On Combinatorial and Stochastic Optimization

Rudy Zhou

April 11, 2023

Abstract

In this dissertation, we study four problems in combinatorial and stochastic optimization. The first two chapters give improved approximation algorithms for classic combinatorial optimization problems. The final two chapters consider combinatorial optimization under stochastic uncertainty. For these problems, we give improved approximations as well as characterize the power of adaptivity.

In the first chapter, we consider generalizations of the k-median problem. In these problems, the goal is to open facilities to serve clients (subject to some constraints) to minimize the total connection cost of each served client to its nearest open facility. We improve the best-known approximations for k-median with outliers and knapsack median to $6.994 + \epsilon$ and $6.387 + \epsilon$, respectively.

In the second chapter, we consider online throughput maximization. In this problem, jobs arrive online with sizes and deadlines, and the goal is to schedule jobs preemptively on m machines to maximize the number of jobs completed by their deadline. We give the first deterministic $O(1)$-competitive algorithm for this problem for any number of machines $m > 1$. This concludes a 20-year line of research since the $m = 1$ case was settled.

In the third chapter, we consider load balancing with stochastic jobs. In this problem, the goal is to assign each job to a machine, which increases the load of the machine by a random size with known distribution, to minimize the expected max load over all machines. First, we give non-adaptive offline and online algorithms that are $O(\frac{\log m}{\log \log m})$-approximate and $O(\log m)$-approximate, respectively, in the most general unrelated machines setting. Both of these algorithms are asymptotically tight for non-adaptive algorithms. In fact, these results hold for much more general stochastic resource allocation problems. Finally, we show how to leverage adaptivity in the special case of related machines load balancing to improve the above approximations.

In the fourth chapter, we consider minimizing the total completion time of stochastic jobs on m identical machines. In this problem, the goal is to schedule the jobs to minimize the expected total completion time of all jobs. We give a $\tilde{O}(\sqrt{m})$-approximation for the special case of Bernoulli jobs. This is the first approximation for this problem that is both independent of the job-size distributions and sublinear in the number of machines m, even for more restrictive special cases.

Contents

Chapter 1

Introduction

Personally, the goal of this dissertation is to document some of the problems in combinatorial optimization that captured my imagination during my PhD. I was drawn to these problems because they are deceptively simple We are given some elementary objects to play with: items with sizes, users with preferences, or tasks to do. We are also given a goal: pack all of the items in the smallest space possible, find the allocation of goods that makes the users the most happy, or schedule all tasks to optimize some quality-of-service metric. The underlying objects are so simple that it seems at first that there is no further mathematical structure other than what is given. However, when one explores further, one realizes that the interaction between the objects and the goal creates a rich underlying structure. For me, this is where the beauty of this area lies.

More technically, in a combinatorial optimization problem, the goal is to make decisions involving discrete objects to optimize an objective function. In this dissertation, we design algorithms for two fundamental classes of problems: clustering and scheduling. In the former, we are given a discrete collection of data points, we wish to summarize the data by partitioning the data points into clusters. We can measure the quality of the clustering via an objective function. In the latter, we are given a collection of jobs and machines, and we wish to schedule jobs on machines, top optimize some objective capturing the quality of a schedule. These problems have been central to developing our theoretical toolkit and understanding of combinatorial optimization.

It is of interest to develop algorithms for such problems that provide good-quality solutions in terms of objective value with small runtime. However, a common theme is that for many problems (for both complexity- and information-theoretic reasons), we cannot hope for the best-of-both worlds: a algorithm that runs quickly (e.g. in polynomial time) and also outputs an *optimal* solution. There are multiple ways to deal with this roadblock, but the one we focus on in this dissertation is by relaxing the latter condition. Thus, we no longer require an optimal solution, but rather a solution whose objective value is *provably comparable to the optimal.* In particular, we are interested in algorithms with bounded *approximation ratio*, which is the maximum over all instances between the objective value of the algorithm and the optimal solution on that instance. This brings us to the subarea of *approximation algorithms* for combinatorial optimization problems. For an introduction to this area, see [WS11].

Approximations algorithms have been a great success. Studying problems through the lens of approximation algorithms gives us a more fine-grained understanding of the structure and difficulty of combinatorial optimization problems through improved approximation algorithms and inapproximability results. Along the way, the community has developed a broad algorithmic toolkit that

has shaped the way we design algorithms today through techniques such as convex relaxations and randomization.

One goal of this dissertation is to develop new algorithmic techniques leading to improved approximation algorithms for fundamental clustering and scheduling problems. A parallel goal is to understand how much information do we really need to solve these problems. For concreteness, what if we do not have complete information about the jobs in a scheduling problem? What if the jobs arrive over time in a real-time system? What if we are uncertain of how long each job will take and only have access to a stochastic prediction? This leads to the neighboring areas of *online algorithms* and *stochastic optimization*. In an online algorithm, the elementary objects of our combinatorial optimization problem are not known up-front but rather arrive sequentially (e.g. over time), and our algorithm must construct a solution incrementally as these objects arrive. In stochastic optimization, some features of the elementary objects are unknown (e.g. the duration of each job in a scheduling problem), and our algorithm must construct a solution given only stochastic information about these features (e.g. the distribution of each job duration.) These simple modifications can completely change the character of a problem. As a result, both of these areas have developed their own unique toolkits and problems with a vibrant interplay among classical approximation, online algorithms, and stochastic optimization. For an introduction to online algorithms, see [Alb03, PST04]. See [KRT00, DGV08] for foundational papers in stochastic optimization.

Next, we will introduce the areas of clustering and scheduling in more detail, define the particular problems that we study, and state our results.

1.1 Clustering

Clustering is a fundamental problem in combinatorial optimization, where we wish to partition a set of data points into *clusters* such that points within the same cluster are more similar than points across different clusters. There are three main ingredients in a clustering problem: a notion of similarity among points, an objective function to measure the quality of the clustering, and constraints to capture other desirable properties of the clustering.

We focus on metric clustering. where the similarity is defined by a distance metric – for every pair of points, we are given a number representing the distance between those two points. The smaller this number is, the more similar the points are. We can define a clustering of these points into, say k clusters, by specifying k candidate points as cluster centers. Then each data point is assigned to its closest cluster center.

Within metric clustering, there are many possible objectives to consider such as the k-center (minimizing the maximum distance from every data point to its cluster center), k-median (minimizing the sum of distances from every data point to its cluster center), and k-means objectives (minimizing the sum of squared distances). Like the names suggest, we have the constraint that we can choose at most k cluster centers to capture the idea that we would like to summarize the data points with few clusters.

In Chapter 2, we study generalizations of the *k-median* problem. In this problem, we are given a set F of facilities, a set C of clients, a metric d on $F \cup C$, and a parameter $k \in \mathbb{N}$. The goal is to choose a set $S \subset F$ of k facilities to open to minimize the sum of *connection costs* of each client to its closest open facility. That is, to minimize the objective $\sum_{j \in C} d(j, S)$, where we define $d(j, S) = \min_{i \in S} d(i, j)$.

The k-median problem is well-studied from the perspective of approximation algorithms, and many

new algorithmic techniques have been discovered while studying it. Examples include linear program rounding [BPR+17, LS16], primal-dual algorithms [JV01], local search [AGK+04], and large data techniques [LG18, MKC+15, GLZ17, GMM+03, IQM+20]. Currently, the best approximation ratio for k-median is 2.671 [CGLS23], and there is a lower bound of $1 + 2/e$ assuming $P \neq NP$ [JMS02].

We are interested in generalizations that impose more complex constraints on the open facilities and served clients [CKMN01, KKN+15]. One such generalization is the *knapsack median* problem. In knapsack median, each facility has a non-negative weight, and we are given budget $B \geq 0$. The goal is to choose a set of open facilities of total weight at most B (instead of having cardinality at most k) to minimize the same objective function. That is, the open facilities must satisfy a knapsack constraint. Another commonly-studied generalization is *k-median with outliers*, also known as *robust k-median*. Here we open k facilities S, as in basic k-median, but we no longer have to serve all the clients; now, we are only required to serve at least m clients $C' \subset C$ of our choice. Formally, the objective function is now $\sum_{j \in C'} d(j, S)$.

Knapsack median and k-median with outliers are much more difficult than the k-median problem. Algorithmic techniques that have been successful in approximating k-median often lead to only a pseudo-approximation for these generalizations—that is, they violate the knapsack constraint or serve fewer than m clients [BPR+18, CKMN01, FKRS19, IQM+20]. Obtaining "true" approximation algorithms requires new ideas beyond those of k-median. The best approximation ratio for both problems is $7.081 + \epsilon$ due to the beautiful iterative rounding framework of Krishnaswamy, Li, and Sandeep [KLS18]. The first and only other true approximation for k-median with outliers is a local search algorithm due to Ke Chen [Che08].

Our Results

Our main result is improved approximation algorithms for knapsack median and k-median with outliers. We obtain both results by considering a more general problem, which we call generalized k-median (GKM). As in k-median, our goal is to open facilities to minimize the connection costs of served clients. In GKM, the open facilities must satisfy r_1 given knapsack constraints, and the served clients must satisfy r_2 given coverage constraints. We define $r = r_1 + r_2$.

For GKM, we show how to round the natural linear program (LP) relaxation to ensure all except $O(r)$ of the variables are integral, and the objective function is increased by at most a 6.387-factor. It is not difficult to show that the iterative rounding framework in [KLS18] can be extended to show a similar result. Indeed, a 7.081-approximation for GKM with at most $O(r)$ fractional facilities is implicit in their work. The improvement in this work is the smaller loss in the objective value.

Our improvement relies on analyzing the extreme points of certain set-cover-like LPs. These extreme points arise at the intermediate steps of our iterative rounding, and by leveraging their structural properties, we obtain our improved pseudo-approximation for GKM. This work reveals some of the structure of such extreme points, and it shows how this structure can lead to improvements.

Using this "pseudo-approximation" for GKM, we can obtain solutions to knapsack median and k-median with outliers with $O(1)$ fractional facilities. Thus, the remaining work is to round a constant number of fractional facilities to obtain an integral solution. To achieve this goal, we apply known sparsification techniques [KLS18] to pre-process the instance, and then develop new post-processing algorithms to round the final $O(1)$ fractional facilities.

We show how to round these remaining variables for knapsack median at arbitrarily small loss, giving a $6.387+\epsilon$-approximation, improving on the best $7.081+\epsilon$-approximation. For k-median with

outliers, a more sophisticated post-processing is needed to round the $O(1)$ fractional facilities. This procedure loses more in the approximation ratio. In the end, we obtain a $6.994 + \epsilon$-approximation, modestly improving on the best known $7.081 + \epsilon$-approximation.

See Chapter 2 for more details. This is joint work with Anupam Gupta and Benjamin Moseley, appearing in the conference International Colloquium on Automata, Languages [GMZ21], and Programming (ICALP) 2021 and in submission to the journal Mathematics of Operations Research.

1.2 Scheduling

In a scheduling problem, we are given a collection of jobs and machines, and our goal is to produce a schedule (what job to assign to each machine and at what time) whose quality is measured by an objective function. Minimally, each job has some processing requirement (or size), but there is a vast landscape of scheduling problems, where the properties of the jobs, machines, and objective differ.

We begin with perhaps the most basic scheduling problem: load balancing. We are given m identical machines and n jobs. Each job j has size x_j such that assigning job j to a machine increases that machine's load by x_j. The goal is to assign each job to some machine to minimize the load of the most-loaded machine (the makespan).

From here, we can obtain new problems by generalizing the machine environment. For example, in related machines, each machine i has a speed s_i such that assigning job j to machine i increases machine i's load by $\frac{x_j}{s_i}$ (i.e. running a job on a machine that is twice as fast takes half as long). More generally, in unrelated machines, assigning job j to machine i increases machine i's load by a machine-dependent size x_{ij}.

Another dimension we can vary is the objective function. Going back to identical machines, we can consider minimizing the total completion time rather than the makespan. Each job j still has size x_j (the same on any machine). However, now we consider the time aspect of the schedule rather than just the load incurred by an assignment. On each machine, we schedule a sequence of jobs (the first job to run on this machine, the second, and so on). Each job in this sequence has a *completion time* c_j, which is exactly the sum of all job sizes up to and including this job in the sequence. The goal is to schedule all jobs to minimize the total completion time $\sum_j c_j$.

Further, we can impose different constraints on the scheduling system. In the above problems, we had to schedule every job and once we scheduled a job, we committed to running it to completion. It could be the case that every job j has a *release date* r_j and *deadline* d_j in addition to its size, so the job is only available to be scheduled in the time interval $[r_j, d_j]$, but our scheduler additionally has the ability to preempt (pause a job to run another) and migrate (resume a job later on a different machine). With these constraints, it may not be possible to complete every job, so we can consider the objective of maximizing the number of completed jobs (the throughput).

From such ingredients, we can construct more and more complicated scheduling problems. However, in this dissertation we focus on the foundational scheduling problems described above. Instead, the main dimension we vary is how much information we have about the jobs. We consider settings where the jobs arrive *online* or we only have *stochastic information* about their processing requirements. The main question we consider is how much we lose when considering restricted information in both of these models.

1.2.1 Online Scheduling

Instead of having access to all jobs immediately, in online scheduling we initially have no knowledge of future jobs (not even their existence). Then, either jobs arrive sequentially one-by-one (the *list model*), or the jobs arrive over time at their release dates (the *over time* model). Each job must be scheduled upon arrival before the next arrives. The key question in online scheduling (and online algorithms more generally) is how much do we lose in terms of objective value by giving up the information of all jobs up-front? The typical way to measure this loss is via the *competitive ratio*, which – for a particular problem and online algorithm – is the maximum ratio over all instances of the objective value of the algorithm versus the objective value of the optimal *offline* algorithm that knows the entire instance up-front. The ideal goal is to design online algorithms with as small competitive ratio as possible or show that any online algorithm must have large competitive ratio.

In this dissertation, we consider the *online* throughput maximization problem. Recall that in throughput maximization, the goal is to preemptively schedule jobs with sizes, release time, and deadlines on m identical machines to maximize the number of jobs that complete by their deadline. In the online version of the problem, the jobs arrive online at their release times at which the scheduler becomes aware of the job and its size and deadline.

More precisely, let J be a collection of jobs such that each $j \in J$ has a *release time* r_j, a *processing time* (or *size*) x_j, and a *deadline* d_j. At each moment of time, the scheduler can specify up to m released jobs to run, and the remaining processing time of the jobs that are run is decreased at a unit rate. Thus we allow the scheduler to preempt jobs (pause a running job to work on another) and migrate job (continue working on a job on a different machine later). A job is completed if its remaining processing time drops to zero by the deadline of that job. The objective is to maximize the number of completed jobs. We measure the performance of our algorithm by the competitive ratio, compared to the optimal offline schedule (Opt) that is aware of all jobs in advance.

A key concept is the *l*axity of a job j, which is $\ell_j = (d_j - r_j) - x_j$, that is, the maximum amount of time we can not run job j and still possibly complete it.

This problem is well understood for the $m = 1$ machine case. No $O(1)$-competitive deterministic algorithm is possible [BKM+92], but there is a randomized algorithm that is $O(1)$-competitive against an oblivious adversary [KP03], and there is a scalable ($O(1+\epsilon)$-speed $O(1/\epsilon)$-competitive) deterministic algorithm [KP00]. The scalability result in [KP00] was extended to the case of $m > 1$ machines in [LMNY13].

Whether an $O(1)$-competitive algorithm exists for $m > 1$ machines has been open for twenty years. Previous results for the multiple machines setting require resource augmentation or assume that all jobs have high laxity [LMNY13, EMS20].

Our Results

Our main result is an $O(1)$-competitive deterministic algorithm for online throughput maximization on $m > 1$ machines. This is the first constant-competitive algorithm that makes no assumptions on the jobs and does not require resource augmentation. Further, on a single machine there is no constant competitive deterministic algorithm, yet a randomized algorithm exists with constant competitive ratio. Our work shows that once more than one machine is considered, then determinism is sufficient to get a $O(1)$-competitive online algorithm. We summarize our results and prior work in Table 1.1.

The main issue issue in removing these assumptions is determining which machine to assign a job

	Deterministic	Randomized	Speed Augmentation
$m = 1$	$\omega(1)$ [BKM+92]	$O(1)$ [KP03]	$O(1+\epsilon)$-speed $O(1/\epsilon)$-competitive [KP00]
$m > 1$	$O(1)$ [Chapter 3]	$O(1)$ [Chapter 3]	$O(1+\epsilon)$-speed $O(1/\epsilon)$-competitive [LMNY13]

Table 1.1: Competitiveness Results

to. If an online algorithm could determine which machine each job was assigned to in Opt, we could obtain an $O(1)$-competitive algorithm for $m > 1$ machines by a relatively straight-forward adaptation of the results from [KP03]. However, if the online algorithm ends up assigning some jobs to different machines than Opt, then comparing the number of completed jobs is challenging. Further, if jobs have small laxity, then the algorithm can be severely penalized for small mistakes in this assignment. One way to view the speed augmentation (or high laxity assumption) analyses in [LMNY13, EMS20] is that the speed augmentation assumption allows one to avoid having to address this issue in the analyses.

We overcome this issue by running three different algorithms in parallel, each on a subset of the machines (using migration if multiple sub-algorithms run the same job). Conceptually, we partition the optimal schedule in to three types of jobs: jobs with high laxity, jobs with low laxity that are pre-empted and completed later, and jobs with low laxity that are run to completion with no pauses. Each of the three sub-algorithms is responsible for one of the types of jobs, so even though our algorithm operates online and cannot determine in general which class a job belongs to upon arrival, we ensure that at least one sub-algorithm will make the right decision for this job.

See Chapter 3 for more details. This is a joint work with Benjamin Moseley, Kirk Pruhs, and Clifford Stein, appearing in the conference Integer Programming and Combinatorial Optimization (IPCO) 2022 [MPSZ22] and in submission to the journal Mathematical Programming.

1.2.2 Stochastic Scheduling

Another way to restrict our algorithm is to consider a stochastic model of jobs sizes. Instead of knowing the size of a job exactly, we assume each job size is a random variable drawn from a known distribution. Initially, the scheduler knows only the job size distributions, but as it begins scheduling jobs, it can observe the realized size of jobs as they are processed. Thus, the scheduler can *adaptively* make scheduling decisions depending on all previously-observed information.

Stochastic scheduling has a long history in operations research and computer science. Earlier works focused on understanding the performance of simple policies and restricted job size distributions [WP80, WVW86, GI99]. More recently, there is interest in designing algorithms for these problems with bounded approximation ratio that work for any job size distributions [KRT00, MSU99, GKNS21]. Along the way, a complementary question is how adaptive do our algorithms need to be? In principle, we could design an algorithm that is non-adaptive: it bases its decisions only on the job size distributions and does not use the realized sizes at all. How much do we lose by restricting ourselves this way?

As I hope you will see reading this dissertation – answering these questions requires new and interesting algorithmic techniques. To showcase this, we study perhaps two of the most fundamental

scheduling problems: load balancing (or makespan minimization) and minimizing total completion time. For both problems, we develop a novel techniques, which give us a stronger understanding of how the optimal adaptive policy behaves and enable us to design improved algorithms that break through barriers where previous approaches were stuck.

Load Balancing

We consider the following stochastic model of load balancing (on unrelated machines). Instead of knowing the size of job j on machine i, x_{ij}, for every machine i and job j exactly, we assume that the size of each job is a random variable $X_{ij} \sim \mathcal{D}_{ij}$ with known distribution $\mathcal{D}_{ij}$. Further, the X_{ij}'s are independent across jobs j. Our algorithm can assign jobs adaptively: once it decides to assign job j to machine i, it learns the realized size X_{ij} and can base subsequent assignments on this information. The goal is to minimize the *expected* makespan.

If jobs are deterministic, a 2-approximation is known and unless $P = NP$, the problem is hard to approximate better than $\frac{3}{2}$ [LST90].

We are interested in algorithms for stochastic load balancing that approximate the optimal *adaptive* policy. Previous works mainly focus on designing non-adaptive algorithms (a fixed job-to-machine assignment using only knowledge of the $\mathcal{D}_{ij}$'s) that approximate the optimal non-adaptive policy. There are non-adaptive $O(1)$-approximations known for identical machines [KRT00] and unrelated machines [GKNS21] in this setting.

In contrast, we are interested in approximating the stronger optimal *adaptive* policy. The *adaptivity gap* (the ratio between the expected makespan of the optimal adaptive and non-adaptive policies) can be $\Omega(\frac{\log m}{\log \log m})$ even for the simplest case of identical machines [GKNS21]. Thus, previous work on approximating the optimal non-adaptive policy does not immediately give any non-trivial approximation guarantees for our setting. The only prior work that gives some guarantees is for special cases where *non-clairvoyant* algorithms (those that do not even need to observe the job size until after it is scheduled) can perform well; in particular, there is a 2-approximation for identical machines [Gra69] (the size of a job does not depend on the machine) and $O(\log m)$-approximation for restricted assignment [ANR95] (identical machines, but each job can only be assigned on some subset of machines).

Further, we consider an additional dimension of uncertainty, namely the knowledge of the job set. In the *offline* setting, the set of jobs and the distributions of their sizes are known up-front, and they can be selected and assigned to the machines irrevocably in any order. In the *online* setting, jobs are not known in advance and they are revealed one-by-one (online-list model). The algorithm learns the job size distributions upon its arrival, and must assign this job without knowledge of future arrivals. Afterwards the next request arrives.

Our Results

As our first main result, we present non-adaptive algorithms for offline and online stochastic load balancing. We design *non-adaptive* algorithms that achieve a $O(\frac{\log m}{\log \log m})$-approximation offline and a $O(\log m)$-approximation online, both compared to the optimal offline adaptive policy.

These results are asymptotically tight as shown by the lower bound of $\Omega(\frac{\log m}{\log \log m})$ on the adaptivity gap [GKNS21] and the lower bound of $\Omega(\log m)$ on the competitive ratio of any deterministic online algorithm, even for deterministic requests [ANR95]. In particular, our work implies that the adaptivity gap for stochastic load balancing is $\Theta(\frac{\log m}{\log \log m})$.

Also, note that in the online setting our algorithm assigns the jobs in fixed order (based on their arrival order), while the offline benchmark is able to assign the jobs in an adaptively-chosen order. In the deterministic setting, this distinction makes no difference (the offline benchmark is a fixed job-to-machine assignment with no notion of order). However, this is non-trivial in the stochastic setting: What is the advantage of adaptively choosing the order rather than using a fixed order to assign the jobs? Our result shows that this advantage is bounded by a $O(\log m)$-factor.

In fact, the above results hold for a much more general resource allocation problem, which we call *configuration balancing* (also sometimes called generalized load balancing). One special case of particular interest is stochastic virtual circuit routing (or congestion minimization). Thus, for these problems we also give asymptotically tight non-adaptive approximations to the optimal adaptive policy.

To improve on the above results, we need to use adaptivity. We show how to leverage adaptivity in the special case of stochastic load balancing on related machines. In this case, we have $X_{ij} = \frac{X_j}{s_i}$, where X_j is the random size of job j and s_i is the speed of machine i. We give an *adaptive* $O(1)$-approximation offline and $O(\log \log m)$-approximation online for stochastic load balancing on related machines.

Because there are good non-clairvoyant algorithms for the simple cases of identical machines and restricted assignment, perhaps the next case to consider is related machines. We give a non-clairvoyant $O(\sqrt{m})$-approximation for related machines load balancing, and further this is asymptotically tight [1]. Thus, using distributional information is necessary to do better than $O(\sqrt{m})$.

All of the above results rely on comparing our non-adaptive policies to a natural assignment LP formulation for stochastic configuration balancing – the most general problem we consider. The main technical challenge is showing that the LP solution is a good proxy for the optimal adaptive policy. We achieve this indirectly by showing that there exists a near-optimal adaptive policy with certain desirable properties that makes it "more" non-adaptive, and that our LP is a good proxy for this near-optimal policy.

See Chapter 4 for more details. This is a joint work with Franziska Eberle, Anupam Gupta, Nicole Megow, and Benjamin Moseley, appearing in the conference Integer Programming and Combinatorial Optimization (IPCO) 2023 [EGM+22].

Completion Time

The second problem we consider in the stochastic model is minimizing the total completion time. We go back to the identical machines setting, so each job j has a size $X_j \sim \mathcal{D}_j$ such that the X_j's are independent across jobs. In this problem, the scheduler proceeds over time rather than constructing a static assignment.

Formally, for each idle machine, an adaptive scheduling policy must choose which job to schedule next on this machine—or it may choose to idle the machine for some time period. In making this decision, it is allowed to use any information it has gained from previously-scheduled jobs. In particular, the policy knows the sizes X_j of all completed jobs j, and if a job j has currently been run for τ time the policy knows that the jobs size is distributed as $(X_j \mid X_j \geq \tau)$. Our goal is to minimize $\sum_j \mathbb{E}[C_j]$, the sum of expected completion times C_j of the jobs.

[1] This is actually my favorite result that I have ever proved; it is very simple but also surprising and asymptotically optimal! Unfortunately it is relegated to an appendix of the original paper, because it is somewhat tangential. Oh well. See § 4.5 for details.

While the deterministic problem can be solved optimally (using the shortest processing time policy) [BJS74], the stochastic setting has long been notorious. Early results for stochastic completion time minimization focused on giving optimal policies only for restricted classes of instances, e.g., the case where all job distributions were exponentials, or where the jobs could be stochastically ordered [WP80, WVW86]. Then, starting with the ground-breaking work of Möhring, Schulz, and Uetz [MSU99], approximation algorithms were given that worked for all stochastic instances. However, almost all such algorithms have approximation ratios with at least linear dependence on the *squared coefficient of variation* $\Delta := \max_j \frac{\mathrm{Var}(X_j)}{(\mathbb{E}X_j)^2}$ [MSU99, SU05, Sch08, SSU16a, GMUX20]. Since this squared coefficient of variation could be very large in general, we want to obtain approximations which are *distribution-independent*, and in particular, do not depend on the coefficient of variation.

The only distribution-independent approximations for stochastic completion time minimization have approximation ratios at least linear in m [IMP15a, EFMM19]. In fact, nothing better than an $O(m)$ approximation is known even for instances consisting of only two types of jobs: identical unit-sized deterministic jobs and identically distributed Bernoulli jobs $X_j \sim s \cdot \mathrm{Ber}(p)$ [EFMM19]. For general job distributions, the best known approximation is $O(m \operatorname{poly} \log n)$ [IMP15a].

There are significant roadblocks to obtaining such distribution-independent guarantees: the known algorithmic toolkit for deterministic jobs relies on greedy policies and linear program-based algorithms [HSSW97, PST04, Pin08]. For the former, the natural *Shortest Expected Processing Time* (SEPT) policy has an approximation ratio no better than $\Omega(n^{1/4})$ [IMP15a]. Moreover, even for instances consisting of only two types of jobs—identical unit-sized deterministic jobs and identical Bernoulli jobs $X_j \sim s \cdot \mathrm{Ber}(p)$, no *index policy* (which assigns each job an "index" depending only on its size distribution, and then schedules the jobs in order of their indices) can have bounded approximation ratio [EFMM19]. Approaches based on linear programming also do not seem to extend to the stochastic setting: the most expressive time-indexed linear program commonly used for such settings has an integrality gap of $\Omega(\Delta)$ [SSU16b]. Finally, we know that *adaptivity gap*—the gap between the optimal adaptive and fixed-assignment policies[2]—is $\Omega(\Delta)$ [SSU16b]. Taken together, these lower bounds rule out most of the tools that work for the setting of deterministic jobs.

Further, there are barriers to obtaining approximations that are sublinear in m: these previous works use "volume" lower bounds, which rely on the fact that the processing capacity of m machines is m times larger than that of a single machine. Indeed, the objective is extremely sensitive to the number of machines m: decreasing the number of machines from m to $\frac{m}{2}$ can change the optimal adaptive policy's objective value by an exponential in m factor. (This is in stark contrast to the deterministic setting, where the optimal solution's objectives for m and $\frac{m}{2}$ machines differ by at most a *constant factor*.) See Appendix C.1 for proofs of these two claims. This gives a sense for why lower bounds on the optimal objective value based on the number of machines m do not generalize well to the stochastic setting, except with a loss of a factor of m.

In summary, the main question we ask is:

> *Can we break through both the Δ- and m-barriers for the basic problem of completion time minimization for stochastic jobs?*

Despite the difficulty in obtaining improved approximations for this problem, it is possible that the problem has a constant-factor approximation!

[2]Such a policy non-adaptively assigns jobs to machines and runs each machines' jobs in a fixed order.

Our Results

In Chapter 5, we consider the case of (non-identical) *Bernoulli jobs*, i.e., with independent processing times $X_j \sim s_j \cdot \mathrm{Ber}(p_j)$ for size $s_j \geq 0$ and probability $p_j \in [0, 1]$. Our main result is the first algorithm that is both distribution-independent and has an approximation ratio sublinear in m. In particular, we give a $\widetilde{O}(\sqrt{m})$-approximation for the special case of Bernoulli jobs. We use the notation $\widetilde{O}(\cdot)$ to hide $\operatorname{poly}\log n$-factors.

Further, our algorithm is a *list schedule*, meaning our algorithm produces a list (i.e., an ordering) of all the jobs, and whenever a machine is free, it schedules the next job according to this ordering. As in the case for online load balancing, our algorithm considers jobs in a fixed order, while the optimal adaptive policy can adaptively order the jobs. Thus, again we can bound the advantage of fixed order versus adaptive order policies by $\widetilde{O}(\sqrt{m})$ for minimizing the completion time of Bernoulli jobs.

Bernoulli jobs already are a significant generalization of the setting of [EFMM19], and our result improves (up to $\operatorname{poly}\log n$-factors) the $O(m)$-approximation of [EFMM19] and the $\widetilde{O}(m)$-approximation of [IMP15a] for this special case of Bernoulli jobs.

See Chapter 5 for more details. This is a joint work with Anupam Gupta and Benjamin Moseley, appearing in the conference Symposium on Discrete Algorithms (SODA) 2023 [GMZ23].

Chapter 2

Generalized k-Median Problems

2.1 Introduction

In this chapter, we develop improved approximations for generalizations of the k-median problem via iterative LP rounding algorithms. Our main technical insight is to leverage novel structural properties of the LP solutions at the intermediate steps of our algorithm to refine previous iterative rounding approaches.

The concrete problem we study is the generalized k-median (GKM) problem. We recall that in this problem, we have collections F of facilities, C of clients, and a metric d on $F \cup C$. The goal is to choose a subset of facilities to open and subset of clients to serve to minimize the total connection cost of each served client to their nearest open facility. The open facilities must satisfy r_1 knapsack constraints, and the served clients must satisfy r_2 coverage constraints, where we define $r = r_1 + r_2$. Specifically, each knapsack constraint specifies a cost to open each facility and a budget. The total cost of the opened facilities must be at most the budget. Similarly, each coverage constraint specifies a profit to serve each client and a quota. The total profit of the served clients must be at least the quota.

Our first result is an improved pseudo-approximation for GKM.

Theorem 2.1.1 (Pseudo-Approximation Algorithm for GKM)**.** *There exists an polynomial time randomized algorithm* PseudoApproximation *that takes as input an instance $\mathcal{I}$ of GKM and outputs a feasible solution to LP_1 with at most $O(r)$ fractional facilities and expected cost at most $6.387 \cdot \mathsf{Opt}(\mathcal{I})$.*

Using the above pseudo-approximation, we can obtain improved "true" approximations for two special cases of particular interest: knapsack median and k-median with outliers.

Theorem 2.1.2 (Approximation Algorithm for Knapsack Median)**.** *There exists a polynomial time randomized algorithm that takes as input an instance $\mathcal{I}$ of knapsack median and parameter $\epsilon \in (0, 1/2)$ and in time $n^{O(1/\epsilon)}$, outputs a feasible solution to $\mathcal{I}$ of expected cost at most $(6.387 + \epsilon) \cdot \mathsf{Opt}(\mathcal{I})$.*

Theorem 2.1.3 (Approximation Algorithm for k-Median with Outliers)**.** *There exists a polynomial time randomized algorithm that takes as input an instance $\mathcal{I}$ of k-median with outliers and parameter $\epsilon \in (0, 1/2)$ and in time $n^{O(1/\epsilon)}$, outputs a feasible solution to $\mathcal{I}$ of expected cost at most $(6.994 + \epsilon) \cdot \mathsf{Opt}(\mathcal{I})$.*

2.1.1 Technical Overview

To illustrate our techniques, we first introduce a natural LP relaxations for GKM. The problem admits an integer program formulation, with variables $\{x_{ij}\}_{i\in F, j\in C}$ and $\{y_i\}_{i\in F}$, where x_{ij} indicates that *client j connects to facility i* and y_i indicates that *facility i is open.* Relaxing the integrality constraints gives the linear program relaxation LP_1.

$$
\begin{array}{lll|lll}
(LP_1): & \min_{x,y} & \sum_{i\in F}\sum_{j\in C} d(i,j)\, x_{ij} & (LP_2): & \min_{y} & \sum_{i\in F}\sum_{j\in C: i\in F_j} d(i,j)\, y_i \\
 & & \sum_{i\in F} x_{ij} \le 1 \qquad \forall j\in C & & & y(F_j)\le 1 \qquad \forall j\in C \\
 & & x_{ij}\le y_i \qquad \forall i\in F, j\in C & & & \\
 & & Wy \le b & & & Wy\le b \\
 & & \sum_{j\in C} a_j(\sum_{i\in F} x_{ij}) \ge c & & & \sum_{j\in C} a_j y(F_j)\ge c \\
 & & x_{ij}, y_i \in [0,1] \qquad \forall i\in F, j\in C & & & y_i\in[0,1] \qquad \forall i\in F
\end{array}
$$

LP_1 is the k-median LP with the extra side constraints. The constraint $Wy \le b$ corresponds to the r_1 knapsack constraints on the facilities y, where $W \in \mathbb{R}_+^{r_1\times F}$ and $b\in\mathbb{R}_+^{r_1}$. These r_1 packing constraints can be thought of as a multidimensional knapsack constraint over the facilities, and ensure that "few" facilities are opened. Next, $\sum_{j\in C} a_j(\sum_i x_{ij}) \ge c$ corresponds to the r_2 coverage constraints on the clients, where $a_j\in\mathbb{R}_+^{r_2}$ for all $j\in C$ and $c\in\mathbb{R}_+^{r_2}$. These coverage constraints ensure that "enough" clients are served. E.g., having one packing constraint $\sum_{i\in F} y_i \le k$ and one covering constraint $\sum_{j\in C}\sum_{i\in F} x_{ij}\ge m$ ensures that at least m clients are covered by at most k facilities; this is the k-median with outliers problem.

Reducing the variables in the LP

We get LP_2 by eliminating the x variables from LP_1, thereby reducing the number of constraints. The idea from [KLS18] is to prescribe a set $F_j\subseteq F$ of permissible facilities for each client j such that x_{ij} is implicitly set to $y_i\mathbf{1}(i\in F_j)$. The details of this reduction and the procedure for creating F_j are given in Proposition 2.2.1. Using this procedure, LP_2 is also a relaxation for GKM. Note that in LP_2, we use the notation $y(F') = \sum_{i\in F'} y_i$ for $F'\subset F$.

Now consider solving LP_2 to obtain an optimal extreme point $\bar{y}$. There must be $|F|$ linearly independent tight constraints at $\bar{y}$, and we call these constraints the *basis* for $\bar{y}$. The tight constraints of interest are the $y(F_j)\le 1$ constraints; in general, there are at most $|C|$ such tight constraints, and we have little structural understanding of the F_j-sets.

Prior Rounding Framework

Consider the family of F_j-sets corresponding to tight constraints, so $\mathcal{F} = \{F_j \mid j\in C, \bar{y}(F_j)=1\}$. If $\mathcal{F}$ is a family of disjoint sets , then the tight constraints of LP_2 form a face of a partition matroid polytope intersected with at most r side constraints (the knapsack and coverage constraints.) Using ideas from, e.g., [KLS18, GRSZ14], we can show that $\bar{y}$ has $O(r)$ fractional variables.

Indeed, the goal of the iterative rounding framework in [KLS18] is to control the set family $\mathcal{F}$ to obtain an optimal extreme point where $\mathcal{F}$ is a disjoint family. To achieve this goal, they iteratively round an auxiliary LP based on LP_2, where they have the constraint $y(F_j)=1$ for all clients j in a special set $C^*\subset C$. Roughly, they regulate what clients are added to C^* and delete constraints $y(F_j)\le 1$ for some clients. The idea is that a client j whose constraint is deleted must be close to

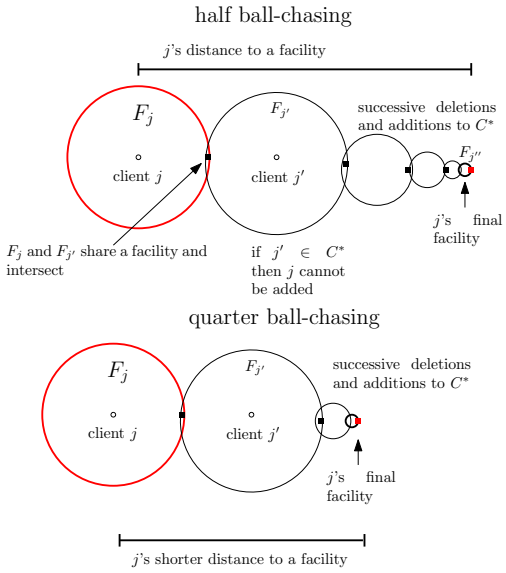

Figure 2.1: Half and quarter ball chasing

some client j' in C^*. Since $y(F_{j'}) = 1$ we can serve j with the facility for j'; the cost is small if j''s facility is close to j.

To get intuition, assume each client j can pay the farthest distance to a facility in F_j, and call this the *radius* of F_j. (Precisely, clients may not be able to afford this distance, but we use this assumption to highlight the ideas behind our algorithmic decisions.) For simplicity, assume every F_j-set is a ball whose radius is a power of two. Over time, this radius shrinks if some y-variables in F_j are set to zero. Consider applying the following iterative steps until none are applicable, in which case C^* corresponds to the tight constraints: (1) delete a constraint for $j \notin C^*$ if the radius of F_j is at least that of some $F_{j'}$ for $j' \in C^*$ and $F_j \cap F_{j'} \neq \emptyset$. (2) add $j \notin C^*$ to C^* if $y(F_j) \leq 1$ is tight and for every $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ it is the case that $F_{j'}$ has a radius strictly larger than F_j. If added then remove all j' from C^* where j's radius is *half* or less of the radius of j' and $F_j \cap F_{j'} \neq \emptyset$.

The performance is bounded by how much a client j with a deleted constraint pays to get to a facility serving a client in C^*. After removing j's constraint, the case to worry about is if j's closest client $j' \in C^*$ is later removed from C^*. This happens only if j'' is added to C^*, with $F_{j''}$ having half the radius of $F_{j'}$. Thus every time we remove j's closest client in C^*, we guarantee that j's cost only increases geometrically. The approximation ratio is proportional to the total distance that j must travel and can be directly related to the distance of "ball-chasing" though these F_j sets. When we remove a client j from C^* due to $j' \in C^*$ such that $F_{j'} \cap F_j \neq \emptyset$ and j' has radius at most half of j, we call this a *half-chasing step*. See Figure 2.1.

New Framework via Structured Extreme Points

The target of our framework is to ensure that the radius decreases in the ball-chasing at a faster rate, in particular *one-quarter*. This gives closer facilities for clients whose constraints are deleted. See Figure 2.1. To achieve this *quarter-chasing step*, we can simply change half to one-quarter in step (2) above.

Making this change immediately decreases the approximation ratio; however, the challenge is that $\mathcal{F}$ is no longer disjoint. Indeed, it can be the case that $j, j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ if their

radii differ by only a one half factor. Instead, our quarter ball-chasing algorithm maintains that $\mathcal{F}$ is not disjoint, but has a *bipartite intersection graph*.

The main technical challenge is obtaining an extreme point with $O(r)$ fractional variables, which is no longer guaranteed as when $\mathcal{F}$ was disjoint. Indeed, if $\mathcal{F}$ has bipartite intersection graph, then the tight constraints form a face of the intersection of two partition matroid polytopes intersected with at most r side constraints. In general, we *cannot upper bound the number of fractional variables* arising in the extreme points of such polytopes. However, such extreme points have a nice combinatorial structure: the intersection graph can be decomposed into $O(r)$ disjoint paths. We exploit this "chain decomposition" of extreme points to discover clients j that can be removed from C^* even if there is not a $j' \in C^*$ where $F_{j'}$ has one quarter of the radius of F_j. We continue this procedure until we are left with only $O(r)$ fractional variables.

The main technical contribution of this work is showing how the problem can be reduced to structural characterization of extreme points corresponding to bipartite matching. This illustrates some of the structural properties of polytopes defined by k-median-type problems. We hope that this helps lead to other structural characterizations of these polytopes and ultimately improved algorithms.

Organization

In § 2.2, we introduce the auxiliary LP for GKM that our iterative rounding algorithm operates on. We note that this is the same LP used in the algorithm of [KLS18]. Then § 2.3–§ 2.5 give the pseudo-approximation for GKM. In particular, § 2.3 describes the basic iterative rounding phase, where we iteratively update the auxiliary LP such that $\mathcal{F}^* = \{F_j \mid j \in C^*\}$ has a bipartite intersection graph. In § 2.4, we characterize the structure of the resulting extreme points and use it to define a new iterative operation, which allows us to reduce the number of fractional variables to $O(r)$. Finally, in § 2.5, we combine the operations from § 2.3 and § 2.4 to obtain our pseduo-approximation algorithm for GKM.

We then obtain true approximations for knapsack median and k-median with outliers: in § 2.6, we describe our framework to turn pseudo-approximation algorithms into true approximations for both problems, and apply it to knapsack median. Then in § 2.7, we consider k-median with outliers.

2.2 Auxiliary LP for Iterative Rounding

In this section, we construct the auxiliary LP, LP_{iter}, that our algorithm will use. We note that we use the same relaxation used in [KLS18]. Recall the two goals of iterative rounding, outlined in § 2.1.1; we want to maintain a set of clients $C^* \subset C$ such that $\{F_j \mid j \in C^*\}$ has bipartite intersection graph, and C^* should provide a good set of open facilities for the clients that are not in C^*. Thus, we want to define LP_{iter} to accommodate moving clients in and out of C^*, while having the LP faithfully capture how much we think the clients outside of C^* should pay in connection costs.

