) \in A$, $\min(S_u) + o_e \leq \max(S_v)$, otherwise no feasible solution can satisfy e and that edge can be removed. A simple randomized strategy that independently assigns each vertex v either $\ell^v_{min} := \min(S_v)$ or $\ell^v_{max} := \max(S_v)$ with equal probability is a 4-approximation. The recent work of Grandoni et al. [77] show that combining this randomized scheme with an appropriate LP-rounding yields a $2\sqrt{2} \approx 2.828$ approximation algorithm for RMAS.

We show that the rounding scheme developed in Section 5.5 can be adapted to obtain an improved approximation algorithm for OffsetRMAS. In Section 5.5, we assigned label i to vertex u with probability $(\frac{1}{2k} + \frac{x^u_i}{2})$. This

can be seen as an instance of the following general strategy: for a vertex u, let p^u denote some prior distribution over its allowed labels. In addition, the LP variables $\{x_i^u\}$ represent another probability distribution over the labels of u. The rounding scheme is then to assign label i to vertex u with probability $(\frac{p_i^u}{2} + \frac{x_i^u}{2})$. For OffsetRMAS, we let p^u be the distribution defined by the simple randomized strategy described above, i.e. $p_i^u = \frac{1}{2}$ for $i \in \{\ell_{min}^u, \ell_{max}^u\}$ and $p_i^u = 0$ for all $i \in S_u \setminus \{\ell_{min}^u, \ell_{max}^u\}$. The rest of this section proves the following theorem which implies Theorem 13.

Theorem 17. *Let $\{x_i^v\}, \{y_{ij}^e\}$ denote an optimal solution to the linear programming relaxation of* OffsetRMAS *described in Section 5.4. Let ℓ be a randomized labeling obtained by independently assigning labels to each vertex v with the following probabilities:*

$$\Pr[\ell(v) = i] = \begin{cases} \frac{1}{4} + \frac{x_i^v}{2} & \text{if } i \in \{\ell_{min}^v, \ell_{max}^v\} \\ \\ \frac{x_i^v}{2} & \text{if } i \in S_v \setminus \{\ell_{min}^v, \ell_{max}^v\} \end{cases} \tag{5.14}$$

Then, for any edge $e = (u, v)$ we have

$$\Pr[\ell(u) + o_e \leq \ell(v)] \geq \frac{1}{4}\left(1 + \frac{1}{\sqrt{2}}\right)\left(\sum_{\substack{i \in S_u, j \in S_v \\ i + o_e \leq j}} y_{ij}^e\right).$$

Proof. Let $\mathcal{S} = \cup_{v \in V} S_v$ denote the set of all labels and let $z_e = \left(\sum_{i+o_e \leq j} y_{ij}^e\right)$

denote the contribution of the edge e to the LP objective. We have,

$$\Pr[\ell(u) + o_e \leq \ell(v)] = \sum_{\substack{i+o_e \leq j \\ i \in S_u, j \in S_v}} \Pr[\ell(u) = i] \Pr[\ell(v) = j]$$

Substituting the assignment probabilities from (5.14) into the above and simplifying we obtain,

$$\begin{aligned} & \Pr[\ell(u) + o_e \leq \ell(v)] \\ = \; & \frac{1}{16} + \frac{1}{8} \left(\sum_{\substack{i \leq \ell^v_{max} - o_e \\ i \in \mathcal{S}}} x^u_i + \sum_{\substack{j \geq \ell^u_{min} + o_e \\ j \in \mathcal{S}}} x^v_j \right) + \frac{1}{4} \left(\sum_{\substack{i+o_e \leq j \\ i,j \in \mathcal{S}}} x^u_i x^v_j \right) \end{aligned}$$

Note that we allow $i, j \in \mathcal{S}$ in the above sums instead of S_u and S_v. This does not affect the analysis as the LP forces $x^u_i = 0$ for $i \notin S_u$ and similarly for v. Now, consider the $|\mathcal{S}| \times |\mathcal{S}|$ matrix $[y^e_{ij}]$. Since x^u_i and x^v_j are equal to the row sums and column sums of this matrix respectively, Lemma 20 guarantees that,

$$\sum_{\substack{i+o_e \leq j \\ i,j \in \mathcal{S}}} x^u_i x^v_j \geq \frac{|\mathcal{S}| - o_e + 1}{2(|\mathcal{S}| - o_e)} \left(\sum_{\substack{i \in S_u, j \in S_v \\ i+o_e \leq j}} y^e_{ij} \right)^2 \geq \frac{(|\mathcal{S}| - o_e + 1)}{2(|\mathcal{S}| - o_e)} z_e^2 \geq \frac{z_e^2}{2}.$$

We thus have,

$$\begin{aligned}
& \Pr[\ell(u) + o_e \leq \ell(v)] \\
\geq\ & \frac{1}{16} + \frac{1}{8}\left(\sum_{i \leq \ell^v_{max} - o_e} x^u_i + \sum_{j \geq \ell^u_{min} + o_e} x^v_j\right) + \frac{z_e^2}{8} \\
\geq\ & \frac{1}{16} + \frac{1}{8}\left(\sum_{i \leq \ell^v_{max} - o_e} \left(\sum_{j \geq i + o_e} y^e_{i,j}\right) + \sum_{j \geq \ell^u_{min} + o_e} \left(\sum_{i + o_e \leq j} y^e_{i,j}\right)\right) + \frac{z_e^2}{8} \\
=\ & \frac{1}{16} + \frac{1}{8}\left(2 \sum_{\substack{i + o_e \leq j \\ i \in S_u, j \in S_v}} y^e_{i,j}\right) + \frac{z_e^2}{8} \\
\geq\ & \frac{1}{16} + \frac{z_e}{4} + \frac{z_e^2}{8} \\
=\ & \frac{z_e}{4}\left(1 + \frac{1}{2}\left(\frac{1}{2z_e} + z_e\right)\right) \geq \frac{z_e}{4}\left(1 + \frac{1}{\sqrt{2}}\right),
\end{aligned}$$

where the last inequality uses $t + 1/at \geq 2/\sqrt{a}$ for $a, t > 0$. □

5.7 Sherali-Adams Integrality Gap for Max-k-Ordering

We now show that the rounding algorithm developed in Section 5.5 is tight. In fact, we show that even after augmenting the LP with nearly polynomial rounds of the Sherali-Adams hierarchy, there is a $2 - \varepsilon$ integrality gap. We begin with a simple construction of an n-vertex digraph which is a $(2 - 2/n)$

integrality gap for the standard LP relaxation for MAX-k-ORDERING in Figure 5.1, for $2 \leq k \leq n$.

Claim 10. *Let $D = (V, A)$ be the complete digraph on n vertices, i.e. having a directed edge for every ordered pair (u, v) of distinct vertices u and v. Thus, $|A| = 2\binom{n}{2}$. Let $k \in \{2, \ldots, n\}$. Then,*

- *The optimum of* MAX-k-ORDERING *on D is at most $\frac{1}{2}\left(1 - \frac{1}{k}\right)\left(\frac{n}{n-1}\right)$.*
- *There is a solution to the standard LP relaxation for* MAX-k-ORDERING *on D with value $\left(1 - \frac{1}{k}\right)$.*

In particular, the above implies a $(2 - 2/n)$ integrality gap for the LP relaxation in Figure 5.1.