2.2.1 Defining F-balls

Our starting point is LP_1. First, we construct LP_2 by finding appropriate F_j-sets for each $j \in C$. We can find such sets by solving LP_1 and splitting the y-variables such that $x_{ij} \in \{0, y_i\}$ for all clients j and facilities i. The proof of the next proposition is standard; see Appendix A.1 for proof.

Proposition 2.2.1. *There exists a polynomial time algorithm that given GKM instance $\mathcal{I}$, duplicates facilities and outputs sets $F_j \subseteq F$ for $j \in C$ such that $\mathsf{Opt}(LP_2) \leq \mathsf{Opt}(\mathcal{I})$.*

In § 2.1.1, we assumed the radii of the F_j sets were powers of two. To formalize this idea, we discretize the distances to powers of $\tau > 1$ (up to some random offset.) The choice of τ is to optimize the final approximation ratio. The main ideas of the algorithm remain the same if we discretize to powers of, say 2, with no random offset. Our discretization procedure is the following:

Fix some $\tau > 1$ and sample the random offset $\alpha \in [1, \tau)$ such that $\log_e \alpha$ is uniformly distributed in $[0, \log_e \tau)$. Without loss of generality, we may assume that the smallest non-zero inter-point distance is 1. Then we define the possible discretized distances, $L(-2) = -1, L(-1) = 0, \ldots, L(\ell) = \alpha\tau^\ell$ for all $\ell \in \mathbb{N}$.

For each $p, q \in F \cup C$, we round $d(p, q)$ *up* to the next largest discretized distance. Let $d'(p, q)$ denote the rounded distances. Observe that $d(p, q) \leq d'(p, q)$ for all $p, q \in F \cup C$. See Appendix A.1 for proof of the following proposition, which we use to bound the cost of discretization.

Proposition 2.2.2. *For all $p, q \in F \cup C$, we have $\mathbb{E}[d'(p, q)] = \frac{\tau - 1}{\log_e \tau} d(p, q)$*

Now using the discretized distances, we can define the *radius level* of F_j for all $j \in C$ by $\ell_j = \min_{\ell \geq -1} \{\ell \mid d'(j, i) \leq L(\ell) \quad \forall i \in F_j\}$. One should imagine that F_j is a ball of radius $L(\ell_j)$ in terms of the d'-distances. Thus, we will often refer to F_j as the *F-ball of client j*. Further, to accommodate "shrinking" the F_j sets, we define the *inner ball of F_j* by: $B_j = \{i \in F_j \mid d'(j, i) \leq L(\ell_j - 1)\}$. Note that we defined $L(-2) = -1$ so that if $\ell_j = -1$, then $B_j = \emptyset$.

2.2.2 Constructing LP_{iter}

Our auxiliary LP will maintain three sets of clients: C_{part}, C_{full}, and C^*. C_{part} consists of all clients, whom we have not yet decided whether we should serve them or not. Then for all clients in C_{full} and C^*, we decide to serve them fully. The difference between the clients in C_{full} and C^* is that for the former, we remove the constraint $y(F_j) = 1$ from the LP, while for the latter we still require $y(F_j) = 1$. Thus although we commit to serving C_{full}, such clients rely on C^* to find an open facility to connect to. Using the discretized distances, radius levels, inner balls, and these three sets of clients, we are ready to define LP_{iter}:

$$
\begin{aligned}
\min_{y} \quad & \sum_{j \in C_{part}} \sum_{i \in F_j} d'(i, j) y_i + \sum_{j \in C_{full} \cup C^*} \Big(\sum_{i \in B_j} d'(i, j) y_i + (1 - y(B_j)) L(\ell_j) \Big) && (LP_{iter}) \\
\text{s.t.} \quad & y(F_j) \leq 1 \quad \forall j \in C_{part} \\
& y(B_j) \leq 1 \quad \forall j \in C_{full} \\
& y(F_j) = 1 \quad \forall j \in C^* \\
& Wy \leq b \\
& \sum_{j \in C_{part}} a_j y(F_j) \geq c - \sum_{j \in C_{full} \cup C^*} a_j \\
& 0 \leq y \leq 1
\end{aligned}
$$

Note that we use the *rounded* distances in the definition of LP_{iter} rather than the original distances. Keeping this in mind, if $C_{part} = C$ and $C_{full}, C^* = \emptyset$, then LP_{iter} is the same as LP_2 up to the discretized distances, so the following proposition is immediate.

Proposition 2.2.3. *Suppose $C_{part} = C$ and $C_{full}, C^* = \emptyset$. Then $\mathbb{E}[\mathsf{Opt}(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau} \mathsf{Opt}(LP_2)$.*

We now take some time to parse the definition of LP_{iter}. Initially, all clients are in C_{part}. For clients in C_{part}, we are not sure yet whether we should serve them or not. Thus for these clients, we simply require $y(F_j) \leq 1$, so they can be served any amount, and in the objective, the contribution of a client from C_{part} is exactly its connection cost (up to discretization) to F_j.

The clients in C_{full} correspond to the "deleted" constraints in § 2.1.1. Importantly, for $j \in C_{full}$, we do not require that $y(F_j) = 1$; rather, we relax this condition to $y(B_j) \leq 1$. Recall that we made the assumption that every client can pay the radius of its F_j set in § 2.1.1. To realize this idea, we require that each $j \in C_{full}$ pays its connection costs to B_j in the objective. Then, to serve j fully, j must find $(1 - y(B_j))$ units of open facility to connect to beyond B_j. Now j truly pays its radius, $L(\ell_j)$, for this $(1 - y(B_j))$ units of connections in LP_{iter}, so we can do "ball-chasing" to C^* to find these facilities. In this case, we say that we *re-route* the client j to some *destination*.

For clients in C^*, we require $y(F_j) = 1$. Note that the contribution of a $j \in C^*$ to the objective of LP_{iter} is exactly its connection cost to F_j. The purpose of C^* is to provide destinations for C_{full}.

Finally, because we have decided to fully serve all clients in C_{full} and C^*, regardless of how much they are actually served in their F-balls, we imagine that they every $j \in C_{full} \cup C^*$ contributes a_j to the coverage constraints, which is reflected in LP_{iter}.

2.2.3 Properties of LP_{iter}

Throughout our algorithm, we will modify the data of LP_{iter} - we will move clients between C_{part}, C_{full}, and C^* and modify the F-balls and radius levels. However, we still want the data of LP_{iter} to satisfy some consistent properties, which we call our *Basic Invariants*.

Definition 2.2.4 (Basic Invariants)**.** We call the following properties our *Basic Invariants*:

1. $C_{part} \cup C_{full} \cup C^*$ partitions C.
2. For all $j \in C$, we have $d'(j, i) \leq L_{\ell_j}$ for all $i \in F_j$.
3. For all $j \in C$, we have $B_j = \{i \in F_j \mid d'(j, i) \leq L_{\ell_j - 1}\}$.
4. For all $j \in C$, we have $\ell_j \geq -1$.
5. (Distinct Neighbors) For all $j_1, j_2 \in C^*$, if $F_{j_1} \cap F_{j_2} \neq \emptyset$, then $|\ell_{j_1} - \ell_{j_2}| = 1$. In words, if the F-balls of two clients in C^* intersect, then they differ by exactly one radius level.

We want to emphasize Basic Invariant (5), which we call the *Distinct Neighbors Property*. This is our relaxation of the condition that $\{F_j \mid j \in C^*\}$ is disjoint, which is the invariant used in [KLS18]. The Distinct Neighbors Property implies that the sub-collections of F-balls in C^* with even- and odd-radius levels are both disjoint. Using this observation, the next proposition is immediate.

Definition 2.2.5 (Intersection Graph)**.** Let $\mathcal{F} = \{F_j \mid j \in C^*\}$ be a set family indexed by C^*. The intersection graph of $\mathcal{F}$ is the undirected graph with vertex set C^* such that two vertices j and j' are connected by an edge if any only if $F_j \cap F_{j'} \neq \emptyset$.

Proposition 2.2.6. *Suppose LP_{iter} satisfies the Distinct Neighbors Property. Then the intersection graph of $\mathcal{F} = \{F_j \mid j \in C^*\}$ is bipartite. In particular, each facility is in at most two F-balls for clients in C^*.*

We summarize the relevant properties of LP_{iter} in the following lemma. The algorithm described by the lemma is exactly the steps we took in this section. By our construction of the F- and B-balls and the fact $C_{part} = C$, it is immediate that LP_{iter} satisfies all Basic Invariants.

Lemma 2.2.7. *There exists a polynomial time algorithm that takes as input a GKM instance $\mathcal{I}$ and outputs LP_{iter} with $C_{part} = C$, $C_{full} = \emptyset$, and $C^* = \emptyset$ such that $\mathbb{E}[\mathsf{Opt}(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau}\mathsf{Opt}(\mathcal{I})$ and LP_{iter} satisfies all Basic Invariants.*

2.3 Basic Iterative Rounding Phase

We now describe the iterative rounding phase of our algorithm. This phase has two main goals: (a) to simplify the constraint set of LP_{iter}, and (b) to decide which clients to serve and how. To make these two decisions, we repeatedly solve LP_{iter} to obtain an optimal extreme point, and then use the structure of tight constraints to update LP_{iter}, and reroute clients accordingly.

2.3.1 The Algorithm

Our algorithm repeatedly solves LP_{iter} to obtain an optimal extreme point $\bar{y}$, and then performs one of the following three possible updates, based on the tight constraints:

1. If some facility i is set to zero in $\bar{y}$, we delete it from the instance.
2. If constraint $\bar{y}(F_j) \leq 1$ is tight for some $j \in C_{part}$, then we decide to fully serve client j by moving j to either C_{full} or C^*. Initially, we add j to C_{full} then run Algorithm 2 to decide if j should be in C^* instead.
3. If constraint $\bar{y}(B_j) \leq 1$ is tight for some $j \in C_{full}$, we shrink F_j by one radius level (so j's new F-ball is exactly B_j.) Then we possibly move j to C^* by running Algorithm 2 for j.

These steps are made formal in Algorithm 1 (IterativeRound) and Algorithm 2 (ReRoute). IterativeRound relies on the subroutine ReRoute, which gives our criterion for moving a client to C^*. This criterion for adding clients to C^* is the key way in which our algorithm differs from that of [KLS18]. In [KLS18], the criterion used ensures that $\{F_j \mid j \in C^*\}$ is a family of disjoint sets. In contrast, we allow F-balls for clients in C^* to intersect, as long as they satisfy the Distinct Neighbors Property (5).

The modifications made by IterativeRound do not increase $\mathsf{Opt}(LP_{iter})$, so upon termination of our algorithm, we have an optimal extreme point $\bar{y}$ to LP_{iter} such that LP_{iter} is still a relaxation of GKM and no non-negativity constraint, C_{part}-constraint, or C_{full}-constraint is tight for $\bar{y}$.

2.3.2 Sketch of Analysis

Recall the goals from the beginning of the section: procedure IterativeRound achieves goal (a) of making $\{F_j \mid j \in C^*\}$ simpler while maintaining the Distinct Neighbors Property. Since we moved facilities between C^* and C_{full}, achieving goal (b) means deciding which facilities to open, and guaranteeing that each client has a "close-by" open facility. (Recall from § 2.2 that C^* is the set of clients such that their F_j-balls are guaranteed to contain an open facility, and C_{full} are the clients which are guaranteed to be served but using facilities opened in C^*.)

To achieve goal (b), we observe that ReRoute always gives quarter-chasing steps. That is, if we move a client j from C^* to C_{full}, then we are guaranteed a neighboring client $j' \in C^*$ with radius

Algorithm 1: IterativeRound

Input: LP_{iter} satisfying all Basic Invariants
Result: Modifies LP_{iter} and outputs an optimal extreme point of LP_{iter}

repeat
  Solve LP_{iter} to obtain optimal extreme point $\bar{y}$.
  if *there exists a facility* $i \in F$ *such that* $\bar{y}_i \geq 0$ *is tight* **then**
    Delete i from F.
  end
  else if *there exists a client* $j \in C_{part}$ *such that* $y(F_j) \leq 1$ *is tight* **then**
    Move j from C_{part} to C_{full}.
    ReRoute(j)
  end
  else if *there exists a client* $j \in C_{full}$ *such that* $\bar{y}(B_j) \leq 1$ *is tight* **then**
    Update $F_j \leftarrow B_j$ and decrement ℓ_j by 1.
    Update $B_j \leftarrow \{i \in F_j \mid d'(j,i) \leq L(\ell_j - 1)\}$.
    ReRoute(j)
  end
  else
    Output $\bar{y}$ and Terminate.
  end
until *termination*

Algorithm 2: ReRoute

Input: Client $j \in C_{full}$
Result: Decide whether to move j to C^* or not

if $\ell_j \leq \ell_{j'} - 1$ *for all* $j' \in C^*$ *such that* $F_j \cap F_{j'} \neq \emptyset$ **then**
  Move j from C_{full} to C^*.
  For all $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \geq \ell_j + 2$, move j' from C^* to C_{full}.
end

level at least two smaller than j. Thus, each time we re-route j to a further destination (i.e. if j' is subject to another quarter-chasing step), the extra distance j must travel decreases geometrically. In the end, we can show that j will have an open facility within $O(1)$ times its radius.

2.4 Iterative Operation for Structured Extreme Points

In this section, we have two goals: (a) we show that the extreme points of LP_{iter} obtained from IterativeRound are highly structured, and admit a *chain decomposition*. Then, (b) we exploit this chain decomposition to define a *new* iterative operation that is applicable whenever $\bar{y}$ has "many" (i.e. more than $O(r)$) fractional variables. We emphasize that this characterization of the extreme points is what enables the new iterative rounding algorithm.

2.4.1 Chain Decomposition

A chain is a sequence of clients in C^* where the F-ball of each client j contains exactly two facilities—one shared with the previous ball and other with the next.

Definition 2.4.1 (Chain)**.** A *chain* is a sequence of clients $(j_1, \ldots, j_p) \subseteq C^*$ satisfying:

- $|F_{j_q}| = 2$ for all $q \in [p]$, and
- $F_{j_q} \cap F_{j_{q+1}} \neq \emptyset$ for all $q \in [p-1]$.

Our chain decomposition is a partition of the *fractional* C^*-clients given in the next theorem, which is our main structural characterization of the extreme points of LP_{iter}. We say that a client j is fractional if all facilities in F_j are fractional; we denote the fractional clients in C^* by $C^*_{<1}$.

Theorem 2.4.2 (Chain Decomposition)**.** *Suppose LP_{iter} satisfies all Basic Invariants. Let $\bar{y}$ be an extreme point of LP_{iter} such that no C_{part}-, C_{full}-, or non-negativity constraint is tight. Then there exists a partition of $C^*_{<1}$ into at most $3r$ chains, along with a set of at most $2r$ violating clients (clients that are not in any chain.)*

The proof relies on analyzing the extreme points of a set-cover-like polytope with r side constraints; we defer it to § 2.8 and proceed instead to define the new iterative operation.

2.4.2 Iterative Operation for Chain Decompositions

Consider an optimal extreme point $\bar{y}$ of LP_{iter} that admits a chain decomposition as in Theorem 2.4.2. We show that if the number of fractional variables in $\bar{y}$ is sufficiently large, there exists a useful structure, which we call a *candidate configuration*.

Definition 2.4.3 (Candidate Configuration)**.** Let $\bar{y}$ be an optimal extreme point of LP_{iter}. A candidate configuration is a pair of two clients $(j, j') \subset C^*_{<1}$ such that:

1. $F_j \cap F_{j'} \neq \emptyset$
2. $\ell_{j'} \leq \ell_j - 1$
3. Every facility in F_j and $F_{j'}$ is in at exactly two F-balls for clients in C^*
4. $|F_j| = 2$ and $|F_{j'}| = 2$

Lemma 2.4.4. *Suppose LP_{iter} satisfies all Basic Invariants, and let $\bar{y}$ be an optimal extreme point of LP_{iter} such that no C_{part}-, C_{full}-, or non-negativity constraint is tight. If $|F_{<1}| \geq 15r$, then there exist a candidate configuration in $C^*_{<1}$.*

Proof. We claim that in order for $C^*_{<1}$ to have a candidate configuration, it suffices to have a chain of length at least four in $C^*_{<1}$. To see this, let $(j_1, j_2, j_3, j_4, \dots) \subset C^*_{<1}$ be a chain of length at least four. Then $F_{j_2} \cap F_{j_3} \neq \emptyset$, and by the Distinct Neighbors Property, either $\ell_{j_3} = \ell_{j_2} - 1$ or $\ell_{j_2} = \ell_{j_3} - 1$.

We only consider the former case, because both cases are analogous. Thus, if $\ell_{j_3} = \ell_{j_2} - 1$, then we claim that (j_2, j_3) forms a candidate configuration. We already have the first two properties of a candidate configuration. Now we verify the last two. Because j_2 and j_3 are part of a chain, we have $|F_{j_2}| = 2$ and $|F_{j_3}| = 2$. Further, j_2 has neighbors j_1 and j_3 along the chain. By Proposition 2.2.6, each facility in F_{j_2} is in at most two F-balls for clients in C^*. In particular, one of the facilities in F_{j_2} is shared by F_{j_1} and F_{j_2}, and the other must be shared by F_{j_2} and F_{j_3}. Thus, each facility in F_{j_2} is in exactly two F-balls for clients in C^*. An analogous argument holds for F_{j_3}, so (j_2, j_3) satisfies all properties of a candidate configuration, as required.

Now suppose $|F_{<1}| \geq 15r$. Applying Lemma 2.8.4, we have:

$$|F_{<1}| \leq dim(C^*_{<1}) + r \leq |C^*_{<1}| + r,$$

which implies $|C^*_{<1}| \geq 14r$. Finally, by Theorem 2.4.2, $C^*_{<1}$ admits a chain decomposition into at most $3r$ chains and a set of at most $2r$ violating clients. Then at least $12r$ of the clients in $C^*_{<1}$ belong to the $3r$ chains. By averaging, there must exist a chain with size at least $\frac{12r}{3r} = 4$, as required. □

Our new iterative operation is easy to state: Find a candidate configuration (j, j') and move j from C^* to C_{full}.

Algorithm 3: ConfigReRoute

Input: An optimal extreme point $\bar{y}$ to LP_{iter} s.t. there exists an candidate configuration
Result: Modify LP_{iter}

2.1 Let $(j, j') \subset C^*_{<1}$ be any candidate configuration.
2.2 Move j from C^* to C_{full}.

2.4.3 Sketch of Analysis

The first two properties of candidate configurations are used to re-route j to j'. Observe a key difference between ReRoute and ConfigReRoute: In the former, we always guarantee quarter-chasing steps. On the other hand, in ConfigReRoute, we only guarantee a neighboring client of at least one radius level smaller, which corresponds to a half-chasing step. This raises the worry that if all re-routings are due to ConfigReRoute, any potential gains by ReRoute are not realized in the worst case. However we show that, roughly speaking, the last two properties of candidate configurations guarantee that the half-chasing steps of ConfigReRoute happen at most half the time.

In particular, suppose client j is re-routed via ConfigReRoute to j' , which is exactly one radius level smaller. If j' is later re-routed via ReRoute, then we can re-route j to j' and then this new destination. This gives one half- and one quarter-chasing step. The concern is if j' is later

re-routed via ConfigReRoute, which would give j two half-chasing steps in a row. By analyzing the interactions between ReRoute and ConfigReRoute, we show that there must exist a j'' that gives j a quarter-chasing step. See Figure 2.2.

Again, we defer the analysis of the re-routing cost of ConfigReRoute to Theorem 2.5.2, where we analyze the interactions between ConfigReRoute and ReRoute, and present our final pseudo-approximation algorithm next.

2.5 Pseudo-Approximation Algorithm for GKM

The pseudo-approximation algorithm for GKM combine the iterative rounding algorithm IterativeRound from § 2.3 with the re-routing operation ConfigReRoute from § 2.4 to construct a solution to LP_{iter}.

Theorem 2.1.1 (Pseudo-Approximation Algorithm for GKM)**.** *There exists an polynomial time randomized algorithm* PseudoApproximation *that takes as input an instance $\mathcal{I}$ of GKM and outputs a feasible solution to LP_1 with at most $O(r)$ fractional facilities and expected cost at most $6.387 \cdot \mathsf{Opt}(\mathcal{I})$.*

Algorithm 4: PseudoApproximation

Input: LP_{iter} satisfying all Basic Invariants
Result: Modifies LP_{iter} and outputs an optimal extreme point of LP_{iter}

```
3.1 repeat
3.2   | Run IterativeRound to obtain an optimal extreme point ȳ of LP_iter
3.3   | if there exists a candidate configuration then
3.4   |   | Run ConfigReRoute
3.5   | end
3.6   | else
3.7   |   | Output ȳ and Terminate
3.8   | end
3.9 until Termination
```

2.5.1 Analysis of PseudoApproximation

There are two main components to analyzing PseudoApproximation. First, we show that the output extreme point has $O(r)$ fractional variables, which follows from Lemma 2.4.4. Second, we bound the re-routing cost, which follows from the sketches in § 2.3 and § 2.4. In particular, for each client, we can charge each of its half-chasing steps to a quarter-chasing step. This improves on [KLS18], where every re-routing is via half-chasing steps. Optimizing the choice of τ (the discretization factor) gives our final approximation ratio.

Lemma 2.5.1. PseudoApproximation *is a polynomial time algorithm that maintains all Basic Invariants, weakly decreases $\mathsf{Opt}(LP_{iter})$, and outputs an optimal extreme point of LP_{iter} with at most $15r$ fractional variables.*

Proof. First we show that PseudoApproximation maintains all Basic Invariants. It is clear that both IterativeRound and ConfigReRoute maintain that $C_{part} \cup C_{full} \cup C^*$ partitions C. Next, we only update the F-balls in IterativeRound by shrinking the F-ball and B-ball by one radius

level or deleting a facility. Further, we only shrink F_j if $\bar{y}(B_j) = 1$, so it cannot be the case that we shrink a F_j when $\ell_j = -1$, because this would imply $B_j = \emptyset$. We conclude, the first four Basic Invariants, which ensure that the F- and B-balls have the correct semantics, are maintained. For the Distinct Neighbors Property, by definition of ReRoute, if we move a client to C^*, it intersects no C^*-ball of weakly smaller radius level, and we subsequently remove any C^*-balls that are at least two radius levels larger. Thus, any remaining C^*-ball that intersects the newly-added ball has radius level exactly one larger.

Next we show that PseudoApproximation terminates in polynomial time. It is clear that ConfigReRoute runs in polynomial time. For IterativeRound, it suffices to show that the number of iterations of IterativeRound is polynomial. In each iteration, we make one of three actions. We either delete a facility from F, move a client from C_{part} to C_{full} or shrink a F-ball by one radius level for a client in $j \in C_{full}$.

We can delete each facility from F at most once, so we make at most $|F|$ deletions. Each client can move from C_{part} to C_{full} at most once, because we never move clients back from C_{full} to C_{part}, so we do this operations at most $|C|$ times. Finally, observe that $\ell_j \geq -1$ for all $j \in C$ over all iterations. We conclude that we can shrink each F-ball only polynomially many times.

For the runtime of PseudoApproximation, it suffices to show that the number of calls to IterativeRound and ConfigReRoute is polynomial. In every iteration of PseudoApproximation, either we terminate or we are guaranteed to move a client from C^* to C_{full} in ConfigReRoute. Each client can be removed from C^* only polynomially many times, because each time a client is removed, in order to be re-added to C^*, it must be the case that we shrunk the F-ball of that client. However, again because $\ell_j \geq -1$ for all $j \in C$, we can shrink each F-ball only polynomially many times.

To see that $\mathsf{Opt}(LP_{iter})$ weakly decreases, it is easy to check that each iteration of IterativeRound and ConfigReRoute maintains the feasibility of the extreme point of LP_{iter} computed in that iteration and does not change the objective value.

Finally, upon termination of IterativeRound, we have an optimal extreme point where no non-negativity-, $C_{full}-$, or C^*-constraint is tight, so Lemma 2.4.4 implies that when PseudoApproximation terminates, we have an optimal extreme point with at most $15r$ fractional variables (facilities.) □

We now bound the re-routing cost by analyzing how C^* evolves throughout PseudoApproximation. This is one of the main technical contributions of our paper, and it is where our richer C^*-set and relaxed re-routing rules are used. [KLS18] prove an analogous result about the re-routing cost of their algorithm. In the language of the following theorem statement, they show that $\alpha = \frac{\tau+1}{\tau-1}$ for the case $\beta = 1$. We improve on this factor by analyzing the interactions between ReRoute and ConfigReRoute. Interestingly, analyzing each of ReRoute and ConfigReRoute separately would not yield any improvement over [KLS18] in the worst case, even with our richer set C^*. It is only by using the properties of candidate configurations and analyzing sequences of calls to ReRoute and ConfigReRoute that we get an improvement.

Theorem 2.5.2 (Re-Routing Cost)**.** *Upon termination of* PseudoApproximation*, let $S \subset F$ be a set of open facilities and $\beta \geq 1$ such that $d(j, S) \leq \beta L(\ell_j)$ for all $j \in C^*$. Then for all $j \in C_{full} \cup C^*$, $d(j, S) \leq (2 + \alpha)L(\ell_j)$, where $\alpha = \max(\beta, 1 + \frac{1+\beta}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.*

We will need the following discretized version of the triangle inequality.

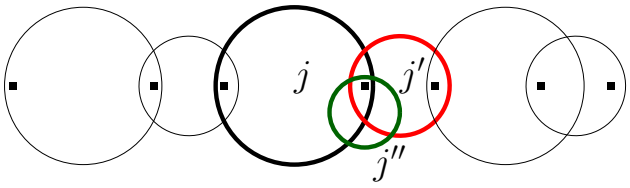

Figure 2.2: A chain of balls in C^*, where squares indicate facilities. First j is removed from C^* as part of candidate configuration (j, j'), so j' has strictly smaller radius than j. Then j'' is added to C^*, which has strictly smaller radius than j'. This gives j a destination that is at least two radius levels smaller.

Proposition 2.5.3. *Let $j, j' \in C$ such that F_j and $F_{j'}$ intersect. Then $d(j, j') \leq L(\ell_j) + L(\ell_{j'})$.*

Proof. Let $i \in F_j \cap F_{j'}$. Then using the triangle inequality we can bound:

$$d(j, j') \leq d(j, i) + d(i, j') \leq d'(j, i) + d'(i, j') \leq L(\ell_j) + L(\ell_{j'}).$$

□

The next lemma analyzes the life-cycle of a client that enters C^* at some point in PseudoApproximation. Our improvement over [KLS18] comes from this lemma.

Lemma 2.5.4. *Upon termination of* PseudoApproximation*, let $S \subset F$ be a set of open facilities and $\beta \geq 1$ such that $d(j, S) \leq \beta L(\ell_j)$ for all $j \in C^*$. Suppose client j is added to C^* at radius level ℓ during* PseudoApproximation *(it may be removed later.) Then upon termination of* PseudoApproximation*, we have $d(j, S) \leq \alpha L(\ell)$, where $\alpha = \max(\beta, 1 + \frac{1+\beta}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.*

Proof. Consider a client j added to C^* with radius level ℓ. If j remains in C^* until termination, the lemma holds for j because $\alpha \geq \beta$. Thus, consider the case where j is later removed from C^* in PseudoApproximation. Note that the only two operations that can possibly cause this removal are ReRoute and ConfigReRoute. We prove the lemma by induction on $\ell = -1, 0, \ldots$. If $\ell = -1$, then j remains in C^* until termination because it has the smallest possible radius level and both ReRoute and ConfigReRoute remove a client from C^* only if there exists another client with strictly smaller radius level.

Similarly, if $\ell = 0$, we note that ReRoute removes a client from C^* only if there exists another client with radius level at least two smaller, which is not possible for j. Thus, if j does not remain in C^* until termination, there must exist some j' that is later added to C^* with radius level at most $\ell - 1 = -1$ such that $F_j \cap F_{j'} \neq \emptyset$. We know that j' remains in C^* until termination since it is of the lowest radius level. Thus:

$$d(j, S) \leq d(j, j') + d(j', S) \leq L(0) + L(-1) + \beta L(-1) = L(0).$$

Now consider $\ell > 0$ where j can possibly be removed from C^* by either ReRoute or ConfigReRoute. In the first case, j is removed by ReRoute, so there exists j' that is added to C^* such that $\ell_{j'} \leq \ell - 2$ and $F_j \cap F_{j'} \neq \emptyset$. Applying the inductive hypothesis to j', we can bound:

$$d(j, S) \leq d(j, j') + d(j', S) \leq L(\ell) + L(\ell - 2) + \alpha L(\ell - 2) \leq (1 + \frac{1 + \alpha}{\tau^2}) L(\ell).$$

It is easy to verify by routine calculations that $1 + \frac{1+\alpha}{\tau^2} \leq \alpha$ given that $\alpha \geq \frac{\tau^3+2\tau^2+1}{\tau^3-1}$.

For our final case, suppose j is removed by ConfigReRoute. Then there exists $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \leq \ell - 1$. Further, $|F_{j'}| = 2$. If j' remains in C^* until termination, then:

$$d(j, S) \leq d(j, j') \leq L(\ell) + L(\ell - 1) + \beta L(\ell - 1) \leq (1 + \frac{1+\beta}{\tau})L(\ell).$$

Otherwise, j' is removed by ReRoute at an even later time because some j'' is added to C^* such that $\ell_{j''} \leq \ell_{j'} - 2$ and $F_{j'} \cap F_{j''} \neq \emptyset$. Applying the inductive hypothesis to j'', we can bound:

$$d(j, S) \leq d(j, j') + d(j', j'') + d(j'', S) \leq (1 + \frac{2}{\tau} + \frac{1+\alpha}{\tau^3})L(\ell).$$

where $\alpha \geq \frac{\tau^3+2\tau^2+1}{\tau^3-1}$ implies $1 + \frac{2}{\tau} + \frac{1+\alpha}{\tau^3} \leq \alpha$.

Now, we consider the case where j' is later removed by ConfigReRoute. To analyze this case, consider when j was removed by ConfigReRoute. At this time, we have $|F_{j'}| = 2$ by definition of Candidate Configuration. Because $F_j \cap F_{j'} \neq \emptyset$, consider any facility $i \in F_j \cap F_{j'}$. When j is removed from C^* by ConfigReRoute, we have that i is in exactly two F-balls for clients in C^*, exactly F_j and $F_{j'}$. However, after removing j from C^*, i is only in one F-ball for clients in C^* - namely $F_{j'}$.

Later, at the time j' is removed by ConfigReRoute, it must be the case that $|F_{j'}| = 2$ still, so $F_{j'}$ is unchanged between the time that j is removed and the time that j' is removed. Thus the facility i that was previously in $F_j \cap F_{j'}$ must still be present in $F_{j'}$. Then this facility must be in exactly two F-balls for clients in C^*, one of which is j'. It must be the case that the other F-ball containing i, say $F_{j''}$, was added to C^* between the removal of j and j'.

Note that the only operation that adds clients to C^* is ReRoute, so we consider the time between the removal of j and j' when j'' is added to C^*. Refer to Figure 2.2. At this time, we have $j' \in C^*$, and $F_{j'} \cap F_{j''} \neq \emptyset$ because of the facility i. Then it must be the case that j'' has strictly smaller radius level than j', so $\ell_{j''} \leq \ell_{j'} - 1 \leq \ell - 2$. To conclude the proof, we note that $F_j \cap F_{j''} \neq \emptyset$ due to the facility i, and apply the inductive hypothesis to j'':

$$d(j, S) \leq d(j, j'') + d(j'', S) \leq (1 + \frac{1+\alpha}{\tau^2})L(\ell,)$$

which is at most $\alpha L(\ell)$. □

Now using the above lemma, we can prove Theorem 2.5.2.

Proof of Theorem 2.5.2. Consider any client j that is in $C_{full} \cup C^*$ upon termination of PseudoApproximation. It must be the case that ReRoute(j) was called at least once during PseudoApproximation. Consider the time of the last such call to ReRoute(j). If j is added to C^* at this time, note that its radius level from now until termination remains unchanged, so applying Lemma 2.5.4 gives that $d(j, S) \leq \alpha L(\ell_j)$, as required. Otherwise, if j is not added to C^* at this time, then there must exist some $j' \in C^*$ such that $F_j \cap F_{j'} \neq \emptyset$ and $\ell_{j'} \leq \ell_j$. Then applying Lemma 2.5.4 to j', we have:

$$d(j, S) \leq d(j, j') + d(j', S) \leq L(\ell_j) + L(\ell_{j'}) + \alpha L(\ell_{j'}) \leq (2 + \alpha)L(\ell_j).$$

□

2.5.2 Putting it all Together: Pseudo-Approximation for GKM

In this section, we prove Theorem 2.1.1. In particular, we use the output of PseudoApproximation to construct a setting of the x-variables with the desired properties.

Proof of Theorem 2.1.1. Given as input an instance $\mathcal{I}$ of GKM, our algorithm is first to run the algorithm guaranteed by Lemma 2.2.7 to construct LP_{iter} from LP_1 such that $\mathbb{E}[\mathsf{Opt}(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau}\mathsf{Opt}(\mathcal{I})$ and LP_{iter} satisfies all Basic Invariants. Note that we will choose $\tau > 1$ later to optimize our final approximation ratio. Then we run PseudoApproximation on LP_{iter}, which satisfies all Basic Invariants, so by Lemma 2.5.1, PseudoApproximation outputs in polynomial time LP_{iter} along with an optimal solution $\bar{y}$ with $O(r)$ fractional variables.

Given $\bar{y}$, we define a setting $\bar{x}$ for the x-variables: for all $j \in C_{part}$, connect j to all facilities in F_j by setting $\bar{x}_{ij} = \bar{y}_i$ for all $i \in F_j$. For all $j \in C^*$, we have $\bar{y}(F_j) = 1$, so connect j to all facilities in F_j. Finally, to connect every $j \in C_{full}$ to one unit of open facilities, we use the following modification of Theorem 2.5.2:

Proposition 2.5.5. *When* PseudoApproximation *terminates, for all $j \in C_{full} \cup C^*$, there exists one unit of open facilities with respect to $\bar{y}$ within distance $(2+\alpha)L(\ell_j)$ of j, where $\alpha = \max(1, 1+\frac{2}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.*

The proof of the above proposition is analogous to that of Theorem 2.5.2 in the case $\beta = 1$, so we omit it. To see this, note that for all $j \in C^*$, we have $\bar{y}(F_j) = 1$. This implies that each $j \in C^*$ has one unit of fractional facility within distance $L(\ell_j)$. Following an analogous inductive argument as in Lemma 2.5.4 gives the desired result.

By routine calculations, it is easy to see that $\alpha = \frac{\tau^3+2\tau^2+1}{\tau^3-1}$ for all $\tau > 1$. Now, for all $j \in C_{full}$, we connect j to all facilities in B_j. We want to connect j to one unit of open facilities, so to find the remaining $1 - \bar{y}(B_j)$ units, we connect j to an arbitrary $1 - \bar{y}(B_j)$ units of open facilities within distance $(2+\alpha)L(\ell_j)$ of j, whose existence is guaranteed by Proposition 2.5.5. This completes the description of $\bar{x}$.

It is easy to verify that $(\bar{x}, \bar{y})$ is feasible for LP_1, because $\bar{y}$ satisfies all knapsack constraints, and every client's contribution to the coverage constraints in LP_1 is exactly its contribution in LP_{iter}. Thus it remains to bound the cost of this solution. We claim that $LP_1(\bar{x}, \bar{y}) \leq (2+\alpha)\mathsf{Opt}(LP_{iter})$, because each client in C_{part} and C^* contributes the same amount to LP_1 and LP_{iter} (up to discretization), and each client $j \in C_{full}$ has connection cost at most $2+\alpha$ times its contribution to LP_{iter}.

In conclusion, the expect cost of the solution $(\bar{x}, \bar{y})$ to LP_1 is at most:

$$(2+\alpha)\,\mathbb{E}[\mathsf{Opt}(LP_{iter})] \leq \frac{\tau-1}{\log_e \tau}\left(2 + \frac{\tau^3+2\tau^2+1}{\tau^3-1}\right)\,\mathsf{Opt}(\mathcal{I}).$$

Choosing $\tau > 1$ to minimize $\frac{\tau-1}{\log_e \tau}(2+\frac{\tau^3+2\tau^2+1}{\tau^3-1})$ gives $\tau = 2.046$ and $\frac{\tau-1}{\log_e \tau}(2+\frac{\tau^3+2\tau^2+1}{\tau^3-1}) = 6.387$. □

2.6 From Pseudo-Approximation to Approximation

In this section, we leverage the pseudo-approximation algorithm for GKM defined in § 2.5 to construct improved approximation algorithms for two special cases of GKM: knapsack median and k-median with outliers.

Recall that knapsack median is an instance of GKM with a single arbitrary knapsack constraint and a single coverage constraint that states we must serve every client in C. Similarly, k-median with outliers is an instance of GKM with a single knapsack constraint, stating that we can open at most k facilities, and a single coverage constraint, stating that we must serve at least m clients. Note that both special cases have $r = 2$.

We restate our main results for these two problems:

Theorem 2.1.2 (Approximation Algorithm for Knapsack Median)**.** *There exists a polynomial time randomized algorithm that takes as input an instance $\mathcal{I}$ of knapsack median and parameter $\epsilon \in (0, 1/2)$ and in time $n^{O(1/\epsilon)}$, outputs a feasible solution to $\mathcal{I}$ of expected cost at most $(6.387 + \epsilon) \cdot \mathit{Opt}(\mathcal{I})$.*

Theorem 2.1.3 (Approximation Algorithm for k-Median with Outliers)**.** *There exists a polynomial time randomized algorithm that takes as input an instance $\mathcal{I}$ of k-median with outliers and parameter $\epsilon \in (0, 1/2)$ and in time $n^{O(1/\epsilon)}$, outputs a feasible solution to $\mathcal{I}$ of expected cost at most $(6.994 + \epsilon) \cdot \mathit{Opt}(\mathcal{I})$.*

2.6.1 Overview

The centerpiece for both of our approximation algorithms is the pseudo-approximation algorithm PseudoApproximation for GKM. For both of these special cases, we can obtain via PseudoApproximation a solution to LP_{iter} with only $O(1)$ fractional facilities and bounded re-routing cost. Now our remaining task is to turn this solution with $O(1)$ fractional facilities into an integral one.

Unfortunately, the basic LP relaxations for knapsack median and k-median with outliers have an unbounded integrality gap. To overcome this bad integrality gap, we use known sparsification tools to pre-process the instance [KLS18]. Our main technical contribution in this section is a post-processing algorithm that rounds the final $O(1)$ fractional variables at a small cost increase for the special cases of knapsack median and k-median with outliers.

2.6.2 Approximation Algorithm for Knapsack Median

To illustrate our approach, we give the pre- and post-processing algorithms for knapsack median, which is the simpler of the two variants. Our pre-processing instance modifies the data of the input instance, so the next definition is useful to specify the input instance and pre-processed instance.

Definition 2.6.1 (Instance of Knapsack Median)**.** An instance of knapsack median is of the form $\mathcal{I} = (F, C, d, w, B)$, where F is a set of facilities, C is a set of clients, d is a metric on $F \cup C$, $w \in \mathbb{R}_+^F$ is the weights of the facilities, and $B \geq 0$ is the budget.

Note that for knapsack median, the two side constraints in LP_{iter} are the knapsack constraint, $\sum_{i \in F} w_i y_i \leq B$, and the coverage constraint, $\sum_{j \in C_{part}} y(F_j) \geq |C| - |C_{full} \cup C^*|$.

We utilize the same pre-processing as in [KLS18]. Roughly, given a knapsack median instance $\mathcal{I}$, we first handle the expensive parts of the optimal solution using enumeration. Once we pre-open the facilities and decide what clients should be assigned there for this expensive part of the instance, we are left with a sub-instance, say $\mathcal{I}'$. In $\mathcal{I}'$ - the "cheap" part of the input instance - the our goal is to open some more facilities to serve the remaining clients. When we construct LP_{iter} for this sub-instance, we initialize additional invariants which we call our *Extra Invariants*.

To state our Extra Invariants, we need to define the *P-ball centered at p with radius r* for any $P \subset F \cup C$, $p \in F \cup C$, and $r \geq 0$, which is the set:

$$B_P(p, r) = \{q \in P \mid d(p, q) \leq r\}.$$

Definition 2.6.2 (Extra Invariants for Knapsack Median)**.** Let $\rho, \delta \in (0, 1/2)$, $U \geq 0$, $S_0 \subset F$, and $R \in \mathbb{R}_+^C$ be given. Then we call the following properties our *Extra Invariants*:

1. For all $i \in S_0$, there exists a dummy client $j(i) \in C^*$ such that $F_{j(i)} = \{i' \in F \mid i' \text{ collocated with } i\}$ with radius level $\ell_{j(i)} = -1$. We let $C_0 \subset C$ be the collection of these dummy clients.
2. For all $i \in F$ that is not collocated with some $i' \in S_0$, we have $\sum_{j \mid i \in F_j} d(i, j) \leq 2\rho U$
3. For all $j \in C$, we have $L(\ell_j) \leq \tau R_j$
4. For all $j \in C$ and $r \leq R_j$, we have: $|B_C(j, \delta r)| r \leq \rho U$.

Extra Invariant (1) guarantees that we open the set of guessed facilities S_0 in our final solution. Then for all non-guessed facilities, so the set $F \setminus S_0$, Extra Invariant (2) captures the idea that these facilities are "cheap." Taken together, Extra Invariants (3) and (4) capture the idea that all remaining clients are "cheap."

The next theorem describes our pre-processing algorithm for knapsack median, which is a convenient re-packaging of the pre-processing used in [KLS18]. The theorem essentially states that given ρ, δ, and U, we can efficiently guess a set $C \setminus C'$ of clients and S_0 of facilities that capture the expensive part of the input instance $\mathcal{I}$. Then when we construct LP_{iter} for the cheap sub-instance, we can obtain the Extra Invariants, and the cost of extending a solution of the sub-instance to the whole instance is bounded with respect to U, which one should imagine is $\mathsf{Opt}(\mathcal{I})$.

Theorem 2.6.3 (Pre-Processing for Knapsack Median)**.** *Let $\mathcal{I} = (F, C, d, w, B)$ be an instance of knapsack median. Then, given as input instance $\mathcal{I}$, parameters $\rho, \delta \in (0, 1/2)$, and an upper bound U on $\mathsf{Opt}(\mathcal{I})$, there exists an algorithm that runs in time $n^{O(1/\rho)}$ and outputs $n^{O(1/\rho)}$-many sub-instances of the form $\mathcal{I}' = (F, C' \subset C, d, w, B)$ along with the data for LP_{iter} on $\mathcal{I}'$, a set of facilities $S_0 \subset F$, and a vector $R \in \mathbb{R}_+^{C'}$ such that:*

1. *LP_{iter} satisfies all Basic and Extra Invariants*
2. $\frac{\log_e \tau}{\tau - 1} \mathbb{E}[\mathsf{Opt}(LP_{iter})] + \frac{1-\delta}{1+\delta} \sum_{j \in C \setminus C'} d(j, S_0) \leq U$

The proof is implicit in [KLS18]. For completeness, we prove the analogous theorem for k-median with outliers, Theorem 2.6.11, in Appendix A.4.

We will show that if LP_{iter} satisfies the Extra Invariants for knapsack median, then we can give a post-processing algorithm with bounded cost. It is not difficult to see that PseudoApproximation maintains the Extra Invariants as well, so we use the Extra Invariants in our post-processing.

Proposition 2.6.4. PseudoApproximation *maintains all Extra Invariants for knapsack median.*

Now we move on to describing our post-processing algorithm. Suppose we run the pre-processing algorithm guaranteed by Theorem 2.6.3 to obtain LP_{iter} satisfying all Basic- and Extra Invariants. Then we can run PseudoApproximation to obtain an optimal extreme point of LP_{iter} with $O(1)$ fractional facilities, and LP_{iter} still satisfies all Basic- and Extra Invariants.

It turns out, to round these $O(1)$ fractional facilities, it suffices to open one facility in each F-ball for clients in C^*. Then we can apply Theorem 2.5.2 to bound the re-routing cost. The main difficulty in this approach is that we must also round some fractional facilities down to zero to maintain the knapsack constraint.

Note that closing a facility can incur an unbounded multiplicative cost in the objective. To see this, consider a fractional facility i that is almost open, so $\bar{y}_i \sim 1$. Then suppose there exists $j \in C_{full}$ such that $i \in B_j$ and $d(j,i) \ll L(\ell_j)$. Then j's contribution to the objective of LP_{iter} is $\sim d(j,i)$. However, if we close i, then j's contribution increases to $L(\ell_j) \gg d(j,i)$.

To bound the cost of closing facilities, we use the Extra Invariants. In particular, we use the next technical lemma, which states that if we want to close down a facility i, and every client j that connects to i has a back-up facility to go to within distance $O(1)L(\ell_j)$, then closing i incurs only a small increase in cost. For proof, see Appendix A.2.

Lemma 2.6.5. *Suppose LP_{iter} satisfies all Basic and Extra Invariants for knapsack median, and let $S \subset F$ and $\alpha \geq 1$. Further, consider a facility $i \notin S \cup S_0$ and set of clients $C' \subset C$ such that for all $j \in C'$, we have $i \in F_j$ and there exists some facility in S within distance $\alpha L(\ell_j)$ of j. Then $\sum_{j \in C'} d(j,S) = O(\frac{\rho}{\delta})U$.*

Further, the coverage constraint for knapsack median, $\sum_{j \in C_{part}} y(F_j) \geq |C| - |C_{full} \cup C^*| = |C_{part}|$ implies that $y(F_j) = 1$ for all $j \in C_{part}$. Thus the next proposition is immediate.

Proposition 2.6.6. *Upon termination of* PseudoApproximation *on a knapsack median instance, we have $C_{part} = \emptyset$.*

To summarize, the goal of our post-processing algorithm is to find an integral setting of the $O(1)$ fractional facilities in the output of PseudoApproximation such that the knapsack constraint is satisfied and there is an open facility in each F-ball for clients in C^*.

Lemma 2.6.7. *Upon termination of* PseudoApproximation *on a knapsack median instance, let $\bar{y}$ be the outputted extreme point of LP_{iter}, and suppose LP_{iter} satisfies all Basic- and Extra Invariants. Then there exists an integral setting of the fractional facilities such that the knapsack constraint is satisfied, there is an open facility in each F-ball for clients in C^*, and every facility in S_0 is open.*

Proof. Consider the following LP:

$$\mathcal{LP} = \min_y \{\sum_{i \in F} w_i y_i \mid y(F_j) = 1 \quad \forall j \in C^*,\, y_i = 1 \quad \forall i \in F_{=1},\, y \in [0,1]^F\}.$$

The first constraint states that we want one open facility in each F-ball for clients in C^*, and the second states that our solution should agree on the integral facilities in $\bar{y}$.

Because LP_{iter} satisfies all Basic Invariants, the intersection graph of $\{F_j \mid j \in C^*\}$ is bipartite by Proposition 2.2.6. Then the feasible region of $\mathcal{LP}$ is a face of the intersection of two partition

matroids (each side of the biparitition of $\{F_j \mid j \in C^*\}$ defines one parititon matroid), and thus $\mathcal{LP}$ is integral.

To conclude the proof, we observe that $\bar{y}$ is feasible for $\mathcal{LP}$, so $\mathsf{Opt}(\mathcal{LP}) \leq \sum_{i \in F} w_i \bar{y}_i \leq B$. Thus there exists an integral setting of facilities that opens one facility in each F-bal for all clients in C^*, agrees with all of $\bar{y}$'s integral facilities, and has total weight at most B. Finally, by Extra Invariant (1), $C_0 \subset C^*$, so we open every facility in S_0. □

Thus, in light of Lemma 2.6.7, our post-processing algorithm is to enumerate over all integral settings of the fractional variables to find one that satisfies the knapsack constraint, opens one facility in each F-ball for clients in C^*, and opens S_0. Combining our post-processing algorithm with PseudoApproximation gives the following theorem.

Theorem 2.6.8. *There exists a polynomial time algorithm that takes as input LP_{iter} for knapsack median instance $\mathcal{I}$ satisfying all Basic- and Extra Invariants and outputs a feasible solution to $\mathcal{I}$ such that the solution opens all facilities in S_0 and has cost at most $(2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\rho/\delta)U$, where $\alpha = \frac{\tau^3+2\tau^2+1}{\tau^3-1}$.*

Proof. Our algorithm is to run PseudoApproximation on LP_{iter} and then run our post-processing algorithm, which is to enumerate over all integral settings of the fractional variables, and then output the feasible solution that opens S_0 of lowest cost (if such a solution exists.)

Let $\bar{y}$ be the optimal extreme point of LP_{iter} output by PseudoApproximation, which has $O(1)$ fractional variables by Lemma 2.5.1. Because $\bar{y}$ has $O(1)$ fractional variables, our post-processing algorithm is clearly efficient, which establishes the runtime of our overall algorithm.

Note that upon termination, LP_{iter} still satisfies all Basic- and Extra Invariants. Then by Lemma 2.6.7, there exists an integral setting of the fractional variables that is feasible, opens S_0, and opens a facility in each F-ball for clients in C^*. It suffices to bound the cost of this solution. Let $S \subset F$ denote the facilities opened by this integral solution, so $d(j,S) \leq L(\ell_j)$ for all $j \in C^*$. Applying Theorem 2.5.2 with $\beta = 1$, we obtain that $d(j,S) \leq (2+\alpha)L(\ell_j)$ for all $j \in C_{full} \cup C^*$, where $\alpha = \max(1, 1+\frac{2}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$. It is easy to check that $\alpha = \frac{\tau^3+2\tau^2+1}{\tau^3-1}$ for all $\tau > 1$.

To bound the cost of the solution S relative to $\mathsf{Opt}(LP_{iter})$, we must bound the cost of closing the $O(1)$-many facilities in $F_{<1} \setminus S$. We recall that by Proposition 2.6.6, we have $C = C_{full} \cup C^*$, so all clients must be fully connected in LP_{iter}.

First we consider any client $j \in C$ that is not supported on any facility in $F_{<1} \setminus S$. Such a client is not affected by closing $F_{<1} \setminus S$, so if F_j is empty, then $d(j,S) \leq (2+\alpha)L(\ell_j)$, which is at most $(2+\alpha)$ times j's contribution to LP_{iter}. Otherwise, F_j contains an integral facility in S to connect to, so $d(j,S)$ is at most j's contribution to LP_{iter}.

It remains to consider the clients whose F-balls contain a facility in $F_{<1} \setminus S$. Because there are only $O(1)$-many facilities in $F_{<1} \setminus S$, it suffices to show that for each $i \in F_{<1} \setminus S$, the additive cost of connecting all clients supported on i is at most $O(\rho/\delta)U$. Here we apply Lemma 2.6.5 to the set of clients $C' = \{j \in C \mid i \in F_j\}$ to obtain $\sum_{j \in C'} d(j,S) = O(\rho/\delta)U$.

To summarize, the cost of connecting the clients not supported on $F_{<1} \setminus S$ is at most $(2+\alpha)\mathsf{Opt}(LP_{iter})$, and the cost of the remaining clients is $O(\rho/\delta)U$, as required. □

Now our complete approximation for knapsack median follows from combining the pre-processing with the above theorem and tuning parameters.

Proof of Theorem 2.1.2. Let $\epsilon' > 0$. We will later choose ϵ' with respect to the given ϵ to obtain the desired approximation ratio and runtime. First, we choose parameters $\rho, \delta \in (0, 1/2)$ and $U \geq 0$ for our pre-processing algorithm guaranteed by Theorem 2.6.3. We take $\rho = \epsilon'^2$ and $\delta = \epsilon'$. We require that U is an upper bound on $\mathsf{Opt}(\mathcal{I})$. Using a standard binary search idea, we can guess $\mathsf{Opt}(\mathcal{I})$ up to a multiplicative $(1+\epsilon')$-factor in time $n^{O(1/\epsilon')}$, so we guess U such that $\mathsf{Opt}(\mathcal{I}) \leq U \leq (1+\epsilon')\mathsf{Opt}(\mathcal{I})$.

With these choices of parameters, we run the algorithm guaranteed by Theorem 2.6.3 to obtain $n^{O(1/\epsilon')}$ many sub-instances such that one such sub-instance is of the form $\mathcal{I}' = (F, C' \subset C, d, w, B)$, where LP_{iter} for $\mathcal{I}'$ satisfies all Basic- and Extra Invariants, and we have:

$$\frac{\log_e \tau}{\tau - 1}\mathbb{E}[\mathsf{Opt}(LP_{iter})] + \frac{1-\epsilon'}{1+\epsilon'}\sum_{j \in C \setminus C'} d(j, S_0) \leq U \tag{2.1}$$

Then for each sub-instance output by the pre-processing, we run the algorithm guaranteed by Theorem 2.6.8 to obtain a solution to each sub-instance. Finally, out of these solutions, we output the one that is feasible for the whole instance with smallest cost. This completes the description of our approximation algorithm for knapsack median. The runtime is $n^{O(1/\epsilon')}$, so it remains to bound the cost of the output solution and to choose the parameters ϵ' and τ.

To bound the cost, it suffices to consider the solution output on the instance $\mathcal{I}'$ where LP_{iter} satisfies all Basic- and Extra Invariants and Equation 2.1. By running the algorithm guaranteed by Theorem 2.6.8 on this LP_{iter}, we obtain a feasible solution $S \subset F$ to $\mathcal{I}'$ such that $S_0 \subset S$, and the cost of connecting C' to S is at most $(2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\epsilon')U$, where $\alpha = \frac{\tau^3+2\tau^2+1}{\tau^3-1}$. To extend this solution on the sub-instance to a solution on the whole instance $\mathcal{I}$, we must connect $C \setminus C'$ to S. Because $S_0 \subset S$, applying Equation 2.1 allows us to upper bound the expected cost of connecting C to S by:

$$(2+\alpha)\mathbb{E}[\mathsf{Opt}(LP_{iter})] + O(\epsilon')U + \sum_{j \in C \setminus C'} d(j, S_0) \leq (2+\alpha)\frac{\tau - 1}{\log_e \tau}\frac{1+\epsilon'}{1-\epsilon'}U + O(\epsilon')U.$$

Now choosing $\tau > 1$ to minimize $(2+\frac{\tau^3+2\tau^2+1}{\tau^3-1})\frac{\tau-1}{\log_e \tau}$ gives $\tau = 2.046$ and $\frac{\tau-1}{\log_e \tau}(2+\frac{\tau^3+2\tau^2+1}{\tau^3-1}) = 6.387$. Thus the expected cost of this solution is at most $6.387\frac{1+\epsilon'}{1-\epsilon'}U + O(\epsilon')U$, where $U \leq (1+\epsilon')\mathsf{Opt}(\mathcal{I})$. Finally, by routine calculations, we can choose $\epsilon' = \theta(\epsilon)$ so that expected cost is at most $(6.387 + \epsilon)\mathsf{Opt}(\mathcal{I})$, as required. Note that the runtime of our algorithm is $n^{O(1/\epsilon')} = n^{O(1/\epsilon)}$. □

2.6.3 Approximation Algorithm for k-Median with Outliers

Our approximation algorithm for k-median with outliers follows the same general steps as our algorithm for knapsack median. We state the analogous Extra Invariants for k-median with outliers and pre-processing algorithm here. The only differences between the Extra Invariants for knapsack median and k-median with outliers is in the final Extra Invariant.

Definition 2.6.9 (Instance of k-Median with Outliers)**.** An instance of k-median with outliers is of the form $\mathcal{I} = (F, C, d, k, m)$, where F is a set of facilities, C is a set of clients, d is a metric on $F \cup C$, k is the number of facilities to open, and m is the number of clients to serve.

Note that for k-median with outliers, the two side constraints in LP_{iter} are the knapsack constraint, $y(F) \leq k$, and the coverage constraint, $\sum_{j \in C_{part}} y(F_j) \geq m - |C_{full} \cup C^*|$.

Definition 2.6.10 (Extra Invariants for k-Median with Outliers)**.** Let $\rho, \delta \in (0, 1/2)$, $U \geq 0$, $S_0 \subset F$, and $R \in \mathbb{R}_+^C$ be given. Then we call the following properties our *Extra Invariants*:

1. For all $i \in S_0$, there exists a dummy client $j(i) \in C^*$ such that $F_{j(i)} = \{i' \in F \mid i' \text{ colocated with } i\}$ with radius level $\ell_{j(i)} = -1$. We let $C_0 \subset C$ be the collection of these dummy clients.
2. For all $i \in F$ that is not collocated with some $i' \in S_0$, we have $\sum_{j | i \in F_j} d(i, j) \leq \rho(1 + \delta)U$
3. For all $j \in C$, we have $L(\ell_j) \leq \tau R_j$
4. For every $t > 0$ and $p \in F \cup C$, we have:

$$|\{j \in B_C(p, \frac{\delta t}{4 + 3\delta}) \mid R_j \geq t\}| \leq \frac{\rho(1 + 3\delta/4)}{1 - \delta/4} \frac{U}{t}.$$

Again, the pre-processing of [KLS18] gives the next theorem. For proof, see Appendix A.4.

Theorem 2.6.11 (Pre-Processing for k-Median with Outliers)**.** *Let $\mathcal{I} = (F, C, d, k, m)$ be an instance of k-median with outliers with optimal solution (S^*, C^*). Then, given as input instance $\mathcal{I}$, parameters $\rho, \delta \in (0, 1/2)$, and an upper bound U on $\mathsf{Opt}(\mathcal{I})$, there exists an algorithm that runs in time $n^{O(1/\rho)}$ and outputs $n^{O(1/\rho)}$-many sub-instances of the form $\mathcal{I}' = (F, C' \subset C, d, k, m' = m - |C^* \setminus C'|)$ along with the data for LP_{iter} on $\mathcal{I}'$, a set of facilities $S_0 \subset F$, and a vector $R \in \mathbb{R}_+^{C'}$ such that:*

1. *LP'_{iter} satisfies all Basic and Extra Invariants*
2. $\frac{\log_e \tau}{(\tau - 1)(1 + \delta/2)} \mathbb{E}[\mathsf{Opt}(LP_{iter})] + \frac{1 - \delta}{1 + \delta} \sum_{j \in C^* \setminus C'} d(j, S_0) \leq U$

It is easy to check that PseudoApproximation maintains all Extra Invariants for k-median with outliers as well, and we have an analogous technical lemma to bound the cost of closing facilities. For proof of the lemma, see Appendix A.2.

Proposition 2.6.12. PseudoApproximation *maintains all Extra Invariants for k-median with outliers.*

Lemma 2.6.13. *Suppose LP_{iter} satisfies all Basic and Extra Invariants for k-median with outliers, and let $S \subset F$ and $\alpha \geq 1$. Further, consider a facility $i \notin S \cup S_0$ and set of clients $C' \subset C$ such that for all $j \in C'$, we have $i \in F_j$ and there exists some facility in S within distance $\alpha L(\ell_j)$ of j. Then $\sum_{j \in C'} d(j, S) = O(\frac{\rho}{\delta})U$.*

Now we focus on the main difference between the two algorithms: the post-processing. In particular, the coverage constraint of k-median with outliers introduces two difficulties in rounding the final $O(1)$ fractional facilities: (a) we are no longer guaranteed that $C_{part} = \emptyset$, and (b) we must satisfy the coverage constraint.

The difficulty with (a) is that now rounding a facility up to one can also incur an unbounded multiplicative cost in the objective. To see this, consider a fractional facility i that is almost closed, so $\bar{y}_i \sim 0$. Consider rounding this facility up to one. Then for a client $j \in C_{part}$ that fractionally

connects to i in the solution $\bar{y}$, if we fully connect j to i, this costs $d(j,i) \gg d(j,i)\bar{y}_i$. The solution here is to use Extra Invariant (2) to bound the additive cost of opening facilities.

The more troublesome issue is (b). Note that the same approach that we used to prove that there exists a good integral setting of the $O(1)$ fractional variables in Lemma 2.6.7 does not work here because putting the coverage constraint in the objective of the LP could result in a solution covering the same client multiple times. Our solution to (b) is a more sophisticated post-processing algorithm that first re-routes clients in C_{part}. After re-routing, we carefully pick facilities to open that do not double-cover any remaining C_{part}-clients. We defer the details of our post-processing algorithm for § 2.7. For now, we present the guarantees of our pseudo-approximation combined with post-processing:

Theorem 2.6.14. *There exists a polynomial time algorithm that takes as input LP_{iter} for k-median with outliers instance $\mathcal{I}$ satisfying all Basic- and Extra Invariants and outputs a feasible set of facilities $S \supset S_0$ such that the cost of connecting m clients to S is at most $(2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\rho/\delta)U$, where $\alpha = \max(3 + 2\tau^{-c}, 1 + \frac{4+2\tau^{-c}}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$ for any constant $c \in \mathbb{N}$.*

Combining the pre-processing of Theorem 2.6.11 with Theorem 2.6.14 and tuning parameters gives our final approximation algorithm for k-median with outliers. The proof of Theorem 2.1.3 is analogous to Theorem 2.1.2, so we defer it to Appendix A.2.

2.7 Post-Processing for k-Median with Outliers

In this section we develop the post-processing algorithm for k-median with outliers that is guaranteed by Theorem 2.6.14. The structure of our algorithm is recursive. First, we give a procedure to round at least one fractional facility or serve at least one client. Then we recurse on the remaining instance until we obtain an integral solution.

2.7.1 Computing Partial Solutions

In this section, we show how to round at least one fractional facility or serve at least one client. We interpret this algorithm as computing a *partial solution* to the given k-median with outliers instance.

The main idea of this algorithm is to re-route clients in C_{part}. In particular, we maintain a subset $\bar{C} \subset C^*$ such that for every client in $\bar{C}$, we guarantee to open an integral facility in their F-ball. We also maintain a subset $C_{covered} \subset C_{part}$ of C_{part}-clients that we re-route; that is, we guarantee to serve them even if no open facility is in their F-balls. Crucially, every client in $C_{part} \setminus C_{covered}$ is supported on at most one F-ball for clients in $\bar{C}$. Thus, we do not have to worry about double-covering those clients when we round the facilities in $F(\bar{C})$.

The partial solution we output consists of one open facility for each client in $\bar{C}$ (along with the facilities that happen to be integral already), and we serve the clients in C_{full}, C^*, $C_{covered}$, and the C_{part}-clients supported on our open facilities. See Algorithm 5 (ComputePartial) for the formal algorithm to compute partial solutions. Note that $c \in \mathbb{N}$ is a parameter of ComputePartial.

We note that to define a solution for k-median with outliers, it suffices to specify the set of open facilities S, because we can choose the clients to serve as the m closest clients to S. Thus when we output a partial solution, we only output the set of open facilities.

We summarize the performance of ComputePartial with the next theorem, which we prove in § 2.7.4. In the next section, we use Theorem 2.7.1 to define our recursive post-processing algorithm.

Algorithm 5: ComputePartial

Input: LP_{iter} and $\bar{y}$ output by PseudoApproximation on a k-median with outliers instance such that $C^*_{<1} \not\subset C_0$

Output: Output a partial solution $S \subset F$ and modify LP_{iter}

4.1 Initialize $C_{covered} = \emptyset$ and $\bar{C} = \{j \in C^*_{<1} \mid F_j \cap S_0 = \emptyset\}$

4.2 **for** *all clients* $\bar{j} \in \bar{C}$ *in increasing order of* $\ell_{\bar{j}}$ **do**

4.3 For all $j \in \bar{C}$ such that $j \neq \bar{j}$ and $F_j \cap F_{\bar{j}} \neq \emptyset$, remove j from $\bar{C}$

4.4 **while** *there exists a client* $j' \in C_{part} \setminus C_{covered}$ *such that* $F_{j'}$ *intersects* $F_{\bar{j}}$ *and* F_j *for some other* $j \in \bar{C}$ **do**

4.5 **if** $\ell_{j'} \leq \ell_{\bar{j}} - c$ **then**

4.6 Remove j from $\bar{C}$

4.7 **end**

4.8 **else**

4.9 Add j' to $C_{covered}$

4.10 **end**

4.11 **end**

4.12 **end**

4.13 For all $i \in F$, we define $w_i = |\{j \in C_{part} \setminus C_{covered} \mid i \in F_j\}|$

4.14 Construct the set $\bar{S} \subset F(\bar{C})$ by greedily picking the facility $i \in F_j$ with largest w_i for each $j \in \bar{C}$

4.15 Define the set $C_{part}(\bar{S}) = \{j \in C_{part} \mid F_j \cap \bar{S} \neq \emptyset\}$

4.16 Define the partial set of facilities by $S = (\bar{S} \cup F_{=1}) \setminus S_0$ and the partial set of clients by $C' = C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0)$

4.17 Update LP_{iter} by deleting S and $F(\bar{C})$ from F, deleting all clients in C' from C, decrementing k by $|S|$, and decrementing m by $|C'|$

4.18 Output the partial solution S

Theorem 2.7.1. *Let LP_{iter} and $\bar{y}$ be the input to* ComputePartial. *Then let S be the partial solution output by* ComputePartial *and LP^1_{iter} be the modified LP. Then LP^1_{iter} satisfies all Basic- and Extra Invariants and we have:*

$$\mathsf{Opt}(LP^1_{iter}) + \frac{1}{2+\alpha}\sum_{j\in C'} d(j, S\cup S_0) \leq \mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U,$$

where $\alpha = \max(3+2\tau^{-c}, 1+\frac{4+2\tau^{-c}}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$.

We want LP^1_{iter} to satisfy all Basic- and Extra Invariants so we can continue recursing on LP^1_{iter}. The inequality given by Theorem 2.7.1 allows us to extend a solution computed on LP^1_{iter} with the partial solution S.

2.7.2 Recursive Post-Processing Algorithm

To complete our post-processing algorithm, we recursively apply ComputePartial until we have an integral solution.

The main idea is that we run PseudoApproximation to obtain an optimal extreme point with $O(1)$ fractional variables. Then using this setting of the y-variables, we construct a partial solution consisting of some open facilities along with the clients that they serve. However, if there are still fractional facilities remaining, we recurse on LP_{iter} (after the modifications by ComputePartial.) Our final solution consists of the union of all recursively computed partial solutions. See Algorithm 6 (OutliersPostProcess.)

Algorithm 6: OutliersPostProcess

Input: LP_{iter} for a k-median with outliers instance satisfying all Basic- and Extra Invariants

Output: Solution $S \subset F$

Run PseudoApproximation to obtain extreme point $\bar{y}$ to LP_{iter}
if *$\bar{y}$ is integral* **then**
  Output the solution $F_{=1}$
end
else if $C^*_{<1} \subset C_0$ **then**
  By Lemma 2.7.2, $\bar{y}$ has at most two fractional variables, say $a, b \in F$.
  Without loss of generality, we may assume $|\{j \in C_{part} \mid a \in F_j\}| \geq |\{j \in C_{part} \mid b \in F_j\}|$
  Output the solution $F_{=1} \cup \{a\}$
end
else
  Run ComputePartial to obtain partial solution S' and update LP_{iter}
  Run ComputePartial on updated LP_{iter} to obtain partial solution S''
  Output solution $S' \cup S''$
end

For proof of the next lemma, see Appendix A.3.

Lemma 2.7.2. *If $C^*_{<1} \subset C_0$, then $\bar{y}$ has at most two fractional variables.*

2.7.3 Analysis of OutliersPostProcess

In this section, we show that OutliersPostProcess satisfies the guarantees of Theorem 2.6.14. All missing proofs can be found in Appendix A.3. We let $\bar{y}$ be the output of PseudoApproximation in the first line of OutliersPostProcess and $\alpha = \max(3 + 2\tau^{-c}, 1 + \frac{4+2\tau^{-c}}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$. First, we handle the base cases of OutliersPostProcess.

The easiest base is when $\bar{y}$ is integral. Here, we do not need the Extra Invariants - all we need is that every client $j \in C_{full} \cup C^*$ has an open facility within distance $(2+\alpha)L(\ell_j)$.

Lemma 2.7.3. *If $\bar{y}$ is integral, then the output solution $F_{=1}$ is feasible, contains S_0, and connecting m clients costs at most $(2+\alpha)\mathsf{Opt}(LP_{iter})$.*

Now, in the other base case, $\bar{y}$ is not integral, but we know that $C^*_{<1} \subset C_0$. By Lemma 2.7.2, we may assume without loss of generality $\bar{y}$ has exactly two fractional facilities, say $a, b \in F_{<1}$. Further, we may assume that the k-constraint is tight, because opening more facilities can only improve the objective value. It follows that $\bar{y}_a + \bar{y}_b = 1$. For the sake of analysis, we define the sets $C_{part}(a) = \{j \in C_{part} \mid a \in F_j\}$ and $C_{part}(b) = \{j \in C_{part} \mid b \in F_j\}$ where we may assume $|C_{part}(a)| \geq |C_{part}(b)|$.

It remains to bound the cost of the solution $F_{=1} \cup \{a\}$. One should imagine that we obtain this solution from $\bar{y}$ by closing down the facility b and opening up a. First we handle the degenerate case where $a \in S_0$. In this case, a and b must both be co-located copies of a facility in S_0, so opening one or the other does not change the cost. Thus, we may assume without loss of generality that $a \notin S_0$. Here, we need to Extra Invariants to bound the cost of opening a and closing b

Lemma 2.7.4. *Suppose $a \notin S_0$. Then the output solution $F_{=1} \cup \{a\}$ is feasible, contains S_0, and connecting m clients costs at most $(2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$.*

This completes the analysis of the base cases. To handle the recursive step, we apply Theorem 2.7.1.

Proof of Theorem 2.6.14. First, we show that OutliersPostProcess terminates in polynomial time. It suffices to show that the number of recursive calls to ComputePartial is polynomial. To see this, note that for each recursive call, it must be the case that $C^*_{<1} \not\subset C_0$. In particular, there exists some non-dummy client in $C^* \setminus C_0$. Thus, we are guaranteed to remove at least once client from C in each recursive call.

Now it remains to show that the output solution is feasible, contains S_0, and connecting m clients costs at most $(2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$. Let $\bar{y}$ be the extreme point computed by PseudoApproximation in the first line of OutliersPostProcess. If $\bar{y}$ is integral or $C^*_{<1} \subset C_0$, then we are done by the above lemmas.

Then it remains to consider the case where $C^*_{<1} \not\subset C_0$. Let LP_{iter} denote the input to OutliersPostProcess and LP^1_{iter} the updated LP_{iter} at the end of ComputePartial as in the statement of Theorem 2.7.1. We note that Theorem 2.7.1 implies that LP^1_{iter} satisfies all Basic and Extra Invariants, so LP^1_{iter} is a valid input to OutliersPostProcess. Then we may assume inductively that the recursive call to OutliersPostProcess on LP^1_{iter} outputs a feasible solution S'' to LP^1_{iter} such that $S_0 \subset S''$ and the cost of connecting m^1 clients from C^1 to S'' is at most $(2+\alpha)\mathsf{Opt}(LP^1_{iter}) + O(\frac{\rho}{\delta})U$.

Further, let S' be the partial solution output by ComputePartial on LP_{iter}. Now we combine the solutions S' and S'' to obtain the solution output by OutliersPostProcess. First, we check that

$S' \cup S''$ is is feasible. This follows, because $|S''| \leq k^1 \leq k - |S'|$ by definition of ComputePartial. Also, $S_0 \subset S''$ by the inductive hypothesis.

It remains to bound the cost of connecting m clients to $S' \cup S''$. Consider serving the m^1 closest clients in C^1 with S'' and C' with $S' \cup S_0$. Because $m_1 = m - |C'|$, this is enough clients. Connecting the m^1 closest clients in C^1 to S'' costs at most $(2+\alpha)\mathsf{Opt}(LP^1_{iter}) + O(\frac{\rho}{\delta})U$ by the inductive hypothesis. Now we use the guarantee of Theorem 2.7.1, which we recall is: $\mathsf{Opt}(LP^1_{iter}) + \frac{1}{2+\alpha}\sum_{j \in C'} d(j, S' \cup S_0) \leq \mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$. Thus, the total connection cost is at most: $(2+\alpha)\mathsf{Opt}(LP^1_{iter}) + O(\frac{\rho}{\delta})U + \sum_{j \in C'} d(j, S' \cup S_0) \leq (2+\alpha)\mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$. Note that the additive $O(\frac{\rho}{\delta})U$ terms which we accrue in each recursive call are still $O(\frac{\rho}{\delta})U$ overall. This is because we keep recursing on a subset of the remaining *fractional* facilities – which is always $O(1)$ – and we open/close each fractional facility at most once over all recursive calls. Thus, we can bound the additive cost of each opening/closing by $O(\frac{\rho}{\delta})U$. □

2.7.4 Proof of Theorem 2.7.1

We let LP_{iter} and $\bar{y}$ denote the input to ComputePartial and LP^1_{iter} the updated LP that is output at the end ComputePartial. We begin with three properties of ComputePartial that will be useful throughout our analysis.

The first is immediate by definition of ComputePartial.

Proposition 2.7.5. *Upon termination of* ComputePartial, *the set family* $\{F_j \mid j \in \bar{C}\}$ *is disjoint, and every client* $j \in C_{part} \setminus C_{covered}$, F_j *intersects at most one* F*-ball for clients in* $\bar{C}$.

Proposition 2.7.6. ComputePartial *initializes and maintains the invariants that* $C_{covered} \subset C_{part}$ *and* $\bar{C} \subset \{j \in C^*_{<1} \mid F_j \cap S_0 = \emptyset\}$

Proof. We initialize $C_{covered} = \emptyset$ and only add clients from C_{part} to $C_{covered}$. Similarly, we initialize $\bar{C} = \{j \in C^*_{<1} \mid F_j \cap S_0 = \emptyset\}$ and only remove clients from $\bar{C}$. □

Lemma 2.7.7. *Every* $\bar{j} \in \bar{C}$ *that is reached by the* For *loop remains in* $\bar{C}$ *until termination.*

Proof. Assume for contradiction that there exists $\bar{j}$ that is reached by the For loop, but $\bar{j}$ does not remain in $\bar{C}$ until termination. Note that $\bar{j}$ cannot be removed from $\bar{C}$ in the iteration that it is considered in the For loop. Thus there must exist a later iteration for client, say j in which $\bar{j}$ is removed from $\bar{C}$. In the iteration for client j, there are only two possible ways that $\bar{j}$ is removed from $\bar{C}$. Either $F_j \cap F_{\bar{j}} \neq \emptyset$ or there exists a client $j' \in C_{part} \setminus C_{covered}$ such that $F_{j'}$ intersects both F_j and $F_{\bar{j}}$ and $\ell_{j'} \leq \ell_j - c$.

In the former case, because we consider $\bar{j}$ before j, it must be the case that we removed j from $\bar{C}$ in $\bar{j}$'s iteration. This is a contradiction. Similarly, in the second case if such a j' exists, then in $\bar{j}$'s iteration, we either remove j from $\bar{C}$ or add j' to $C_{covered}$. In either case, this is a contradiction. □

Now we are ready to prove both properties of Theorem 2.7.1. It is not difficult to see that LP^1_{iter} satisfies all Basic- and Extra Invariants by construction.

Lemma 2.7.8. LP^1_{iter} *satisfies all Basic- and Extra Invariants.*

Proof. By assumption, the input to ComputePartial, LP_{iter}, satisfies all Basic and Extra Invariants. To obtain LP^1_{iter} from LP_{iter}, we delete some clients and facilities. Thus the only change to the F- and B-balls for clients in C^1 is that we possibly remove some facilities from their F- and

B-balls; importantly, the radius levels, ℓ_j for all clients j, remain the same. Thus, it is easy to see that LP^1_{iter} satisfies all Basic Invariants.

Similarly, for all remaining clients j, we have not changed ℓ_j or R_j, so the only Extra Invariant that requires some care to verify is Extra Invariant (1). However, we recall that to obtain C^{*1}, we delete all clients in $C^* \setminus C_0$ from the instance, so $C^{*1} = C_0$. This is because $C_0 \subset C^*$ by the assumption that LP_{iter} satisfies all Extra Invariants. □

Now it remains to show $\mathsf{Opt}(LP^1_{iter}) + \frac{1}{2+\alpha} \sum_{j \in C'} d(j, S \cup S_0) \leq \mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$. To do this, we partition C into C^1 and $C' = C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0)$. For each client in C^1, we show that its contribution to the objective of LP^1_{iter} is at most its contribution to LP_{iter}. Then for each client $j \in C'$, either $d(j, S \cup S_0)$ is at most $2 + \alpha$ times j's contribution to $\mathsf{Opt}(LP_{iter})$ or we can charge j's connection cost to an additive $O(\frac{\rho}{\delta})U$ term.

First, we focus on C^1. For these clients, it suffices to show that $\bar{y}$ (restricted to F^1) is feasible for LP^1_{iter}. This is because for all $j \in C^1$, either $j \in C^1_{part} \subset C_{part}$ or $j \in C_0$. The clients in C_0 contribute zero to the cost of LP^1_{iter} and LP_{iter}. This is because both LP_{iter} and LP^1_{iter} satisfy Extra Invariant (1), so every dummy client $j(i) \in C_0$ is co-located with one unit of open facility corresponding to $i \in S_0$.

Thus it remains to consider the clients $j \in C^1_{part}$. We recall that $C^1_{part} \subset C_{part}$ and $F^1_j \subset F_j$ for all $j \in C^1$, so each $j \in C^1_{part}$ costs less in LP^1_{iter} than in LP_{iter}.

To complete the cost analysis of C^1, we go back to prove feasibility. The main difficulty is showing that the coverage constraint is still satisfied. Recall that we construct $\bar{S}$ by greedily opening the facility in each F-ball for clients in $\bar{C}$ that covers the most $C_{part} \setminus C_{covered}$-clients. Proposition 2.7.5 ensures that this greedy choice is well-defined (because $\{F_j \mid j \in \bar{C}\}$ is disjoint), and that we do not double-cover any $C_{part} \setminus C_{covered}$-clients.

Then by definition of greedy, we show that our partial solution covers more clients than the fractional facilities we delete. This proposition is the key to showing that the coverage constraint is still satisfied.

Proposition 2.7.9. *Upon termination of* ComputePartial, *we have* $|C_{part}(\bar{S})| \geq \sum_{j \in \bar{C}} \sum_{i \in F_j} w_i \bar{y}_i$.

Proof. For each $j \in \bar{C}$, let $i(j) \in \bar{S}$ be the unique open facility in F_j. By definition of $C_{part}(\bar{S})$, for all $j \in C_{part}(\bar{S})$ we have $F_j \cap \bar{S} \neq \emptyset$. Further, by Proposition 2.7.5, F_j intersects exactly one F-ball among clients in $\bar{C}$, so each $i(j)$ for $j \in \bar{C}$ covers a unique set of clients. This implies the equality: $|C_{part}(\bar{S})| = \sum_{j \in \bar{C}} w_{i(j)}$. Combining this equality with the facts that for all $j \in \bar{C}$, we have $\bar{y}(F_j) = 1$ and $w_{i(j)} \geq w_i$ for all $i \in F_j$ gives the desired inequality:

$$|C_{part}(\bar{S})| = \sum_{j \in \bar{C}} w_{i(j)} \geq \sum_{j \in \bar{C}} w_{i(j)} \bar{y}(F_j) \geq \sum_{j \in \bar{C}} \sum_{i \in F_j} w_i \bar{y}_i.$$

□

Finally, we can complete the analysis of C^1. It is easy to check that constraints except for the coverage constraint are satisfied. To handle the coverage constraint, we use Proposition 2.7.9.

Lemma 2.7.10. *$\bar{y}$ restricted to F^1 is feasible for LP^1_{iter}.*

Proof. We note that $C^1 = C \setminus (C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0))$ and $F^1 = F \setminus (F_{=1} \cup F(\bar{C}) \setminus S_0)$. Then we have $C^1_{part} = C_{part} \setminus (C_{part}(\bar{S}) \cup C_{covered})$, $C^1_{full} = \emptyset$, and $C^{*1} = C_0$.

It suffices to show that all C^1_{part}-constraints, C^{*1}-constraint, the k-constraint, and the coverage constraint are satisfied by $\bar{y}$ restricted to F^1.

Consider any $j \in C^1_{part}$. We observe that the $F^1_j \subset F_j$ for all $j \in C^1_{part}$ and $C^1_{part} \subset C_{part}$.Then $\bar{y}(F^1_j) \leq \bar{y}(F_j) \leq 1$ for all $j \in C^1_{part}$.

Now for any $j \in C^{*1} = C_0$, it suffices to show that we do not delete any copies of $i \in S_0$ when going from from F to F^1, but this is immediate because $S_0 \subset F$, and we do not delete any facility from S_0 to obtain F^1. Thus, every C^{*1}-constraint is satisfied.

For the k-constraint, we have $\bar{y}(F) \leq k$. We want to show $\bar{y}(F^1) \leq k - |S|$. By definition $|\bar{S}| = \sum_{j \in \bar{C}} \bar{y}(F_j) = \bar{y}(F(\bar{C}))$, where the final equality follows because $\{F_j \mid j \in \bar{C}\}$ is a disjoint collection by Proposition 2.7.5. Then we compute: $\bar{y}(F^1) = \bar{y}(F) - \bar{y}(F(\bar{C})) - |F_{=1} \setminus S_0| \leq k - |\bar{S}| - |F_{=1} \setminus S_0| = k - |S|$, as required.

Finally, for the coverage constraint, we want to show:

$$\sum_{j \in C^1_{part}} \bar{y}(F^1_j) \geq m^1 - |C^1_{full} \cup C^{*1}|,$$

where $C^1_{part} = C_{part} \setminus (C_{part}(\bar{S}) \cup C_{covered})$, $m^1 = m - |C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0)|$, and $C^1_{full} = \emptyset$ and $C^{*1} = C_0$, so $|C^1_{full} \cup C^{*1}| = |C_0|$. Thus we can re-write the coverage constraint as:

$$\sum_{j \in C^1_{part}} \bar{y}(F^1_j) \geq m - |C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup C^*|.$$

Recall that the coverage constraint of LP_{iter} implies: $\sum_{j \in C_{part}} \bar{y}(F_j) \geq m - |C_{full} \cup C^*|$. By splitting this inequality into the contribution by C^1_{part} and $C_{part} \setminus C^1_{part} = C_{part}(\bar{S}) \cup C_{covered}$, we obtain:

$$\begin{aligned}
\sum_{j \in C_{part}} \bar{y}(F_j) &\geq m - |C_{full} \cup C^*| \\
\sum_{j \in C^1_{part}} \bar{y}(F_j) + \sum_{j \in C_{part}(\bar{S}) \cup C_{covered}} \bar{y}(F_j) &\geq m - |C_{full} \cup C^*| \\
\sum_{j \in C^1_{part}} \bar{y}(F_j) + \sum_{j \in C_{part}(\bar{S})} \bar{y}(F_j) &\geq m - |C_{covered} C_{full} \cup C^*|
\end{aligned}$$

, where in the final inequality we use the fact that $\bar{y}(F_j) \leq 1$ for all $j \in C_{covered} \subset C_{part}$. Now, we recall that $F^1_j = F_j \setminus F(\bar{C})$ for all $j \in C^1_{part}$, because $C^1_{part} \subset C_{part}$. We can re-write: $\sum_{j \in C^1_{part}} \bar{y}(F_j) = \sum_{j \in C^1_{part}} (\bar{y}(F^1_j) + \bar{y}(F_j \cap F(\bar{C}))$. To show that the coverage constraint is satisfied, it suffices to show: $\sum_{j \in C_{part^1}} \bar{y}(F_j \cap F(\bar{C})) + \sum_{j \in C_{part}(\bar{S})} \bar{y}(F_j) \leq |C_{part}(\bar{S})|$. To see this, observe that the first sum is over all clients in $C_{part} \setminus C_{covered}$ supported on some facility in $F(\bar{C}) \setminus \bar{S}$ but none in $\bar{S}$ (otherwise these clients would be in $C_{part}(\bar{S})$.) The second sum is over all clients in $C_{part} \setminus C_{covered}$ supported

on some facility in $\bar{S}$. Thus, recalling that $w_i = |\{j \in C_{part} \setminus C_{covered} \mid i \in F_j\}|$, we have: $\sum_{j \in C_{part^1}} \bar{y}(F_j \cap F(\bar{C})) + \sum_{j \in C_{part}(\bar{S})} \bar{y}(F_j) = \sum_{j \in \bar{C}} \sum_{i \in F_j} w_i \bar{y}_i \leq |C_{part}(\bar{S})|$, where in the final inequality we apply Proposition 2.7.9. □

For the final property, we must upper bound the connection cost of $C' = C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0)$ to $S \cup S_0$. We bound the connection cost in a few steps. First, we bound the re-routing cost of $C_{full} \cup C^*$. Second, we show that every $j \in C_{covered}$ has a "back-up" facility within $O(1)L(\ell_j)$. This is used to bound the additive cost of connecting $C_{covered}$. Finally, for the clients in $C_{part}(\bar{S})$, we guarantee to open a facility in their F-balls, so we also bound their additive cost.