Proof. The number of forward edges is simply the number of ordered pairs of vertices (u, v) with distinct labels. By Turan's Theorem, the optimal integral solution is to partition the vertices into k subsets whose sizes differ by at most 1, giving each subset a distinct label from $\{1, \ldots, k\}$. This implies that there are at most $\frac{n^2}{2}\left(1 - \frac{1}{k}\right)$ forward edges. Hence, the optimal integral solution has value,

$$\frac{1}{2}\left(1 - \frac{1}{k}\right)\left(\frac{n}{n-1}\right).$$

On the other hand, consider an LP solution that assigns $x_i^u = \frac{1}{k}$ for all $u \in V$ and $i \in [k]$, and $y_{i,i+1}^e = \frac{1}{k}$ for all $e = (u, v) \in A$ and $i \in [k-1]$. Each edge e contributes,

$$\sum_{i=1}^{k-1} y_{i,i+1}^e = \left(1 - \frac{1}{k}\right),$$

to the objective. □

We now claim that the above integrality gap is essentially retained even after near polynomial rounds of the Sherali-Adams constraints given in Figure 5.2. In particular, we prove the following that implies Theorem 14.

Theorem 18. *For any constant $\varepsilon > 0$, there is $\gamma > 0$ such that for large enough $n \in \mathbb{Z}^+$ and any $k \in \{2, \ldots, n\}$, there is a weighted digraph $D^* = (V^*, A^*)$ satisfying,*

- *The optimum of* Max-k-Ordering *on D^* is at most $\frac{1}{2}\left(1 - \frac{1}{k}\right) + \varepsilon$.*
- *The LP relaxation for* Max-k-Ordering *augmented with $n^{(\gamma/\log\log k)}$ rounds of Sherali-Adams constraints has objective value at least $\left(1 - \varepsilon\right)\left(1 - \frac{1}{k}\right)$.*

The rest of this section is devoted to proving the above theorem. Our construction of the integrality gap uses the techniques of Lee [86] who proved a similar gap for a variant of the *graph pricing* problem. We begin by showing that a sparse, random subgraph D', of the complete digraph D mentioned

above, also has a low optimum solution. For this, we require the following result on *ε-samples* [90] for finite set systems that follows from Hoeffding's bound. The reader is referred to Theorem 3.2 in [91] for a proof.

Theorem 19. *Let* $(\mathcal{U}, \mathcal{S})$ *denote a finite set system*[2]*. Suppose* $\tilde{A}$ *is a multi-set obtained by sampling from* $\mathcal{U}$ *independently and uniformly* m *times where* $m \geq \frac{1}{2\varepsilon^2} \ln \frac{2|\mathcal{S}|}{\delta}$*. Then with probability at least* $1 - \delta$*,*

$$\left| \frac{|\tilde{A} \cap S|}{|\tilde{A}|} - \frac{|S|}{|\mathcal{U}|} \right| \leq \varepsilon, \quad \forall S \in \mathcal{S}.$$

$\tilde{A}$ *is referred to as an* ε*-sample for* $(\mathcal{U}, \mathcal{S})$*.*

In order to construct a solution that satisfies Sherali-Adams constraints for a large number of rounds, we require the instance to be *locally sparse*, i.e. the underlying undirected graph is almost a tree on subgraphs induced by large (but bounded) vertex sets. We use the notion of *path decomposability* as defined by Charikar et al. [88] as a measure of local sparsity.

Definition 7. [Path Decomposability] *A graph* G *is* l*-path decomposable if every 2-connected subgraph* H *of* G *contains a path of length* l *such that every vertex of the path has degree 2 in* H*.*

[2] A set system $(\mathcal{U}, \mathcal{S})$ consists of a ground set $\mathcal{U}$ and a collection of its subsets $\mathcal{S} \subseteq 2^{\mathcal{U}}$. It is called finite if $|\mathcal{U}|$ is finite.

We proceed to show that the sparse graph D' obtained as above can be further processed so that it is locally sparse. Applying the techniques in [88] and [86] yields a solution with high value that satisfies the Sherali-Adams constraints.

5.7.1 Constructing a Sparse Instance

Lemma 21. *Let $D = (V, A)$ be the complete digraph on n vertices, let $k \in \{2, \ldots, n\}$ and $\varepsilon > 0$ be a small constant. The weighted digraph $D' = (V, A')$ obtained by sampling $\Omega(\frac{n \log k}{\varepsilon^2})$ edges uniformly at random satisfies $Opt(D') \leq \frac{1}{2}\left(1 - \frac{1}{k}\right) + \varepsilon$ with high probability, where Opt denotes the optimum of* Max-k-Ordering.

Proof. Let $V = [n]$ and $A \subseteq [n] \times [n]$ denote the vertices and edges of the digraph D. Let $\rho : [n] \to [k]$ denote some labeling of the vertices. Let $S_\rho := \{(i, j) \mid \rho(i) < \rho(j), (i, j) \in A\}$ denote the subset of edges that are satisfied by ρ, and $\mathcal{S} := \{S_\rho \mid \forall \text{ labelings } \rho\}$ denote the collection of such subsets induced by all feasible labelings. Since the number of distinct labelings is k^n, we have that $|\mathcal{S}| \leq k^n$.

We now construct an $(\varepsilon/2)$-sample for the set system $(A, \mathcal{S})$ by randomly sampling edges (with replacement) as per Theorem 19. Let $\tilde{A}$ denote the

bag of randomly chosen $\left(\frac{2}{\varepsilon^2}\ln\frac{2k^n}{\delta}\right)$ edges. Substituting $\delta = \frac{1}{n}$, we get that $|\tilde{A}| = \Omega(\frac{n\log k}{\varepsilon^2})$ and with probability at least $1 - \frac{1}{n}$ we have,

$$\left|\frac{|S_\rho \cap \tilde{A}|}{|\tilde{A}|} - \frac{|S_\rho|}{|A|}\right| \leq \varepsilon/2, \quad \forall\rho. \tag{5.17}$$

In order to avoid multi-edges in the construction, we define the weight of an edge $w(u,v)$ to be the number of times that edge is sampled in $\tilde{A}$ and let A' denote the set of thus weighted edges obtained from $\tilde{A}$. Equation (5.17) along with Claim 10 guarantees that the optimum integral solution of the weighted graph D' induced by the edges A' is bounded by $Opt(D') \leq Opt(D) + \varepsilon \leq \frac{1}{2}\left(1-\frac{1}{k}\right)\left(\frac{n}{n-1}\right) + \varepsilon/2 \leq \frac{1}{2}\left(1-\frac{1}{k}\right) + \varepsilon$ as desired. □

Given a digraph, let its corresponding undirected multigraph be obtained by replacing every directed edge by the corresponding undirected one. Note that if the digraph contains both (u,v) and (v,u) edge for some pair of vertices, then the undirected multigraph contains two parallel edges between u and v.

We now show that D' obtained in Lemma 21 can be slightly modified so that its corresponding underlying multigraph is almost regular, has high girth, and is locally sparse i.e. all small enough subgraphs are l-path decomposable for an appropriate choice of parameters.

Lemma 22. *Let $k = \{2, \ldots, n\}$ and $\varepsilon > 0$ be a small enough constant. Given the complete digraph $D = (V, A)$ on n vertices, let $D' = (V, A')$ be obtained by sampling (with replacement) $\Theta(\frac{n \log k}{\varepsilon^2})$ edges uniformly at random. Then, with high probability there exists a subgraph $D'' = (V, A'')$ of D' obtained by removing at most $\varepsilon|A'|$ edges, such that the undirected multigraph G'' underlying D'' satisfies the following properties:*

1. Bounded Degree: *The maximum degree of any vertex is at most 2Δ and G'' has $\Omega(\Delta n)$ edges, where $\Delta = \Theta(\frac{\log k}{\varepsilon^2})$.*

2. High Girth: *G'' has girth at least $l = O(\frac{\log n}{\log \Delta})$.*

Proof. Since D' is obtained by sampling $\Theta(\frac{n \log k}{\varepsilon^2})$ edges uniformly, the probability that any given edge is selected is $p = \Theta(\frac{\log k}{\varepsilon^2 n})$. In addition, these events are negatively correlated. Therefore given any set of edges S, the probability that all the edges in S are sampled is upper bounded by $p^{|S|}$.