The next lemma and corollary allow us to bound the cost of $C_{full} \cup C^*$.

Lemma 2.7.11. *Upon termination of* ComputePartial, *for all $j \in C^*$, we have $d(j, S \cup S_0) \leq (3 + \frac{2}{\tau^c})L(\ell_j)$.*

Proof. There are a few cases to consider. First, if $j \in C^*_{=1}$, then the lemma is trivial because by definition there exists an integral facility in F_j. Otherwise, if $j \in C^*_{<1}$, but $F_j \cap S_0 \neq \emptyset$, then again the lemma is trivial.

Thus it remains to consider clients $j \in \{j \in C^*_{<1} \mid F_j \cap S_0 = \emptyset\}$. We note that such a client j is initially in $\bar{C}$. If j remains in $\bar{C}$ until termination, then we are done, because we are guaranteed to open a facility in F_j for all $j \in \bar{C}$ (this is exactly the set $\bar{S}$ of facilities.)

For the final case, we suppose client j is removed from $\bar{C}$ in the iteration where we consider $\bar{j} \in \bar{C}$. Then either $F_j \cap F_{\bar{j}} \neq \emptyset$ or there exists $j' \in C_{part} \setminus C_{covered}$ such that $F_{j'}$ intersects both F_j and $F_{\bar{j}}$ and $\ell_{j'} \leq \ell_{\bar{j}} - c$. Note that because the For loop considers clients in increasing order of radius level, we have $\ell_j \geq \ell_{\bar{j}}$

In the former case, by the Distinct Neighbors Property, we have $\ell_j \geq \ell_{\bar{j}} + 1$. Further, by Lemma 2.7.7, we know that $\bar{j}$ remains in $\bar{C}$ until termination, so $d(\bar{j}, S) \leq L(\ell_{\bar{j}})$. Then we can upper bound: $d(j, S) \leq d(j, \bar{j}) + d(\bar{j}, S) \leq L(\ell_j) + L(\ell_{\bar{j}}) + L(\ell_{\bar{j}}) \leq (1 + \frac{2}{\tau})L(\ell_j) < 3L(\ell_j)$, where in the final inequality, we use the fact that $\tau > 1$.

In the latter case, we again have $d(\bar{j}, S) \leq L(\ell_{\bar{j}})$. Then we can bound the distance from j to S by first going from j to j', then from j' to $\bar{j}$, and finally from $\bar{j}$ to S, where $\ell_{j'} \leq \ell_{\bar{j}} - c$ and $\ell_{\bar{j}} \leq \ell_j$:

$$d(j, S) \leq d(j, j') + d(j', \bar{j}) + d(\bar{j}, S) \leq L(\ell_j) + L(\ell_{j'}) + L(\ell_{j'}) + L(\ell_{\bar{j}}) + L(\ell_{\bar{j}}) \leq (3 + \frac{2}{\tau^c})L(\ell_j).$$

□

Corollary 2.7.12. *Upon termination of* ComputePartial, *for all $j \in C_{full} \cup C^*$, we have $d(j, S \cup S_0) \leq (2 + \alpha)L(\ell_j)$, where $\alpha = \max(3 + 2\tau^{-c}, 1 + \frac{4+2\tau^{-c}}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$*

Proof. Apply Theorem 2.5.2 with $\beta = 3 + 2\tau^{-c}$. □

Similarly, the next lemma ensures that $C_{covered}$ can also be re-routed.

Lemma 2.7.13. *Upon termination of* ComputePartial, *for all $j \in C_{covered}$, we have $d(j, \bar{S}) \leq (1 + 2\tau^c)L(\ell_j)$.*

Proof. Because $j \in C_{covered}$, it must be the case that we put j in $C_{covered}$ in some iteration of the FOR loop where we consider client $\bar{j} \in \bar{C}$. Thus we have $\ell_j \geq \ell_{\bar{j}} - c + 1$ and $F_j \cap F_{\bar{j}} \neq \emptyset$. Also, by Lemma 2.7.7, $\bar{j}$ remains in $\bar{C}$ until termination, so $d(\bar{j}, \bar{S}) \leq L(\ell_{\bar{j}})$. Using these facts, we can bound: $d(j, S) \leq d(j, \bar{j}) + d(\bar{j}, S) \leq L(\ell_j) + L(\ell_{\bar{j}}) + L(\ell_{\bar{j}}) \leq (1 + 2\tau^{c-1})L(\ell_j)$. □

Using the above lemmas, we are ready to bound the connection cost of our partial solution. Note that we bound the cost of serving C' not with S, which is the partial solution we output, but rather with $S \cup S_0$. Thus, we are implicitly assuming that S_0 will be opened by the later recursive calls.

We recall that $C' = C_{part}(\bar{S}) \cup C_{covered} \cup C_{full} \cup (C^* \setminus C_0)$ and $S \cup S_0 = F_{=1} \cup \bar{S} \cup S_0$.

To begin, we bound the cost of connecting $C_{part}(\bar{S})$ to $\bar{S}$. By definition, every client $j \in C_{part}(\bar{S})$ has some facility from $\bar{S}$ in its F-ball. Further, $\bar{S} \subset F(\bar{C})$ by definition, and $F(\bar{C}) \cap S_0 = \emptyset$ using Proposition 2.7.5. Thus we can apply Extra Invariant (2) to each facility in $\bar{S}$. Further, we know that $|\bar{S}| = O(1)$, because there are only $O(1)$ fractional facilities, and every facility in $\bar{S}$ is fractional by definition. Then we can bound: $\sum_{j \in C_{part}(\bar{S})} d(j, \bar{S}) \leq \sum_{i \in \bar{S}} \sum_{j \in C_{part}(\bar{S}) | i \in F_j} d(j, i) \leq O(\rho)U$, where we apply Extra Invariant (2) for each $i \in \bar{S}$. Thus, we have shown that the connection cost of $C_{part}(\bar{S})$ is at most an additive $O(\rho)U$.

Now we move on to the rest of C', that is - the clients in $C_{covered}$, C_{full} and $C^* \setminus C_0$. For the clients in $C_{covered}$, we know by Lemma 2.7.13 that every client $j \in C_{covered}$ has an open facility in $\bar{S}$ at distance at most $(1 + 2\tau^c)L(\ell_j)$. Further, by definition of $C_{covered}$, each $j \in C_{covered}$ is supported on a fractional facility not in S_0. To see this, note that for all $j \in C_{covered}$, there exists $\bar{j} \in \bar{C}$ such that $F_j \cap F_{\bar{j}} \neq \emptyset$, and $F_{\bar{j}} \cap S_0 = \emptyset$.

Then we can use Lemma 2.6.13 for each fractional facility $i \notin S_0$ to bound the cost of connecting all $C_{covered}$-clients supported on i to $\bar{S}$. For each such fractional facility $i \notin S_0$, the connection cost of these clients is at most $O(\frac{\rho}{\delta})U$. By summing over all $O(1)$-many fractional $i \notin S_0$, we connect all of $C_{covered}$ at additive cost at most $O(\frac{\rho}{\delta})U$.

Finally, we handle the clients in $C_{full} \cup (C^* \setminus C_0)$. For convenience, we denote this set of clients by $\hat{C}$. Using Lemma 2.7.11, every $j \in \hat{C}$ has a facility in $S \cup S_0$ at distance at most $(2+\alpha)L(\ell_j)$. There are a few cases to consider. For the first case, we consider the clients $\{j \in \hat{C} \mid F_j \setminus (S_0 \cup F_{=1}) \neq \emptyset\}$, that is the set of $\hat{C}$-clients whose F-balls contain some cheap, fractional facility. By an analogous argument as for $C_{covered}$, we can apply Lemma 2.6.13 to each fractional $i \notin S_0$ to bound the cost of connecting $\{j \in \hat{C} \mid F_j \setminus (S_0 \cup F_{=1}) \neq \emptyset\}$ to $S \cup S_0$ by $O(\frac{\rho}{\delta})U$.

Then it remains to consider the clients $j \in \hat{C}$ such that $F_j \subset F_{=1} \cup S_0$. For such a client j, if $F_j = \emptyset$, then j's contribution to the objective of LP_{iter} is exactly $L(\ell_j)$, so connecting j to $S \cup S_0$ costs at most $(2 + \alpha)$ times j's contribution to the objective of LP_{iter}.

Similarly, if F_j happens to contain a facility in $F_{=1} \cup S_0$, then we can simply connect j to the closest such facility in F_j. Note that $F_{=1} \cup S_0 \subset S \cup S_0$, so j's connection cost in this case it as most its contribution to the objective in LP_{iter}. In conclusion, the connection cost of $\{j \in \bar{C} \mid F_j \subset F_{=1} \cup S_0\}$ is at most $(2 + \alpha)\mathsf{Opt}(LP_{iter})$. Summing the costs of these different groups of clients completes the proof.

2.8 Chain Decompositions of Extreme Points

In this section, we prove a more general version of Theorem 2.4.2 that applies to set-cover-like polytopes with r side constraints. In particular, we consider polytopes of the form:

$$\mathcal{P} = \{y \in \mathbb{R}^F \mid y(F_j) = 1 \quad \forall j \in C^*,\, 0 \leq y \leq 1,\, Ay \leq b\}$$

, where F, C^*, and the F-balls are defined as in LP_{iter}, and $Ay \leq b$ is an arbitrary system of r linear inequalities. We note that other than the C_{part}-, and C_{full}-constraints, $\mathcal{P}$ generalizes the feasible region of LP_{iter} by taking the system $Ay \leq b$ to be the r_1 knapsack constraints and r_2 coverage constraints.

Although we phrase $\mathcal{P}$ in terms of facilities and clients, one can interpret $\mathcal{P}$ as a set cover polytope with side constraints as saying that we must choose elements in F to cover each set in the family $\mathcal{F} = \{F_j \mid j \in C^*\}$ subject to the constraints $Ay \leq b$. The main result of this section is that if $\mathcal{F}$ has bipartite intersection graph, there exists a chain decomposition of the extreme points of $\mathcal{P}$.

Note that $\mathcal{P}$ can also also be interpreted as the intersection of two partition matroid polytopes or as a bipartite matching polytope both with r side constraints. Our chain decomposition theorem shares some parallels with the work of Grandoni, Ravi, Singh, and Zenklusen, who studied the structure of bipartite matching polytopes with r side constraints [GRSZ14].

Theorem 2.8.1 (General Chain Decomposition)**.** *Suppose we have a polytope:* $\mathcal{P} = \{y \in \mathbb{R}^F \mid y(F_j) = 1 \quad \forall j \in C^*,\, 0 \leq y \leq 1,\, Ay \leq b\}$*, such that F is a finite ground set of elements (facilities),* $\{F_j \subset F \mid j \in C^*\}$ *is a set family indexed by C^* (clients), and $Ay \leq b$ is a system of r linear inequalities. Further, let $\bar{y}$ be an extreme point for $\mathcal{P}$ such that no non-negativity constraint is tight. If $\mathcal{F}$ has bipartite intersection graph, then $C^*_{<1}$ admits a partition into $3r$ chains along with at most $2r$ violating clients (clients that are not in any chain.)*

We will use the following geometric fact about extreme points of polyhedra:

Claim 2.8.2. *Let $\mathcal{P}$ be a polyhedron in $\mathbb{R}^n$. Then $x \in \mathbb{R}^n$ is an extreme point of $\mathcal{P}$ if and only if there exist n linearly independent constraints of $\mathcal{P}$ that are tight at x. We call such a set of constraints a* basis *for x.*

Theorem 2.4.2 follows almost immediately from Theorem 2.8.1.

Proof of Theorem 2.4.2. Let $\bar{y}$ be an extreme point of LP_{iter} such that no C_{part}-, C_{full}-, or non-negativity constraint is tight, and suppose LP_{iter} satisfies the Distinct Neighbors Now consider the polytope:

$$\mathcal{P} = \{y \in \mathbb{R}^F \mid y(F_j) = 1 \quad \forall j \in C^*,\, 0 \leq y \leq 1,\, Ay \leq b\},$$

, where $Ay \leq b$ consists of the r_1 knapsack constraints and r_2 coverage constraints of LP_{iter}. We claim that $\bar{y}$ is an extreme point of $\mathcal{P}$. To see this, note that $\bar{y}$ is an extreme point of LP_{iter}, so fix any basis for $\bar{y}$ using tight constraints of LP_{iter}. By assumption, this basis uses no C_{part}-, C_{full}-, or non-negativity constraint. In particular, it only uses constraints of LP_{iter} that are also present in $\mathcal{P}$, so this basis certifies that $\bar{y}$ is an extreme point of $\mathcal{P}$.

Further, by Proposition 2.2.6, the set family $\{F_j \mid j \in C^*\}$ has bipartite intersection graph. Then we can apply Theorem 2.8.1 to $\bar{y}$ and polytope $\mathcal{P}$, which gives the desired result. □

2.8.1 Proof of Theorem 2.8.1

Now we go back to prove the more general chain decomposition theorem: Theorem 2.8.1. Throughout this section, let $\mathcal{P} = \{y \in \mathbb{R}^F \mid y(F_j) = 1 \quad \forall j \in C^*,\ 0 \le y \le 1,\ Ay \le b\}$ be a polytope satisfying the properties of Theorem 2.8.1. In particular, the intersection graph of $\mathcal{F} = \{F_j \mid j \in C^*\}$ is bipartite. Further, let $\bar{y}$ be an extreme point of $\mathcal{P}$ such that no non-negativity constraint is tight for $\bar{y}$.

The crux is the next lemma, which allows us to bound the complexity of the intersection graph with respect to the number of side constraints r. The lemma follows by constructing an appropriate basis for $\bar{y}$.

Definition 2.8.3. For any subset $C' \subset C^*$, let $dim(C')$ denote the maximum number of linearly independent C'-constraints, so the constraint set $\{y(F_j) = 1 \mid j \in C'\}$.

Lemma 2.8.4. *Let $\bar{y}$ be an extreme point of $\mathcal{P}$ such that no non-negativity constraint is tight. Then the number of fractional facilities in $\bar{y}$ satisfies $|F_{<1}| \le dim(C^*_{<1}) + r$ (r is the number of constraints of $Ay \le b$.)*

Proof. We construct a basis $\bar{y}$. First, for each integral facility $i \in F_{=1}$, we add the integrality constraint $\bar{y}_i \le 1$ to our basis. Thus we currently have $|F_{=1}|$ constraints in our basis.

It remains to choose $|F_{<1}|$ further linearly independent constraints to add to our basis. Note that we have already added all tight integrality constraints to our basis, and no non-negativity constraint is tight. Then the only remaining constraints we can add are the C^*-constraints and the r constraints of $Ay \le b$. Every $C^*_{=1}$-constraint is linearly dependent with the tight integrality constraints, which we already chose. It follows, the only possible constraints we can choose are the $C^*_{<1}$-constraints and the r constraints of $Ay \le b$ so $|F_{<1}| \le dim(C^*_{<1}) + r$. □

Now, to find a chain decomposition of $C^*_{<1}$, first we find the violating clients. We note that every F-ball contains at least two facilities. The violating clients will be those clients whose F-balls contain strictly more than two facilities, so we let $V = \{j \in C^*_{<1} \mid |F_j| > 2\}$ be the set of violating clients. We show that V has the desired size by a standard counting argument.

Proposition 2.8.5. $|V| \le 2r$

Proof. Note that the F-balls for clients in $C^*_{<1}$ are supported on only factional facilities, and each such ball contains at least two fractional facilities. Further, because $\mathcal{F}$ has bipartite intersection graph, each fractional facility is in at most two F-balls for clients in V. Combining these observations give:

$$|V| \le \sum_{j \in C^*} (|F_j| - 2) \le 2|F_{<1}| - 2|C^*|.$$

Applying Lemma 2.8.4 completes the proof:

$$|V| \le 2(dim(C^*_{<1}) + r) - 2|C^*_{<1}| \le 2r.$$

□

It remains to partition $C^*_{<1} \setminus V$ into the desired chains. Importantly, for all $j \in C^*_{<1} \setminus V$, we have $|F_j| = 2$. To find our chains, we consider the intersection graph of $C^*_{<1}$, so the intersection graph of the set family $\{F_j \mid j \in C^*_{<1}\}$. We let G denote this graph. Note that G is a subgraph of the standard intersection graph, so it is also bipartite by assumption.

We consider deleting the vertices V from G, which breaks G into some connected components, say $H_1, \ldots, H_\ell$. Let V_k denote the vertex set of H_k, so we have that $V_k \cup \cdots \cup V_k$ partitions $C^*_{<1} \setminus V$. Further, for all $k \in [\ell]$, every F-ball for clients in V_k contains exactly two facilities, and every facility is in at most two F-balls. Translating these statements into properties of the intersection graph, we can see that every vertex of H_k has degree at most two, and H_k is connected, so we can conclude that each H_k is a path or even cycle (we eliminate the odd cycle case because the intersection graph is bipartite.) Thus, each V_k forms a chain.

To complete the proof, it remains to upper bound ℓ, the number of chains. To do so, we first split the inequality given by Lemma 2.8.4 into the contribution by each H_k. Importantly, we observe that the $F(V_k)$'s are disjoint for all k because the V_k's correspond to distinct connected components. Then we have:

$$\sum_{k\in[\ell]} |F(V_k)| \leq |F_{<1}| \leq dim(C^*_{<1}) + r \leq \sum_{k\in[\ell]} dim(V_k) + dim(V) + r \leq \sum_{k\in[\ell]} dim(V_k) + 3r.$$

The way to interpret this inequality is that each chain V_k has a budget of $dim(V_k)$ fractional facilities to use in its chain, but we have an extra $3r$ facilities to pay for any facilities beyond each V_k's allocated budget. We will show that each chain uses at least one extra facility from this $3r$ surplus, which allows us to upper bound ℓ by $3r$.

Proposition 2.8.6. *For all $k \in [\ell]$, we have $|F(V_k)| > dim(V_k)$.*

Proof. There are two cases to consider based on if H_k is a path or even cycle. Suppose H_k is a path, say $j_1 \to \cdots \to j_p$. We first handle the degenerate case where there exists $q \in [p-1]$ such that $|F_{j_q} \cap F_{j_{q+1}}| = 2$. Because each F-ball for these clients has size exactly two, the path H_k has length exactly two and both clients have exactly the same F-ball. Thus, $|F(V_k)| = 2$, but $dim(V_k) = 1$.

Otherwise, we may assume for all $q \in [p-1]$, we have $|F_{j_q} \cap F_{j_{q+1}}| = 1$. Then each non-leaf client shares one facility with the previous client on the path and one with the next, and each leaf client has one facility all to itself. With these observations, we can bound:

$$dim(V_k) \leq |V_k| = \frac{1}{2}\sum_{j\in V_k} |F_j| = \frac{1}{2}(2|F(V_k)| - 2) = |F(V_k)| - 1.$$

It remains to consider the case where H_k is an even cycle. Then each facility in $F(V_k)$ is in exactly two F-balls, where each such ball contains exactly two facilities. Thus we have $|V_k| = \frac{1}{2}\sum_{j\in V_k} |F_j| = \frac{1}{2}(2|F(V_k)|) = |F(V_k)|$. It remains to show $dim(V_k) < |V_k|$. This follows because the constraints $\{y(F_j) = 1 \mid j \in V_k\}$ are not linearly independent. To see this, note that H_k is an even cycle , so consider the linear combination of constraints where odd-indexed clients along the cycle have coefficient 1 and even-indexed ones have coefficient -1. □

With the above proposition, we complete the proof by bounding ℓ: $\sum_{k\in[\ell]} dim(V_k) + 3r \geq \sum_{k\in[\ell]} |F(V_k)| \geq \sum_{k\in[\ell]} dim(V_k) + \ell \Rightarrow 3r \geq \ell$.

2.9 Conclusion

In this chapter, we gave improved approximation algorithms for knapsack median and k-median with outliers. Conceptually, our work refines the typical iterative rounding framework. Typically, iterative rounding algorithms have the following form:

1. Solve LP relaxation for problem
2. If solution is (near) integral, then we are done
3. Else modify the LP and re-solve

However, our iterative rounding algorithms work as follows:

1. Solve LP relaxation for problem
2. If solution is (near) integral, then we are done
3. **Else if solution is highly-structured, then modify the LP – exploiting the structure – and re-solve**
4. Else modify the LP and re-solve

The particular structure we use is the *chain decomposition*, which enable our new iterative operation. By exploiting the structure of extreme points before we terminate with a near-integral solution, we are able to find a better solution than the typical iterative rounding framework. We hope that – beyond k-median problems – this conceptual innovation will inspire improved iterative rounding algorithms for other combinatorial optimization problems. In fact, our quarter-ball chasing technique has already been applied to other clustering problems [Den22].

Some concrete open problems are:

- Can we use such an iterative rounding framework to improve the best known k-median approximation, which is 2.671 due to [CGLS23] (without any side constraints)?
- Recently, there has been interest in colorful k-clustering problems, where the clients are partitioned into, say ℓ, color classes and we must serve at least some quota of clients from each color class [BIPV19]. For the k-center objective, there is a $O(1)$-approximation for this colorful version of the problem [AAKZ22, GJS23]. Note that the colorful k-median problem is a special case of GKM with one coverage constraint per color class; thus, we can obtain a $O(1)$ approximate solution with $O(\ell)$ fractional facilities. Can we obtain a $O(1)$-approximation for colorful k-median by rounding the final $O(\ell)$ fractional facilities?
- One natural avenue to improve our approach is to consider eighth-ball chasing (rather than quarter-ball chasing) – i.e. we only delete a client when we have a neighboring ball that is at least eight times smaller. In principle, this could lead to improved re-routing cost and thus approximation ratio, but at the cost of more complex set structures arising at the LP extreme points. Can we understand the structure of such extreme points and exploit them to obtain further improvements?

Chapter 3

Online Throughput Maximization

3.1 Introduction

We give the first constant-competitive online algorithm for throughput maximization. We recall that the goal is to preemptively schedule jobs that arrive online at their release date r_j with sizes x_j and deadlines d_j on m identical machines to maximize the number of jobs that complete by their deadline.

Our main result is the following theorem.

Theorem 3.1.1. *For all $m > 1$, there exists a deterministic $O(1)$-competitive algorithm for throughput maximization on m machines.*

3.1.1 Scheduling Policies

We give some basic definitions and notations about scheduling policies.

A job j is *feasible* at time t (with respect to some schedule) if it can still be feasibly completed, so $x_j(t) > 0$ and $t + x_j(t) \leq d_j$, where $x_j(t)$ is the remaining processing time of job j at time t (with respect to the same schedule.) We define the *l*axity of a job j, as $\ell_j = (d_j - r_j) - x_j$, that is, the maximum amount of time we can not run job j and still possibly complete it.

A schedule $\mathcal{S}$ of jobs J is defined by a map from time/machine pairs (t, i) to a job j that is run on machine i at time t, with the constraint that no job can be run one two different machines at the same time. We conflate $\mathcal{S}$ with the scheduling policy as well as the set of jobs *completed* by the schedule. Thus, the objective value achieved by this schedule is $|\mathcal{S}|$.

A schedule is *non-migratory* if for every job j there exists a machine i such that if j is run at time t then j is run on machine i. Otherwise the schedule is *migratory*.

If $\mathcal{S}$ is a scheduling algorithm, then $\mathcal{S}(J, m)$ denotes the schedule that results from running $\mathcal{S}$ on instance J on m machines. Similarly, $\mathsf{Opt}(J, m)$ denotes the optimal schedule on instance J on m machines. We will sometimes omit the J and/or the m if they are clear from context. Sometimes we will abuse notation and let Opt denote a nearly-optimal schedule that additionally has some desirable structural property.

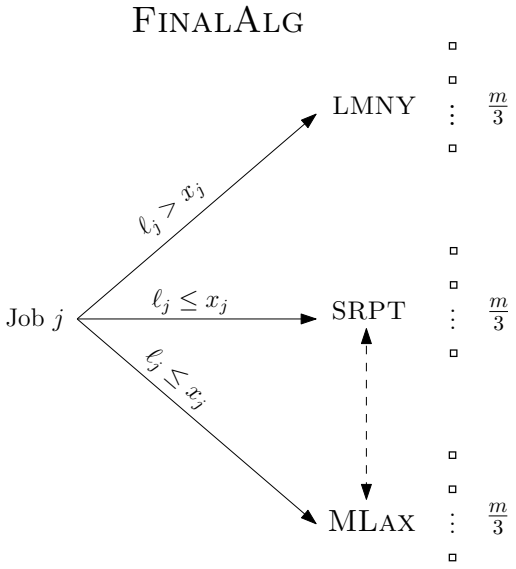

Figure 3.1: Summary of FinalAlg. *A job will be given to* LMNY *if its laxity is sufficiently high. Otherwise, it will be migrated between* SRPT *and* MLax.

3.1.2 Algorithms and Technical Overview

A simple consequence of the results in [KP01] and [KP03] is an $O(1)$-competitive randomized algorithm in the case that $m = O(1)$. Thus we concentrate on the case that m is large. We also observe that since there is an $O(1)$-approximate non-migratory schedule [KP01], changing the number of machines by an $O(1)$ factor does not change the optimal objective value by more than an $O(1)$ factor. This is because we can always take an optimal non-migratory schedule on m machines and create a new schedule on m/c machines whose objective value decreases by at most a factor of c, by keeping the m/c machines that complete the most jobs.

These observations about the structure of near-optimal schedules allow us to design a $O(1)$-competitive algorithm that is a combination of various deterministic algorithms. In particular, on an instance J, our algorithm, FinalAlg, will run a deterministic algorithm, LMNY, on $m/3$ machines on the subinstance $J_{hi} = \{j \in J \mid \ell_j > x_j\}$ of high laxity jobs, a deterministic algorithm SRPT on $m/3$ machines on the subinstance $J_{lo} = \{j \in J \mid \ell_j \leq x_j\}$ of low laxity jobs, and a deterministic algorithm MLax on $m/3$ machines on the subinstance J_{lo} of low laxity jobs. Note that we run SRPT and MLax on the same jobs. To achieve this, if both algorithms decide to run the same job j, then the algorithm in which j has shorter remaining processing time actually runs job j, and the other simulates running j. See Figure 3.1.

We will eventually show that for all instances, at least one of these three algorithms is $O(1)$-competitive, from which our main result will follow. Roughly, each of the three algorithms is responsible for a different part of Opt.

Our main theorem about FinalAlg is the following:

Theorem 3.1.2. *For any $m \geq 48$,* FinalAlg *is a $O(1)$-competitive deterministic algorithm for Throughput Maximization on m machines.*

We now discuss these three component algorithms of FinalAlg.

LMNY

The algorithm LMNY is the algorithm from [LMNY13] with the following guarantee.

Lemma 3.1.3. *[LMNY13] For any number of machines m, and any job instance J,* LMNY *is an $O(1)$-competitive deterministic algorithm on the instance J_{hi}.*

SRPT

The algorithm SRPT is the standard shortest remaining processing time algorithm, modified to only run jobs that are feasible.

Definition 3.1.4 (SRPT)**.** At each time, run the m feasible jobs with shortest remaining processing time. If there are less than m feasible jobs, then all feasible jobs are run.

We will show that SRPT is competitive with the low laxity jobs completed in Opt that are not preempted in Opt.

MLax

The final, most challenging, component algorithm of FinalAlg is MLax, which intuitively we want to be competitive on low-laxity jobs in Opt that are preempted.

To better understand the challenge of achieving this goal, consider $m = 1$ and an instance of *disagreeable jobs*. A set of jobs is disagreeable if, for any two jobs j and k, if j has an earlier release date than k, it also has a later deadline than k. Further, suppose all but one job in Opt is preempted and completed at a later time.

To be competitive, MLax must preempt almost all the jobs that it completes, but cannot afford to abandon too many jobs that it preempts. Because the jobs have low laxity, this can be challenging as it can only preempt each job for a small amount of time, and its hard to know which of the many options is the "right" job to preempt for. This issue was resolved in [KP03] for the case of $m = 1$ machine, but the issue gets more challenging when $m > 1$, because we also have to choose the "right" machine for each job.

We now describe the algorithm MLax. Let α be a sufficiently large constant (chosen later.) MLax maintains m stacks (last-in-first-out data structures) of jobs (one per machine), $H_1, \ldots, H_m$. The stacks are initially empty. At all times, MLax runs the top job of stack H_i on machine i. We define the *frontier* F to be the set consisting of the top job of each stack (i.e. all currently running jobs.) It remains to describe how the H_i's are updated.

There are two types of events that cause MLax to update the H_i's: reaching a job's pseudo-release time (defined below) or completing a job.

Definition 3.1.5 (Viable Jobs and Pseudo-Release Time)**.** The pseudo-release time (if it exists) $\tilde{r}_j$ of job j is the earliest time in $[r_j, r_j + \frac{\ell_j}{2}]$ such that there are at least $\frac{7}{8}m$ jobs j' on the frontier satisfying $\alpha x_{j'} \geq \ell_j$.

We say a job j is *viable* if $\tilde{r}_j$ exists and *non-viable* otherwise.

At job j's pseudo-release time (note $\tilde{r}_j$ can be determined online by MLax), MLax does the following:

a) If there exists a stack whose top job j' satisfies $\alpha x_j \leq \ell_{j'}$, then *push* j onto any such stack.

b) Else if there exist at least $\frac{3}{4}m$ stacks whose second-top job j'' satisfies $\alpha x_j \leq \ell_{j''}$ and further some such stack has top job j' satisfying $\ell_j > \ell_{j'}$, then on such a stack with minimum $\ell_{j'}$, *replace* its top job j' by j.

While the replacement operation in step b) can be implemented as a pop and then push, we view it as a separate operation for analysis purposes. To handle corner cases in these descriptions, one can assume that there is a job with infinite size/laxity on the bottom of each H_i.

When MLax completes a job j that was on stack H_i, MLax does the following:

c) *Pop* j off of stack H_i.

d) Keep *popping* H_i until the top job of H_i is feasible.

Analysis Sketch

There are three main steps in proving Theorem 3.1.2 to show FinalAlg is $O(1)$-competitive:

- In § 3.2, we show how to modify the optimal schedule to obtain certain structural properties that facilitate the comparison with SRPT and MLax.
- In § 3.3, we show that SRPT is competitive with the low-laxity, non-viable jobs. Intuitively, the jobs that MLax is running that prevent a job j from becoming viable are so much smaller than job j, and they provide a witness that SRPT must also be working on jobs much smaller than j.
- In § 3.4, we show that SRPT and MLax together are competitive with the low-laxity, viable jobs. First, we show that SRPT is competitive with the number of non-preempted jobs in Opt. We then essentially show that MLax is competitive with the number of preempted jobs in Opt. The key component in the design of MLax is the condition that a job j won't replace a job on the frontier unless at there are at least $\frac{3}{4}m$ stacks whose second-top job j'' satisfies $\alpha x_j \leq \ell_{j''}$. This condition most differentiates MLax from m copies of the Lax algorithm in [KP03]. This condition also allows us to surmount the issue of potentially assigning a job to a "wrong" processor, as jobs that satisfy this condition are highly flexible about where they can go on the frontier.

We combine these results in § 3.5 to complete the analysis of FinalAlg.

3.2 Structure of Optimal Schedule

The goal of this section is to introduce the key properties of (near-)optimal scheduling policies that we will use in our analysis.

By losing a constant factor in the competitive ratio, we can use a constant factor fewer machines than Opt, which justifies FinalAlg running each of three algorithms on $\frac{m}{3}$ machines. The proof is an extension of results in [KP01].

Lemma 3.2.1. *For any collection of jobs J, number of machines m, and $c > 1$, we have $|\mathsf{Opt}(J, \frac{m}{c})| = \Omega(\frac{1}{c}|\mathsf{Opt}(J, m)|)$.*

Proof. It is shown in [KP01] that for any schedule on m machines, there exists a non-migratory schedule on at most $6m$ machines that completes the same jobs. Applied to $\mathsf{Opt}(J, m)$, we obtain a non-migratory schedule $\mathcal{S}$ on $6m$ machines with $|\mathcal{S}| = |\mathsf{Opt}(J, m)|$. Keeping the $\frac{m}{c}$ machines that complete the most jobs in $\mathcal{S}$ gives a non-migratory schedule on $\frac{m}{c}$ machines that completes at least $\frac{1}{6c}|\mathcal{S}|$ jobs. □

A non-migratory schedule on m machines can be expressed as m schedules, each on a single machine and on a separate set of jobs. To characterize these single machine schedules, we introduce the concept of forest schedules. Let $\mathcal{S}$ be any schedule. For any job j, we let $f_j(\mathcal{S})$ and $c_j(\mathcal{S})$ denote the first and last times that $\mathcal{S}$ runs the job j, respectively. Note that $\mathcal{S}$ does not necessarily complete j at time $c_j(\mathcal{S})$.

Definition 3.2.2 (Forest Schedule)**.** We say a single-machine schedule $\mathcal{S}$ is a forest schedule if for all jobs j, j' such that $f_j(\mathcal{S}) < f_{j'}(\mathcal{S})$, $\mathcal{S}$ does not run j during the time interval $(f_{j'}(\mathcal{S}), c_{j'}(\mathcal{S}))$ (so the $(f_j(\mathcal{S}), c_j(\mathcal{S}))$-intervals form a laminar family.) Then $\mathcal{S}$ naturally defines a forest (in the graph-theoretic sense), where the nodes are jobs run by $\mathcal{S}$ and the descendants of a job j are the the jobs that are first run in the time interval $(f_j(\mathcal{S}), c_j(\mathcal{S}))$.

A non-migratory m-machine schedule is a forest schedule if all of its single-machine schedules are forest schedules.

With these definitions, we are ready to construct the near-optimal policies to which we will compare SRPT and MLax. The proof relies on modifications to Opt introduced in [KP03].

Lemma 3.2.3. *Let J be a set of jobs satisfying $\ell_j \le x_j$ for all $j \in J$. Then for any times $\hat{r}_j \in [r_j, r_j + \frac{\ell_j}{2}]$ and constant $\alpha \ge 1$, there exist non-migratory forest schedules $\mathcal{S}$ and $\mathcal{S}'$ on the jobs J such that:*

1. *Both $\mathcal{S}$ and $\mathcal{S}'$ complete every job they run.*
2. *Let J_i be the set of jobs that $\mathcal{S}$ runs on machine i. For every machine i and time, if there exists a feasible job in J_i, then $\mathcal{S}$ runs such a job.*
3. *For all jobs $j \in \mathcal{S}$, we have $f_j(\mathcal{S}) = \hat{r}_j$.*
4. *If job j' is a descendant of job j in $\mathcal{S}$, then $\alpha x_{j'} \le \ell_j$*
5. *$|\{\text{leaves of } \mathcal{S}'\}| + |\mathcal{S}| = \Omega(|\mathsf{Opt}(J)|)$.*

Proof. We modify the optimal schedule $\mathsf{Opt}(J)$ to obtain the desired properties. First, we may assume that $\mathsf{Opt}(J)$ is non-migratory by losing a constant factor (Lemma 3.2.1.) Thus, it suffices to prove the lemma for a single machine schedule, because we can apply the lemma to each of the single-machine schedules in the non-migratory schedule $\mathsf{Opt}(J)$. The proof for the single-machine case follows from the modifications given in Lemmas 22 and 23 of [KP03]. We note that [KP03] only show how to ensure $f_j(\mathcal{S}) = t_j$ for a particular $t_j \in [r_j, r_j + \frac{\ell_j}{2}]$, but it is straightforward to verify that the same proof holds for any $t_j \in [r_j, r_j + \frac{\ell_j}{2}]$. □

Intuitively, the schedule $\mathcal{S}$ captures the jobs in Opt that are preempted and $\mathcal{S}'$ captures the jobs in Opt that are not preempted (i.e. the leaves in the forest schedule.)

To summarize, we may assume by losing a constant factor that Opt is a non-migratory forest schedule. Looking ahead, we will apply Lemma 3.2.3 to the non-viable and viable jobs separately. In each case, we will use a combination of SRPT and MLax to handle jobs in $\mathcal{S}$ and SRPT for those in $\mathcal{S}'$.

3.3 SRPT is Competitive with Non-Viable Jobs

The main result of this section is that SRPT is competitive with the number of non-viable, low-laxity jobs of the optimal schedule (Theorem 3.3.1). We recall that a job j is non-viable if for *every time* in $[r_j, r_j + \frac{\ell_j}{2}]$, there are at least $\frac{1}{8}m$ jobs j' on the frontier of MLax satisfying $\alpha x_{j'} < \ell_j$.

Theorem 3.3.1. *Let J be a set of jobs satisfying $\ell_j \leq x_j$ for all $j \in J$. Then for $\alpha = O(1)$ sufficiently large and number of machines $m \geq 16$, we have $|\mathrm{SRPT}(J)| = \Omega(|\mathsf{Opt}(J_{nv})|)$, where J_{nv} is the set of non-viable jobs with respect to $\mathrm{MLax}(J)$.*

In the remainder of this section, we prove Theorem 3.3.1. The main idea of the proof is that for any non-viable job j, MLax is running many jobs that are much smaller than j (by at least an α-factor.) These jobs give a witness that SRPT must be working on these jobs or even smaller ones.

The following technical lemma - stating that SRPT is competitive with the leaves of any forest schedule - is needed in the proof as well as in § 3.4. Intuitively this follows because whenever some schedule is running a feasible job, then SRPT either runs the same job or a job with shorter remaining processing time. We will use this lemma to handle the non-viable jobs that are not preempted.

Lemma 3.3.2. *Let J be any set of jobs and $\mathcal{S}$ be any forest schedule on m machines and jobs $J' \subset J$ that only runs feasible jobs. Let L be the set of leaves of $\mathcal{S}$. Then $|\mathrm{SRPT}(J)| \geq \frac{1}{2}|L|$.*

Proof. It suffices to show that $|L \setminus \mathrm{SRPT}(J)| \leq |\mathrm{SRPT}(J)|$. The main property of SRPT gives:

Proposition 3.3.3. *Consider any leaf $\ell \in L \setminus \mathrm{SRPT}(J)$. Suppose $\mathcal{S}$ starts running ℓ at time t. Then SRPT completes m jobs in the time interval $[f_\ell(\mathcal{S}), f_\ell(\mathcal{S}) + x_\ell]$.*

Proof. At time $f_\ell(\mathcal{S})$ in SRPT (J), job ℓ has remaining processing time at most x_ℓ and is feasible by assumption. Because $\ell \notin \mathrm{SRPT}(J)$, there must exist a first time $t' \in [f_\ell(\mathcal{S}), f_\ell(\mathcal{S}) + x_\ell]$ where ℓ is not run by $\mathrm{SRPT}(J)$. At this time, $\mathrm{SRPT}(J)$ must be running m jobs with remaining processing time at most $x_\ell - (t' - f_\ell(\mathcal{S}))$. In particular, $\mathrm{SRPT}(J)$ must complete m jobs by time $f_\ell(\mathcal{S}) + x_\ell$. □

Using the proposition, we give a charging scheme: Each job $\ell \in L \setminus \mathrm{SRPT}(J)$ begins with 1 credit. By the proposition, we can find m jobs that $\mathrm{SRPT}(J)$ completes in the time interval $[f_\ell(\mathcal{S}), f_\ell(\mathcal{S}) + x_\ell]$. Then ℓ transfers $\frac{1}{m}$ credits each to m such jobs in SRPT.

It remains to show that each $j \in \mathrm{SRPT}(J)$ gets at most 1 credit. Note that j can only get credits from leaves ℓ such that $c_j(\mathrm{SRPT}) \in [f_\ell(\mathcal{S}), f_\ell(\mathcal{S}) + x_\ell]$. There are at most m such intervals (at most one per machine), because we only consider leaves, whose intervals are disjoint if there are on the same machine. □

Now we are ready to prove Theorem 3.3.1.

Proof of Theorem 3.3.1. Let $\mathcal{S}, \mathcal{S}'$ be the schedules guaranteed by Lemma 3.2.3 on the set of jobs J_{nv} with $\hat{r}_j = r_j$ for all $j \in J_{nv}$. We re-state the properties of these schedules for convenience:

1. Both $\mathcal{S}$ and $\mathcal{S}'$ complete every job they run.
2. Let J_i be the set of jobs that $\mathcal{S}$ runs on machine i. For every machine i and time, if there exists a feasible job in J_i, then $\mathcal{S}$ runs such a job.

3. For all jobs $j \in \mathcal{S}$, we have $f_j(\mathcal{S}) = r_j$.

4. If job j' is a descendant of job j in $\mathcal{S}$, then $\alpha x_{j'} \le \ell_j$

5. $|\{\text{leaves of } \mathcal{S}'\}| + |\mathcal{S}| = \Omega(|\mathsf{Opt}(J_{nv})|)$.

By Lemma 3.3.2, we have $|\text{SRPT}(J)| = \Omega(|\{\text{leaves of } \mathcal{S}'\}|)$. Thus, it remains to show the following:

Lemma 3.3.4. *For $\alpha = O(1)$ sufficiently large, $|\text{SRPT}(J)| = \Omega(|\mathcal{S}|)$*

Proof. We first show that for the majority of jobs j in $\mathcal{S}$'s forest, we run j itself on some machine for at least a constant fraction of the time interval $[r_j, r_j + \frac{\ell_j}{2}]$.

Proposition 3.3.5. *For at least half of the nodes j in $\mathcal{S}$'s forest, there exists a closed interval $I_j \subset [r_j, r_j + \frac{\ell_j}{2}]$ of length at least $\frac{\ell_j}{8}$ such that $\mathcal{S}$ runs j on some machine during I_j.*

Proof. We say a node j is a *non-progenitor* if j has less than 2^z descendants at depth z from j for all $z \ge 1$. Because $\mathcal{S}$ satisfies (1), at least half of the nodes in $\mathcal{S}$'s forest are non-progenitors. This follows from Lemma 7 in [KP03].