Bounded Degree: As the maximum degree of any vertex in D is at most $2n$, the expected degree of any vertex $v \in V$ in D' is at most $\Delta = 2pn = \Theta(\frac{\log k}{\varepsilon^2})$. Call a vertex $v \in V$ *bad* if it has degree more than 2Δ in D', and call and edge $(u, v) \in A'$ bad if either u or v is bad. Now, for any edge (u, v), the probability that (u, v) is bad given that $(u, v) \in A'$ is at most $2e^{-\frac{\Delta}{3}}$ by Chernoff bound.

Hence, the expected number of bad edges is at most $2e^{-\frac{\Delta}{3}}|A'|$. Finally, by Markov's inequality, with probability at least $\frac{1}{2}$, the number of bad edges is at most $4e^{-\frac{\Delta}{3}}|A'|$. Deleting all bad edges guarantees that the maximum degree of D' is at most 2Δ and with probability at least half, we only delete $4e^{-\frac{\Delta}{3}}|A'|$ edges which is much smaller than $\varepsilon|A'|$ since $\Delta = \Theta(\frac{\log k}{\varepsilon^2})$.

Girth Control: Let G' denote the undirected multigraph underlying D'. Since the degree of any vertex in D is at most $2n$, we have

$$\mathbb{E}[\text{Number of cycles in } G' \text{ of length } i] \leq n(2n)^{i-1}p^i \leq (C\Delta)^i$$

for some constant C. For $i = O(\frac{\log n}{\log \Delta})$, we get

$$\mathbb{E}[\text{Number of cycles in } G' \text{ of length } i] \leq n^{0.5}$$

Summing up over all i in $2 \dots l = O(\frac{\log n}{\log \Delta})$, we get that the expected number of cycles of length up to l is at most $O(n^{0.6})$ and hence it is less than $O(n^{0.7})$ with high probability. We can then remove one edge from each such cycle (i.e. $o(n)$ edges) to ensure that the graph G'' so obtained has girth at least l. In particular, note that the corresponding digraph D'' has no 2-cycles. $\square$

Ensuring Local Sparsity

Using Lemma 22 we ensure that the subgraph of G'' induced by any subset of n^δ vertices is l-path decomposable for some constant $\delta > 0$. The following lemma shows that 2-connected subgraphs of G'' of polynomially bounded size are sparse.

Lemma 23. *The undirected multigraph G'' underlying the digraph D'' satisfies the following p, i.e., there exists $\delta > 0$ such that every 2-connected subgraph $\tilde{G}$ of G'' containing $t' \leq n^\delta$ vertices has only $(1+\eta)t'$ edges where $\eta = \frac{1}{3l}$.*

Proof. Let G denote the undirected multigraph underlying the graph D that was used as a starting point for Lemma 22. The proof proceeds by counting the number of possible "dense" subgraphs of G and showing that the probability that any of them exist in G'' after the previous sparsification steps is bounded by $o(1)$. We consider two cases based on the value of t'.

Case 1: $4 \leq t' \leq \frac{1}{\eta}$. We first bound the total number of 2-connected subgraphs of G with t' vertices and $t'+1$ edges. It is easy to verify that the only possible degree sequences for such subgraphs are $(4, 2, 2, \ldots)$ or $(3, 3, 2, 2, \ldots)$. Suppose it is $(4, 2, 2, \ldots)$ and let v be the vertex with degree 4. Now, there must be a sequence of $t'+2$ vertices $(v, \ldots, v, \ldots, v)$ that

represents an Eulerian tour. But the number of such sequences is upper bounded by $nt'(n)^{t'-1} = t'n^{t'}$ (n for guessing v, t' for guessing the position of v in the middle, and $n^{t'-1}$ to guess the other $t'-1$ vertices). Now assume that the degree sequence is $(3,3,2,2,\ldots)$ and u, v be the vertices with degree 3. Now, there must a sequence of $t+2$ vertices $(u,\ldots,v,\ldots,u,\ldots,v)$ that represents an Eulerian path from u to v. By a similar argument, the number of such sequences is bounded by $t'^2 n^{t'}$. Hence, we are guaranteed that the total number of 2-connected subgraphs of G with t' vertices and $t'+1$ edges is at most $2t'^2 n^{t'}$.

Therefore, the probability that there exists a subgraph of G'' with t' vertices and $t'+1$ edges for $4 \leq t' \leq \frac{1}{\eta} = 3l$ is at most

$$\sum_{t'=4}^{3l} 2t'^2 n^{t'} p^{t'+1} = \sum_{t'=4}^{3l} 2t'^2 n^{t'} \left(\frac{\Delta}{2n}\right)^{t'+1} \leq \frac{C}{n} l^3 \left(\frac{\Delta}{2}\right)^{3l+1},$$

where C is an appropriate constant. For $l = O(\frac{\log n}{\log \Delta})$, we have that $(\frac{C}{n}) l^3 (\frac{\Delta}{2})^{3l+1} \leq n^{-0.1} = o(1)$.

Case 2: $n^{\delta} \geq t' > \frac{1}{\eta} = 3l$. In this case, we count the number of subgraphs of G with t' vertices and $(1+\eta)t'$ edges. As shown by Lee [86], the number of such subgraphs is bounded by $C\alpha^{t'} n^{t'} (\frac{et'}{2\eta})^{2\eta t'}$ for some constants C and α.

Therefore, the probability that such a subgraph exists in G'' is at most

$$\begin{aligned} C\alpha^{t'}n^{t'}\left(\frac{et'}{2\eta}\right)^{2\eta t'}p^{(1+\eta)t'} &= C\alpha^{t'}n^{t'}\left(\frac{et'}{2\eta}\right)^{2\eta t'}\left(\frac{\Delta}{2n}\right)^{(1+\eta)t'} \\ &\leq (C_1\Delta^2)^{t'}\left(C_2\frac{l^2t'^2}{n}\right)^{t'/3l}, \end{aligned}$$

where C_1 and C_2 are appropriate constants. We choose $l = O(\frac{\log n}{\log \Delta})$ and $\delta \in (0, 0.1)$, such that above quantity is less than $n^{-0.1}$. Summing up over all $t' = 3l, \ldots, n^\delta$, we still have that probability such a subgraph exists is bounded by $o(1)$. □

Finally, we need the following lemma proved by Arora et al. [87] regarding the existence of long paths in sparse, 2-connected graphs.

Lemma 24. [87] *Let $l \geq 1$ be an integer and $0 < \eta < \frac{1}{3l-1}$, and let H be a 2-connected graph with t vertices and at most $(1+\eta)t$ edges and H is not a cycle. Then H contains a path of length at least $l+1$ whose internal vertices have degree 2 in H.*

Corollary 3. *Every subgraph $\tilde{G}$ of G'' that is induced on at most $t' \leq n^\delta$ vertices is $(l-1)$-path decomposable.*

Proof. Consider any 2-connected subgraph H of $\tilde{G}$. If H is not a cycle, then Lemma 23 and Lemma 24 together guarantee that H contains a path of

length at least $l+1$ such that all internal vertices have degree 2 in H, which gives us a path of length $(l-1)$ with all vertices of degree 2 in H. On the other hand, if H is a cycle, then Lemma 22 guarantees that H has at least $l+1$ vertices and hence again the required path exists. □

For convenience we replace $(l-1)$ in Corollary 3 with l, and since $l = \Theta(\frac{\log n}{\log \Delta})$, this does not change any parameter noticeably.

Final Instance

Theorem 20. *Given $k \in \{2, \ldots, n\}$ and constants $\varepsilon, \mu > 0$, there exists a constant $\gamma > 0$, and parameters $\Delta = \Theta(\frac{\log k}{\varepsilon^2})$, and $l = \Theta(\frac{\log n}{\log \Delta})$ such that there is an instance $\hat{D}$ (with underlying undirected graph $\hat{G}$) of* Max-k-Ordering *with the following properties*

- *Low Integral Optimum: $Opt(\hat{D}) \leq \frac{1}{2}(1 - \frac{1}{k}) + \varepsilon$.*
- *Almost Regularity: Maximum Degree of $\hat{G} \leq 2\Delta$, and $\hat{G}$ has $\Omega(\Delta n)$ edges.*
- *Local Sparsity: For $t < n^{\gamma/\log \Delta}$, every induced subgraph of G on $(2\Delta)^l t$ vertices is l-path decomposable.*
- *Large Noise: For $t < n^{\gamma/\log \Delta}$, $(1-\mu)^{l/10} \leq \frac{\mu}{5t}$.*

Note that $n^{\gamma/\log\Delta} = n^{\Omega(1/\log\log k)}$.

Proof. Let D'' be the digraph obtained from Lemma 22. Lemmas 21 and 22 imply that the digraph D'' so obtained (i) has low integral optimum, (ii) is almost regular, and (iii) has girth $\geq l$.

The large noise condition is satisfied by $l \geq (C\gamma \log n/\log\Delta)$ for an appropriate constant C.

Corollary 3 guarantees that the local sparsity condition is satisfied if $(2\Delta)^l t \leq n^\delta$, i.e. $l \leq C'(\delta-\gamma)\log n$ for another constant C'. Hence, by selecting a small enough constant γ and an appropriate $l = \Theta(\frac{\log n}{\log\Delta})$, the instance D'' obtained in Lemma 22 satisfies all the required properties. □

5.7.2 Constructing Local Distributions

Let $D = (V, A)$ be the instance of Max-k-Ordering constructed in Theorem 20 and let $G = (V, E)$ be the underlying undirected graph. We now show that there exists a solution to the LP after $t = n^{\gamma/\log\Delta}$ rounds of the Sherali-Adams hierarchy whose objective is at least $(1-\varepsilon)(1-\frac{1}{k})$. Our proof for the existence of such a solution essentially follows the approach of Lee [86]. Given a set of $t \leq n^{\gamma/\log\Delta}$ vertices $S = \{v_1, v_2, \ldots, v_t\}$, our goal is to give a distribution on events $\{\ell(v_1) = x_1, \ell(v_2) = x_2, \ldots, \ell(v_t) = x_t\}_{x_1,x_2,\ldots,x_t\in[k]}$.

Let $d(u, v)$ be the shortest distance between u and v in the (undirected) graph G. Let $V' \subset V$ be the set of vertices that are at most l distance away from S and let G' be the subgraph induced by V' on G. Since the maximum degree of vertices is bounded by 2Δ, we have $|V'| \leq (2\Delta)^l t$ and hence G' is l-path decomposable by Theorem 20.

The first step of the construction relies on the following theorem by Charikar et al. [92] that shows that if a graph G' is l-path decomposable, then there exists a distribution on partitions of V such that close vertices are likely to remain in the same partition while distant vertices are likely to be separated.

Theorem 21 (Charikar et al. [92])**.** *Suppose $G' = (V', E')$ is an l-path decomposable graph. Let $d(\cdot, \cdot)$ be the shortest path distance on G , and $L = \lfloor l/9 \rfloor$; $\mu \in [1/L, 1]$. Then there exists a probabilistic distribution of multicuts of G' (or in other words random partition of G' into pieces) such that the following properties hold. For every two vertices u and v,*

1. *If $d(u, v) \leq L$, then the probability that u and v are separated by the multicut (i.e. lie in different parts) equals $1 - (1 - \mu)^{d(u,v)}$; moreover, if u and v lie in the same part, then the unique shortest path between u and v also lies in that part.*

2. *If $d(u,v) > L$, then the probability that u and v are separated by the multicut is at least $1-(1-\mu)^L$.*

3. *Every piece of the multicut partition is a tree.*

Based on this random partitioning, we define a distribution on the vertices in S (actually in V'). As each piece of the above partition is a tree, given some vertex u with an arbitrary label i, we can extend it to a labeling ℓ for every other vertex in that piece such that every directed edge (x,y) in the piece satisfies $\ell(y)-\ell(x) = 1 \pmod{k}$.

For vertices u and v with $d(u,v) \leq L$, we say that label i for u and i' for v *match* if the labeling $\ell(u) = i, \ell(v) = i'$ can be extended so that for every directed edge (x,y) on the unique shortest path between u and v, $\ell(y)-\ell(x) = 1 \pmod{k}$. Note that there are exactly k such matching pairs for every u and v. We can now use Theorem 21 to obtain a random labeling as follows.

Corollary 4. *Suppose $G' = (V', E')$ is an l-path decomposable graph. Let $L = \lfloor l/9 \rfloor; \mu \in [1/L, 1]$. Then there exists a random labeling $r : V' \to [k]$ such that*

1. *If $d = d(u,v) \leq L$, then*

$$\Pr[r(u) = i, r(v) = i'] = \begin{cases} \frac{(1-\mu)^d}{k} + \frac{1-(1-\mu)^d}{k^2} & \textit{if } i \textit{ and } i' \textit{ match} \\ \frac{1-(1-\mu)^d}{k^2} & \textit{otherwise} \end{cases}$$

2. *If $d > L$, then*

$$\frac{1-(1-\mu)^L}{k^2} \leq \Pr[r(u) = i, r(v) = i'] \leq \frac{1-(1-\mu)^L}{k^2} + \frac{(1-\mu)^L}{k} \textit{ for any } i, i' \in [k]$$

Proof. We first sample from the distribution of multicuts given by Theorem 21. For every piece obtained, we pick an arbitrary vertex u and assign $r(u)$ to be a uniformly random label from $[k]$. Now, since each piece is a tree, we can propagate this label along the tree so that for every directed edge (v, w) we have $r(w) - r(v) = 1 \pmod{k}$. Note that the final distribution obtained does not depend on the choice of the initial vertex u.

Consider any two vertices u and v. If $d(u, v) \leq L$, then if u and v are in the same piece, then the path connecting u and v in the piece is the shortest path. If i and i' are matching labels, then

$$\begin{aligned} \Pr[r(u) = i, r(v) = i'] &= \Pr[u, v \text{ in the same piece}] \cdot \left(\frac{1}{k}\right) \\ &\quad + \Pr[u, v \text{ are separated}] \cdot \left(\frac{1}{k^2}\right). \end{aligned}$$

On the other hand, if i and i' are not matching,

$$\begin{aligned}\Pr[r(u) = i, r(v) = i'] &= \Pr[u, v \text{ in the same piece}] \cdot 0 \\ &\quad + \Pr[u, v \text{ are separated}] \cdot \left(\frac{1}{k^2}\right).\end{aligned}$$

Similarly, if $d(u, v) > L$, then $\Pr[r(u) = i, r(v) = i']$ is lower bounded by $\Pr[u, v \text{ are separated}]/k^2$ and upper bounded by $\Pr[u, v \text{ in the same piece}]/k + Pr[u, v \text{ are separated}]/k^2$. Substituting the separation probabilities in Theorem 21 proves the desired result. □

The above random labeling defines a distribution ν_S over labels of pairs of vertices as follows.