Now consider any non-progenitor node j. Because $\mathcal{S}$ is a forest, $\mathcal{S}$ is only running j or its descendants on some machine in times $[r_j, r_j + \frac{\ell_j}{2}]$. Further, because j is a non-progenitor and $\mathcal{S}$ satisfies (1) and (4), we can partition $[r_j, r_j + \frac{\ell_j}{2}] = [r_j, a] \cup (a, b) \cup [b, r_j + \frac{\ell_j}{2}]$ such that $[r_j, a]$ and $[b, r_j + \frac{\ell_j}{2}]$ are times where $\mathcal{S}$ is running j, and (a, b) are times where $\mathcal{S}$ is running descendants of j. By taking α sufficiently large, we have $|(a, b)| \le \frac{\ell_j}{4}$. This follows from Lemma 6 in [KP03]. It follows, at least one of $[r_j, a]$ or $[b, r_j + \frac{\ell_j}{2}]$ has length at least $\frac{\ell_j}{8}$. This gives the desired I_j. □

Let $\mathcal{S}'' \subset \mathcal{S}$ be the collection of jobs guaranteed by the proposition, so $|\mathcal{S}''| \ge \frac{1}{2}|\mathcal{S}|$. It suffices to show that $|\text{SRPT}(J)| = \Omega(|\mathcal{S}''|)$. Thus, we argue about $\text{MLax}(J)$ in the interval I_j (guaranteed by Proposition 3.3.5) for some $j \in \mathcal{S}''$.

Proposition 3.3.6. *Consider any job $j \in \mathcal{S}''$. For sufficiently large $\alpha = O(1)$, $\text{MLax}(J)$ starts running at least $\frac{m}{16}$ jobs during I_j such that each such job j' satisfies $[f_{j'}(\text{MLax}(J)), f_{j'}(\text{MLax}(J)) + x_{j'}] \subset I_j$.*

Proof. We let I' be the prefix of I_j with length exactly $\frac{|I_j|}{2} \ge \frac{\ell_j}{16}$. Recall that $j \in \mathcal{S}''$ is non-viable. Thus, because $I' \subset I_j \subset [r_j, r_j + \frac{\ell_j}{2}]$, $\text{MLax}(J)$ is always running at least $\frac{1}{8}m$ jobs j' satisfying $\alpha x_{j'} < \ell_j$ during I'.

We define J' to be the set of jobs that $\text{MLax}(J)$ runs during I' satisfying $\alpha x_{j'} < \ell_j$. We further partition J' into size classes, $J' = \bigcup_{z \in \mathbb{N}} J'_z$ such that J'_z consists of the jobs in J' with size in $(\frac{\ell_j}{\alpha^{z+1}}, \frac{\ell_j}{\alpha^z}]$.

For each machine i, we let T_z^i be the times in I' that $\text{MLax}(J)$ is running a job from J'_z on machine i. Note that each T_z^i is the union of finitely many intervals. Then because $\text{MLax}(J)$ is always running at least $\frac{1}{8}m$ jobs j' satisfying $\alpha x_{j'} \le \ell_j$ during I', we have:

$$\sum_{z \in \mathbb{N}} \sum_{i \in [m]} |T_z^i| \ge \frac{m}{8} |I'|.$$

By averaging, there exists some z with $\sum_{i \in [m]} |T_z^i| \ge \frac{m}{8} \frac{|I'|}{2^{z+1}}$.

Fix such a z. It suffices to show that there exist at least $\frac{m}{16}$ jobs in J'_z that LAX starts in I'. This is because every job $j' \in J'_z$ has size at most $\frac{\ell_j}{\alpha^z}$ and $|I_j \setminus I'| \geq \frac{\ell_j}{16}$. Taking $\alpha \geq 16$ gives that $f_{j'}(\text{LAX}) + x_{j'} \in I_j$.

Note that every job in J'_z has size within a α-factor of each other, so there can be at most one such job per stack at any time. This implies that there are at most m jobs in J'_z that *don't* start in I' (i.e. the start before I'.) These jobs contribute at most $m\frac{\ell_j}{\alpha^z}$ to $\sum_{i\in[m]}|T^i_z|$. Choosing α large enough, we can ensure that the jobs in J'_z that start in I' contribute at least $\frac{m}{16}\frac{\ell_j}{\alpha^z}$ to $\sum_{i\in[m]}|T^i_z|$. To conclude, we note that each job in J'_z that starts in I' contributes at most $\frac{\ell_j}{\alpha^z}$ to the same sum, so there must exist at least $\frac{m}{16}$ such jobs. □

Using the above proposition, we define a charging scheme to show that $|\text{SRPT}(J)| = \Omega(|\mathcal{S}''|)$. Each job $j \in \mathcal{S}''$ begins with 1 credit. By the proposition, we can find $\frac{m}{16}$ jobs j' such that $[f_{j'}(\text{MLAX}(J)), f_{j'}(\text{MLAX}(J)) + x_{j'}]$ is contained in the time when $\mathcal{S}$ runs j. There are two cases to consider. If all $\frac{m}{16}$ jobs we find are contained in $\text{SRPT}(J)$, then we transfer $\frac{16}{m}$ credits from j to each of the $\frac{m}{16}$-many jobs. Note that here we are using $m \geq 16$. Otherwise, there exists some such j' that is not in $\text{SRPT}(J)$. Then $\text{SRPT}(J)$ will complete at least m jobs in $[f_{j'}(\text{MLAX}(J)), f_{j'}(\text{MLAX}(J)) + x_{j'}]$. We transfer $\frac{1}{m}$ credits from j to each of the m-many jobs. To conclude, we note that each $j' \in \text{SRPT}(J)$ gets $O(\frac{1}{m})$ credits from at most $O(m)$ jobs in $\mathcal{S}''$. □

□

We showed that SRPT is competitive with the low-laxity, non-viable jobs in Opt. It remains to consider the low-laxity, viable jobs, which we handle in the next section.

3.4 SRPT and MLax are Competitive with Viable Jobs

We have shown that SRPT is competitive with the non-viable, low-laxity jobs. Thus, it remains to account for the viable, low-laxity jobs. We recall that a job j is viable if there exists a time in $[r_j, r_j + \frac{\ell_j}{2}]$ such that there are at least $\frac{7}{8}m$ jobs j' on the frontier satisfying $\alpha x_{j'} \geq \ell_j$. The first such time is the pseudo-release time, $\tilde{r}_j$ of job j. For these jobs, we show that SRPT and MLAX together are competitive with the viable, low-laxity jobs of the optimal schedule.

Theorem 3.4.1. *Let J be a set of jobs satisfying $\ell_j \leq x_j$ for all $j \in J$. Then for $\alpha = O(1)$ sufficiently large and number of machines $m \geq 8$, we have $|\text{SRPT}(J)| + |\text{MLAX}(J)| = \Omega(|\mathsf{Opt}(J_v)|)$, where J_v is the set of viable jobs with respect to $\text{MLAX}(J)$.*

Proof of Theorem 3.4.1. Let $\mathcal{S}, \mathcal{S}'$ be the schedules guaranteed by Lemma 3.2.3 on the set of jobs J_v with $\hat{r}_j = \tilde{r}_j$ for all $j \in J_v$. We re-state the properties of these schedules for convenience:

1. Both $\mathcal{S}$ and $\mathcal{S}'$ complete every job they run.
2. Let J_i be the set of jobs that $\mathcal{S}$ runs on machine i. For every machine i and time, if there exists a feasible job in J_i, then $\mathcal{S}$ runs such a job.
3. For all jobs $j \in \mathcal{S}$, we have $f_j(\mathcal{S}) = \tilde{r}_j$.
4. If job j' is a descendant of job j in $\mathcal{S}$, then $\alpha x_{j'} \leq \ell_j$
5. $|\{\text{leaves of } \mathcal{S}'\}| + |\mathcal{S}| = \Omega(|\mathsf{Opt}(J_v)|)$.

By Lemma 3.3.2, we have $|\text{SRPT}(J)| = \Omega(|\{\text{leaves of } \mathcal{S}'\}|)$. Thus, it suffices to show that $|\text{SRPT}(J)| + |\text{MLax}(J)| = \Omega(|\mathcal{S}|)$. We do this with two lemmas, whose proofs we defer until later. First, we show that MLax pushes (not necessarily completes) many jobs. In particular, we show:

Lemma 3.4.2. $|\text{SRPT}(J)| + \#(\textit{pushes of } \text{MLax}(J)) = \Omega(|\mathcal{S}|)$

The main idea to prove Lemma 3.4.2 is to consider sequences of preemptions in Opt. In particular, suppose Opt preempts job a for b and then b for c. Roughly, we use viability to show that the only way MLax doesn't push any of these jobs is if in between their pseudo-release times, MLax pushes $\Omega(m)$ jobs.

Second, we show that the pushes of MLax give a witness that SRPT and MLax together actually complete many jobs.

Lemma 3.4.3. $|\text{SRPT}(J)| + |\text{MLax}(J)| = \Omega(\#(\textit{pushes of } \text{MLax}(J)))$.

The main idea to prove Lemma 3.4.3 is to upper-bound the number of jobs that MLax pops because they are infeasible (all other pushes lead to completed jobs.) The reason MLax pops a job j for being infeasible is because while j was on a stack, MLax spent at least $\frac{\ell_j}{2}$ units of time running jobs higher than j on j's stack. Either those jobs are completed by MLax, or MLax must have have done many pushes or replacements instead. We show that the replacements give a witness that SRPT must complete many jobs.

Combining these two lemmas completes the proof of Theorem 3.4.1. □

Now we go back and prove Lemma 3.4.2 and Lemma 3.4.3.

3.4.1 Proof of Lemma 3.4.2

Recall that $\mathcal{S}$ is a forest schedule. We say the *first child* of a job j is the child j' of j with the earliest starting time $f_{j'}(\mathcal{S})$. In other words, if j is not a leaf, then its first child is the first job that pre-empts j. We first focus on a sequence of first children in $\mathcal{S}$.

Lemma 3.4.4. *Let $a, b, c \in \mathcal{S}$ be jobs such that b is the first child of a and c is the first child of b. Then* $\text{MLax}(J)$ *does at least one of the following during the time interval $[\tilde{r}_a, \tilde{r}_c]$:*

- *Push at least $\frac{m}{8}$ jobs,*
- *Push job b,*
- *Push a job on top of b when b is on the frontier,*
- *Push c.*

Proof. Because $\mathcal{S}$ is a forest schedule, we have $\tilde{r}_a < \tilde{r}_b < \tilde{r}_c$. It suffices to show that if during $[\tilde{r}_a, \tilde{r}_c]$, $\text{MLax}(J)$ pushes strictly fewer than $\frac{m}{8}$ jobs, $\text{MLax}(J)$ does not push b, and if $\text{MLax}(J)$ does not push any job on top of b when b is on the frontier, then $\text{MLax}(J)$ pushes c.

First, because $\text{MLax}(J)$ pushes strictly fewer than $\frac{m}{8}$ jobs during $[\tilde{r}_a, \tilde{r}_c]$, there exists at least $\frac{7}{8}m$ stacks that receive no push during this interval. We call such stacks *stable*. The key property of stable stacks is that the laxities of their top- and second-top jobs never decrease during this interval, because these stacks are only changed by replacements and pops.

Now consider time $\tilde{r}_a$. By definition of pseudo-release time, at this time, there exist at least $\frac{7}{8}m$ stacks whose top job j' satisfies $\alpha x_{j'} \geq \ell_j$. Further, for any such stack, let j'' be its second-top job. Then because b is a descendant of a in $\mathcal{S}$, we have:

$$\alpha x_b \leq \ell_a \leq \alpha x_{j'} \leq \ell_{j''}.$$

It follows that there exist at least $\frac{3}{4}m$ stable stacks whose second-top job j'' satisfies $\alpha x_b \leq \ell_{j''}$ for the entirety of $[\tilde{r}_a, \tilde{r}_c]$. We say such stacks are b-stable.

Now consider time $\tilde{r}_b$. We may assume b is not pushed at this time. However, there exist at least $\frac{3}{4}m$ that are b-stable. Thus, if we do not replace the top of some stack with b, it must be the case that the top job j' of every b-stable stack satisfies $\ell'_j \geq \ell_b$. Because these stacks are stable, their laxities only increase by time $\tilde{r}_c$, so MLax(J) will push c on some stack at that time.

Otherwise, suppose we replace the top job of some stack with b. Then b is on the frontier at $\tilde{r}_b$. We may assume that no job is pushed directly on top of b. If b remains on the frontier by time $\tilde{r}_c$, then MLax(J) will push c, because $\alpha x_c \leq \ell_b$. The remaining case is if b leaves the frontier in some time in $[\tilde{r}_b, \tilde{r}_c]$. We claim that it cannot be the case that b is popped, because by (2), $\mathcal{S}$ could not complete b by time $\tilde{r}_c$, so MLax(J) cannot as well. Thus, it must be the case that b is replaced by some job, say d at time $\tilde{r}_d$. At this time, there exist at least $\frac{3}{4}m$ stacks whose second-top job j'' satisfies $\alpha x_d \leq \ell_{j''}$. It follows, there exist at least $\frac{m}{2}$ b-stable stacks whose second-top job j'' satisfies $\alpha x_d \leq \ell_{j''}$ at time $\tilde{r}_d$. Note that because $m \geq 8$, there exists at least one such stack, say i, that is not b's stack. In particular, because b's stack has minimum laxity, it must be the case that the top job j' of stack i satisfies $\ell_{j'} \geq \ell_b$. Finally, because stack i is stable, at time $\tilde{r}_c$ we will push c. □

Now using the above lemma, we give a charging scheme to prove Lemma 3.4.2. First note that by Lemma 3.3.2, we have $|\text{SRPT}(J)| = \Omega(\#(\text{leaves of } \mathcal{S}))$. Thus, it suffices to give a charging scheme such that each job $a \in \mathcal{S}$ begins with 1 credit, and charges it to leaves of $\mathcal{S}$ and completions of MLax(J) so that each job is charged $O(1)$ credits. Each job $a \in \mathcal{S}$ distributes its 1 credit as follows:

- (Leaf Transfer) If a is a leaf or parent of a leaf of $\mathcal{S}$, say ℓ, then a charges ℓ for 1 credit.

Else let b be the first child of a and c the first child of b in $\mathcal{S}$

- (Push Transfer) If MLax(J) pushes b or c, then a charges 1 unit to b or c, respectively.
- (Interior Transfer) Else if job b is on the frontier, but another job, say d, is pushed on top of b, then a charges 1 unit to d.
- (m-Push Transfer) Otherwise, by Lemma 3.4.4, MLax(J) must push at least $\frac{m}{8}$ jobs during $[\tilde{r}_a, \tilde{r}_c]$. In this case, a charges $\frac{8}{m}$ units to each of these $\frac{m}{8}$ such jobs.

This completes the description of the charging scheme. It remains to show that each job is charged $O(1)$ credits. Each job receives at most 2 credits due to Leaf Transfers and at most 2 credits due to Push Transfers and Interior Transfers. As each job is in at most $3m$ intervals of the form $[\tilde{r}_a, \tilde{r}_c]$, each job is charged $O(1)$ from m-Push Transfers.

3.4.2 Proof of Lemma 3.4.3

Recall in MLax, there are two types of pops: a job is popped if it is completed, and then we continue popping until the top job of that stack is feasible. We call the former *completion pops* and the later *infeasible pops*. Note that it suffices to prove the next lemma, which bounds the infeasible pops. This is because #(pushes of MLax(J)) = #(completions pops of MLax(J)) + #(infeasible pops of MLax(J)). To see this, note that every stack is empty at the beginning and end of the algorithm, and the stack size only changes due to pushes and pops.

Lemma 3.4.5. *For $\alpha = O(1)$ sufficiently large, we have:*

$$|\text{SRPT}(J)| + |\text{MLax}(J)| + \#(\textit{pushes of } \text{MLax}(J)) \geq 2 \cdot \#(\textit{infeasible pops of } \text{MLax}(J)).$$

Proof. We define a charging scheme such that the completions of SRPT(J) and MLax(J) and the pushes executed by MLax(J) pay for the infeasible pops. Each completion of SRPT(J) is given 2 credits, each completion of MLax(J) is given 1 credit, and each job that MLax(J) pushes is given 1 credit. Thus each job begins with at most 4 credits. For any $z \geq 0$, we say job j' is z-below j (at time t) if j' and j are on the same stack in MLax(J) and j' is z positions below j on that stack at time t. We define z-above analogously. A job j distributes these initial credits as follows:

- (SRPT-transfer) If SRPT(J) completes job j and MLax also ran j at some point, then j gives $\frac{1}{2^{z+1}}$ credits to the job that is z-below j at time $f_j(\text{MLax}(J))$ for all $z \geq 0$.
- (m-SRPT-transfer) If SRPT(J) completes job j at time t, then j gives $\frac{1}{2^{z+1}}\frac{1}{m}$ credits to the job that is z-below the top of each stack in MLax(J) at time t for all $z \geq 0$.
- (MLax-transfer) If MLax(J) completes a job j, then j gives $\frac{1}{2^{z+1}}$ credits to the job that is z-below j at the time j is completed for all $z \geq 0$.
- (Push-transfer) If MLax(J) pushes a job j, then j gives $\frac{1}{2^{z+1}}$ credits to the job that is z-below j at the time j is pushed for all $z \geq 0$.

It remains to show that for $\alpha = O(1)$ sufficiently large, every infeasible pop gets at least 4 credits. We consider any job j that is an infeasible pop of MLax(J). At time $\tilde{r}_j$ when j joins some stack in MLax(J), say H, j's remaining laxity was at least $\frac{\ell_j}{2}$. However, as j later became an infeasible pop, it must be the case that while j was on stack H, MLax(J) was running jobs that are higher than j on stack H for at least $\frac{\ell_j}{2}$ units of time.

Let I be the union of intervals of times that MLax(J) runs a job higher than j on stack H (so j is on the stack for the entirety of I.) Then we have $|I| \geq \frac{\ell_j}{2}$. Further, we partition I based on the height of the job on H that MLax(J) is currently running. In particular, we partition $I = \bigcup_{z \geq 1} I_z$, where I_z is the union of intervals of times that MLax(J) runs a job on H that is exactly z-above j.

By averaging, there exists a $z \geq 1$ such that $|I_z| \geq \frac{\ell_j}{2^{z+1}}$. Fix such a z. We can write I_z as the union of disjoint intervals, say $I_z = \bigcup_{u=1}^{s} [a_u, b_u]$. Because during each sub-interval, MLax(J) is running jobs on H that are much smaller than j itself, these jobs give a witness that SRPT(J) completes many jobs as long as these sub-intervals are long enough. We formalize this in the following proposition, whose proof is omitted in this extended abstract.

Proposition 3.4.6. *In each sub-interval $[a_u, b_u]$ of length at least $4\frac{\ell_j}{\alpha^z}$, job j earns at least $\frac{1}{2^{z+3}}\frac{b_u - a_u}{\ell_j/\alpha^z}$ credits from* SRPT*-transfers and m-*SRPT*-transfers.*

On the other hand, even if the sub-intervals are too short, the job j still gets credits from MLax-transfers and Push-transfers when the height of the stack changes. We formalize this statement in the following proposition, whose proof is omitted in this extended abstract.

Proposition 3.4.7. *For every sub-interval $[a_u, b_u]$, job j earns at least $\frac{1}{2^{z+2}}$ credits from* MLax*-transfers and Push-transfers at time b_u.*

Now we combine the above two propositions to complete the proof of Lemma 3.4.5. We say a sub-interval $[a_u, b_u]$ is *long* if it has length at least $4\frac{\ell_j}{\alpha^z}$ (i.e. we can apply Proposition 3.4.6 to it) and *short* otherwise. First, suppose the aggregate length of all long intervals it at least $4 \cdot 2^{z+3}\frac{\ell_j}{\alpha^z}$. Then by Proposition 3.4.6, job j gets at least 4 credits from the long intervals. Otherwise, the aggregate length of all long intervals is less than $4 \cdot 2^{z+3}\frac{\ell_j}{\alpha^z}$. In this case, recall that the long and short intervals partition I_z, which has length at least $\frac{\ell_j}{2^{z+1}}$. It follows, the aggregate length of the short intervals is at least $\frac{\ell_j}{2^{z+1}} - 4 \cdot 2^{z+3}\frac{\ell_j}{\alpha^z}$. For $\alpha = O(1)$ large enough, we may assume the aggregate length of the short intervals is at least $4 \cdot 2^{z+2}\frac{4\ell_j}{\alpha^z}$. Because each short interval has length at most $4\frac{\ell_j}{\alpha^z}$, there are at least $4 \cdot 2^{z+2}$ short intervals. We conclude, by Proposition 3.4.7, job j gets at least 4 credits from the short intervals. We conclude, in either case job j gets at least 4 credits. □

3.5 Putting it all together

In this section, we prove our main result, Theorem 3.1.1, which follows from the next meta-theorem:

Theorem 3.5.1. *Let J be any set of jobs. Then for number of machines $m \geq 16$, we have $|\text{LMNY}(J_{hi})| + |\text{SRPT}(J_{lo})| + |\text{MLax}(J_{lo})| = \Omega(|\mathsf{Opt}(J)|)$, where $J_{hi} = \{j \in J \mid \ell_j > x_j\}$ and $J_{lo} = \{j \in J \mid \ell_j \leq x_j\}$ partition J into high- and low-laxity jobs.*

Proof. We have $|\text{LMNY}(J_{hi})| = \Omega(|\mathsf{Opt}(J_{hi})|$ by Lemma 3.1.3. Also, we further partition $J_{lo} = J_v \cup J_{nv}$ into the viable and non-viable jobs with respect to $\text{MLax}(J_{lo})$. Then Theorem 3.3.1 and Theorem 3.4.1 together give $|\text{SRPT}(J_{lo})| + |\text{MLax}(J_{lo})| = \Omega(|\mathsf{Opt}(J_v)| + |\mathsf{Opt}(J_{nv})|)$. To complete the proof, we observe that $J = J_{hi} \cup J_v \cup J_{nv}$ partitions J, so $|\mathsf{Opt}(J_{hi})| + |\mathsf{Opt}(J_v)| + |\mathsf{Opt}(J_{nv})| = \Omega(|\mathsf{Opt}(J)|)$. □

The proof of Theorem 3.1.2, which gives our performance guarantee for FinalAlg is immediate:

Proof of Theorem 3.1.2. By combining Theorem 3.5.1 and Lemma 3.2.1, the objective value achieved by FinalAlg is:

$$\begin{aligned}\Omega(|\text{LMNY}(J_{hi}, \frac{m}{3})| + |\text{SRPT}(J_{lo}, \frac{m}{3})| + |\text{MLax}(J_{lo}, \frac{m}{3})|) &= \Omega(|\mathsf{Opt}(J, \frac{m}{3})|) \\ &= \Omega(|\mathsf{Opt}(J, m)|).\end{aligned}$$

□

Finally, we obtain our $O(1)$-competitive deterministic algorithm for all $m > 1$ (recall FinalAlg is $O(1)$-competitive only when $m \geq 48$) by using a two-machine algorithm when m is too small:

Proof of Theorem 3.1.1. Our algorithm is the following: If $1 < m < 48$, then we run the deterministic two-machine algorithm from [KP03] which is $O(1)$-competitive with the optimal single-machine schedule. Thus by Lemma 3.2.1, this algorithm is also $O(m) = O(1)$-competitive for all $m < 48$. Otherwise, $m \geq 48$, so we run FinalAlg. □

3.6 Conclusion

We considered throughput maximization on multiple machines without any high-laxity or speed augmentation assumption. Our final competitive ratio is a large unspecified constant (obtained by choosing $\alpha = O(1)$ sufficiently large.)

Conceptually, our work overcomes barriers faced by previous approaches, which were mainly in the case of low laxity jobs that are pre-empted by the optimal solution and completed later. Our new sub-algorithm for this case, MLax, uses carefully-crafted definitions for pseudo-release times and the replacement rule to ensure that at least a constant-factor of the machines are utilized effectively.

The main open question is if there exists a simpler algorithm and analysis for online throughput maximization on multiple machines. The "carefully crafted" definition of MLax makes its design and analysis arguably fragile and unnatural.

Further, is migration necessary to achieve constant-competitiveness? Note that two out of three of our component algorithms (LMNY and Lax) are non-migratory. Naively, SRPT is migratory, and we also use migration because Lax and SRPT share the low-laxity jobs. We leave it as an open question to develop a *non-migratory* constant-competitive algorithm for throughput maximization on multiple machines.

Chapter 4

Stochastic Load Balancing

4.1 Introduction

We first define the *configuration balancing* problem, which generalizes many resource allocation problems including load balancing. In this problem, there are m resources (machines) and n requests (jobs). Every request j has q_j possible configurations $x_j(1), \ldots, x_j(q_j) \in \mathbb{R}^m_{\geq 0}$. We must choose one configuration $c_j \in [q_j]$ per request, which adds $x_j(c_j)$ to the load vector on the resources. The goal is to minimize the makespan, i.e., the load of the most-loaded resource.

In the stochastic version, *configuration balancing with stochastic requests*, we assume each configuration c of request j is a random vector $X_j(c) \sim \mathcal{D}_j(c)$ with known distribution $\mathcal{D}_j(c)$ such that the $X_j(c)$'s are independent across different requests j. However, the actual realized vector of a configuration c of request j is only observed after *irrevocably* selecting this particular configuration for request j. The objective is to minimize the expected maximum load (i.e., the expected makespan)

$$\mathbb{E}\Big[\max_i \sum_{j=1}^{n} X_{ij}(c_j)\Big],$$

where c_j is the configuration chosen for request j. We assume that we have oracle access to the $\mathcal{D}_j(c)$'s; in particular we assume that in constant time, we can compute any needed statistic of the distribution $\mathcal{D}_j(c)$.

As in stochastic load balancing, we also consider the online setting, where requests are not known in advance and they are revealed one-by-one (online-list model). Upon arrival, each request reveals its $\mathcal{D}_j(c)$'s, from which the algorithm must choose one irrevocably before the next request arrives.

For these general offline and online problem, we give asymptotically tight approximations.

Theorem 4.1.1. *For configuration balancing with stochastic requests there is a randomized* offline *algorithm that computes a non-adaptive policy that is a $O\big(\frac{\log m}{\log\log m}\big)$-approximation and a deterministic* online *algorithm that is a $O(\log m)$-approximation when comparing to the optimal offline adaptive policy. Both algorithms run in polynomial time in the number of resources and the total number of configurations over all requests.*

Note that stochastic load balancing is a special case – the resources are the m machines, and each job has m possible configurations – one corresponding to assigning that job to each machine. Thus, we can efficiently represent all configurations. This gives the following corollary for stochastic load

balancing, where additionally we can make the offline algorithm deterministic by exploiting the simpler LP in this special case.

Theorem 4.1.2. *There exist efficient deterministic algorithms that compute a non-adaptive policy for load balancing on unrelated machines with stochastic jobs that achieve an* $\Theta\left(\frac{\log m}{\log \log m}\right)$*-approximation* offline *and an* $\Theta(\log m)$*-approximation* online *when comparing to the optimal offline adaptive policy.*

Further, if the machines are related ($X_{ij} = \frac{X_j}{s_i}$ for machine speed s_i), we give improved *adaptive approximations*. Interestingly, our adaptive algorithms begin with a similar *non-adaptive* assignment of jobs to machines, but we deviate from the assignment adaptively to obtain our improved algorithms.

Theorem 4.1.3. *For load balancing on related machines with stochastic jobs, there exist efficient deterministic algorithms that compute an adaptive* offline $O(1)$*-approximation and an adaptive* online $O(\log \log m)$*-approximation when comparing to the optimal offline adaptive policy.*

We also quantify the power and limitation of non-clairvoyant algorithms for load balancing on related machines (i.e. if our algorithm did not use the distributional information at all). Recall that a non-clairvoyant algorithm has no prior knowledge of the job sizes and must decide where to schedule a job using only the information about current machine loads and speeds. After an algorithm's decision upon assigning a job, an adversary chooses the realized size of the job, which the algorithm then observes for subsequent decisions. We refer to a non-clairvoyant *online* algorithm as a non-clairvoyant algorithm that learns about the existence of jobs (of unknown size) online one by one.

Theorem 4.1.4. *For load balancing on related machines, there exists an efficient* non-clairvoyant online *algorithm that* $O(\sqrt{m})$*-approximates the optimal clairvoyant algorithm. Further, any non-clairvoyant algorithm is at least* $\Omega(\sqrt{m})$*-approximate.*

Thus the *clairvoyance gap* – the maximum ratio between the makespan of an optimal non-clairvoyant algorithm and the optimal makespan achievable by a clairvoyant offline algorithm – is $\Theta(\sqrt{m})$ [1].

Another special case of configuration balancing that is of particular interest is virtual circuit routing (or congestion minimization). In this problem, there is a directed graph $G = (V, E)$ on m edges with edge capacities $c_e > 0$ for $e \in E$, and n requests, each request consisting of a source-sink pair (s_j, t_j) in G and a demand $x_j \geq 0$. The goal is to route each request j from s_j to t_j via some directed path, increasing the load/congestion of each edge e on the path by $\frac{x_j}{c_e}$, while the objective is to minimize the load of the most-loaded edge.

In the stochasic version, the demands $X_j \sim \mathcal{D}_j$ are random with known distributions. This is again a special case of configuration balancing with stochastic requests where the resources are the edges, and each request has a configuration for each possible routing path. Note that in general there are exponentially many configurations, so Theorem 4.1.1 does not immediately imply efficient algorithms for stochastic routing. However, we can exploit the particular structure of the configurations to efficient optimize over the configurations for stochastic virtual circuit rounding, and obtain the same result.

[1] Again, I will advertise this result. If there is one proof you read in this entire dissertation, I think it should be this one. See § 4.5.

Theorem 4.1.5. *For virtual circuit routing with stochastic requests, there exist an efficient randomized* offline *algorithm computing a non-adaptive policy that is a* $\Theta\big(\frac{\log m}{\log\log m}\big)$*-approximation and an efficient deterministic* online *algorithm that computes an* $\Theta(\log m)$*-approximation when comparing to the optimal offline adaptive policy.*

Note that even for deterministic requests, virtual circuit routing is hard to approximate better than $O(\frac{\log m}{\log\log m})$ unless NP $\subseteq$ ZPTIME$(n^{\log\log n})$ [CGKT07], and there is a lower bound of $\Omega(\log n)$ for online algorithms [AAF+97]. Thus, Theorem 4.1.5 implies that stochastic requests are not harder to approximate than deterministic requests.

4.1.1 Technical Overview

We illustrate the main idea behind our non-adaptive policies, which compare to the optimal offline adaptive policy. Throughout this paper, we let Opt denote the optimal adaptive policy as well as its makespan. As in many other stochastic optimization problems, our goal is to give a good deterministic proxy for the makespan of a policy. Then, our algorithm will optimize over this deterministic proxy to obtain a good solution. First, we observe that if all configurations were bounded with respect to $\mathbb{E}[\mathsf{Opt}]$ in every entry, then selecting configurations such that each resource has expected load $O(\mathbb{E}[\mathsf{Opt}])$ gives the desired $O\big(\frac{\log m}{\log\log m}\big)$-approximation by standard concentration inequalities for independent sums with bounded increments. Thus, in this case the expected load on each resource is a good proxy. However, in general, we have no upper bound on $X_{ij}(c)$, so we cannot argue as above. We turn these unbounded random variables into bounded ones in a standard way by splitting each request into *truncated* and *exceptional* parts.

Definition 4.1.6 (Truncated and Exceptional Parts)**.** Let $\tau \geq 0$ be a fixed threshold. For a random variable X, its truncated part (with respect to threshold τ) is $X^T := X \cdot \mathbb{1}_{X<\tau}$. Similarly, its exceptional part is $X^E := X \cdot \mathbb{1}_{X\geq\tau}$. Note that $X = X^T + X^E$.

It is immediate that the truncated parts $X^T_{ij}(c)$ are bounded in $[0,\tau]$. Taking $\tau = O(\mathbb{E}[\mathsf{Opt}])$, we can control their contribution to the makespan using concentration. It remains to find a good proxy for the contribution of exceptional parts to the makespan. This is one of the main technical challenges of our work as we aim to compare against the optimal adaptive policy. We will see that adaptive policies have much better control over the exceptional parts than non-adaptive ones.

Concretely, let c_j be the configuration chosen by some fixed policy for request j. Note that c_j itself can be a random variable in $\{1,\dots,q_j\}$. We want to control the quantity

$$\mathbb{E}\Big[\max_i \sum_{j=1}^{n} X^E_{ij}(c_j)\Big].$$

Because we have no reasonable bound on the $X^E_{ij}(c_j)$'s, for non-adaptive policies, we can only upper bound the expected maximum by the following sum

$$\mathbb{E}\Big[\max_{1\leq i\leq m} \sum_{j=1}^{n} X^E_{ij}(c_j)\Big] \leq \sum_{j=1}^{n} \mathbb{E}\Big[\max_{1\leq i\leq m} X^E_{ij}(c_j)\Big]. \tag{4.1}$$

We call the right hand side *total (expected) exceptional load*. The above inequality is tight up to constants for non-adaptive policies, so it seems like the total expected exceptional load is a good proxy to use for our algorithm. However, it is far from tight for adaptive policies as the example shows.

Example 4.1.7. We define an instance of load balancing on *related* machines, where each machine i has a speed s_i and each job j has processing time X_j such that $X_{ij} = \frac{X_j}{s_i}$. Our instance has one "fast" machine with speed 1 and $m-1$ "slow" machines each with speed $\frac{1}{\tau m}$, where $\tau > 0$ is the truncation threshold. There are m jobs: a stochastic one with processing time $\tau \cdot \mathrm{Ber}\left(\frac{1}{\tau}\right)$ and $m-1$ deterministic jobs with processing time $\frac{1}{m}$. The optimal adaptive policy schedules first the stochastic job on the fast machine. If its realized size is 0, then it schedules all deterministic jobs on the fast machine. Otherwise its realized size is τ and we schedule one deterministic job on each slow machine. This gives $\mathbb{E}[\mathsf{Opt}] = \left(1 - \frac{1}{\tau}\right)\left(\frac{m-1}{m}\right) + \frac{1}{\tau} \cdot \tau = \Theta(1)$. However, the total expected exceptional load (with respect to threshold τ) is $\sum_{i,j} \mathbb{E}\left[X_{ij}^E \cdot \mathbb{1}_{j \to i}\right] = \frac{1}{\tau}(m\tau) = m$, where $j \to i$ denotes that job j is assigned to machine i, i.e., configuration i is chosen for j.

In the example, the optimal adaptive policy accrues a lot of exceptional load, but this does not have a large effect on the makespan. Concretely, (4.1) can be loose by a $\Omega(m)$-factor for adaptive policies. Thus, it seems that the total exceptional load is a bad proxy in terms of lower-bounding Opt. However, we show that, by comparing our algorithm to a *near-optimal* adaptive policy rather than the optimal one, the total exceptional load becomes a good proxy in the following sense. This is the main technical contribution of our work, and it underlies all of our algorithmic techniques.

Theorem 4.1.8. *For configuration balancing with stochastic requests, there exists an adaptive policy with expected maximum load and total expected exceptional load at most* $2 \cdot \mathbb{E}[\mathsf{Opt}]$ *with respect to any truncation threshold* $\tau \geq 2 \cdot \mathbb{E}[\mathsf{Opt}]$. *Further, any configuration* c *selected by this policy satisfies* $\mathbb{E}\left[\max_i X_i(c)\right] \leq \tau$.

The proof of the above relies on carefully modifying the "decision tree" representing the optimal adaptive policy. In light of Theorem 4.1.8, the deterministic proxies we consider are the expected truncated load on each resource and the total expected exceptional load. All of our algorithms then proceed by ensuring that both quantities are bounded with respect to $\mathbb{E}[\mathsf{Opt}]$. In the offline case, we round a natural assignment-type linear program (LP), and in the online case, we use a potential function argument. All of these algorithms actually output non-adaptive policies.

For the special case of related-machines load balancing, we also compute a non-adaptive assignment but instead of following it exactly, we deviate using adaptivity and give improved solutions.

4.2 Configuration Balancing with Stochastic Requests

In this section, we prove our main results for the most general problem we consider: configuration balancing. We give a $O\left(\frac{\log m}{\log \log m}\right)$-approximation offline and a $O(\log m)$-approximation online. Both of our algorithms are non-adaptive. Before describing the algorithms, we give our main structure theorem that enables all of our results. Roughly, we show that instead of comparing to the optimal adaptive policy, by losing only a constant factor in the approximation ratio, we can compare to a near-optimal policy that behaves like a non-adaptive one (with respect to the proxy objectives we consider – namely, the total expected exceptional load).

4.2.1 Structure theorem

The goal of this section is to show that there exists a near-optimal policy as guaranteed by Theorem 4.1.8. To this end, we modify the optimal policy by "restarting" whenever an exceptional request is encountered. Additionally, we ensure that this modified policy never selects a configuration c for a request j with $\mathbb{E}\left[\max_i X_{ij}(c)\right] > \tau$.

We let J denote the set of requests. For any subset $J' \subseteq J$, we let $\mathsf{Opt}(J')$ denote the optimal adaptive policy (and its maximum load) on the set of requests J'. Note that $\mathsf{Opt}(\emptyset) = 0$. Our (existential) algorithm to construct such a policy will begin by running the optimal policy $\mathsf{Opt}(J)$ on all requests. However, once a exceptional request is encountered or the next decision will choose a configuration with too large expected max, we cancel $\mathsf{Opt}(J)$ and instead recurse on all remaining requests, ignoring all previously-accrued loads. The main idea of our analysis is that we recurse with small probability. We now proceed formally.

Theorem 4.1.8. *For configuration balancing with stochastic requests, there exists an adaptive policy with expected maximum load and total expected exceptional load at most $2 \cdot \mathbb{E}[\mathit{Opt}]$ with respect to any truncation threshold $\tau \geq 2 \cdot \mathbb{E}[\mathit{Opt}]$. Further, any configuration c selected by this policy satisfies $\mathbb{E}\big[\max_i X_i(c)\big] \leq \tau$.*

Proof. We prove the theorem by induction on the number of requests $n \in \mathbb{N}$. The base case $n = 0$ is trivial. Now we consider $n > 0$. Let J be the set of n requests. Our algorithm to construct the desired policy $S(J)$ is the following. Throughout, we fix a truncation threshold $\tau \geq 2 \cdot \mathbb{E}[\mathsf{Opt}]$.

Algorithm 7: Policy $S(J)$

```
7.1  R ← J                                                  // remaining requests
7.2  if R = ∅ then
7.3      return empty policy                                // finish
7.4  while R ≠ ∅ do
7.5      j ← first / next request considered by Opt(J)
7.6      c_j ← configuration chosen for request j by Opt(J)
7.8      if E[max_i X_ij(c_j)] > τ then                     // maximum too large
7.9          break
7.10     else
7.11         choose c_j for request j                       // S(J) follows Opt(J)
7.12         R ← R \ {j}                                    // update remaining requests
7.14         if max_i X_ij(c_j) ≥ τ then                    // exceptional configuration observed
7.15             break
7.16 run S(R)                                               // recurse with remaining requests
```

Let R be the random set of requests we recurse on after stopping $\mathsf{Opt}(J)$. We first show that indeed $|R| < |J|$, so we can apply induction. Suppose we did not follow $\mathsf{Opt}(J)$ to completion because a chosen configuration becomes exceptional (7.14); denote this event by $\mathcal{E}$. In this case, there is at least one request for which we have chosen a configuration. Hence, we have $|R| < |J|$, and therefore there is a policy S with the required properties by induction.

Suppose now that $\mathsf{Opt}(J)$ chooses a configuration c_j for request j that is too large (7.8); denote this event by $\mathcal{L}$. We have to show that $|R| < |J|$ holds as well. Suppose for the sake of contradiction that j was the first request considered by $\mathsf{Opt}(J)$. As Opt is w.l.o.g. deterministic, this implies $\mathbb{E}[\mathsf{Opt}] \geq \mathbb{E}\big[\max_i X_{ij}(c_j)\big] > 2\mathbb{E}[\mathsf{Opt}]$, a contradiction. Hence, the desired policy S exists by induction.

The maximum load of this policy is at most $\mathsf{Opt}(J) + S(R)$, where we set $R = \emptyset$ if no decision

results in an exceptional or too large configuration when running $\mathsf{Opt}(J)$. In expectation, we have

$$\begin{aligned}\mathbb{E}[S(R)] &= \sum_{J' \subsetneq J} \mathbb{E}[S(R) \mid R = J']\mathbb{P}[R = J'] = \sum_{J' \subsetneq J} \mathbb{E}[S(J')]\mathbb{P}[R = J'] \leq 2 \sum_{J' \subsetneq J} \mathbb{E}[\mathsf{Opt}(J')]\mathbb{P}[R = J] \\ &\leq 2 \cdot \mathbb{E}[\mathsf{Opt}(J)]\mathbb{P}[R \neq \emptyset].\end{aligned}$$

In the second equality, we use the fact that the realizations of the remaining requests in R are independent of the event $R = J'$. The first inequality uses the inductive hypothesis. The last inequality uses $J' \subseteq J$, so $\mathbb{E}[\mathsf{Opt}(J')] \leq \mathbb{E}[\mathsf{Opt}(J)]$, and $\mathsf{Opt}(\emptyset) = 0$.