Definition 8. *Let $S = \{v_1, v_2, \ldots, v_t\}$ be a fixed set of vertices. For any two vertices $u, v \in S$ and $i, i' \in [k]$, let $\nu_S(u(i), v(i')) = \Pr[x(u) = i, x(v) = i']$ in the local distribution on S defined by r in Corollary 4.*

We now define another distribution ρ over labels for pairs of vertices that is independent of the choice of the set S as follows.

Definition 9. *For any vertices $u \neq v$ and $i, i' \in [k]$, let $\rho(u(i), v(i')) = \Pr[x(u) = i, x(v) = i']$ if $d(u, v) \leq L$, and $\frac{1}{k^2}$ otherwise. Also define $\rho(u(i), u(i)) = \frac{1}{k}$ and $\rho(u(i), u(i')) = 0$ for $i \neq i'$. Since the shortest path*

between u and v is unique when $d(u,v) \leq L$, ρ is uniquely defined by D and G and is independent of the choice of set S.

Lee [86] shows that it is possible to use the ρ and ν_S distributions defined above to produce consistent distributions over events of the form $\{\ell(v_1) = x_1, \ldots, \ell(v_t) = x_t\}_{x_1,\ldots,x_t \in [k]}$. Further, these distributions need to be consistent, i.e., the marginal distribution on $S \cap S'$ does not depend on the choice of its superset (S or S') that is used to obtained the larger local distribution. The key idea here as shown by Charikar et al. [88] is to embed ρ into Euclidean space with a small error to obtain tk vectors $\{v(i)\}_{v \in S, i \in [k]}$ such that $u(i) \cdot v(i') \approx \rho(u(i), v(i'))$. This uses the large noise property in Theorem 20. The following lemma appears as Lemma 5.7 in [86].

Lemma 25 (Lee [86])**.** *There exist tk vectors $\{v(i)\}_{v \in S, i \in [k]}$ such that $||v(i)||_2^2 = \mu + \frac{1}{T+1}$ and $u(i) \cdot v(i') = \frac{\mu}{2} + \rho(u(i), v(i'))$.*

Given such tk vectors, one can use a geometric rounding scheme to define the consistent local distributions. Note that the local distribution is completely defined by the pairwise inner products of the vectors which, for any two vectors, is independent of the subset S. Lee [86] shows that the following simple rounding scheme suffices to obtain a good distribution: choose a random Gaussian vector g, and for each vertex v, let $\ell(v) = \arg\max_i (v(i) \cdot g)$.

Lemma 26 (Lee [86] [3]). *There exists a $\mu > 0$ depending on k and ε such that, in the above rounding scheme, for any edge (u, v) and any label $i \in [k]$ the probability that $\ell(u) = i$ and $\ell(v) = i + 1 \ (mod\ k)$ is at least $\frac{1-12\varepsilon}{k}$.*

Consider the solution to $n^{\gamma/\log\Delta}$ rounds of the Sherali-Adams hierarchy obtained by the above rounding process. For any edge $(u, v) \in A$, its contribution to the objective is

$$\begin{aligned}\sum_{1\le i<i'\le k} \Pr[\ell(u) = i, \ell(v) = i'] &\geq \sum_{i\in[k-1]} \Pr[\ell(u) = i, \ell(v) = i + 1] \\ &\geq \sum_{i\in[k-1]} \frac{1 - 12\varepsilon}{k}\end{aligned}$$

The last inequality follows due to Lemma 26. Thus we have a fractional solution with value at least $(1-12\varepsilon)(1-\frac{1}{k})$. This, along with the low optimum of the instance from Theorem 20 completes the proof of Theorem 18.

5.8 The DED(k) Problem

Recall that the DED(k) problem is to remove the minimum weight subset of edges from a given DAG so that the remaining digraph does not contain any path of length k.

[3] The lemma follows from the proof of Lemma 5.8 of Lee [86] by substituting $l_A(u, v) = 1$.

5.8.1 Combinatorial k-Approximation

In the unweighted case (i.e. all edges have unit weight), the following simple scheme is a k-approximation algorithm. As long as the DAG contains a directed path P of length k, delete *all* edges of that path. It is easy to see that the above scheme guarantees a k-approximation as the optimal solution must delete at least one edge from the path P while the algorithm deletes exactly k edges.

The following slightly modified scheme that uses the local ratio technique yields a k-approximation for weighted DAGs.

Algorithm LocalRatio:

1. $S \leftarrow \{e \in E \mid w(e) = 0\}$

2. While $(V, E \setminus S)$ contains a path P of length k

   (a) $w_{min} \leftarrow \min_{e \in P}(w(e))$

   (b) $w(e) = w(e) - w_{min}, \forall e \in P$

   (c) $S =\leftarrow \{e \in E \mid w(e) = 0\}$

Theorem 22. LocalRatio *is a polynomial time k-approximation to the* DED(k) *problem on weighted DAGs.*

Proof. We note that the LocalRatio terminates in at most $|E|$ iterations as the weight of at least one edge reduces to 0 in each iteration. Also, since one can check if there exists a path of length k in DAG via a dynamic programming, it follows that LocalRatio runs in polynomial time.

Let $\mathcal{O} \subseteq E$ be an optimal solution and $\mathcal{S} \subseteq E$ be the solution returned by LocalRatio. Note that an edge is in $\mathcal{S}$ if its weight is reduced to 0 in some iteration of the algorithm. Thus, the weight of $\mathcal{S}$ is upper bounded by the total reduction in the weight of the edges. At each iteration, for a path P of length k, the reduction is at most k times the minimum weight edge (according to the current weights) on in P. Since there is at least one edge e in P which is in $\mathcal{O}$, we charge this reduction to the weight of e. Then the weight of e decreases by at least $1/k$ factor of what is charged to it, and it cannot decrease beyond 0. Thus, the weight of $\mathcal{S}$ is at most the k times the weight of $\mathcal{O}$. □

5.8.2 k-Approximation via LP Rounding

The natural LP relaxation for DED(k) on an n-vertex DAG $D = (V, E)$ is given in Figure 5.3. This relaxation has $n^{O(k)}$ constraints. However, when the input graph is a DAG, it admits the following polynomial time separation

oracle for any $k \in \{2, \ldots, n-1\}$.

Separation Oracle and Rounding.

For each vertex $v \in V$ and integer $t \in [n]$, define $a_t^v = \min_P(\sum_{e \in P} x_e)$ where P is a path of length t that ends at vertex v. Once we compute all these a_t^v values, then a constraint is violated if and only if there is a vertex v such that $a_k^v < 1$.

On a DAG the $\{a_t^v \mid v \in V, t \in [n]\}$ can be computed by dynamic programming. First assume that the vertices are arranged in a topological order. For any vertex v with no predecessors, set $a_t^v = 0, \forall t$. Otherwise, we have the following recurrence,

$$a_t^v = \min_{u \in \text{predecessors}(v)} (x_{(u,v)} + a_{t-1}^u).$$

It is easy to see that the above recurrence leads to a dynamic program on a DAG. Once we obtain an optimal solution to the LP relaxation, a simple threshold based rounding using a threshold of $1/k$ yields a k-approximation.

Theorem 23. *The standard LP relaxation for* DED(k) *on n-vertex DAGs can be solved in polynomial time for $k = \{2, \ldots, n-1\}$ and yields a k-approximation.*

5.8.3 Hardness of Approximation

For fixed integer $k \geq 2$ and arbitrarily small constant $\varepsilon > 0$, Svensson [84] showed factor $(k-\varepsilon)$ UGC-hardness of the *vertex deletion* version of DED(k), which requires deleting the minimum number of vertices from a given DAG to remove all paths with k vertices. In particular, [84] proves the following structural hardness result.

Theorem 24 (Svensson [84])**.** *For any fixed integer $t \geq 2$ and arbitrary constant $\varepsilon > 0$, assuming the UGC the following is NP-hard: Given a DAG $D(V, E)$, distinguish between the following cases:*

- (Completeness): *There exist t disjoint subsets $V_1, \ldots, V_t \subset V$ satisfying $|V_i| \geq \frac{1-\varepsilon}{t}|V|$ and such that a subgraph induced by any $t-1$ of these subsets has no directed path of t vertices.*

- (Soundness): *Every induced subgraph on $\varepsilon|V|$ vertices has a path with $|V|^{1-\varepsilon}$ vertices.*

The following theorem provides a simple gadget reduction from the above theorem to a hardness for DED(k) on DAGs.

Theorem 25. *Assuming the UGC, for any constant $k \geq 4$ and $\varepsilon > 0$, the* DED(k) *problem on weighted DAGs is NP-hard to approximate with a factor*

better than $(\lfloor k/2 \rfloor - \varepsilon)$.

Proof. Fix $t = \lfloor k/2 \rfloor$. Let $D = (V, E)$ be a hard instance from Theorem 24 for the parameter t and small enough $\varepsilon > 0$. The following simple reduction yields a weighted DAG $H = (V_H, E_H)$ as an instance of DED(k). Assign $w(e) = 2|V|$ to every edge $e \in E$. Split every vertex $v \in V$ into v_{in} and v_{out} and add a directed edge (v_{in}, v_{out}) of weight 1. Also every edge entering v now enters v_{in} while edges leaving v now leave v_{out}. It is easy to see that removing all edges of weight 1 from H eliminates all paths with 2 edges, implying that the optimum solution has weight at most $|V|$. Thus, we may assume that the optimum solution does not delete any edge of weight $2|V|$.

We now show that Theorem 24 implies that it is UG-hard to distinguish whether: (Completeness) H has a solution of cost $\leq (\frac{1}{t} + \varepsilon)|V|$, or (Soundness) H has no solution of cost $(1 - \varepsilon)|V|$. This immediately implies the desired $(t - \varepsilon') = (\lfloor k/2 \rfloor - \varepsilon')$ UGC-hardness for DED(k).

- (Completeness) There exists a subset $S \subseteq V$ of size at most $(\frac{1}{t} + \varepsilon)|V|$, such that removing S eliminates all paths in D of t vertices. Let S' denote the set of edges in H corresponding to the vertices in S. It is easy to observe that $H(V_H, E_H \setminus S')$ has no paths of length (number of edges) $2t$. Thus, S' is a feasible solution to the DED(k) problem of

cost $(\frac{1}{t} + \varepsilon)|V|$.

- (Soundness) Assume for the sake of contradiction, that we have an optimal solution $S' \subseteq E_H$ of cost at most $(1 - \varepsilon)|V|$. Since S' is an optimal solution it only has edges of weight 1, each of which correspond to a vertex in V. Let S denote this set of vertices in V. By construction, since $H(V_H, E_H \setminus S')$ has no paths with k edges, $D[V \setminus S]$ has no induced paths with $\lfloor k/2 \rfloor + 1 = t + 1$ vertices. Further, since $|S'| = |S| \leq (1 - \varepsilon)|V|$, we have $|V \setminus S| \geq \varepsilon|V|$. Thus, we have a set of size $\varepsilon|V|$ that has no induced paths of length $t + 1$. This is a contradiction since every induced subgraph of $\varepsilon|V|$ vertices has a path of length $|V|^{1-\varepsilon} \geq t + 1$.

□

5.9 Linear Soft Precedence Scheduling

In this section, we consider the *Linear Soft Precedence Scheduling* problem. Once again, the input consists of a set n jobs, precedence constraints between the jobs, and a deadline k. We wish to schedule all jobs so that all jobs are completed by their deadline and we try to satisfy the precedence constraints

as well as we can. We now try to capture the scenario where the penalty incurred due to violation of a precedence constraint is proportional to the extent by which the constraint is violated. Let s_j and f_j denote the starting time and finishing time of a job j respectively in a given schedule. We say a precedence constraint $u \prec v$ is satisfied if and only if $s_v \geq f_u$, i.e., job v is processed only after job u has finished processing. However, if the constraint is not satisfied, then we incur a penalty of $(f_u - s_v)$.

Formally, we define the *Linear Soft Precedence Scheduling* problem as follows -

Definition 10. Linear Soft Precedence Scheduling*: We are given a set V of n jobs, their associated integer processing times $\{p(j)\}_{j=1}^{n}$, and an integer deadline k. In addition, we have weighted precedence constraints between jobs where $e = (u, v)$ indicates that u must be finished before v is started and $w(e)$ is the weight of constraint e. The objective is to find a schedule that processes all jobs by the deadline k and minimizes the sum of penalties for every violated constraint. If a constraint $u \prec v$ is violated, we incur a penalty of $(f_u - s_v)$.*

Due to the nature of the penalty, we can formulate the Linear Soft Precedence Scheduling problem naturally as a linear program as shown

in Figure 5.4. As a result, we can solve the problem optimally in polynomial time. The LP has variables s_v and f_v that represents the starting and finishing times of a job $v \in V$ respectively. The variable c_e equals the penalty that constraint $e = (u, v)$ needs to bear.

$$\min \sum_{e \in A} c_e \cdot w(e)$$

subject to,

$$\begin{aligned}
&\forall v \in V, && f_v = s_v + p(v) \\
&\forall v \in V, && f_v \leq k \\
&\forall e = (u, v) && c_{(u,v)} \geq f_u - s_v \\
&\forall e = (u, v), && c_e \geq 0 \\
&\forall v \in V, && s_v \geq 0
\end{aligned}$$

Figure 5.4: LP for Linear Soft Precedence Scheduling

.

5.10 Future Directions

The Max-k-Ordering and related problems that we defined in this chapter lead the way to a number of interesting research directions. In this section, we summarize some of the prominent open problems and future research directions.

Improved Approximation using Semidefinite Programming

In Section 5.5, we develop an LP-rounding based randomized 2-approximation algorithm for the Max-k-Ordering problem for any $k \in [2, n]$. Further we showed that the LP relaxation in Figure 5.1 has an integrality gap of $2 - \varepsilon$ even after augmentation with almost polynomial rounds of the Sherali-Adams hierarchy for all $k \in [2, n]$. This lower bound rules out the existence of better algorithms based on the natural LP relaxation.

On the other hand, Guruswami et al. [80] show that the Maximum Acyclic Subgraph problem (Max-k-Ordering when $k = n$) is hard to approximate with a factor better than 2 assuming the Unique Games Conjecture. Hence for general k, we cannot hope for a better than 2 approximation. However

much better results are known in the special case of Max-DiCut (Max-k-Ordering when $k = 2$). Following the breakthrough work of Goemans and Williamson [93], a series of papers [76, 75, 74, 73] gave successively improved approximations for Max-DiCut leading to the best-known approximation factor of ≈1.144. On the other hand, Khot et al. [82] show that Max-DiCut is hard to approximate with a factor better than 1.1382 assuming the Unique Games Conjecture. Figure 5.5 shows the current status of the approximation factor for Max-k-Ordering for different values of k.