Note that on the event $R \neq \emptyset$, we have that $\mathsf{Opt}(J)$ chooses a configuration that becomes exceptional or that is too large in expectation. By definition of the policy S, the events $\mathcal{E}$ and $\mathcal{L}$ are disjoint. By definition of $\mathcal{E}$, we have $\mathsf{Opt}(J) \cdot \mathbb{1}_{\mathcal{E}} \geq \tau \cdot \mathbb{1}_{\mathcal{E}}$. Observe that the event $\mathcal{L}$ implies that there is a request j^* with configuration c^* chosen by $\mathsf{Opt}(J)$ with $\mathbb{E}\big[\max_i X_{ij^*}(c^*)\big] \geq \tau$. Since the realization of $\max_i X_{ij^*}(c^*)$ is independent of the choice c^*, this implies $\mathbb{E}[\mathsf{Opt} \mid \mathcal{L}] \geq \mathbb{E}[\max_i X_{ij^*}(c^*) \mid \mathcal{L}] = \mathbb{E}[\max_i X_{ij^*}(c^*)] \geq \tau$. Thus,

$$\mathbb{E}[\mathsf{Opt}(J)] \geq \mathbb{P}[\mathcal{E}]\mathbb{E}[\mathsf{Opt}(J) \mid \mathcal{E}] + \mathbb{P}[\mathcal{L}]\mathbb{E}[\mathsf{Opt}(J) \mid \mathcal{L}] \geq \mathbb{P}[\mathcal{E}]\tau + \mathbb{P}[\mathcal{L}]\tau \geq 2\mathbb{P}[R \neq \emptyset]\mathbb{E}[\mathsf{Opt}(J)].$$

Rearranging yields $\mathbb{P}[R \neq \emptyset] \leq \frac{1}{2}$. Hence, we can bound the expected makespan of policy $S(J)$ by

$$\mathbb{E}[\mathsf{Opt}(J)] + \mathbb{E}[S(R)] \leq \mathbb{E}[\mathsf{Opt}(J)] + 2\mathbb{E}[\mathsf{Opt}(J)]\mathbb{P}[R \neq \emptyset] \leq 2\mathbb{E}[\mathsf{Opt}(J)].$$

The computation for the total expected exceptional load is similar. We let $j \to c$ denote the event that our policy chooses configuration c for request j. Then, we can split the exceptional load into two parts based on whether a configuration is chosen by $\mathsf{Opt}(J)$ or $S(R)$

$$\sum_{j=1}^{n}\sum_{c=1}^{q_j}\Big(\max_i X_{ij}^E(c)\Big) \cdot \mathbb{1}_{j\to c} = \sum_{j=1}^{n}\sum_{c=1}^{q_j}\Big(\max_i X_{ij}^E(c)\Big) \cdot \mathbb{1}_{j \xrightarrow{J} c} + \sum_{j=1}^{n}\sum_{c=1}^{q_j}\Big(\max_i X_{ij}^E(c)\Big) \cdot \mathbb{1}_{j \xrightarrow{R} c},$$

where we let $j \xrightarrow{J} c$ and $j \xrightarrow{R} c$ denote the events that configuration c is chosen for request j by $\mathsf{Opt}(J)$ up to the first too large configuration or up to and including the first exceptional configuration, or by $S(R)$, respectively.

We first bound the former term, corresponding to the configurations chosen in $\mathsf{Opt}(J)$. In case of event $\mathcal{L}$ or if $\mathsf{Opt}(J)$ is run to completion, we have $\sum_{j,c}\big(\max_i X_{ij}^E(c)\big) \cdot \mathbb{1}_{j \xrightarrow{J} c} = 0$. Otherwise, let $j^* \to c^*$ be the first (and only) exceptional configuration chosen by $\mathsf{Opt}(J)$. Then, $\sum_{j,c}\big(\max_i X_{ij}^E(c)\big) \cdot \mathbb{1}_{j \xrightarrow{J} c} = \max_i X_{ij^*}^E(c^*) \leq \mathsf{Opt}(J)$. Combining and taking expectations yields

$$\mathbb{E}\Big[\sum_{j=1}^{n}\sum_{c=1}^{q_j}\max_i X_{ij}^E(c) \cdot \mathbb{1}_{j \xrightarrow{J} c}\Big] \leq \mathbb{E}[\mathsf{Opt}(J)].$$

For the latter term, we condition again on the events $R = J'$ and apply the inductive hypothesis. All exceptional parts are defined with respect to the fixed threshold $\tau \geq 2 \cdot \mathbb{E}[\mathsf{Opt}(J)] \geq 2 \cdot \mathbb{E}[\mathsf{Opt}(J')]$ for $J' \subset J$. Therefore,

$$\mathbb{E}\Big[\sum_{j=1}^{n}\sum_{c=1}^{q_j}\big(\max_i X_{ij}^E(c)\big) \cdot \mathbb{1}_{j \xrightarrow{R} c}\Big] = \sum_{J' \subsetneq J}\mathbb{E}\Big[\sum_{j \in J'}\sum_{c=1}^{q_j}\big(\max_i X_{ij}^E(c)\big) \cdot \mathbb{1}_{j \xrightarrow{R} c} \,\Big|\, R = J'\Big] \cdot \mathbb{P}[R = J']$$

$$= \sum_{J' \subsetneq J} \mathbb{E}\Big[\sum_{j \in J'} \sum_{c=1}^{q_j} \big(\max_i X_{ij}^E(c) \big) \cdot \mathbb{1}_{j \xrightarrow{J'} c} \Big] \cdot \mathbb{P}[R = J']$$
$$\leq 2 \sum_{J' \subsetneq J} \mathbb{E}[\mathsf{Opt}(J')] \cdot \mathbb{P}[R = J']$$
$$\leq 2 \cdot \mathbb{E}[\mathsf{Opt}(J)] \cdot \mathbb{P}[R \neq \emptyset] \leq \mathbb{E}[\mathsf{Opt}(J)].$$

Note that we define $j \xrightarrow{J'} c$ to be the event that the policy $S(J')$ chooses configuration c for request j. In conclusion, by combining our bounds for these two terms we have

$$\mathbb{E}\Big[\sum_{j=1}^{n} \sum_{c=1}^{q_j} \big(\max_i X_{ij}(c)^E \big) \cdot \mathbb{1}_{j \to c} \Big] \leq 2\mathbb{E}[\mathsf{Opt}(J)].$$

To conclude, our constructed policy has expected maximum truncated load and total expected exceptional load both at most $2 \cdot \mathbb{E}[\mathsf{Opt}]$ (by the above calculations), and it never chooses a configuration with $\mathbb{E}\big[\max_i X_i(c)\big] > \tau$ (because we stop running $\mathsf{Opt}(J)$ right before it chooses such a configuration, and by induction we subsequently do not as well.) □

Having this near-optimal policy at hand, the upshot is that we can bound our subsequent algorithms with respect to the following LP relaxation ($\mathsf{LP_C}$) for configuration balancing with stochastic requests. The variable y_{cj} denotes selecting configuration c for request j. We take our threshold between the truncated and exceptional parts to be τ. Using the natural setting of the y-variables defined by the policy guaranteed by Theorem 4.1.8, it is straight-forward to show that the following LP relaxation is feasible, formalized in Lemma 4.2.1.

$$\begin{aligned}
\textstyle\sum_{c=1}^{q_j} y_{cj} &= 1 && \forall\, j \in [n] \\
\textstyle\sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}[X_{ij}^T(c)] \cdot y_{cj} &\leq \tau && \forall\, i \in [m] \\
\textstyle\sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}[\max_i X_{ij}^E(c)] \cdot y_{cj} &\leq \tau && \qquad\qquad (\mathsf{LP_C}) \\
y_{cj} &= 0 && \forall\, j \in [n], \forall\, c \in [q_j] : \mathbb{E}[\max_i X_{ij}(c)] > \tau \\
y_{cj} &\geq 0 && \forall\, j \in [n], \forall\, c \in [q_j]
\end{aligned}$$

Lemma 4.2.1. ($\mathsf{LP_C}$) *has a feasible solution for any* $\tau \geq 2 \cdot \mathbb{E}[\mathit{Opt}]$.

Proof. Consider the adaptive policy guaranteed by Theorem 4.1.8, and let the events $j \to c$ be with respect to this policy. We consider the natural setting of the y-variables given this policy: for all j and $c \in [q_j]$, we take $y_{cj} = \mathbb{P}(j \to c)$. It is clear that $\sum_{c=1}^{q_j} y_{cj} = 1$ for all j and $0 \leq y_{cj} \leq 1$ for all j and for all $c \in [q_j]$. Moreover, as by Theorem 4.1.8 the policy does not select a configuration c with $\mathbb{E}\big[\max_i X_i(c)\big] > \tau$, the pruning constraints $y_{cj} = 0$ if $\mathbb{E}\big[\max_i X_i(c)\big] > \tau$ are also satisfied.

It remains to verify the exceptional and truncated load constraints. For the exceptional constraint, we observe

$$\sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}\Big[\max_i X_{ij}^E(c) \Big] \cdot y_{cj} = \sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}\Big[\max_i X_{ij}^E(c) \cdot \mathbb{1}_{j \to c} \Big] \leq 2 \cdot \mathbb{E}[\mathsf{Opt}] \leq \gamma,$$

where in the first step we use the fact that the decision to choose configuration c for request j is independent of its realization, and in the second we use the properties of the policy. Similarly, for the truncated constraint for each i,

$$\sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}\big[X_{ij}^T(c) \big] \cdot y_{cj} = \sum_{j=1}^{n} \sum_{c=1}^{q_j} \mathbb{E}\big[X_{ij}^T(c) \cdot \mathbb{1}_{j \to c} \big] \leq 2 \cdot \mathbb{E}[\mathsf{Opt}] \leq \tau.$$

□

4.2.2 Offline Setting

Our offline algorithm is based on the natural randomized rounding of ($\mathsf{LP_C}$). For the truncated parts, we use the following maximal inequality to bound their contribution to the makespan. See Appendix B.1 for proof. The independence is only required for the random variables constituting a particular sum S_i, but is not necessary for random variables appearing in different sums.

Lemma 4.2.2. *Let $S_1, \dots, S_m$ be sums of independent random variables, that are bounded in $[0, \tau]$ for some $\tau > 0$, such that $\mathbb{E}[S_i] \le \tau$ for all $1 \le i \le m$. Then, $\mathbb{E}[\max_i S_i] = O\big(\frac{\log m}{\log\log m}\big)\tau$.*

To bound the contribution of the exceptional parts, we use (4.1) (i.e. the total expected exceptional load.) Using binary search for the correct choice of τ and re-scaling the instance down by the current value of τ, it suffices to give an efficient algorithm that either

- outputs a non-adaptive policy with expected makespan $O\big(\frac{\log m}{\log\log m}\big)$, or
- certifies that $\mathbb{E}[\mathsf{Opt}] > 1$.

This is because for $\tau \in \big(\mathbb{E}[\mathsf{Opt}], 2 \cdot \mathbb{E}[\mathsf{Opt}]\big]$, the re-scaling guarantees $\mathbb{E}[\mathsf{Opt}] \in [\frac{1}{2}, 1)$ on the scaled instance, in which case the algorithm achieves expected makespan $O\big(\frac{\log m}{\log\log m}\big) = O\big(\frac{\log m}{\log\log m}\big) \cdot \mathbb{E}[\mathsf{Opt}]$.

To that end, we use natural independent randomized rounding of ($\mathsf{LP_C}$). That is, if ($\mathsf{LP_C}$) has a feasible solution y^*, for request j, we choose configuration c as configuration c_j independently with probability y^*_{cj}; see Algorithm 8. If the configurations are given explicitly as part of the input,

Algorithm 8: Offline Configuration Balancing with Stochastic Requests

```
8.1 try to solve (LP_C) with τ = 2
8.2 if (LP_C) is feasible then
8.3     let y* be the outputted feasible solution
8.4     for each request j do
8.5         independently sample c ∈ [q_j] with probability y*_cj
8.6         choose sampled c as c_j
8.7 else
8.8     return "E[Opt] > 1"
```

then ($\mathsf{LP_C}$) can be solved in polynomial time and, thus, Algorithm 8 runs in polynomial time. Hence, the desired $O\big(\frac{\log m}{\log\log m}\big)$-approximate non-adaptive policy for configuration balancing with stochastic requests (Theorem 4.1.1) follows from the next lemma.

Lemma 4.2.3. *If ($\mathsf{LP_C}$) can be solved in polynomial time, Algorithm 8 is a polynomial-time randomized algorithm that either outputs a non-adaptive policy with expected makespan $O\big(\frac{\log m}{\log\log m}\big)$, or certifies correctly that $\mathbb{E}[\mathsf{Opt}] > 1$.*

Proof. We need to show that Algorithm 8 either outputs a non-adaptive policy with expected makespan $O\big(\frac{\log m}{\log\log m}\big)$ or certifies that $\mathbb{E}[\mathsf{Opt}] > 1$. There are two cases.

If ($\mathsf{LP_C}$) is feasible for $\tau = 2$, then we output a (randomized) non-adaptive policy. We show this policy has expected makespan $O\big(\frac{\log m}{\log\log m}\big)$. We let $j \to c$ denote the event that our algorithm chooses configuration c for request j. Thus, the truncated load on resource i can be written as

$$L_i = \sum_{j=1}^{n} \left(\sum_{c=1}^{q_j} X^T_{ij}(c) \cdot \mathbb{1}_{j \to c} \right).$$

Note that the random variables $\sum_{c=1}^{q_j} X_{ij}^T(c) \cdot \mathbb{1}_{j\to c}$ are independent for different j because the X_{ij} are and we sample the configuration c_j independently for each j. Further, they are bounded in $[0,2]$ by truncation. With the constraints of ($\mathsf{LP_C}$), we can bound the expectation by

$$\mathbb{E}[L_i] = \sum_{j=1}^{n}\sum_{c=1}^{q_j} \mathbb{E}\Big[X_{ij}^T(c)\cdot \mathbb{1}_{j\to c}\Big] = \sum_{j=1}^{n}\sum_{c=1}^{q_j} \mathbb{E}\Big[X_{ij}^T(c)\Big]\cdot y_{cj}^* \le 2,$$

where we used the independence of $\mathbb{1}_{j\to c}$ and $X_{ij}(c)$ in the second equality. By Lemma 4.2.2, $\mathbb{E}[\max_i L_i] = O\big(\frac{\log m}{\log\log m}\big)$. Using (4.1) we upper bound the total expected exceptional load by

$$\mathbb{E}\Big[\max_{1\le i\le m}\sum_{j=1}^{n}\sum_{c=1}^{q_j} X_{ij}^E(c)\cdot\mathbb{1}_{j\to c}\Big] \le \sum_{j=1}^{n}\sum_{c=1}^{q_j}\mathbb{E}\Big[\max_{1\le i\le m} X_{ij}^E(c)\cdot\mathbb{1}_{j\to c}\Big] = \sum_{j=1}^{n}\sum_{c=1}^{q_j}\mathbb{E}\Big[\max_{1\le i\le m} X_{ij}^E(c)\Big]\cdot y_{cj}^* \le 2.$$

Combining our bounds for the truncated and exceptional parts completes the proof. The expected makespan of our algorithm is given by

$$\mathbb{E}\Big[\max_{1\le i\le m}\sum_{j=1}^{n}\sum_{c=1}^{q_j} X_{ij}(c)\cdot\mathbb{1}_{j\to c}\Big] \le \mathbb{E}\Big[\max_{1\le i\le m} L_i\Big] + \mathbb{E}\Big[\max_{1\le i\le m}\sum_{j=1}^{n}\sum_{c=1}^{q_j} X_{ij}^E(c)\cdot\mathbb{1}_{j\to c}\Big] = O\Big(\frac{\log m}{\log\log m}\Big).$$

In the other case ($\mathsf{LP_C}$) is infeasible, so the contrapositive of Lemma 4.2.1 gives $\mathbb{E}[\mathsf{Opt}] > 1$. □

4.2.3 Online Setting

We now consider online configuration balancing where n stochastic requests arrive online one-by-one, and for each request, one configuration has to be irrevocably selected before the next request appears. We present a non-adaptive online algorithm that achieves a competitive ratio of $O(\log m)$, which is best possible due to the lower bound of $\Omega(\log m)$ on the competitive ratio of any deterministic algorithm for online load balancing on unrelated machines [ANR95].

First, by a standard guess-and-double scheme, we may assume we have a good guess of $\mathbb{E}[\mathsf{Opt}]$.

Lemma 4.2.4. *Given an instance of online configuration balancing with stochastic requests, suppose there exists an online algorithm that, given parameter $\lambda > 0$, never creates an expected makespan more than $\alpha\cdot\lambda$, possibly terminating before handling all requests. Further, if the algorithm terminates prematurely, then it certifies that $\mathbb{E}[\mathsf{Opt}] > \lambda$. Then, there exists an $O(\alpha)$-competitive algorithm for online configuration balancing with stochastic requests. Further, the resulting algorithm preserves non-adaptivity.*

We omit the proof, which is analogous to its virtual-circuit-routing counterpart in [AAF+97].

We will build on the same technical tools as in the offline case. In particular, we wish to compute a non-adaptive assignment online with small expected truncated load on each resource and small total expected exceptional load. To achieve this, we generalize the greedy potential function approach of [AAF+97]. Our two new ingredients are to treat the exceptional parts of a request's configuration as a resource requirement for an additional, artificial resource and to compare the potential of our solution directly with a *fractional* solution to ($\mathsf{LP_C}$).

Now we describe our potential function, which is based on an exponential/soft-max function. Let λ denote the current guess of the optimum as required by Lemma 4.2.4. We take $\tau = 2\lambda$ as our

truncation threshold. Given load vector $L \in \mathbb{R}^{m+1}$, our potential function is

$$\phi(L) = \sum_{i=0}^{m} (3/2)^{L_i/\tau}.$$

For $i = 1, \ldots, m$, we will ensure the ith entry of L is the *expected* truncated load on resource i. We use the 0th entry as a virtual resource that is the total expected exceptional load. For any request j, we let L_{ij} be the ith entry of the expected load vector after handling the first j requests. We define $L_{i0} := 0$ for all i. Let L_j be the expected load vector after handling the first j requests.

Algorithm 9 works as follows: Upon arrival of request j, we try to choose the configuration $c_j \in [q_j]$ that minimizes the increase in potential. Concretely, we choose c_j to minimize

$$\left((3/2)^{(L_{0j-1}+\mathbb{E}[\max_{i\in[m]} X_{ij}^E(c_j)])/\tau} + \sum_{i=1}^{m} (3/2)^{(L_{ij-1}+\mathbb{E}[X_{ij}^T(c_j)])/\tau}\right) - \phi(L_{j-1}).$$

Algorithm 9: Online Configuration Balancing with Stochastic Requests

```
9.1  ℓ ← log_{3/2}(2m + 2)
9.2  λ ← current guess of 𝔼[Opt]
9.3  τ ← 2λ truncation threshold
9.4  upon arrival of request j do
9.6      c_j ← arg min_{c∈[q_j]} ((3/2)^{(L_{0j-1}+𝔼[max_{i∈[m]} X^E_{ij}(c)])/τ} + Σ_{i=1}^m (3/2)^{(L_{ij-1}+𝔼[X^T_{ij}(c)])/τ}) − φ(L_{j-1})
9.7      if L_{ij-1} + 𝔼[X^T_{ij}(c_j)] ≤ ℓτ for all i ∈ [m] and L_{0j-1} + 𝔼[max_{i∈[m]} X^E_{ij}(c_j)] ≤ ℓτ then
9.8          choose c_j for j
9.9          L_{ij} ← L_{ij-1} + 𝔼[X^T_{ij}(c_j)] for all i ∈ [m]
9.10         L_{0j} ← L_{0j-1} + 𝔼[max_{i∈[m]} X^E_{ij}(c_j)]
9.11     else
9.12         return "𝔼[Opt] > λ"
```

To analyze this algorithm, we compare its makespan with a solution to ($\mathsf{LP_C}$). This LP has an integrality gap of $\Omega\left(\frac{\log m}{\log\log m}\right)$, which follows immediately from the path assignment LP for virtual circuit routing [LRS98]. Hence, a straightforward analysis of Algorithm 9 comparing to a rounded solution to ($\mathsf{LP_C}$) gives an assignment with expected truncated load per machine and total expected exceptional load $O\left(\log m \cdot \frac{\log m}{\log\log m}\right) \cdot \mathbb{E}[\mathsf{Opt}]$. The $O\left(\frac{\log m}{\log\log m}\right)$ factor is due to the aforementioned integrality gap while the second factor $O(\log m)$ is due to the use of the potential function. To get a tight competitive ratio of $O(\log m)$, we avoid the integrality gap by comparing to a *fractional* solution to ($\mathsf{LP_C}$), and we use a slightly different maximal inequality for the regime where the mean of the sums is larger than the increments by at most a $O(\log m)$-factor. The particular maximal inequality is the following, which we prove in Appendix B.1.

Lemma 4.2.5. *Let $S_1, \ldots, S_m$ be sums of independent random variables bounded in $[0, \tau]$ such that $\mathbb{E}[S_i] \le O(\log m)\tau$ for all $1 \le i \le m$. Then, $\mathbb{E}[\max_i S_i] \le O(\log m)\tau$.*

Now we give our main guarantee for Algorithm 9, which implies the desired $O(\log m)$-competitive online algorithm for configuration balancing with stochastic requests.

Lemma 4.2.6. *Suppose the minimizing configuration in Line 9.6 can be found in polynomial time. Then Algorithm 9 runs in polynomial time. Further, Algorithm 9 is a deterministic, non-adaptive algorithm that correctly solves the subproblem of Lemma 4.2.4 for $\alpha = O(\log m)$.*

Proof. The running time of Algorithm 9 is clear. It remains to show that the algorithm creates expected makespan at most $O(\log m)\lambda$ or correctly certifies $\mathbb{E}[\mathsf{Opt}] > \lambda$ if it terminates prematurely.

We first show the former. By definition, upon termination of the algorithm after, say n' requests, the expected truncated load on each resource and total expected exceptional load are both at most $O(\log m) \cdot \lambda$. Let the configuration choices c_j be with respect to our algorithm. We can split the makespan into the contributions by the truncated and exceptional parts.

$$\mathbb{E}\left[\max_{1\le i\le m}\sum_{j=1}^{n'} X_{ij}(c_j)\right] \le \mathbb{E}\left[\max_{1\le i\le m}\sum_{j=1}^{n'} X^T_{ij}(c_j)\right] + \mathbb{E}\left[\max_{1\le i\le m}\sum_{j=1}^{n'} X^E_{ij}(c_j)\right].$$

Each truncated part is bounded in $[0, 2\lambda]$ and each resource has expected truncated load at most $O(\log m)\lambda$. By Lemma 4.2.5, we can bound the contribution of the truncated parts by $\mathbb{E}\big[\max_{1\le i\le m}\sum_{j=1}^{n'} X^T_{ij}(c_j)\big] = O(\log m)\lambda$. For the exceptional parts, applying (4.1) gives

$$\mathbb{E}\left[\max_{1\le i\le m}\sum_{j=1}^{n} X^E_{ij}(c_j)\right] \le \mathbb{E}\left[\sum_{j=1}^{n}\max_{1\le i\le m} X^E_{ij}(c_j)\right] \le O(\log m)\lambda.$$

Combining both bounds gives that the expected makespan is at most $O(\log m)\lambda$, as required.

It remains to show that if $\mathbb{E}[\mathsf{Opt}] \le \lambda$, then the algorithm successfully assigns all requests. We do so by bounding the increase in the potential function. Note that if $\mathbb{E}[\mathsf{Opt}] \le \lambda$, then ($\mathsf{LP_C}$) is feasible for our choice of $\tau = 2\lambda$ by Lemma 4.2.1. Thus, let (y_{cj}) be a feasible solution to ($\mathsf{LP_C}$). For simplicity, let $x_{0j}(c) := \mathbb{E}[\max_{1\le i\le m} X^E_{ij}(c)]$ be the exceptional part of configuration c and $x_{ij}(c) := \mathbb{E}[X^T_{ij}(c)]$ its truncated part on resource i.

For each request j, as Algorithm 9 chooses a configuration c_j minimizing the increase in Φ,

$$\phi(L_{j-1} + x_j(c_j)) - \phi(L_{j-1}) \le \phi(L_{j-1} + x_j(c)) - \phi(L_{j-1})$$

for all configurations $c \in [q_j]$. As $\sum_{c=1}^{q_j} y_{cj} = 1$ by feasibility of (y_{cj}), this implies

$$\phi(L_{j-1} + x_j(c_j)) - \phi(L_{j-1}) \le \sum_{c=1}^{q_j} y_{cj}\phi(L_{j-1} + x_j(c)) - \phi(L_{j-1}). \tag{4.2}$$

We recall that $L_0 = 0 \in \mathbb{R}^{m+1}$. We bound the increase in potential incurred by Algorithm 9:

$$\begin{aligned}
\phi(L_n) - \phi(L_0) &= \sum_{j=1}^{n}\sum_{i=0}^{m}(3/2)^{L_{ij-1}/\tau}\Big((3/2)^{x_{ij}(c_j)/\tau} - 1\Big) \\
&\le \sum_{j=1}^{n}\sum_{c=1}^{q_j} y_{cj}\sum_{i=0}^{m}(3/2)^{L_{ij-1}/\tau}\Big((3/2)^{x_{ij}(c)/\tau} - 1\Big) \\
&\le \sum_{i=0}^{m}(3/2)^{L_{in}/\tau}\sum_{j=1}^{n}\sum_{c=1}^{q_j} y_{cj}\Big((3/2)^{x_{ij}(c)/\tau} - 1\Big),
\end{aligned}$$

where the first inequality holds due to (4.2), and the second inequality holds because the load on machine i only increases over time. Standard estimates of e^z give $(3/2)^z - 1 \leq (1/2)z$ for $z \in [0,1]$. As y_{cj} is feasible, we know that $x_{ij}(c) \leq \tau$ for all c with $y_{cj} > 0$. Hence,

$$\phi(L_n) - \phi(L_0) \leq \sum_{i=0}^{m} (3/2)^{L_{in}/\tau} \sum_{j=1}^{n} \sum_{c=1}^{q_j} y_{cj} \frac{x_{ij}(c)}{2\tau}$$

Using that y_{cj} is feasible and satisfies $\sum_{j=1}^{n} \sum_{c=1}^{q_j} x_{ij}(c) \cdot y_{cj} \leq \tau$ for resource $i = 0$ by the exceptional constraint and for all resources $i = 1, \ldots, m$ by the truncated constraints, we get

$$\phi(L_n) - \phi(L_0) \leq (1/2) \sum_{i=0}^{m} (3/2)^{L_{in}/\tau} = (1/2)\phi(L_n).$$

After rearranging, we have $\phi(L_n) \leq 2\phi(L_0) = 2(m+1)$ by choice of L_0. Taking logarithms and using that $\log_{3/2}(z)$ is monotonically increasing, we conclude that $\max_{0 \leq i \leq m} L_{in} \leq \log_{3/2}(2m+2)\tau$. Note that we chose $\ell = \log_{3/2}(2m+2)$ implying that Algorithm 9 never fails if $\mathbb{E}[\mathsf{Opt}] \leq \lambda$. □

Summary

Our main theorem for configuration balancing follows from the above two algorithms.

Theorem 4.1.1. *For configuration balancing with stochastic requests there is a randomized* offline *algorithm that computes a non-adaptive policy that is a* $O\big(\frac{\log m}{\log \log m}\big)$*-approximation and a deterministic* online *algorithm that is a* $O(\log m)$*-approximation when comparing to the optimal offline adaptive policy. Both algorithms run in polynomial time in the number of resources and the total number of configurations over all requests.*

Proof. Note that the configurations are given explicitly, implying that both, Algorithms 8 and 9, run in polynomial time. Hence, Lemma 4.2.3 gives the $O(\frac{\log m}{\log \log m})$-approximate offline algorithm and Lemmas 4.2.4 and 4.2.6 give the $O(\log m)$-approximate online algorithm. □

4.3 Unrelated Load Balancing and Virtual Circuit Routing

In this section, we apply our algorithms for configuration balancing to stochastic load balancing on unrelated machines (Theorem 4.1.2) as well as stochastic virtual circuit routing (Theorem 4.1.5).

4.3.1 Unrelated Load Balancing with Stochastic Jobs

We recall that in load balancing on unrelated machines with stochastic jobs, we have m machines and n jobs such that the size of job j on machine i is a random variable X_{ij}. These X_{ij}s are independent across jobs. This is a special case of configuration balancing with stochastic requests by taking m resources (corresponding to the m machines) and n requests such that each request j has m possible configurations, one for each machine choice job j has. Precisely, we define the configurations $c = 1, \ldots, m$ for job j by setting

$$X_{ij}(c) = \begin{cases} X_{cj} & \text{if } i = c, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Theorem 4.1.2. *There exist efficient deterministic algorithms that compute a non-adaptive policy for load balancing on unrelated machines with stochastic jobs that achieve an* $\Theta\big(\frac{\log m}{\log \log m}\big)$*-approximation* offline *and an* $\Theta(\log m)$*-approximation* online *when comparing to the optimal offline adaptive policy.*

Proof. Each request has m possible configurations, so the total size of the resulting configuration balancing instance is polynomial. Thus, we may assume the configurations are given explicitly. Hence, Theorem 4.1.1 immediately gives a *randomized* $O(\frac{\log m}{\log \log m})$-approximation offline and $O(\log m)$-approximation online for load balancing on unrelated machines with stochastic jobs.

However, to obtain a deterministic offline algorithm, we can de-randomize Algorithm 8 for this special case because here, ($\mathsf{LP_C}$) is equivalent to the generalized assignment LP considered by Shmoys and Tardos, which has a constant-factor rounding algorithm [ST93]. □

4.3.2 Routing with Stochastic Demands

Virtual circuit routing is another special case of configuration balancing: Given any instance of the former, which consists of a directed graph with m edges and n source-sink pairs, the resulting configuration balancing instance has m resources, one for each edge, and n requests, one for each source-sink pair, such that there is one configuration per source-sink path in the graph.

Note that unlike load balancing, where a request has at most m configurations, for routing, a request can have exponentially many configurations. Thus, to obtain polynomial-time algorithms for routing, we cannot explicitly represent all configurations. In particular, to prove Theorem 4.1.5 in the offline setting, we need to show how to solve ($\mathsf{LP_C}$) efficiently in the special case of routing, and to prove Theorem 4.1.5 in the online setting, we need to show how to find the configuration (i.e. the path) that minimizes the increase in the potential Φ.

Offline Routing: Solving ($\mathsf{LP_C}$)

We first re-write ($\mathsf{LP_C}$) for routing. Recall that τ is the truncation threshold. For a request j, let E_j be the set of all edges with $\mathbb{E}[X_{ej}] \leq \tau$ and let $\mathcal{P}_j$ be the collection of all s_j-t_j paths in the auxiliary graph $G_j = (V, E_j)$. For each request j and path $P \in \mathcal{P}_j$, the variable y_{Pj} denotes the decision to route request j along path P. Thus, we want to solve the following path assignment LP.

$$\begin{aligned} \textstyle\sum_{P \in \mathcal{P}_j} y_{Pj} &= 1 && \forall\, j \in [n] \\ \textstyle\sum_{j=1}^{n} \sum_{P \in \mathcal{P}_j} \mathbb{E}[X^T_{ej}] \cdot y_{Pj} &\leq \tau && \forall\, e \in E \\ \textstyle\sum_{j=1}^{n} \sum_{P \in \mathcal{P}_j} \mathbb{E}[\max_{e \in P} X^E_{ej}] \cdot y_{Pj} &\leq \tau \\ y_{Pj} &\geq 0 && \forall\, j \in [n],\ P \in \mathcal{P}_j \end{aligned} \tag{$\mathsf{LP_P}$}$$

To see that ($\mathsf{LP_P}$) is equivalent to ($\mathsf{LP_C}$), note that the pruning constraints of the latter ensure that any configuration/path P with $\mathbb{E}\big[\max_{e \in P} X_{ej}\big] > \tau$ will not be selected. Re-writing gives $\mathbb{E}\big[\max_{e \in P} X_{ej}\big] = \mathbb{E}\big[\max_{e \in P} \frac{1}{c_e} X_j\big] = \max_{e \in P} \mathbb{E}[X_{ej}]$. Thus, the pruning constraint ensures that no edge with $\mathbb{E}[X_{ej}] > \tau$ will be selected. This is exactly encoded by the set of feasible paths $\mathcal{P}_j$.

Note that ($\mathsf{LP_P}$) has an exponential number of variables. For classical congestion minimization problems, the path-assignment LP formulation is equivalent to a flow formulation [LRS98], which can be solved optimally in polynomial time using results about flows in networks. In our case, however, we additionally have the third constraint (the exceptional load constraint) in the LP, which does not allow for a straight-forward equivalent flow formulation. Therefore, we use LP duality in order to solve it efficiently: We give a separation oracle for its dual LP. For obtaining the dual, we add the trivial objective to maximize $0^T y$ to ($\mathsf{LP_P}$). Hence, the dual of ($\mathsf{LP_P}$) is

$$\begin{aligned} & \min \textstyle\sum_j a_j + \sum_e b_e \cdot \tau + c \cdot \tau & & \\ s.t. \quad & a_j + \textstyle\sum_{e \in P} b_e \cdot \mathbb{E}[X_{ej}^T] + c \cdot \mathbb{E}[\max_{e \in P} X_{ej}^T] & \geq 0 \qquad & \forall j \in [n],\ P \in \mathcal{P}_j \\ & b_e & \geq 0 \qquad & \forall\ e \in E \\ & c & \geq 0. & \end{aligned} \tag{$\mathsf{D_P}$}$$

For solving ($\mathsf{D_P}$), consider a request j and a path $P \in \mathcal{P}_j$. The expected exceptional part of routing j along P is $\mathbb{E}\big[\max_{e \in P} X_{ej}^E\big] = \max_{e \in P} \mathbb{E}\big[\big(X_j/c_e\big)^E\big]$, which is the expected exceptional part of the smallest-capacity edge $\bar{e}_j$ along the path. Given this edge $\bar{e}_j$, a particular choice of the dual variables is feasible for the first constraint of ($\mathsf{D_P}$) if and only if

$$\min_{P \in \mathcal{P}_j \,:\, \min_{e \in P} c_e \geq c_{\bar{e}_j}} \left(\sum_{e \in P} b_e \cdot \mathbb{E}\big[X_{ej}^T\big] + c \cdot \mathbb{E}\big[\max_{e \in P} X_{ej}^T\big] \right) \geq -a_j \quad \forall\ j.$$

Hence, for each request j, it remains to find a path $P \in \mathcal{P}_j$ that is minimal w.r.t. edge weights $b_e \cdot \mathbb{E}[X_{ej}^T]$. By letting every edge $\bar{e}$ be the smallest-capacity edge and removing any smaller-capacity edges from the graph, this becomes a shortest s_j-t_j path problem.

Algorithm 10: Separation oracle for ($\mathsf{D_P}$)

```
10.1  (a, b, c) ← current solution of (D_P)
10.2  if b_e < 0 for some e ∈ E or c < 0 then
10.3      return the violated non-negativity constraint
10.4  else
10.5      for j and edge ē ∈ E_j do
10.6          E_ē ← {e ∈ E_j : c_e ≥ c_ē}
10.7          G_ē ← (V, E_ē)
10.8          P_jē ← shortest s_j-t_j path in G_ē w.r.t. edge weights b_e · E[X_ej^T]
10.9          if Σ_{e∈P_jē} b_e · E[X_ej^T] + c · E[max_{e∈P_jē} X_ej^T] < −a_j then
10.10             return the separating hyperplane Σ_{e∈P_jē} b_e · E[X_ej^T] + c · E[max_{e∈P_jē} X_ej^T] ≥ −a_j
10.11     return "feasible"
```

Lemma 4.3.1. *Algorithm 10 is a polynomial time separation oracle for* ($\mathsf{D_P}$).

Proof. Since shortest paths can be found in polynomial time [Bel58], Algorithm 10 indeed runs in polynomial time. Further, if there is an edge $e \in E$ with $b_e < 0$ or if $c < 0$, then Algorithm 10 correctly outputs the violated constraint.

It remains to consider the case when $b, c \geq 0$. In this case, we need to show for each request j that we find some $\bar{e} \in E$ such that $P_{j\bar{e}}$ achieves $\min_{P \in \mathcal{P}_j} \big(\sum_{e \in P} b_e \cdot \mathbb{E}[X_{ej}^T] + c \cdot \mathbb{E}[\max_{e \in P} X_{ej}^T] \big)$.

Consider any request j such that the minimum is achieved by some s_j-t_j path P^*. Let e^* be the smallest-capacity edge along P^*. We claim for the correct guess $\bar{e} = e^*$, $P_{j\bar{e}}$ achieves the minimum. To see this, observe that P^* is a s_j-t_j path in the graph $G_{\bar{e}}$, so the algorithm will choose $P_{j\bar{e}}$ with $\sum_{e \in P_{j\bar{e}}} b_e \cdot \mathbb{E}[X_{ej}^T] \leq \sum_{e \in P^*} b_e \cdot \mathbb{E}[X_{ej}^T]$ by definition of the edge weights. Further, we have $c \cdot \mathbb{E}[\max_{e \in P_{j\bar{e}}} X_{ej}^T] \leq c \cdot \mathbb{E}[\max_{e \in P^*} X_{ej}^T]$, because the latter maximum is achieved by edge $\bar{e}$, and by definition of the residual graph, $P_{j\bar{e}}$ cannot use any edges with smaller capacity than $\bar{e}$. Combining these two bounds shows that $P_{j\bar{e}}$ achieves the minimum. This in turn implies that Algorithm 10 returns a constraint violated by (a, b, c) if such a constraint exists. □

Online Routing: Minimizing Increase in Φ

For online virtual circuit routing, we assume that a sequence of source sink-pairs (s_j, t_j) for $j = 1, \ldots, n$ arrive online. To implement Algorithm 9 efficiently, given a load vector $L = (L_0, \ldots, L_m)$ and source-sink pair (s_j, t_j) with random demand $X_j \geq 0$, we need to choose a s_j-t_j path in G that minimizes the increase in Φ with respect to some fixed truncation threshold τ. Recall that we index the edges of G by $1, \ldots, m$, while L_0 is the load of an additional, artificial resource that captures the total expected exceptional load.

As in the offline setting, the expected exceptional part of the configuration corresponding to choosing a particular s_j-t_j path is the expected exceptional part of the smallest-capacity edge along the path. Thus, the increase in potential due to choosing a path P is

$$\left((3/2)^{(L_0+(\max_{e\in P}\mathbb{E}[X_{ej}^E])/\tau} - (3/2)^{L_0/\tau}\right) + \sum_{e\in P}\left((3/2)^{(L_e+\mathbb{E}[X_{ej}^T])/\tau} - (3/2)^{L_e/\tau}\right).$$

The first term is the increase due to the exceptional part (the smallest-capacity edge along P), and the remaining terms are the per-edge contributions due to the truncated parts. Analogously to the previous section, we only consider edges in $E_j = \{e \in E : \mathbb{E}[X_{ej}] \leq \tau\}$, guess the smallest-capacity edge, and solve a shortest s_j-t_j path problem to find the minimizing path; see Algorithm 11.

Algorithm 11: Minimizing increase in Φ for virtual circuit routing

11.1 $L \leftarrow$ current load vector
11.2 **for** $\bar{e} \in E_j$ **do**
11.3 $\quad E_{\bar{e}} \leftarrow \{e \in E_j : c_e \geq c_{\bar{e}}\}$
11.4 $\quad G_{\bar{e}} \leftarrow (V, E_{\bar{e}})$
11.5 $\quad P_{j\bar{e}} \leftarrow$ shortest s_j-t_j path in $G_{\bar{e}}$ w.r.t. edge weights $\left((3/2)^{(L_e+\mathbb{E}[X_{ej}^T])/\tau} - (3/2)^{L_e/\tau}\right)$
11.6 $P_j \leftarrow \arg\min_{P_{j\bar{e}}}$ increase in Φ

Lemma 4.3.2. *Given a request j and load vector L, Algorithm 11 returns in polynomial time a path P_j that minimizes the increase in Φ.*

Proof. It is clear that Algorithm 11 runs in polynomial time. To see that it also finds a path that minimizes the increase in Φ, let P_j^* be such an optimal path with smallest-capacity edge e^*. For the correct guess $\bar{e} = e^*$, P_j^* is a s_j-t_j path in the graph $G_{\bar{e}}$. Thus, Algorithm 11 chooses $P_{j\bar{e}}$ such that the per-edge contributions due to the truncated parts of $P_{j\bar{e}}$ are at most those due to P_j^* by definition of the edge weights. Further, the exceptional part of $P_{j\bar{e}}$ is at most that of P_j^*, because $P_{j\bar{e}}$ does not use any edges with capacity smaller than $c_{\bar{e}}$ by definition of the graph $G_{\bar{e}}$. We conclude that $P_{j\bar{e}}$ is also a s_j-t_j path that minimizes the increase in Φ. □

Summary

Combining the above two algorithms for the offline and online cases allow us to efficiently optimize over the possibly exponentially many configurations in stochastic virtual circuit routing.

Theorem 4.1.5. *For virtual circuit routing with stochastic requests, there exist an efficient randomized* offline *algorithm computing a non-adaptive policy that is a $\Theta\left(\frac{\log m}{\log\log m}\right)$-approximation and an efficient deterministic* online *algorithm that computes an $\Theta(\log m)$-approximation when comparing to the optimal offline adaptive policy.*

Proof. For offline virtual circuit routing, Lemma 4.3.1 guarantees that ($\mathsf{LP_P}$) can be solved optimally in polynomial time by LP duality. Thus, Lemma 4.2.3 implies that Algorithm 8 runs in polynomial time and achieves a maximum congestion of $O\big(\frac{\log m}{\log\log m}\big)\mathbb{E}[\mathsf{Opt}]$.

For the online problem, Lemma 4.3.2 guarantees that, for each request j, a path P_j that minimizes the increase in the potential function Φ is found in polynomial time. Thus, Lemma 4.2.3 implies that Algorithm 9 runs in polynomial time and guarantees a maximum congestion of $O(\log m)\mathbb{E}[\mathsf{Opt}]$. □

4.4 Load Balancing on Related Machines

In this section, we improve on Theorem 4.1.2 in the special case of related machines, where each machine i has a speed parameter $s_i > 0$ and each job j an independent size X_j such that $X_{ij} = \frac{X_j}{s_i}$. Recall that we gave a non-adaptive $O\big(\frac{\log m}{\log\log m}\big)$-approximation for unrelated machines. However, the adaptivity gap is $\Omega\big(\frac{\log m}{\log\log m}\big)$ even for load balancing on identical machines where every machine has the same speed. Thus, to improve on Theorem 4.1.2, we need to use adaptivity.

The starting point of our improved algorithms is the same non-adaptive assignment for unrelated-machine load balancing. However, instead of non-adaptively assigning a job j to the specified machine i, we adaptively assign j to the least loaded machine with similar speed to i. In the first part, we formalize this idea and then we focus on offline and online load balancing on related machines.

4.4.1 Machine Smoothing

In this section, we define a notion of *smoothed machines*. We show that by losing a constant factor in the approximation ratio, we may assume that the machines are partitioned into at most $O(\log m)$ groups such that machines within a group have the same speed and the size of the groups shrinks geometrically. Thus, by "machines with similar speed to i," we mean machines in the same group.

Formally, we would like to transform an instance $\mathcal{I}$ of load balancing on m related machines with stochastic jobs into an instance $\mathcal{I}_s$ with so-called "smoothed machines" and the same set of jobs with the following three properties:

(i) The machines are partitioned into $m' = O(\log m)$ groups such that group k consists of m_k machines with speed exactly s_k such that $s_1 < s_2 < \cdots < s_{m'}$.

(ii) For all groups $1 \le k < m'$, we have $m_k \ge \frac{3}{2} m_{k+1}$.