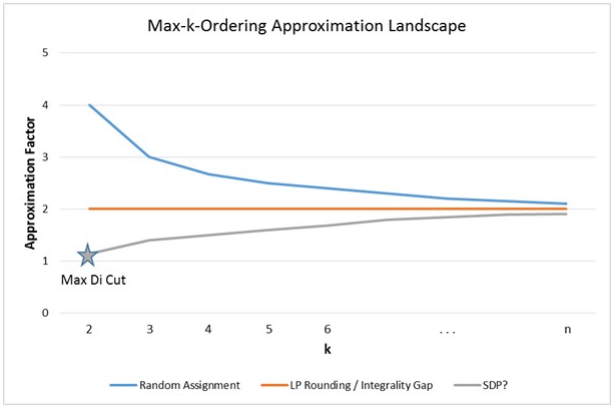

Figure 5.5: Approximation Landscape for Max-k-Ordering

It remains unclear if the gray line in Figure 5.5 can be achieved. Our first open problem is to design an SDP-rounding based algorithm for the Max-k-Ordering problem that improves on the 2-approximation for small value of k.

Improved Algorithms for DAGs

The Max-k-Ordering problem is motivated by the scheduling jobs with soft precedence constraints by a hard deadline. Indeed, in most practical scenarios, the underlying precedence graph for a set of jobs is acyclic. However, our algorithms and analysis do not make use of this fact and instead assume that the precedence graph G is a general directed graph. Our next open question is to design improved algorithms for the important case when the underlying graph is a DAG.

Indeed, since Max-DiCut remains NP-complete on a DAG [94], Max-k-Ordering remains a hard problem even on DAGs. Surprisingly even for Max-DiCut on DAGs, better approximations than on a general digraph are not known. Hence, we continue our quest for improved approximation algorithms.

Closing the Gap for OffsetRMAS

In Section 5.6, we develop an LP-rounding based randomized 2.344-approximation algorithm for the OffsetRMAS problem. Since OffsetRMAS generalizes the Max-k-Ordering problem, it retains the inapproximability beyond a factor of 2 (assuming the UGC). Our final open question is to close this gap

in the understanding of OffsetRMAS by either finding a 2-approximation algorithm or by improving the lower bound.

Chapter 6

Conclusion

Job scheduling and resource allocation are among the most fundamental problems in Computer Science. While scheduling algorithms are mechanisms that assign tasks to machines, more broadly, they play a major role of assigning scare resources to competing jobs. To quote from an article by Birmal et al. (SIGACT News, [3]), "Scheduling, it turns out, comes down to deciding how to spend money." New computing environments bring new challenges and the design and analysis of appropriate scheduling models is fundamental to realize the true potential of these advances.

In this thesis, we have studied various scheduling problems motivated by the need to minimize energy costs as well as network congestion in datacenters.

In Chapter 2, we introduced our framework for network-aware energy-efficient scheduling and designed approximation algorithms for the same. Our model simultaneously generalizes previous work on energy-efficient scheduling such as the machine activation problem [9] as well as fundamental graph theoretic problems such as hypergraph covering with hard capacities [49, 50]. Modern datacenters are composed of tens of thousands machines interconnected with a fast, high bandwidth network. Consequently any job has a large amount of flexibility in the machine on which it can be processed and yet not all machines are identical. Traditional models do not accurate capture this flexibility and either treat all machines as identical (For e.g. [9]) or limit the number of machines to which a job can be assigned (For e.g. [49, 50]). Our results indicate that effectively balancing energy efficiency as well as network congestion leads to new challenges that cannot be overcome by obvious extensions of prior work in either domain. Although we obtain algorithms that yield provably good schedules with respect to all the three criteria (machine activation cost, makespan, network congestion) in polynomial time, our algorithms are not yet applicable in practice. In particular, our algorithms rely on iterative rounding of auxiliary linear programs and are thus prohibitively slow to use in a real datacenter. Our primary focus in this work is to demonstrate that it

is possible to obtain good approximation algorithms in this highly non-trivial scheduling scenario. We believe that the existence of such a result should provide an impetus to future work that yields both better approximation algorithms as well as fast heuristics that work well in practical settings.

In Chapters 3 and 4, we have considered two causes for congestion in datacenter networks. Application frameworks such as MapReduce [6] and Hadoop [7] encourage the development of applications that alternate between computation and communication phases. Massively parallelizable applications distribute their computation tasks over hundreds of machines and the intermediate data generated is then "shuffled" and grouped together to be passed on to the next computation stage during a communication phase. Traditional scheduling frameworks that treat each job as an atomic unit fail to account for application-level objectives in this setting. Co-flow scheduling is a recent networking abstraction introduced by Chowdhury and Stoica [13] to accurately capture these application-level scheduling objectives. In Chapter 3, we have shown that co-flow scheduling shares greater similarities with the well-studied concurrent open shop scheduling problem than previously believed and utilized this connection to obtain improved approximation algorithms. In contrast to the data transfer necessitated due to actual communication requirements for

jobs, data often needs to be routed through a network for monitoring and security purposes. In Chapter 4, we focused on the data detours that are caused due to the deployment of middleboxes in a datacenter. By exploiting the hierarchical layout of datacenters, we have developed efficient algorithms for middlebox deployment that are either optimal (in the soft-capacitated case) or have small approximation factors (in the hard-capacitated case). Note that while the co-flow scheduling problem models the datacenter as a single non-blocking switch, machines in a datacenter are usually connected in some hierarchical fashion. In this work we leave open the question of scheduling co-flows over a hierarchical datacenter network.

While the previous chapters deal with the question of *where* should a job be scheduled and *how* should its data be transferred over the network, in Chapter 5 we study the question of in *what order* should jobs be processed. For a large number of applications, precedence constraints specify a partial order over the set of jobs that need to be executed. However, in the presence of hard deadline constraints, it is often impossible to find a schedule that satisfies all precedence constraints. Since some precedence constraints are more important than others, we introduced the Max-k-Ordering problem to maximize the total weight of satisfied constraints while completing all

the jobs by their deadline and design tight approximation algorithms for the same. Soft precedence constraints add a new dimension to many well studied scheduling problems and we believe that it will inspire more interesting algorithmic work in the future.

Bibliography

[1] Cisco Visual Networking. Cisco Global Cloud Index: Forecast and Methodology, 2012-2017 White Paper, 2013.

[2] Data Center Efficiency Assessment. `http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf`.

[3] Ken Birman, Gregory Chockler, and Robbert van Renesse. Toward a cloud computing research agenda. *ACM SIGACT News*, 40(2):68–80, 2009.

[4] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. CONGA: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514. ACM, 2014.

[5] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 407–420. ACM, 2015.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] Tom White. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012. `https://hadoop.apache.org`.

[8] Koyel Mukherjee, Samir Khuller, and Amol Deshpande. Algorithms for the thermal scheduling problem. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 949–960. IEEE, 2013.

[9] Samir Khuller, Jian Li, and Barna Saha. Energy efficient scheduling via partial shutdown. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1360–1372. Society for Industrial and Applied Mathematics, 2010.

[10] Manish Purohit and Barna Saha. A Framework for Network-Aware Energy-Efficient Scheduling. *Submitted*, 2016.

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

[12] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[13] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

[14] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication, SIGCOMM*, pages 443–454, 2014.

[15] Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM*, pages 393–406, 2015.

[16] Zhen Qiu, Cliff Stein, and Yuan Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 294–303. ACM, 2015.

[17] Yangming Zhao, Kai Chen, Wei Bai, Minlan Yu, Chen Tian, Yanhui Geng, Yiming Zhang, Dan Li, and Sheng Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 424–432. IEEE, 2015.

[18] Samir Khuller and Manish Purohit. Improved Approximation Algorithms for Scheduling Co-Flows. *Submitted*, 2016.

[19] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10):892–901, 1985.

[20] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009.

[21] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the 2013 ACM Conference on Special Interest Group on Data Communication, SIGCOMM*, pages 27–38. ACM, 2013.

[22] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 227–240, Berkeley, CA, USA, 2013. USENIX Association.

[23] Seungjoon Lee, Manish Purohit, and Barna Saha. Firewall placement in cloud data centers. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 52. ACM, 2013.

[24] Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1):29–38, 1999.

[25] Fabián A Chudak and Dorit S Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25(5):199–204, 1999.

[26] Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[27] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[28] Eugene L Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[29] François Margot, Maurice Queyranne, and Yaoguang Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.

[30] Andreas S Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *Integer Programming and Combinatorial Optimization*, pages 301–315. Springer, 1996.

[31] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 745–754. ACM, 2010.

[32] David Lesaint, Deepak Mehta, Barry O'Sullivan, Luis Quesada, and Nic Wilson. A soft global precedence constraint. In *IJCAI*, pages 566–571, 2009.

[33] Piotr Jaskowski and Anna Sobotka. Using soft precedence relations for reduction of the construction project duration. *Technological and Economic Development of Economy*, 18(2):262–279, 2012.

[34] Sreyash Kenkre, Vinayaka Pandit, Manish Purohit, and Rishi Saket. On the approximability of digraph ordering. In *Algorithms-ESA 2015*, pages 792–803. Springer, 2015.

[35] Christos H Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM journal on computing*, 19(2):322–328, 1990.

[36] Evripidis Bampis, Rodolphe Giroudeau, and Alexander Kononov. Scheduling tasks with small communication delays for clusters of processors. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 314–315. ACM, 2001.

[37] David Bernstein and Izidor Gertner. Scheduling expressions on a pipelined processor with a maximal delay of one cycle. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1):57–66, 1989.

[38] Ph Chrétienne and C Picouleau. Scheduling with communication delays: A survey. *Scheduling theory and its applications*, pages 65–90, 1995.

[39] Daniel W Engels, Jon Feldman, David R Karger, and Matthias Ruhl. Parallel processor scheduling with delay constraints. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 577–585. Society for Industrial and Applied Mathematics, 2001.

[40] Theodora Varvarigou, Vwani P Roychowdhury, T Kallath, Eugene Lawler, et al. Scheduling in and out forests in the presence of communication delays. *Parallel and Distributed Systems, IEEE Transactions on*, 7(10):1065–1074, 1996.

[41] Bart Veltman, BJ Lageweg, and Jan Karel Lenstra. Multiprocessor scheduling with communication delays. *Parallel computing*, 16(2):173–182, 1990.

[42] Rodolphe Giroudeau and Jean-Claude König. General scheduling non-approximability results in presence of hierarchical communications. *European Journal of Operational Research*, 184(2):441–457, 2008.

[43] Christophe Picouleau. New complexity results on scheduling with small communication delays. *Discrete Applied Mathematics*, 60(1):331–342, 1995.

[44] Cynthia Phillips, Clifford Stein, and Joel Wein. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10(4):573–598, 1997.

[45] Sungjin Im and Benjamin Moseley. Scheduling in bandwidth constrained tree networks. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, pages 171–180. ACM, 2015.

[46] Laurence Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4), 1982.

[47] Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, 2006.

[48] Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. In *Automata, Languages and Programming*, pages 164–175. Springer, 2003.

[49] Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *Automata, Languages, and Programming*, pages 762–773. Springer, 2012.

[50] Wang Chi Cheung, Michel X Goemans, and Sam Chiu-wai Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1714–1726. SIAM, 2014.

[51] Lisa Fleischer. Data center scheduling, generalized flows, and submodularity. In *ANALCO*, pages 56–65. SIAM, 2010.

[52] Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. Online mixed packing and covering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 85–100. SIAM, 2013.

[53] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1-3):461–474, 1993.

[54] Cloud dataflow. `https://cloud.google.com/dataflow/`.

[55] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[56] Monaldo Mastrolilli, Maurice Queyranne, Andreas S Schulz, Ola Svensson, and Nelson A Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

[57] Zhi-Long Chen and Nicholas G Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072–1089, 2007.

[58] Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order scheduling models: hardness and algorithms. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 96–107. Springer, 2007.

[59] Joseph Y-T Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007.

[60] Guoqing Wang and TC Edwin Cheng. Customer order scheduling to minimize total weighted completion time. *Omega*, 35(5):623–626, 2007.

[61] Maurice Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993.

[62] Zhen Qiu, Cliff Stein, and Yuan Zhong. Experimental Analysis of Algorithms for Coflow Scheduling. *ArXiv e-prints*, 2016.

[63] Nikhil Bansal, Kang-Won Lee, Viswanath Nagarajan, and Murtaza Zafer. Minimum congestion mapping in a cloud. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 267–276. ACM, 2011.

[64] Debojyoti Dutta, Michael Kapralov, Ian Post, and Rajendra Shinde. Embedding paths into trees: Vm placement to minimize congestion. In *ESA*, pages 431–442. Springer, 2012.

[65] Xitao Wen, Kai Chen, Yan Chen, Yongqiang Liu, Yong Xia, and Chengchen Hu. Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter. In *ICDCS*, pages 12–21. IEEE, 2012.

[66] Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms (TALG)*, 2009.

[67] Aravind Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 416–416. IEEE Computer Society, 1997.

[68] Stavros G Kolliopoulos and Clifford Stein. *Approximating disjoint-path problems using greedy algorithms and packing integer programs*. Springer, 1998.

[69] Chandra Chekuri and Sanjeev Khanna. Edge disjoint paths revisited. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 628–637. Society for Industrial and Applied Mathematics, 2003.

[70] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67(3):473–496, 2003.

[71] Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 233–242. IEEE, 2012.

[72] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[73] Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *Integer Programming and Combinatorial Optimization*, pages 67–82, 2002.

[74] Shiro Matuura and Tomomi Matsui. 0.863-approximation algorithm for MAX DICUT. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, APPROX*, pages 138–146, 2001.

[75] Uri Zwick. Analyzing the MAX 2-SAT and MAX DI-CUT approximation algorithms of Feige and Goemans. *Manuscript*, 2000.

[76] Uri Feige and Michel X Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proc. ISTCS*, pages 182–189, 1995.

[77] Fabrizio Grandoni, Tomasz Kociumaka, and Michał Włodarczyk. An LP-rounding-approximation for restricted maximum acyclic subgraph. *Information Processing Letters*, 115(2):182–185, 2015.

[78] Alantha Newman. *Approximating the maximum acyclic subgraph*. PhD thesis, Massachusetts Institute of Technology, 2000.

[79] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, STOC*, pages 767–775, 2002.

[80] Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 573–582, 2008.

[81] Johan Håstad. Some optimal inapproximability results. *JACM*, 48(4):798–859, 2001.

[82] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[83] Doowon Paik, Sudhakar Reddy, and Sartaj Sahni. Deleting vertices to bound path length. *IEEE Transactions on Computers*, 43(9):1091–1096, 1994.

[84] Ola Svensson. Hardness of vertex deletion and project scheduling. In *Proc. APPROX*, pages 301–312, 2012.

[85] Luca Trevisan. Parallel approximation algorithms by positive linear programming. *Algorithmica*, 21(1):72–88, 1998.

[86] Euiwoong Lee. Hardness of graph pricing through generalized Max-Dicut. *Proceedings of the 47th ACM Symposium on Theory of Computing, STOC*, pages 391–399, 2015.

[87] Sanjeev Arora, Béla Bollobás, and László Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS*, pages 313–313, 2002.

[88] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Integrality gaps for Sherali-Adams relaxations. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, pages 283–292, 2009.

[89] Alan Frieze and Mark Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, 1997.

[90] Vladimir Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[91] Hubert Chan. COMP8601 Lecture-9 Notes. Department of Computer Science, HKU, Fall 2013. `http://i.cs.hku.hk/~hubert/teaching/c8601_2013/notes9.pdf`.

[92] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Local global tradeoffs in metric embeddings. *SIAM Journal on Computing*, 39(6):2487–2512, 2010.

[93] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[94] Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011.