(iii) $\mathsf{Opt}(\mathcal{I}_s) = O(\mathsf{Opt}(\mathcal{I}))$.

To this end, we suitably decrease machine speeds and delete machines from the original instance $\mathcal{I}$. Our algorithm is the following.

We show that this algorithm creates the desired smoothed machines instance.

Lemma 4.4.1. *Given an instance $\mathcal{I}$ of load balancing with m related machines and stochastic jobs, Algorithm 12 efficiently computes an instance $\mathcal{I}_s$ of smoothed machines with the same set of jobs satisfying Properties (i) to (iii).*

Proof sketch (full proof in Appendix B.2). It is clear that the algorithm is efficient and outputs $\mathcal{I}_s$ satisfying (i) and (ii). For showing (iii), we analyze the increase of the cost of Opt due to each step. For Step 12.6, we schedule the jobs assigned by Opt to deleted machines on the fastest machine,

Algorithm 12: Machine Smoothing

```
12.1  s_max ← max_i s_i
12.2  for i = 1 to m do
12.3      s_i ← s_i / s_max
12.4      if s_i ≤ 1/m then
12.6          delete machine i
12.7      else
12.9          s_i ← 2^⌊log s_i⌋
12.10 partition machines by speeds such that group k has m_k machines of speed s_k
12.11 index the groups in order of increasing speed
12.12 for k = 1 to ⌈log m⌉ do
12.13     if m_k < (3/2) m_{k+1} then
12.15         delete group k
```

increasing its load by at most $(m-1)\frac{1}{m}\mathsf{Opt}$. Step 12.9 increases Opt by at most a factor 2. For Step 12.15, we schedule all jobs assigned by Opt to a deleted machine on machines of the next faster remaining group, following a fixed mapping of the deleted machines to the remaining machines. Because of (ii), we can bound the increase in load on each remaining machine by $O(\mathsf{Opt})$. □

To summarize, by losing a constant-factor in our final approximation ratio, we may assume we are working with an instance of smoothed machines. Looking ahead, if our non-adaptive policy assigns job j to machine i, then we instead adaptively assign j to the least-loaded machine in the group containing machine i. We will use the properties of smoothed machines to show that this leads to a $O(1)$-approximation offline and $O(\log\log m)$-approximation online.

A similar idea for machine smoothing has been employed by Im et al. [IKPS18] for deterministic load balancing on related machines. In their approach, they ensure that the *total processing power* of the machines in a group decreases geometrically rather than the number of machines.

4.4.2 Offline Setting

In this section, we give our $O(1)$-approximate adaptive policy for load balancing on related machines with stochastic jobs. Our algorithm has three parts: machine smoothing (Algorithm 12), non-adaptive assignment (Algorithm 8), and adaptively turning a job-to-machine assignment into a coarser job-to-group assignment. Precisely, our algorithm is the following.

Algorithm 13: Offline Related Load Balancing

```
13.1 smooth machines with Algorithm 12
13.2 create configuration balancing instance as in (4.3)
13.3 obtain assignment of jobs to machines by Algorithm 8
13.4 for j ∈ J do
13.5     i ← machine of j
13.6     schedule j on least loaded machine i' with s_i = s_{i'}
```

Note that as in the case of unrelated machine load balancing (Theorem 4.1.2), we can derandomize

this algorithm by employing the GAP LP rounding algorithm by [ST93].

We show that this algorithm gives the desired $O(1)$-approximation. Note that our previous analysis of Algorithm 8 gave a $O(\frac{\log m}{\log \log m})$-approximation. We improve on this using the properties of smoothed machines and our adaptive decisions. We give a stronger maximal inequality using the fact that group sizes are geometrically increasing. See Appendix B.1 for proof of the following.

Lemma 4.4.2. *Let $c_1, \dots, c_m \in \mathbb{N}_{\geq 1}$ be constants such that $c_i \geq \frac{3}{2} c_{i+1}$ for all $1 \leq i \leq m$. Let $S_1, \dots, S_m$ be sums of independent random variables bounded in $[0, \tau]$ such that $\mathbb{E}[S_i] \leq c_i \tau$ for all $1 \leq i \leq m$. Then, $\mathbb{E}\big[\max_i \frac{S_i}{c_i}\big] \leq O(\tau)$.*

Now we analyze Algorithm 13.

Lemma 4.4.3. *For offline load balancing on related machines with stochastic jobs, Algorithm 13 efficiently either outputs an adaptive policy with expected makespan $O(1)$ or certifies $\mathbb{E}[\mathsf{Opt}] > 1$.*

Proof. It is clear that the algorithm runs in polynomial time. By the contrapositive of Lemma 4.2.1, if ($\mathsf{LP_C}$) is infeasible for $\tau = 2$, then we correctly certify $\mathbb{E}[\mathsf{Opt}] > 1$. Thus, it remains to consider the case where the LP is feasible.

In this case, we obtain a job-to-machine assignment with expected truncated load at most 2 on every machine and expected exceptional load at most 2. Summing up the truncated loads within each group, we have that group k has expected truncated load at most $2m_k$. Again, we split the makespan of our policy into truncated and exceptional parts

$$\mathbb{E}\left[\max_{1 \leq i \leq m} \sum_{j \to i} X_{ij}\right] \leq \mathbb{E}\left[\max_{1 \leq i \leq m} \sum_{j \to i} X_{ij}^T\right] + \mathbb{E}\left[\max_{1 \leq i \leq m} \sum_{j \to i} X_{ij}^E\right],$$

where the events $j \to i$ are with respect to our final adaptive assignment. We can upper bound the contribution of the exceptional parts (the latter term) by 2 using (4.1). It remains to bound the contribution of truncated parts. We do so by considering the makespan on each *group*. For group k, we let $j \to k$ denote the event that we non-adaptively assign job j to a machine in group k, and X_{kj} be the size of job j on any machine in group k (recall that they all have the same speed). Then,

$$\mathbb{E}\left[\max_{1 \leq i \leq m} \sum_{j \to i} X_{ij}^T\right] = \mathbb{E}\left[\max_k \max_{i \in G_k} \sum_{j \to i} X_{ij}^T\right],$$

where G_k is the collection of all machines in group k. Similar to the analysis of list scheduling by [Gra69], i.e, by an averaging argument, we obtain

$$\max_{i \in G_k} \sum_{j \to i} X_{ij}^T \leq \frac{1}{m_k} \sum_{j \to k} X_{kj}^T + \max_{j \to k} X_{kj}^T \leq \frac{1}{m_k} \sum_{j \to k} X_{kj}^T + 2. \tag{4.4}$$

The expected truncated load on group k is at most $2m_k$, and we have $m_k \geq \frac{3}{2} m_{k+1}$ for all k by the properties of smoothed machines. Lemma 4.4.2 bounds the expected maximum of (4.4) for all k by $O(1)$. This upper bounds the expected contribution of the truncated parts by $O(1)$, as required. □

4.4.3 Online Load Balancing on Related Machines

In this section, we apply the same framework to the online setting. As main difference, we compute the non-adaptive assignment online using Algorithm 9. We must be careful, because our online configuration balancing algorithm loses a logarithmic factor in the number of resources, so to obtain a $O(\log \log m)$-approximation, we aggregate each group (in the smoothed-machines instance) as a single resource. Thus, our configuration balancing instance will have only $O(\log m)$ resources.

We first describe how to construct the configuration balancing instance. Suppose we have smoothed machines with m' groups. We define $m'+1$ resources indexed $0, \dots, m'$ such that the 0th resource is a virtual resource collecting exceptional parts, and the resources $1, \dots m'$ index the groups of smoothed machines and collect the respective truncated parts. Each job j has m' configurations $c = 1, \dots, m'$ (one per job-to-group assignment) defined by

$$X_{kj}(c) = \begin{cases} \mathbb{E}\left[X_{kj}^E\right] & \text{if } k = 0, \\ \frac{1}{m_k}\mathbb{E}\left[X_{kj}^T\right] & \text{if } k = c, \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

for a fixed truncation threshold τ. Note that these configurations are deterministic. Intuitively, this definition captures the fact that we will average all jobs assigned to group k over the m_k machines in the group. Now we give our algorithm for the online setting.

Algorithm 14: Online Related Load Balancing

```
14.1 smooth machines with Algorithm 12
14.2 λ ← current guess of 𝔼[Opt]
14.3 τ ← 2λ truncation threshold
14.4 upon arrival of request j do
14.5     construct configurations as in (4.5)
14.6     k ← configuration chosen by Algorithm 9
14.7     schedule j on least-loaded machine in group k
```

Lemma 4.4.4. *For online load balancing on related machines with stochastic jobs, Algorithm 14 runs in polynomial time and correctly solves the subproblem of Lemma 4.2.4 for $\alpha = O(\log \log m)$.*

Proof. It is clear that the algorithm runs in polynomial time.

Let Opt_S be the optimal makespan of the smoothed load balancing instance. We first claim that the resulting configuration balancing instance has optimal makespan at most $O(\mathbb{E}[\mathsf{Opt}_S])$ for any truncation threshold $\tau \geq 2\mathbb{E}[\mathsf{Opt}_S]$. To see this, observe that ($\mathsf{LP_C}$) for the smoothed load balancing instance is feasible for τ. Then, ($\mathsf{LP_C}$) for the deterministic configuration balancing instance is obtained from this LP by aggregating the truncated constraints for all machines in the same group and dividing by m_k. Thus, this latter LP is also feasible for the same threshold, and it admits a constant-factor approximation by Shmoys-Tardos [ST93]. Thus, Opt_D, the optimal solution of the deterministic configuration balancing instance defined by (4.5), is at most $O(\mathbb{E}[\mathsf{Opt}_S])$.

Further, by the properties of smoothed machines, the resulting configuration balancing instance has $m'+1 = O(\log m)$ resources. As argued in the proof of Theorem 4.1.3, the potential function guarantees that Algorithm 9 never creates makespan (for the configuration balancing instance) more than $O(\log \log m)\lambda$ if $\mathsf{Opt}_D \leq \lambda$. Hence, by (4.5), the total expected exceptional load and

average expected truncated load within each group are at most $O(\log\log m)\lambda$. We translate these bounds into bounds on the makespan for the load balancing instance.

Similar to Lemma 4.4.3, we can upper bound the contribution of the exceptional parts by $O(\log\log m)\lambda$ using (4.1) and the truncated parts by $O(\log\log m)\lambda$ using Lemma 4.4.2. Thus, the expected makespan of the load balancing instance is at most $O(\log\log m)\lambda$, as required.

Finally, we recall that if Algorithm 9 fails, then $\mathsf{Opt}_D > \lambda$, which implies $\mathbb{E}[\mathsf{Opt}_S] \geq \Omega(\lambda)$. □

Summary

The proof of our main theorem for related machines follows immediately from Lemma 4.4.3 and Lemma 4.4.4.

Theorem 4.1.3. *For load balancing on related machines with stochastic jobs, there exist efficient deterministic algorithms that compute an adaptive* offline $O(1)$*-approximation and an adaptive* online $O(\log\log m)$*-approximation when comparing to the optimal offline adaptive policy.*

4.5 Clairvoyance Gap for Load Balancing on Related Machines

In this section, we prove the lower- and upper bounds of Theorem 4.1.4:

Theorem 4.1.4. *For load balancing on related machines, there exists an efficient* non-clairvoyant online *algorithm that $O(\sqrt{m})$-approximates the optimal clairvoyant algorithm. Further, any non-clairvoyant algorithm is at least $\Omega(\sqrt{m})$-approximate.*

To show this result, we first give a lower bound on the clairvoyance gap and then complement it by an upper bound achieved by a non-clairvoyant online list scheduling algorithm.

Lemma 4.5.1. *The clairvoyance gap for scheduling on related machines is $\Omega(\sqrt{m})$.*

Proof. Consider an instance with m machines, one fast machine with speed 1 and $m-1$ slow machines with speed $\frac{1}{\sqrt{m}}$. There are m jobs, one big job with size 1 and $m-1$ small jobs with size $\frac{1}{\sqrt{m}}$. There is a schedule of makespan 1, which is achieved by assigning the big job to the fast machine and one small job per slow machine.

We show that any non-clairvoyant algorithm has makespan $\Omega(\sqrt{m})$ which implies the lemma. To that end, we define the following adversarial strategy: the adversary reveals only small jobs (unless it runs out of small jobs, in which case the final job is big) until the first time that the algorithm decides to use a slow machine. Then the adversary makes this job big. If an algorithm only uses the fast machine, then it has makespan $1+(m-1)\cdot\frac{1}{\sqrt{m}} = \Omega(\sqrt{m})$. Otherwise, the algorithm must use a slow machine, which receives the big job from the adversary, which also gives makespan $\sqrt{m}$. Hence, any non-clairvoyant algorithm incurs a makespan $\Omega(\sqrt{m})$. □

In the next lemma, we show an upper bound on the clairvoyance gap. We refer to *list scheduling* as the algorithm that assigns the next job to the currently least-loaded machine. Here, the load of a machine is the total processing time of jobs that have been assigned to the machine. We use list scheduling only on a subset of machines that are "fast enough" and leave the remaining machines idle. More precisely, we list schedule the jobs in the order of their arrival on the machines with speeds within a factor $\frac{1}{\sqrt{m}}$ of the fastest.

Lemma 4.5.2. *The clairvoyance gap for related machine scheduling is $O(\sqrt{m})$.*

Proof. For convenience, we re-scale the instance such that the speed of the fastest machine is 1. Hence, our algorithm uses all machines i with $s_i \in \left[\frac{1}{\sqrt{m}}, 1\right]$; we call such machines *fast* and the remaining ones *slow*. Clearly, list scheduling on the fast machines is both, online and non-clairvoyant. It remains to bound the competitive ratio. To this end, we observe

$$C_{\max} \leq \frac{\sum_{j \in J} p_j}{\sum_{i:s_i \geq 1/\sqrt{m}} s_i} + \frac{\max_{j \in J} p_j}{\min_{i:s_i \geq 1/\sqrt{m}} s_i},$$

where the first term bounds the starting time and the second term bounds the processing time of any job. We bound both terms separately by comparing to the optimal solution Opt, where Opt denotes the optimal schedule as well as its makespan. Clearly, $\max_{j \in J} p_j \leq \mathsf{Opt}$, which implies that the second term is at most $\sqrt{m} \cdot \mathsf{Opt}$.

For the first term, note that the total size of jobs that Opt schedules on fast machines is at most $m' \cdot \mathsf{Opt}$, where m' is the number of fast machines. The total size of jobs that Opt schedules on slow machines is at most $m \cdot \frac{1}{\sqrt{m}} \cdot \mathsf{Opt} = \sqrt{m} \cdot \mathsf{Opt}$ as there are at most m slow machines. Further, the total speed of the fast machines is at least 1 (because the fastest machine has speed 1) and also at least $m' \frac{1}{\sqrt{m}}$. Therefore,

$$\frac{\sum_{j \in J} p_j}{\sum_{i:s_i \geq 1/\sqrt{m}} s_i} \leq \frac{m' \cdot \mathsf{Opt} + \sqrt{m} \cdot \mathsf{Opt}}{\max\{1, m'/\sqrt{m}\}} \leq \left(\frac{m'}{m'/\sqrt{m}} + \frac{\sqrt{m}}{1}\right) \cdot \mathsf{Opt} = O(\sqrt{m}) \cdot \mathsf{Opt},$$

concluding the proof. □

Conclusion

We considered the configuration balancing problem under uncertainty. In contrast to the (often overly optimistic) clairvoyant settings and the (often overly pessimistic) non-clairvoyant settings, we consider the stochastic setting where each request j presents a set of random vectors, and we need to (adaptively) pick one of these vectors, to minimize the *expected* maximum load over the m resources. We give logarithmic bounds for several general settings (which are existentially tight), and a much better $O(1)$ offline and $O(\log \log m)$ online bound for the related machines setting.

The main theme in this work is that the optimal adaptive policy may do unintuitive things: Without seeing Example 4.1.7, it seems like the optimal adaptive policy cannot possible accrue large total exceptional load – after all, as soon as a single configuration becomes exceptional, then Opt has missed its expected makespan target. To overcome this, we argue that there exists a near-optimal adaptive policy that is more well-behaved and thus more amenable to our classical toolkit from deterministic combinatorial optimization: LP rounding and potential functions.

The main open problem is to improve on the $O(\frac{\log m}{\log \log m})$-approximation for offline stochastic load balancing on unrelated machines using adaptivity beyond the related machines setting. One particularly simple open setting is *stochastic edge orientation*: We are given an undirected graph G and probablility parameter p. The goal is to orient every edge of G_p – the random subraph where every edge of G appears independently with probability p – to minimize the expected maximum in-degree over every vertex of G. However, our algorithm does not observe G_p directly; rather, our algorithm adaptively queries edges of G, upon which we learn if the edge is in G_p or not. If it is, then at this point we must irrevocably orient it. This problem is a special case of stochastic load balancing by taking one job per edge in G such that the resulting job can only be assigned to the two endpoints (machines) of that edge. Note that this special case avoids the adaptivity gap lower

bound of $\Omega(\frac{\log m}{\log \log m})$ due to [GKNS21], because in that lower bound, all jobs can be assigned to all machines.

Another open question is closing the gap for online related-machines load balancing. It would be particularly interesting to show a super-constant lower bound for any online algorithm for stochastic load balancing on related machines, because this would separate the deterministic (where a $O(1)$-competitive algorithm is known [BCK00]) and stochastic settings.

Chapter 5

Stochastic Completion Time Minimization

5.1 Introduction

We consider the problem of scheduling n stochastic jobs on m identical machines to minimize the expected total completion time of the jobs. Each job j is an independent random variable $X_j \sim \mathcal{D}_j$ with known distribution $\mathcal{D}_j$. Now the completion time C_j of job j is a random variable, which depends on the random job sizes (and on any random decisions our algorithm may make). Our goal is to minimize $\sum_j \mathbb{E}[C_j]$.

Our main result is the first algorithm for this problem that is both distribution-independent and has an approximation ratio sublinear in m for the special case of Bernoulli jobs ($X_j \sim s_j \cdot \mathrm{Ber}(p_j)$ for size $s_j \geq 0$ and probability $p_j \in [0, 1]$).

Theorem 5.1.1. *There exists an efficient deterministic algorithm for completion time minimization for Bernoulli jobs that computes a list schedule that $\widetilde{O}(\sqrt{m})$-approximates the optimal adaptive policy.*

We remark that a consequence of our techniques is a simple $\tilde{O}(m)$-approximation for Bernoulli jobs, matching (up to $\operatorname{poly}\log n$-factors) the result of [IMP15a] in this special case (see § 5.4.2 for details.)

5.1.1 Technical Overview

Our algorithm design will be informed by a proxy objective function, which we call the *weighted free time*. We first observe that to bound the completion time of a job, it suffices to bound its starting time, S_j. This is because on identical machines, we have $C_j = S_j + X_j$, where $\sum_j X_j$ is a lower-bound on the optimal total completion time. The key idea of our proxy objective is to relate the per-job starting times to a more global quantity, which we call the *free time*.

Definition 5.1.2 (Free Time)**.** Consider any fixed schedule. The ith free time of the schedule, which we denote by $F(i)$ is the first time when i jobs have been started and at least one machine is free to start the $(i+1)$st job.

For schedules that do not idle machines, the ith free time is the load of the least-loaded machine after starting i jobs. By definition of free time, there are $\Theta(n/2^k)$ jobs with starting times in

$[F(n-n/2^{k-1}), F(n-n/2^k)]$ for all $k = 1, \ldots, \log n$ (all logarithms are base 2 in this paper.) Thus we have:

$$\sum_j S_j = \sum_{k=1}^{\log n} \Theta(n/2^k) F(n - n/2^k). \tag{5.1}$$

We call this final expression, $\sum_{k=1}^{\log n} n/2^k \cdot F(n-n/2^k)$, the *weighted free time* of the schedule. We can view this objective as defining $\log n$ work checkpoints for our algorithm. These checkpoints are the time that we have $n/2^1$ jobs left to start (i.e. $F(n-n/2^1)$), $n/2^2$ left to start (i.e. $F(n-n/2^2)$), and so on. Roughly, the goal of our algorithm is to ensure that at each work checkpoint, our free time is comparable with the optimal schedule's free time at the same checkpoint.

We can now illustrate the reason for considering free times rather than the completion time directly. Indeed, let $C(i)$ be the time that we complete i jobs (and note the difference with C_j, which is the time at which we finish a specific job j). We analogously have $\sum_j C_j = \Theta(\sum_k n/2^k \cdot C(n-n/2^k))$. However, one difficulty of stochastic jobs is we cannot easily control what are the first $n-n/2^k$ jobs *to complete*. On the other hand, for free times, we have complete control over what $n-n/2^k$ jobs we decide *to start* first, which then contribute to $F(n-n/2^k)$. This suggests two natural informal subproblems for our algorithm design:

- **Subset Selection:** Compute nested sets of jobs $J_1 \subset J_2 \subset \ldots$ such that for all k, J_k is comparable to the first $n-n/2^k$ jobs of the optimal adaptive policy (i.e. the jobs contributing to $F(n-n/2^k)$ for Opt.)
- **Batch Free Time Minimization:** Given nested sets of jobs $J_1 \subset J_2 \subset \ldots$, schedule the J_k's such that our free time after scheduling J_k is comparable to $F(n-n/2^k)$ for Opt. Our schedule must satisfy the *batch constraint* that we schedule J_k (i.e. start every job in J_k) before $J_{k+1} \setminus J_k$ for all k.

The main technical challenge in both subproblems is the interaction between the free time and the batch constraint. Since our final algorithm will be a list schedule, the J_k-sets are chosen non-adaptively. However, the optimal policy chooses its first $n-n/2^k$ jobs *adaptively*, so it is not clear that there even exist good sets J_k. Our first contribution is that we can indeed efficiently find good J_k sets non-adaptively by delaying slightly more jobs than Opt.

Theorem 5.1.3 (Subset Selection, Informal)**.** *Given Bernoulli jobs, we can efficiently find nested sets of jobs $J_1 \subset \cdots \subset J_{\log n}$ such that $|J_k| = n - \widetilde{O}(n/2^k)$ and J_k is a subset of the first $n-n/2^k$ jobs of the optimal adaptive policy for all realizations.*

Our subset selection algorithm is a simple greedy algorithm: to construct J_k, for each possibly Bernoulli size parameter (which we may assume there are $O(\log n)$ of by standard discretization techniques) we remove the $n/2^k$ jobs with largest probability parameters. Thus, for each size, we keep the "smallest possible" jobs. The analysis relies on the following structural characterization of the optimal adaptive policy for Bernoulli jobs.

Lemma 5.1.4. *Consider a collection of Bernoulli jobs. Then for each possible size parameter, the optimal adaptive completion time schedule for these jobs starts the jobs with this size parameter in increasing order of their probabilities for all realizations of the job sizes.*

In light of Theorem 5.1.3, it remains to compute good list schedule of the J_k's subject to the batch constraint. Our free time minimization algorithm is also greedy: For each batch $J_k \setminus J_{k-1}$, we list-schedule them in increasing order of size parameter.

Theorem 5.1.5 (Batch Free Time Minimization, Informal)**.** *Given the nested sets of Bernoulli jobs $J_1 \subset \cdots \subset J_{\log n}$ guaranteed by Theorem 5.1.3, list scheduling them in increasing order of size parameter (subject to the batch constraint) is $\widetilde{O}(\sqrt{m})$-approximate for weighted free time.*

Combining Theorems 5.1.3 and 5.1.5 gives our desired $\widetilde{O}(\sqrt{m})$-approximation for completion time minimization for Bernoulli jobs. We now give an overview of our analysis.

For a moment, suppose that we could list-schedule the batches output by Theorem 5.1.3, $J_1 \subset \ldots J_{\log n}$, *optimally* subject to the batch constraint (i.e. for each $J_k \setminus J_{k-1}$, we compute an ordering of the jobs and start the jobs in this order for all $k = 1, \ldots, \log n$) to minimize the weighted free time. At first glance, it might seem like we are done, because J_k is always a subset of Opt's first $n - n/2^k$ jobs, so we are only scheduling fewer jobs than Opt at every work checkpoint. However, this reasoning fails because of the batch constraint. Indeed, let us contrast the classic makespan minimization problem (schedule *deterministic* jobs to minimize the load of the most-loaded machine) with its free time analogue (schedule n deterministic jobs to minimize the nth free time). While an arbitrary list schedule is $O(1)$-approximate for makespan, it is $\Omega(m)$-approximate for nth free time:

Lemma 5.1.6. *For all $m > 1$, there exists a set of n jobs J and a list-schedule of J whose nth free time $\Omega(m)$-approximates the optimal nth free time.*

Proof. Consider m "small" jobs of size 1 and $m-1$ "big" jobs of size m. The optimal free time schedule is to first schedule one small job on each machine, and then one big job on $m-1$ machines. Thus the optimal nth free time is 1. Now consider the list-schedule of all big jobs before small jobs. Then each big job is scheduled on a separate machine, and all m small jobs are scheduled on the remaining machine. This gives nth free time m. □

The instances from the above lemma suggest that we should schedule small jobs before bigger ones so that the big jobs do not clog up the machines and delay the starting times of the small jobs. Implicitly, this is why previous work loses a m-factor: While Opt has m machines to schedule small jobs before the machines are clogged by big ones, it can be the case that Alg first clogs all but one machine with big jobs and then schedules all small jobs on a single machine. For our algorithm, because of the batch constraint, it could be the case that Opt does some small jobs in $J_k \setminus J_{k-1}$ much earlier than our algorithm when fewer machines are clogged by big jobs. This is the main technical challenge that we overcome to obtain our improvement.

Concretely, consider scheduling J_k subject to the batch constraint. For this subproblem, we say a job is big if its realized size is larger than Opt's (random) $n-n/2^k$th free time. Scheduling such jobs effectively turns off a machine for the remainder of the schedule. We call such machines *clogged*. One should imagine that afterwards we are averaging the volume of the remaining small jobs over one less machine. Our first insight is a stronger lower bound on the rate that Opt clogs machines using Lemma 5.1.4. We let J_k^* be the adaptively-chosen set of the first $n - n/2^k$ jobs started by Opt. Because $J_{k'} \subset J_{k'}^*$ for every batch k' *for every realization of job sizes*, the number of big jobs in J_k' is at most the number of big jobs in $J_{k'}^*$. This implies that our algorithm clogs machines at a slower rate than Opt for every realization.

Similarly, Lemma 5.1.4 also implies that the total size of small jobs in $J_{k'}$ is at most the total size of small jobs in $J^*_{k'}$ for every $k' \le k$. However, this does *not* guarantee that the total size of small jobs in $J_{k'} \setminus J_{k'-1}$ is at most that in $J^*_{k'} \setminus J^*_{k'-1}$, so although our algorithm has more unclogged machines, we may also be trying to average more small jobs over these machines. Our second insight is a delicate charging argument that characterizes how the free times of previous batches affect the current one. Roughly, we argue that for our particular batches, while moving a small job to an earlier batch allows us to average it over more unclogged machines, this also delays the free time of all later batches. This ensures that there is not much benefit to moving small jobs to earlier batches. To achieve this, we initiate a systematic study of the free time.

5.1.2 Comparison to prior work

Prior $O(\Delta)$-approximations rely on bounding with respect to an LP solution, e.g. [MSU99, SSU16b]. These have an integrality gap of $\Omega(\Delta)$. On the other hand, our algorithm is combinatorial and avoids this gap by comparing directly to the optimal adaptive policy.

The distribution-independent approximation of [IMP15a] also partitions jobs into batches (as in our subset selection problem). Roughly, they guarantee that their batches are "better" than the optimal solution's jobs "in expectation." However, we will show that our algorithm's batches are better than the optimal solution's jobs for *every* realization (using our structural characterization of the optimal policy Lemma 5.1.4.)

Further, their algorithm schedules jobs within batches in arbitrary order (i.e. they give a trivial solution to the subproblem we call free time minimization). It seems likely that a loss of $\Omega(m)$ is necessary if one considers an arbitrary list schedule because of the lower bound in Lemma 5.1.6. To overcome this, we choose a particular list schedule (i.e. in increasing order of size parameter), which we show is $\widetilde{O}(\sqrt{m})$-approximate. To summarize, using a deeper technical understanding, we give more refined guarantees for both subset selection and free time minimization than [IMP15a].

The only other work is [EFMM19], which considers even more restricted instances: those with only two types of jobs, identical deterministic and identical Bernoulli. Their algorithm is to schedule either all deterministic jobs first or all Bernoullis first (depending on the relative number of each type of jobs.) Our subset selection algorithm vastly generalizes this idea to arbitrary Bernoulli jobs with varying size and probability parameters. Further, while our algorithm in Theorem 5.1.5 runs an index policy within each $J_k \setminus J_{k-1}$-batch, we overcome the lower bound on index policies due to [EFMM19] because our subset selection algorithm constructs the batches by taking into account the relative number of different types of jobs—not only the distributions of individual jobs.

5.2 Subset Selection

The goal of this section is to solve the *subset selection subproblem* for Bernoulli jobs: we want to find nested sets of jobs $J_1 \subset \cdots \subset J_{\log n}$ such that J_k is comparable to the first $n - n/2^k$ jobs of the optimal adaptive completion time schedule. Formally, let J^*_k be the random set consisting of the first $n - n/2^k$ jobs scheduled by the optimal completion time schedule. Our main theorem here is the following:

Theorem 5.2.1 (Subset Selection)**.** *Let L be the number of distinct Bernoulli size parameters. There is an algorithm* ChooseJobs *that outputs sets J_k satisfying:*

(i) $J_1 \subset \cdots \subset J_{\log n} \subset J$
(ii) $|J_k| \in [n - L \cdot n/2^k, n - n/2^k]$

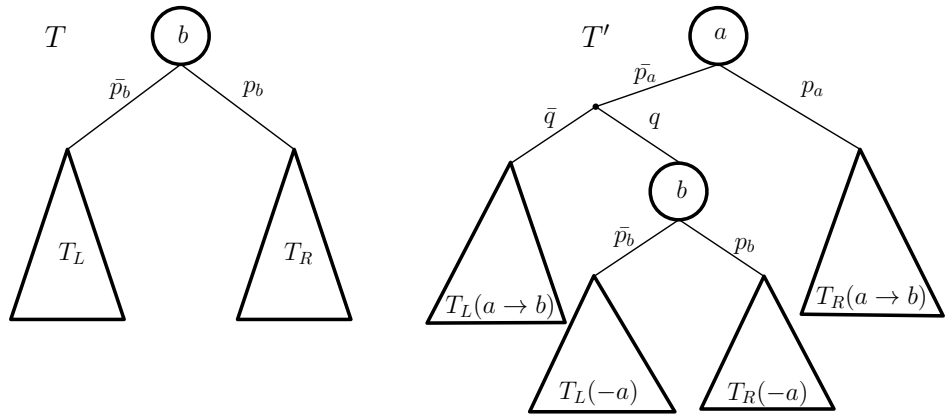

Figure 5.1: Original decision tree T and modified decision tree T'

(iii) $J_k \subset J_k^*$ for all k and all realizations.

We show later how to use standard rescaling and discretization techniques to assume $L = O(\log n)$ while losing only an extra constant factor in our final approximation ratio.

It is convenient to think of the $n/2^k$ jobs that the optimal schedule *excludes* from J_k^* rather than the jobs it chooses to start. Similarly, we specify our algorithm's set of jobs also by the exclusions. The next lemma gives our structural characterization for the optimal adaptive completion time schedule for Bernoulli jobs, which allows us to characterize which jobs the optimal schedule chooses to exclude.

Because jobs are Bernoullis, upon scheduling a job j, the scheduler immediately learns the realized size of X_j because it is either 0 or s_j. Thus, the optimal schedule can be represented by a decision tree, where each node is labeled by a job j, corresponding to the decision to schedule j on the currently least loaded machine, and has a left- and right child, corresponding to the realized size of j being 0 or s_j, respectively. Every root-leaf path on this tree gives an ordering to schedule the jobs for a particular realization of job sizes.

Lemma 5.1.4. *Consider a collection of Bernoulli jobs. Then for each possible size parameter, the optimal adaptive completion time schedule for these jobs starts the jobs with this size parameter in increasing order of their probabilities for all realizations of the job sizes.*

Proof Sketch. Our proof is an exchange argument. Consider the optimal decision tree (as described above), and suppose there exists a root-leaf path that schedules job $X_b \sim s \cdot \mathrm{Ber}(p_b)$ before $X_a \sim s \cdot \mathrm{Ber}(p_a)$ with $p_a \le p_b$. Then there exists a subtree T rooted at b such that a is scheduled on every root-leaf path in this subtree.

We now show how to modify T to start a before b while not increasing the expected completion time. Let T_L and T_R be the left- and right subtrees (corresponding to the root job b coming up size 0 or s) of T, respectively. We define $T_L(a \to b)$ to be T_L with the job a replaced by job b and $T_L(-a)$ to be T_L, but at a's node, we do not schedule anything and instead go to a's left child. The subtrees $T_R(a \to b)$ and $T_R(-a)$ are defined analogously. See Figure 5.1 for the modified tree T'.

We choose the parameter q so that the probability that T' enters $T_L(a \to b)$ or $T_L(-a)$ is exactly $\bar{p}_b$. This is our replacement for the event that T enters T_L. A calculation now shows that the expected completion time weakly decreases from T to T'. (See the proof in Appendix C.2 for details.) □

By definition Opt can exclude only $n/2^k$ jobs from J_k^*. On the other hand, our algorithm will exclude $n/2^k$ jobs of each Bernoulli size *simultaneously*. Lemma 5.1.4 suggests that we might as well exclude the jobs with largest p_j's. In particular, our algorithm to choose sets of jobs that are comparable to the J_k^*'s is the following:

ChooseJobs: For each $k = 1, \ldots, \log n$, let J_k be the set of jobs constructed as follows:

i. Initialize $J_k = J$.
ii. For each Bernoulli size s, remove from J_k the $n/2^k$ jobs of size s with largest p_j's.
iii. Output J_k.

Proof of Theorem 5.2.1. It is immediate that the sets $J_1, \ldots, J_{\log n}$ are nested and have the desired size. It remains to show that $J_k \subset J_k^*$. Note that Opt excludes at most $n/2^k$ jobs of the largest probabilities of each size by Lemma 5.1.4. This holds for all realizations. On the other hand, we exclude $n/2^k$ jobs of largest probability of all sizes simultaneously. □

Morally, Theorem 5.2.1 states that we know the jobs that Opt starts to achieve $F^*(n - n/2^k)$ for all k (up to a L-factor.) This suggests that we should first schedule J_1 to get free time comparable to $F^*(n - n/2^1)$, and then $J_2 \setminus J_1$ to get free time comparable to $F^*(n - n/2^2)$, and so on.

The goal of the next section is to show how to schedule the J_k's subject to this batch constraint (we must schedule all jobs in J_{k-1} before any in $J_k \setminus J_{k-1}$ for all k) such that our weighted free time is comparable to that of Opt. Note that in general, even though $J_k \subset J_k^*$ for all k, the *optimal* completion time schedule may not satisfy the batch constraint—this is precisely is the main technical challenge that we have to overcome in the next section.

5.3 Batch Free Time Minimization

We now turn to the *batch free time minimization* problem. Our starting point is the nested sets of jobs $J_1 \subset \cdots \subset J_{\log n} \subset J$ output by ChooseJobs. Recall that J_k^* is the random set of the first $n - n/2^k$ jobs scheduled by the optimal completion time policy Opt.

5.3.1 Free Time Basics

To motivate our final algorithm, we explore some basic properties of the free time. We first recall the lower bound instance from Lemma 5.1.6: there are m small jobs of size 1 and $m - 1$ big jobs of size m. The optimal nth free time is 1 by first scheduling all small jobs and then all big jobs. Observe that the final $m - 1$ big jobs do not contribute to the optimal nth free time. This gave the intuition that we should schedule small jobs before big ones clog up the machines. This intuition turns out to be correct, which we formalize in the next lemma.

Lemma 5.3.1. *List-scheduling deterministic jobs in increasing order of size is a 4-approximation for nth free time.*

Proof. Let J be the set of input jobs and Opt the optimal nth free time. We partition J into *small* and *big* jobs: a job is big if its size is strictly greater than Opt, and is small otherwise. This definition means that Opt can schedule at most one big job per machine. Moreover, there are strictly less than m big jobs, otherwise Opt would need to schedule at least one big job per machine, which contradicts the optimal nth free time.

On the other hand, our algorithm starts all small jobs before all big jobs. We claim that the nth free time of our algorithm, which we denote by $F(J)$, is at most the makespan of our algorithm after only scheduling the small jobs, which we denote by $M(S)$. To see this, consider the time right before we start the first big job. All machines have loads within $[M(S) - \mathsf{Opt}, M(S)]$ (the lower bound follows by noting that we list-scheduled only jobs of size at most Opt up until this point.) Thus, we schedule the at most $m-1$ remaining big jobs each on separate machines. All of the big jobs are started by time $M(S)$, and there exists a machine that schedules no big job, which is free by $M(S)$ as well. It follows, $F(J) \leq M(S)$.

Now, because every list-schedule is 2-approximate for makespan, we have $M(S) \leq 2M^{\mathsf{Opt}}(S)$, where $M^{\mathsf{Opt}}(S)$ is the makespan of the small jobs under Opt (i.e. when Opt finishes its final small job.) To complete the proof, we relate $M^{\mathsf{Opt}}(S)$ with Opt. It suffices to show $M^{\mathsf{Opt}}(S) \leq 2\mathsf{Opt}$. To see this, consider time Opt in the optimal schedule. At this time, all machines are either free or working on their final job. In particular, any machine working on a small job completes by time $\mathsf{Opt} + \mathsf{Opt}$. □

While we do not apply Lemma 5.3.1 directly for our algorithm, the ideas in the analysis will be crucial. In particular, a key concept in our analysis is to differentiate between small and big jobs. Roughly, when we consider J_k, a job is small if its size is at most $F^*(n - n/2^k)$ and big otherwise. We are concerned about the volume (total size) of the small jobs and number of big jobs. However, because of the batch constraint, we cannot ensure that all small jobs are scheduled before all big jobs in general.

As we schedule batches $J_1, J_2 \setminus J_1, \ldots, J_k \setminus J_{k-1}$, more and more machines are getting clogged by big jobs. For the purposes of scheduling J_k, these machines are effectively turned off. Thus, as we proceed through the batches, we are averaging the volume of small jobs over fewer and fewer machines. The goal of our algorithm will be to ensure that we do small jobs as early as possible (subject to the batch constraint) so that we have the most unclogged machines available.

5.3.2 Final Algorithm

With the goal of § 5.3.1 in mind, we are ready to describe our final algorithm, which is the $\widetilde{O}(\sqrt{m})$-approximation guaranteed by Theorem 5.1.1. Although we cannot ensure that within a batch, jobs are scheduled in increasing order of *realized* size as in the analysis of Lemma 5.3.1, because our jobs are Bernoullis, we can ensure that all jobs that come up heads (have realized size s_j) are scheduled in increasing order of realized size. Here, we crucially use the fact that our jobs are Bernoullis, so if the come up tails (have size 0), they do not affect the free time. For the rest of the description, we make the following assumption, which we justify in Appendix C.3.

Assumption 5.3.2. We assume that there are $L = O(\log n)$ distinct Bernoulli size parameters s_j, each at most n^8.

By losing a constant factor in the final approximation ratio, we may assume Assumption 5.3.2 for the rest of the analysis.

Lemma 5.3.3. *Let $m \geq 2$. Suppose there exists an algorithm for completion time minimization for Bernoulli jobs on m machines satisfying Assumption 5.3.2 that outputs a list schedule with expected completion time at most $\alpha(\mathbb{E}\mathsf{Opt} + O(1))$. Then there exists a $O(\alpha)$-approximate algorithm for the same problem without the assumption. Further, the resulting algorithm is also a list schedule, and it preserves efficiency and determinism.*

The proof uses standard rescaling and discretization ideas, but it is more involved because of the stochastic jobs; we justify it in Appendix C.3. Our final algorithm is now the following:

StochFree: Given input collection J of Bernoulli jobs:

i. Run ChooseJobs to obtain nested sets of jobs $J_1 \subset \cdots \subset J_{\log n} \subset J$

ii. List-schedule each batch $J_k \setminus J_{k-1}$ in increasing order of Bernoulli size parameter s_j for all batches $k = 1, \ldots, \log n$.

iii. List-schedule all remaining jobs $J \setminus J_{\log n}$ in arbitrary order.

It is clear that StochFree outputs a list schedule in polynomial time and is deterministic. Our main approximation guarantee for StochFree is the following.

Theorem 5.3.4 (Batch Free Time Minimization)**.** *Given Bernoulli jobs, if $m \geq 2$ and Assumption 5.3.2 holds, then* StochFree *outputs a list schedule with expected completion time at most $\widetilde{O}(\sqrt{m}) \cdot \big(\mathbb{E}[\mathsf{Opt}] + O(1)\big)$, where Opt is the optimal adaptive policy.*

Note that composing Theorem 5.3.4 with Lemma 5.3.3 gives the desired $\widetilde{O}(\sqrt{m})$ without the assumption for all $m \geq 2$. For the remaining case of $m = 1$, scheduling the jobs in increasing order of their expected processing times is an optimal policy [Rot66]. This gives the desired $\widetilde{O}(\sqrt{m})$-approximation for all m, and completes the proof of Theorem 5.1.1. In the remainder of the paper, we analyze StochFree (Theorem 5.3.4.)

5.4 Analysis of the StochFree Algorithm

The goal of this section is to prove Theorem 5.3.4, given Assumption 5.3.2. Our proof has four conceptual steps.

i. Bound the weighted free time of Alg by averaging the volume of small jobs within each batch over the unclogged machines—those that have not yet scheduled a big job. (§ 5.4.1)

ii. Show that StochFree is $\widetilde{O}(m)$-approximate for all $m \geq 2$. This serves as a warm-up to the improved $\widetilde{O}(\sqrt{m})$-approximation, and it allows us to focus on the remaining case where $m = \Omega(1)$ is sufficiently large. (§ 5.4.2)

iii. Control the rate that machines become clogged by a large job. We show that the rate that machines become clogged for Alg is slow enough so that Opt cannot benefit much by putting small volume in earlier batches than Alg. (§ 5.4.3)

iv. Finally, bound the contribution of the volume of small jobs to the free time. Here we handle the main challenge, which is that Opt may schedule small volume "in the past" compared to Alg. (§ 5.4.4-§ 5.4.6)

5.4.1 Weighted free time

First, we pass from total completion time to our proxy objective of weighted free time. We let Opt denote the optimal adaptive completion time policy as well as its completion time. Recall that J_k^* is the first $n - n/2^k$ jobs scheduled by this policy achieving free time $F^*(n - n/2^k)$. Analogously, we let Alg denote the completion time of our algorithm, and $F(J_k)$ the free time of our algorithm after scheduling J_k.

Lemma 5.4.1. *We have* $\mathsf{Alg} = O(\log n)(\sum_k n/2^k \cdot F(J_k) + \mathsf{Opt})$ *and* $\mathsf{Opt} = \Omega(\sum_k n/2^k \cdot F^*(n - n/2^k))$.

Proof. We rewrite $\mathsf{Alg} = \sum_j C_j = \sum_j S_j + \sum_j X_j$. First, note that $\sum_j X_j \le \mathsf{Opt}$. It remains to bound $\sum_j S_j$. Recall that in StochFree, first we list-schedule $J_{\log n}$ subject to the batch constraint and then $J \setminus J_{\log n}$. We first handle the starting times of $J_{\log n}$. For all k, note that $J_k \setminus J_{k-1}$ consists of at most $O(L) \cdot n/2^k$ jobs with starting times in $[F(J_{k-1}), F(J_k)]$ by Theorem 5.2.1. Thus we have $\sum_{j \in J_{\log n}} S_j = O(L) \cdot (\sum_k n/2^k \cdot F(J_k)) = O(\log n) \cdot (\sum_k n/2^k \cdot F(J_k))$ using Assumption 5.3.2.

For the jobs in $J \setminus J_{\log n}$, by Theorem 5.2.1, there are at most $O(L) = O(\log n)$ such jobs. Each of these jobs completes by the makespan of Alg's schedule. Further, Alg is a list schedule, which is 2-approximate for makespan, so the makespan of Alg is at most twice the makespan of Opt. The makespan of Opt is a lower bound on Opt (because some job must complete at this time.) We conclude, $\sum_{j \in J \setminus J_{\log n}} S_j = O(\log n)\mathsf{Opt}$. Combining our bounds for $J_{\log n}$ and $J \setminus J_{\log n}$ gives the desired result for Alg.

The bound on Opt follows from (5.1). □

We refer to $\sum_k n/2^k \cdot F(J_k)$ as Alg's weighted free time and $\sum_k n/2^k \cdot F^*(n - n/2^k)$ as Opt's. The remainder of the analysis will focus on bounding Alg's weighted free time with respect to Opt's. Our main result is the following:

Theorem 5.4.2. *If* $m = \Omega(1)$ *is sufficiently large, then the weighted free time of* Alg *satisfies:*

$$\mathbb{E}\left[\sum_k n/2^k \cdot F(J_k)\right] = \widetilde{O}(\sqrt{m}) \cdot \left(\mathbb{E}\left[\sum_k n/2^k \cdot F^*(n - n/2^k)\right] + \mathbb{E}[\mathsf{Opt}]\right) + O(1).$$

Note that Theorem 5.4.2 along with Lemma 5.4.1 implies the desired guarantee in Theorem 5.3.4 for the case $m = \Omega(1)$ sufficiently large.

We now introduce some notations. For all k, we call $I_k = J_k \setminus J_{k-1}$ the kth *batch* of jobs. Recall that the J_k's are nested, so the batch constraint says we schedule in order $I_1, \ldots, I_{\log n}$. We define $I_k^* = J_k^* \setminus J_{k-1}^*$ analogously. For any set of jobs, J' and $\tau \ge 0$, we define $J'(= \tau)$ to be the random subset consisting of all jobs in J' with *realized* size exactly τ. We define $J'(> \tau)$ and $J'(\le \tau)$ analogously. Further, for a set of jobs J', we let $\mathrm{Vol}(J') = \sum_{j \in J'} X_j$ be the volume of J'. Finally, we say job j is τ-big for $\tau \ge 0$ if $X_j > \tau$. Otherwise j is τ-small.

As in the analysis for minimizing the free time for a single batch of deterministic jobs (Lemma 5.3.1), the key concept is to differentiate between small and big jobs. To this end, for all k we define the random threshold $\tau_k = 2 \cdot \max(\mathbb{E}F^*(n - n/2^k), F^*(n - n/2^k))$. Morally, one should imagine that τ_k is $F^*(n - n/2^k)$, but there is an edge case where $F^*(n - n/2^k) < \mathbb{E}F^*(n - n/2^k)$ and a multiplicative factor for concentration. When bounding $F(J_k)$, we will take τ_k to be our threshold between small- and big jobs. This threshold has the following crucial property that Alg always has at least as many unclogged machines as Opt. In particular, Alg always has at least one unclogged machine.

Proposition 5.4.3. *For all k, the following holds per-realization: $|J_k(> \tau_k)| \le |J_k^*(> \tau_k)| < m$.*

Proof. The first inequality follows from Theorem 5.2.1, because $J_k(> \tau_k) \subset J_k^*(> \tau_k)$ per-realization. For the second inequality, note that $\tau_k \ge F^*(n - n/2^k)$, so by definition of free time, Opt schedules strictly less than m jobs bigger than τ_k to achieve $F^*(n - n/2^k)$. □

Using this threshold, we re-write $F(J_k)$ by averaging the volume of small jobs over the unclogged machines (the ones with no big job.)

Lemma 5.4.4. *For all k, the following holds per-realization:*

$$F(J_k) \le F(J_{k-1}) + \frac{\mathrm{Vol}(I_k(\le \tau_k))}{m - |J_{k-1}(> \tau_k)|} + 2\tau_k.$$

Proof. First, we note by Proposition 5.4.3 that the denominator $m - |J_{k-1}(> \tau_k)| \ge 1$. Then consider time $F(J_{k-1})$. There are at least $m - |J_{k-1}(> \tau_k)|$ machines that have not scheduled a τ_k-big job in J_{k-1}. At this time, each machine is either free or working on its final job in J_{k-1}. In particular, each machine that has not scheduled a τ_k-big job in J_{k-1} is free to start working on I_k by time $F(J_{k-1}) + \tau_k$, and there are at least $m - |J_{k-1}(> \tau_k)|$ such machines.

We need the following monotonicity property of list schedules.

Lemma 5.4.5. *Consider a set of deterministic jobs and a fixed list schedule of those jobs. Then increasing the initial load or decreasing the number of machines weakly increase the free time of the schedule.*

Proof. Let J be the set of jobs. Consider initial load vectors $\ell, \ell' \in \mathbb{R}^m$, where the ith entry of each vector denotes the initial load on machine i. Now suppose $\ell \le \ell'$, entry-wise. It suffices to show that $F(J, \ell) \le F(J, \ell')$, where $F(J, \ell)$ is the free time achieved by our list-schedule with initial load ℓ. This suffices, because we can decrease the number of machines by making the initial loads of some machines arbitrarily large so that they will never be used.

We prove $F(J, \ell) \le F(J, \ell')$ by induction on the number of jobs, $|J|$. In the base case, $|J| = 0$, so the claim is trivial because $\ell \le \ell'$. For $|J| > 0$, let j be the first job in the list, which is scheduled, without loss of generality, on the first machine for both initial loads ℓ and ℓ'. Then:

$$F(J, \ell) = F(J \setminus \{j\}, \ell + s_j e_1) \le F(J \setminus \{j\}, \ell' + s_j e_1) = F(J, \ell'),$$

where e_1 is the first standard basis vector, so we have $\ell + s_j e_1 \le \ell' + s_j e_i 1$ entry-wise. Then we assumed inductively that $F(J \setminus \{j\}, \ell + s_j e_1) \le F(J \setminus \{j\}, \ell' + s_j e_1)$. □

By Lemma 5.4.5, we can upper-bound $F(J_k)$ by list-scheduling I_k with initial load $F(J_{k-1}) + \tau_k$ on $m - |J_{k-1}(> \tau_k)|$ machines that have not scheduled a τ_k-big job in J_{k-1}. Recall that Alg list-schedules I_k in increasing order of size parameter, so - ignoring jobs that come up tails with realized size 0 - we schedule all τ_k-small jobs in I_k before any τ_k-big one. Further, $|I_k(> \tau_k)| < m - |J_{k-1}(> \tau_k)|$ by Proposition 5.4.3, so there exists some machine that schedules only τ_k-small jobs in I_k. This machine is free by time $F(J_k) \le F(J_{k-1}) + \tau_k + \frac{\mathrm{Vol}(I_k(\le \tau_k))}{m - |J_{k-1}(> \tau_k)|} + \tau_k$. □

Using Lemma 5.4.4 and the exponentially decreasing weights, we can re-write Alg's weighted free time as:

$$\sum_k n/2^k \cdot F(J_k) = O\left(\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\le \tau_k))}{m - |J_{k-1}(> \tau_k)|} + \sum_k n/2^k \cdot \tau_k \right) \tag{5.2}$$

By definition of τ_k, the second sum is $O(\mathbb{E} \sum_k n/2^k \cdot F^*(n - n/2^k))$ in expectation, which is exactly Opt's weighted free time. It remains to bound the first sum.

5.4.2 Warm up: $\widetilde{O}(m)$-approximation

Before proceeding with the proof of Theorem 5.4.2, we observe that Equation (5.2) along with our basic weighted free time properties is enough to give a $\widetilde{O}(m)$-approximation. Interestingly, this gives a simple proof that nearly matches the previously best-known guarantees for Bernoulli jobs.

Lemma 5.4.6. *Given Bernoulli jobs, if $m \geq 2$ and Assumption 5.3.2 holds, then* StochFree *outputs a list schedule whose expected completion time $\widetilde{O}(m)$-approximates the optimal adaptive policy.*

Proof. Starting from (5.2):

$$\sum_k n/2^k \cdot F(J_k) = O\Big(\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} + \sum_k n/2^k \cdot \tau_k\Big),$$

we note that $I_k \subset J_k^*$ by Theorem 5.2.1 and $m - |J_{k-1}(> \tau_k)| \geq 1$ by Proposition 5.4.3. Thus, we can bound:

$$\frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} \leq \mathrm{Vol}(J_k^*(\leq \tau_k)).$$

We claim that $\mathrm{Vol}(J_k^*(\leq \tau_k)) = O(m) \cdot \tau_k$. To see this, observe that by averaging the volume of $J_k^*(\leq \tau_k)$ over the m machines, after scheduling J_k^*, each machine in Opt has load at least $\frac{\mathrm{Vol}(J_k^*(\leq \tau_k))}{m} - \tau_k$. This gives $F^*(n - n/2^k) \geq \frac{\mathrm{Vol}(J_k^*(\leq \tau_k))}{m} - \tau_k$. Noting that $\tau_k \geq F^*(n - n/2^k)$ completes the proof that $\mathrm{Vol}(J_k^*(\leq \tau_k)) = O(m) \cdot \tau_k$.

Applying this to our above expression gives $\frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} = O(m) \cdot \tau_k$, so Alg's weighted free time satisfies:

$$\sum_k n/2^k \cdot F(J_k) = O(m) \cdot (\sum_k n/2^k \cdot \tau_k).$$

Taking expectations and applying Lemma 5.4.1 completes the proof. □

The loss of m in the above proof was because Alg averages the small volume over at least 1 unclogged machine, but Opt may average the same volume over at most m machines. Intuitively, this reasoning is why previous work loses a m-factor as well.

Further, this is the main technical challenge that we will overcome to get our improvement. Indeed, even though $J_k \subset J_k^*$ for all k, it is not true that $I_k \subset I_{k^*}$. This means that while we are averaging $\mathrm{Vol}(I_k(\leq \tau_k))$ over $m - |J_{k-1}(> \tau_k)|$ machines (which is at least as many machines as Opt has for batch k), it can be the case that Opt actually did jobs in I_k in much earlier batches. In the remainder of our analysis, we do a more fine-grained analysis of the rate that Alg and Opt clog machines, and when they choose to do the same small volume. This allows us to break through the linear dependence in m.

5.4.3 Bounding the unclogged machines

In this section, we are interested in controlling the quantity $m - |J_{k-1}(> \tau_k)|$, which is the number of machines we have left to schedule I_k (the unclogged machines.) Note that there are two sources of randomness: the realizations of jobs in J_{k-1} and the threshold τ_k.

Our strategy is to control $m - \mathbb{E}|J_{k-1}(> \tau)|$ for a fixed threshold τ. Then because $|J_{k-1}(> \tau)|$ is a sum of independent $\{0,1\}$-valued random variables, a Chernoff-union bound argument allows us to control $m - |J_{k-1}(> \tau)|$ as well.

However, we will see that concentration alone is not enough; this is because there is an unbounded difference between $|J_{k-1}(> \tau)| = m$ and $|J_{k-1}(> \tau)| < m - 1$. In the former case, all machines are clogged by big jobs, whose size we cannot upper bound. Thus, we cannot make any progress towards reaching time $F(J_k)$ (by starting more jobs.) In the latter, we have at least one machine, so we can still make some progress towards $F(J_k)$. The situation to keep in mind is when $\mathbb{E}|J_{k-1}(> \tau)|$ is close to m, so concentration around the mean will fail to preserve this hard constraint that we need at least one unclogged machine. To remedy this, we will combine concentration arguments with the per-realization properties of StochFree.

We begin with the concentration arguments, so we wish to understand $m - \mathbb{E}|J_{k-1}(> \tau)|$. We first use the properties of ChooseJobs to bound $\mathbb{E}|J_{k-1}(> \tau)|$:

Proposition 5.4.7. *For all fixed thresholds τ and batches k, we have $\mathbb{E}|I_k(> \tau)| \geq \frac{1}{2}\mathbb{E}\tau|I_{k-1}(> \tau)|$.*

Proof. By summing over the relevant sizes, it suffices to prove $\mathbb{E}|I_k(= s)| \geq \frac{1}{2}\mathbb{E}|I_{k-1}(= s)|$ for any Bernoulli size parameter s. We may assume $\mathbb{E}|I_{k-1}(= s)| > 0$ or else the proposition is trivial.

Then when ChooseJobs constructs J_{k-1}, it includes at least one job with size parameter s. It follows, there exist $n/2^{k-1}$ remaining jobs in $J \setminus J_{k-1}$ with size parameter s. When constructing J_k, ChooseJobs will include $n/2^k$ of these remaining jobs. In conclusion, I_{k-1} has at most $n/2^{k-1}$ jobs with size parameter s, while I_k has at least $n/2^k$. The result follows because ChooseJobs includes jobs in increasing order of p_j. □

Proposition 5.4.7 allows us to relate the expected number of machines left (with respect to fixed threshold τ) at batch k with the number of machines left at $k' \leq k$:

Lemma 5.4.8. *For all fixed thresholds τ and batches $k' \leq k$, we have $m' - \mathbb{E}|J_{k-1}(> \tau)| \geq 2^{-(k-k'+1)} \cdot (m' - \mathbb{E}|J_{k'-1}(> \tau)|)$, where $m' \geq \mathbb{E}|J_k(> \tau)|$.*

Proof. We may assume $\mathbb{E}|I_k(> \tau)| > 0$ or else the lemma is trivial, because by definition of ChooseJobs, if $\mathbb{E}|I_k(> \tau)| = 0$, then $\mathbb{E}|J_{k-1}(> \tau)| = 0$ and $\mathbb{E}|J_{k'-1}(> \tau)| = 0$.

In particular, we may assume $m' - \mathbb{E}|J_{k-1}(> \tau)| \geq \mathbb{E}|I_k(> \tau)| > 0$. Then we compute:

$$\frac{m' - \mathbb{E}|J_{k'-1}(> \tau)|}{m' - \mathbb{E}|J_{k-1}(> \tau)|} = 1 + \frac{\mathbb{E}|I_{k'}(> \tau)| + \cdots + \mathbb{E}|I_{k-1}(> \tau)|}{m' - \mathbb{E}|J_{k-1}(> \tau)|} \leq 1 + \frac{\mathbb{E}|I_{k'}(> \tau)| + \cdots + \mathbb{E}|I_{k-1}(> \tau)|}{\mathbb{E}|I_k(> \tau)|}.$$

Repeatedly applying Proposition 5.4.7 to the numerator gives:

$$1 + \frac{\mathbb{E}|I_{k'}(> \tau)| + \cdots + \mathbb{E}|I_{k-1}(> \tau)|}{\mathbb{E}|I_k(> \tau)|} \leq 1 + (2^{k-k'} + \cdots + 2^1) \leq 2^{k-k'+1}.$$

□

To see the utility of Lemma 5.4.8, suppose $\mathbb{E}|J_k(> \tau)| = m$. Then roughly the lemma says in expectation, we lose at most half of our remaining machines between each batch. However, in the weighted free time the coefficient $n/2^k$ (corresponding to the number of jobs delayed by the current batch) also halves between each batch. Thus, although we are losing half of our machines, only half as many jobs are affected by this loss.

First, we bound the expectation of $|J_k(> \tau)|$ when τ is sufficiently large (i.e. for all possible realizations of τ_k.) The proof uses a Chernoff bound along with the definition of big jobs; see Appendix C.4 for proof.

Lemma 5.4.9. *Let $m = \Omega(1)$ be sufficiently large. Then there exists a constant $c \geq 0$ such that for all batches k and thresholds $\tau > 2\mathbb{E}F^*(n - n/2^k)$, we have $\mathbb{E}|J_k(> \tau)| \leq m + c\sqrt{m}$.*

Now, because $\mathbb{E}|J_k(> \tau)| = O(m)$, we can bound the deviation of $|J_k(> \tau)|$ by $\widetilde{O}(\sqrt{m})$ with high probability.

We define the notation $|J_k(> \tau)| \overset{\pm\Delta}{\approx} \mathbb{E}|J_k(> \tau)|$ to denote the event

$$| \, |J_k(> \tau)| - \mathbb{E}|J_k(> \tau)| \, | \leq \Delta.$$

The proof of the next lemma is a Chernoff-union argument; see Appendix C.4 for proof.

Lemma 5.4.10. *Let $\Delta = O(\sqrt{m} \log n)$ and $m = \Omega(1)$ be sufficiently large. Then with probability at least $1 - \frac{1}{\text{poly}(n)}$, the following events hold:*

$$\{|J_k(> \tau)| \overset{\pm\Delta}{\approx} \mathbb{E}|J_k(> \tau)| \quad \forall \text{ batches } k \text{ and thresholds } \tau > 2\mathbb{E}F^*(n - n/2^k)\}. \tag{5.3}$$

Combining Lemma 5.4.8 and Lemma 5.4.10, we can show the number of remaining machines is concentrated as well. Here we also need to bring in the per-realization properties of StochFree to handle the case where concentration is not enough to ensure that we have at least one remaining machine. This is the main result of this section. Recall that we defined $\tau_k = 2\max(\mathbb{E}F^*(n - n/2^k), F^*(n - n/2^k))$, so in particular $\tau_k \geq 2\mathbb{E}F^*(n - n/2^k)$.

Lemma 5.4.11. *Suppose Event (5.3) holds. Then for all pairs of batches $k' \leq k$, we have $m - |J_{k-1}(> \tau_k)| \geq (3\Delta)^{-1}2^{-(k-k'+1)}(m - |J_{k'-1}(> \tau_k)|)$, where $\Delta = O(\sqrt{m} \log n)$.*

Proof. Consider fixed batches $k' \leq k$, and let $\mu_k = \mathbb{E}|J_k(> \tau_k)|$ and $\mu_{k'} = \mathbb{E}|J'_k(> \tau_k)|$. Note that $\tau_k \geq 2\mathbb{E}F^*(n - n/2^k) \geq 2\mathbb{E}F^*(n - n/2^{k'})$, so Event (5.3) gives $|J_k(> \tau_k)| \overset{\pm\Delta}{\approx} \mu_k$ and $|J_{k'}(> \tau_k)| \overset{\pm\Delta}{\approx} \mu_{k'}$. Further, we may choose $\Delta = O(\sqrt{m} \log n)$ large enough so that $\mu_k \leq m + \Delta$. Using these approximations with Lemma 5.4.8 gives:

$$\begin{aligned} m - |J_k(> \tau_k)| &= m + \Delta - |J_k(> \tau_k)| - \Delta \\ &\geq m + \Delta - \mu_k - 2\Delta \\ &\geq 2^{-(k-k'+1)}(m + \Delta - \mu_{k'}) - 2\Delta \\ &\geq 2^{-(k-k'+1)}(m - |J_k(> \tau_k)|) - 2\Delta. \end{aligned}$$

Finally, by Proposition 5.4.3, $m - |J_k(> \tau_k)| \geq 1$, so rearranging gives:

$$3\Delta(m - |J_k(> \tau_k)|) \geq m - |J_k(> \tau_k)| + 2\Delta \geq 2^{-(k-k'+1)}(m + |J_k(> \tau_k)|).$$

□

To summarize, we showed that up to a multiplicative $\widetilde{O}(\sqrt{m})$-factor, the number of unclogged machines with respect to threshold τ_k at worst halves in each batch up to k.

5.4.4 Bounding small-in-the-past jobs

Recall that our goal is to bound $\sum_k n/2^k \cdot \frac{\text{Vol}(I_k(\leq \tau_k))}{m-|J_{k-1}(>\tau_k)|}$. To this end, consider fixed k. Because $I_k \subset J_k^* = \cup_{k' \leq k} I_k^*$ (by Theorem 5.2.1), we can write:

$$\frac{\text{Vol}(I_k(\leq \tau_k))}{m-|J_{k-1}(>\tau_k)|} = \sum_{k' \leq k} \frac{\text{Vol}(I_k \cap I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k-1}(>\tau_k)|} + \sum_{k' \leq k} \frac{\text{Vol}(I_k \cap I_{k'}^*(>\tau_{k'}, \leq \tau_k))}{m-|J_{k-1}(>\tau_k)|}.$$

Thus, we split I_k depending on which batch Opt decided to schedule that job in. Further, we split $I_k \cap I_{k'}^*$ (i.e. the jobs our algorithm does in batch k that Opt did in the past batch $k' \leq k$) into the jobs that are small-in-the-past (size at most $\tau_{k'}$) and big-in-the-past (size greater than $\tau_{k'}$ and at most τ_k.)

The goal of this section is to bound the small-in-the-past jobs. This formalizes the idea that the rate at which we lose machines, guaranteed by Lemma 5.4.11, is offset by the number of jobs Opt is delaying, captured by the exponentially decreasing weights $n/2^k$. More precisely, if Opt decides to do a small job from I_k in an earlier batch, say $I_{k'}^*$, then Opt is averaging this small volume over at most a $\widetilde{O}(\sqrt{m}) \cdot 2^{k-k'}$-factor more unclogged machines. However, the weight of this term in Opt's weighted free time increased by a $2^{k-k'}$-factor as well, corresponding to the number of jobs delayed by batch k'. Thus, up to a $\widetilde{O}(\sqrt{m})$-factor, there is no benefit to doing the small-in-the-past jobs any earlier. We show the following.

Lemma 5.4.12. *Suppose Event (5.3) holds. Then the small-in-the-past jobs satisfy:*

$$\sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\text{Vol}(I_k \cap I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k-1}(>\tau_k)|} = \widetilde{O}(\sqrt{m}) \cdot \sum_k n/2^k \cdot \tau_k.$$

Proof. Because there are $O(\log n)$ batches, it suffices to show for fixed k and $k' \leq k$ that we have

$$\frac{\text{Vol}(I_k \cap I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k-1}(>\tau_k)|} = O(\Delta) \cdot 2^{k-k'} \tau_{k'},$$

where $\Delta = O(\sqrt{m} \log n)$. Summing over all k and $k' \leq k$ would give the desired result.

We upper bound the numerator using $I_k \cap I_{k'}^* \subset I_{k'}^*$ and apply Lemma 5.4.11 to the denominator. This gives:

$$\frac{\text{Vol}(I_k \cap I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k-1}(>\tau_k)|} = O(\Delta) \cdot 2^{k-k'} \frac{\text{Vol}(I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k'-1}(>\tau_k)|} = O(\Delta) \cdot 2^{k-k'} \frac{\text{Vol}(I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k'-1}^*(>\tau_{k'})|}.$$

In the final step, we used $J_{k'-1} \subset J_{k'-1}^*$ (by Theorem 5.2.1) and $\tau_k \geq \tau_{k'}$.

Finally, we show

$$\frac{\text{Vol}(I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k'-1}^*(>\tau_{k'})|} = O(\tau_{k'}).$$

Recall that $\tau_{k'} > F^*(n-n/2^{k'})$, so Opt schedules at most one $\tau_{k'}$-big job per machine in $J_{k'}^*$. Further, Opt schedules $I_{k'}^*(\leq \tau_{k'})$ only on the $m-|J_{k'-1}^*(>\tau_{k'})|$ machines that have not yet scheduled a $\tau_{k'}$-big job yet. By averaging, after scheduling $I_{k'}^*(\leq \tau_{k'})$, every such machine in Opt has load at least

$$\frac{\text{Vol}(I_{k'}^*(\leq \tau_{k'}))}{m-|J_{k'-1}^*(>\tau_{k'})|} - \tau_{k'}.$$

One of these machines must achieve the free time $F^*(n - n/2^{k'})$, because every other machine has already scheduled a $\tau_{k'}$-big job. This implies

$$F^*(n - n/2^{k'}) \geq \frac{\mathrm{Vol}(I^*_{k'}(\leq \tau_{k'}))}{m - |J^*_{k'-1}(> \tau_{k'})|} - \tau_{k'}.$$

Rearranging and using $\tau_{k'} > F^*(n - n/2^{k'})$ give the desired result. □

Thus, the contribution of the small-in-the-past jobs to Alg's weighted free time is comparable to Opt's weighted free time, up to a $\widetilde{O}(\sqrt{m})$-factor.

5.4.5 Bounding big-in-the-past jobs

The goal of this section is to bound the big-in-the-past jobs, that is:

$$\sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(I_k \cap I^*_{k'}(> \tau_{k'}, \leq \tau_k))}{m - |J_{k-1}(> \tau_k)|}.$$

For convenience, we define $I_{kk'} = I_k \cap I^*_{k'}(> \tau_{k'}, \leq \tau_k)$. Note that we cannot apply volume arguments as in § 5.4.4, because the big-in-the-past jobs are $\tau_{k'}$-big. Instead, we will use the fact that Opt schedules at most one $I_{kk'}$-job per machine.

There are two types of jobs in $j \in I_{kk'}$: We say j is *blocked* if Opt later schedules a τ_k-big job in J_{k-1} on the same machine as j (recall that $J_{k-1} \subset J^*_{k-1}$ by Theorem 5.2.1.) Otherwise, j is *unblocked*. Further, a machine is blocked/unblocked if the $I_{kk'}$-job scheduled on that machine is blocked/unblocked. Thus we can partition $I_{kk'} = B_{kk'} \cup U_{kk'}$ into blocked and unblocked jobs, respectively.

By splitting the volume of jobs into unblocked and blocked, we can rewrite:

$$\sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(I_k \cap I^*_{k'}(> \tau_{k'}, \leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} = \sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(U_{kk'})}{m - |J_{k-1}(> \tau_k)|} + \sum_k n/2^k \cdot \frac{\mathrm{Vol}(B_{kk'})}{m - |J_{k-1}(> \tau_k)|}.$$

Intuitively, the unblocked jobs are not problematic because there can be at most $m - |J_{k-1}(> \tau_k)|$ such jobs.

Lemma 5.4.13. *The unblocked jobs satisfy* $\sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(U_{kk'})}{m - |J_{k-1}(> \tau_k)|} \leq O(\log n) \cdot \sum_k n/2^k \cdot \tau_k$.

Proof. Because there are $O(\log n)$ batches, it suffices to show for fixed k and $k' \leq k$ that

$$\frac{\mathrm{Vol}(U_{kk'})}{m - |J_{k-1}(> \tau_k)|} \leq \tau_k.$$

We recall that every job in $U_{kk'}$ is τ_k-small, so:

$$\frac{\mathrm{Vol}(U_{kk'})}{m - |J_{k-1}(> \tau_k)|} \leq \tau_k \cdot \frac{|U_{kk'}|}{m - |J_{k-1}(> \tau_k)|}.$$

We note that every job in $U_{kk'}$ is $\tau_{k'}$-big, and Opt schedules these jobs in batch $I^*_{k'}$. Thus, there is at most one $U_{kk'}$-job per unblocked machine. Further, there are at most $m - |J_{k-1}(> \tau_k)|$ unblocked machines, because each τ_k-big job in J_{k-1} must be scheduled on a separate machine of Opt (because $J_{k-1} \subset J^*_k$ by Theorem 5.2.1.) We conclude, $\frac{|U_{kk'}|}{m - |J^*_{k-1}(> \tau_k)|} \leq 1$, as required. □

It remains to handle the blocked jobs. Again, the central issue is that Opt does blocked jobs in an earlier batch before some machines get clogged. On the other hand, ChooseJobs puts these jobs in a later batch when we have fewer machines.

Unlike our previous arguments, for the blocked jobs we will charge the volume of these jobs to the completion time of Opt directly. Because these jobs are blocked, Opt must schedule a τ_k-big job later on the same machine. In particular, Opt must have kept scheduling Bernoulli jobs with size parameter at least τ_k until one comes up heads. We will charge $B_{kk'}$ to the completion time of all of these coin flips.

As before, we consider a fixed threshold τ and later union bound over all relevant thresholds. In this section, for any batch k and threshold τ, we define $p_{k\tau} \in [0, 1]$ to be the largest probability parameter across all jobs j in J_{k-1} with $s_j > \tau$ (if no such job exists, then we follow the convention $p_{k\tau} = 0$.) Note that $p_{k\tau}$ is deterministic for fixed τ. We first relate the number of remaining machines with the expected number of heads in the kth batch.

Proposition 5.4.14. *Consider any batches k, $k' \le k$ and threshold $\tau \ge 2\mathbb{E}F^*(n - n/2^k)$. Suppose Event (5.3) holds. Then we have $m - |J_{k-1}(> \tau)| \ge p_{k\tau} \cdot n/2^k - O(\Delta)$, where $\Delta = O(\sqrt{m} \log n)$.*

Proof. First, if $p_{k\tau} = 0$, then $|J_{k-1}(> \tau)| = 0$, so the proposition is trivial. Thus, we may assume $p_{k\tau} > 0$. In particular, ChooseJobs included at least one job $j \in J_{k-1}$ with $s_j > \tau$ and $p_j = p_{k\tau}$. It follows, ChooseJobs will include $n/2^k$ further jobs in I_k with size parameter larger than τ_k and probability parameter at least $p_{k\tau}$. Thus, we have $\mathbb{E}|I_k(> \tau)| \ge p_{k\tau} \cdot n/2^k$. Rewriting $\mathbb{E}|I_k(> \tau)| = \mathbb{E}|J_k(> \tau)| - \mathbb{E}|J_{k-1}(> \tau)|$ and applying Lemma 5.4.9 and Event (5.3) to the first and second expectations, respectively gives:

$$p_{k\tau} \cdot n/2^k \le \mathbb{E}|I_k(> \tau)| = \mathbb{E}|J_k(> \tau)| - \mathbb{E}|J_{k-1}(> \tau)| \le (m + O(\sqrt{m})) - (|J_{k-1}(> \tau)| - O(\Delta)).$$

Rearranging gives the desired result. □

To see the utility of Proposition 5.4.14, we assume for a moment that τ_k is deterministic and ignore the additive $O(\Delta)$ term in the proposition. Then we could rewrite $n/2^k \cdot \frac{\mathrm{Vol}(B_{kk'})}{m - |J_{k-1}(> \tau_k)|} \lesssim \frac{1}{p_{k\tau}} \cdot \mathrm{Vol}(B_{kk'})$.

To relate this expression with Opt, we note that Opt schedules a τ_k-big job on top of each job in $B_{kk'}$. In particular, Opt must schedule enough Bernoulli jobs j with $s_j > \tau_k$ until at one comes up heads on each such machine. Each such job also satisfies $p_j \le p_{k\tau_k}$, so - roughly - in Opt we expect each blocked job to delay at least $\frac{1}{p_{k\tau}}$ jobs in order for that machine to become blocked. This would give $\frac{1}{p_{k\tau_k}} \cdot \mathrm{Vol}(B_{kk'}) \lesssim \mathsf{Opt}$, as required.

5.4.6 Coin Game

It remains to formalize this idea using a martingale argument. We begin by defining an (artificial) game, which will model the process of a machine becoming blocked.

Definition 5.4.15 (Coin Game)**.** The game is played with n coins and m machines by a single player. The coins are independent such that coin j comes up heads with probability p_j. Initially, all machines are *available*. At each turn, the player can either choose to flip a previously unflipped coin on an available machine or to end the game. In the former case, if the coin comes up heads, then the machine becomes unavailable. The game ends when the player chooses to, or if we run out of unflipped coins or available machines.

Now we are ready to interpret Opt as implicitly playing a coin game to block machines.

Definition 5.4.16 (Induced Coin Game)**.** Consider pairs of batches $k' \le k$ and thresholds $\tau' \le \tau$. Then the (k', k, τ', τ)-induced coin game (with respect to policy Opt) is a distribution over coin games defined as follows:

- The machines are the ones of Opt whose final job in $J^*_{k'}$ has size exactly τ'.
- For every job in $j \in J_{k-1} \setminus J^*_{k'}$ with $s_j > \tau$, we have a coin with the same probability parameter.

The player of the coin game simulates Opt as follows. Starting from after Opt schedules $J^*_{k'}$, if Opt subsequently schedules a job on a machine that is still available (in the coin game), then the player flips the corresponding coin (if such a coin exists) on the same machine. The player decides to stop when it runs of out coins or all machines are unavailable.

One should imagine that the machines in the induced coin game are exactly those that can become blocked. Thus, a machine becoming unavailable in the coin game corresponds to it becoming blocked in Opt, and the total number of flipped coins records how many jobs were delayed by τ'.

Using a martingale argument, we relate the number of machines that become unavailable with the number of flipped coins. The next lemma formalizes the idea that to block a machine, we expect Opt to flip $\frac{1}{p_{k\tau_k}}$ coins per blocked machine. Recall that for any batch k and threshold τ, we define $p_{k\tau}$ to be the largest probability parameter across all jobs j in J_{k-1} with $s_j > \tau$.

Lemma 5.4.17. *With probability* $1 - \frac{1}{\text{poly}(n)}$*, the following event holds:*

$$\{\#(\textit{unavailable machines}) \le p_{k\tau} \cdot \#(\textit{flipped coins}) + \Delta \quad \forall\ (k', k, \tau', \tau)\textit{- induced coin games}\}, \tag{5.4}$$

where $\Delta = O(\sqrt{m} \log n)$.

Proof. Because there are $O(\log n)$ batches and $L = O(\log n)$ relevant thresholds, by union-bounding over all pairs of batches and thresholds, it suffices to show that a fixed (k', k, τ', τ)-induced coin game satisfies:

$$\mathbb{P}(\#(\text{unavailable machines}) \le p_{k\tau} \cdot \#(\text{flipped coins}) + \Delta) = 1 - \frac{1}{\text{poly}(n)}.$$

We will define a martingale to count the number of unavailable machines. For all $t \ge 0$, let A_t be the (adaptively chosen) set of the first t coins flipped by the player. If the player stops before flipping t coins, then we define $A_t = A_{t-1}$. Now consider the sequence of random variables $M_t = \sum_{j \in A_t} C_j - \sum_{j \in A_t} p_j$ for all $t \ge 0$, where $C_j \sim Ber(p_j)$ is the distribution of coin j. Note that $\sum_{j \in A_t} C_j$ is exactly the number of heads in the first t coin flips, which is the number of unavaiable machines.

We claim that M_t is a martingale. Consider any $t \ge 0$. There are two cases. If $A_t = A_{t-1}$, then $M_t = M_{t-1}$, so trivially $\mathbb{E}[M_t \mid M_{t-1}, \ldots, M_0] = M_{t-1}$. Otherwise, $A_t = A_{t-1} \cup \{j\}$ for some adaptively chosen coin j. It suffices to show the martingale property conditioned on the next coin being j for any fixed coin j:

$$\begin{aligned}\mathbb{E}[M_t \mid M_{t-1}, \ldots, M_0, A_t = A_{t-1} \cup \{j\}] &= \mathbb{E}[M_{t-1} + C_j - p_j \mid M_{t-1}, \ldots, M_0, A_t = A_{t-1} \cup \{j\}] \\ &= M_{t-1} + p_j - p_j = M_{t-1},\end{aligned}$$

as required.

To bound the deviation of M_t, we apply Freedman's inequality [Fre75] to the martingale difference sequence of M_t.

Proposition 5.4.18 (Freedman's inequality)**.** *Consider a real-valued martingale sequence $\{X_t\}_{t\geq 0}$ such that $X_0 = 0$ and $|X_t| \leq M$ almost surely for all t. Let $Y_t = \sum_{s=0}^{t} \mathbb{E}[X_s^2 \mid X_{s-1}, \ldots, X_0]$ denote the quadratic variation process of $\{X_t\}_t$. Then for any $\ell \geq 0, \sigma^2 > 0$ and stopping time τ, we have:*

$$\mathbb{P}(|\sum_{t=0}^{\tau} X_t| \geq \ell \text{ and } Y_\tau \leq \sigma^2) \leq 2 \cdot \exp\left(-\frac{\ell^2/2}{\sigma^2 + M\ell/3}\right).$$

We let X_t denote the martingale difference sequence of M_t, which is defined as $X_0 = 0$ and $X_t = M_t - M_{t-1}$ for all $t > 0$. Because M_t is a martingale, X_t is as well. Furthermore, we have $|X_t| \leq 1$ almost surely for all t. For any $t \geq 0$, we let j_t be the (adaptively chosen) tth coin flip. Then we can bound the quadratic variation process by:

$$\begin{aligned} Y_t = \sum_{s=0}^{t} \mathbb{E}[X_s^2 \mid X_{s-1}, \ldots, X_0] &= \sum_{s=0}^{t} \mathbb{E}[(C_{j_s} - p_{j_s})^2 \mid X_{s-1}, \ldots, X_0] \\ &\leq \sum_{s=0}^{t} \mathbb{E}[C_{j_s}^2 \mid X_{s-1}, \ldots, X_0] \\ &= \sum_{s=0}^{t} \mathbb{E}[C_{j_s} \mid X_{s-1}, \ldots, X_0]. \end{aligned}$$

Note that the $C_{j_1} + \ldots C_{j_t} \leq m$ almost surely, because the induced coin game has at most m machines, and any adaptive policy can flip at most one heads per machine. Thus, we have $Y_t \leq m$ for all t.

Now let T be the stopping time when the induced coin game ends, so T is exactly the number of flipped coins. Then Freedman's inequality gives:

$$\mathbb{P}(|\sum_{t=0}^{T} X_t| \geq \Delta) = \mathbb{P}(|\sum_{t=0}^{T} X_t| \geq \Delta \text{ and } Y_T \leq m) \leq 2 \cdot \exp(-\frac{\Delta^2/2}{m + \Delta/3}).$$

Taking $\Delta = O(\sqrt{m} \log n)$ gives $\mathbb{P}(|\sum_{t=0}^{T} X_t| \geq \Delta) \leq \frac{1}{\text{poly}(n)}$.

Finally, we observe that #(unavailable machines) $= \sum_{j \in A_T} C_j$. Further, we have $p_{k\tau} \cdot$#(flipped coins) $\geq \sum_{j \in A_T} p_j$, because every coin j corresponds to a job in J_{k-1} with $s_j > \tau$, so $p_j \leq p_{k\tau}$ for all coins. Thus, we conclude:

$$\mathbb{P}(\#(\text{unavailable machines}) > p_{k\tau} \cdot \#(\text{flipped coins}) + \Delta) \leq \mathbb{P}(|\sum_{t=0}^{T} X_t| > \Delta) \leq \frac{1}{\text{poly}(n)}.$$

□

Combining Proposition 5.4.14 and Lemma 5.4.17, we can bound the blocked jobs:

Lemma 5.4.19. *Suppose Events (5.3) and (5.4) hold. Then the blocked jobs satisfy:*

$$\sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\text{Vol}(B_{kk'})}{m - |J_{k-1}(> \tau_k)|} = \widetilde{O}(\sqrt{m})(\sum_k n/2^k \cdot \tau_k + \mathit{Opt}).$$

Proof. Because there are $O(\log n)$ batches k, it suffices to show for fixed $k, k' \le k$ that $n/2^k \cdot \frac{\mathrm{Vol}(B_{kk'})}{m-|J_{k-1}(>\tau_k)|} = O(\Delta \log n)(n/2^k \cdot \tau_k + \mathsf{Opt})$ for $\Delta = O(\sqrt{m}\log n)$. We consider two cases.

First, on the event that $p_{k\tau_k} = 0$, we have $m - |J_{k-1}(>\tau_k)| = m$. Recall that every job in $B_{kk'}$ is $\tau_{k'}$-big and in $I^*_{k'}$, so there is at most one such job per machine in Opt. Then we can bound:

$$n/2^k \cdot \frac{\mathrm{Vol}(B_{kk'})}{m - |J_{k-1}(>\tau_k)|} \le n/2^k \cdot \tau_k \frac{m}{m} = n/2^k \cdot \tau_k.$$

Otherwise, we have $p_{k\tau_k} > 0$. Here we related the blocked jobs to the induced coin games:

$$\begin{aligned}
\mathrm{Vol}(B_{kk'}) &= \sum_{\tau' \le \tau_k} \tau' \cdot |\{j \in B_{kk'} \mid X_j = \tau'\}| \\
&= \sum_{\tau' \le \tau_k} \tau' \cdot \#(\text{unavailable machines in } (k', k, \tau', \tau_k)\text{-induced coin game}) \\
&\le \sum_{\tau' \le \tau_k} \tau' \cdot (p_{k\tau_k} \cdot \#(\text{flipped coins in } (k', k, \tau', \tau_k)\text{-induced coin game}) + \Delta) \\
&\le O(\log n) \cdot p_{k\tau_k} \cdot \mathsf{Opt} + O(\Delta \log n) \cdot \tau_k.
\end{aligned}$$

where the first inequality follows from Event (5.4). The second follows because there are $O(\log n)$ relevant thresholds $\tau' \le \tau_k$, and every flipped coin in the (k', k, τ', τ_k)-induced coin game corresponds to Opt scheduling a job on a machine that already scheduled some job with size τ', so every such job has completion time at least τ'. It follows:

$$n/2^k \cdot \frac{\mathrm{Vol}(B_{kk'})}{m - |J_{k-1}(>\tau_k)|} \le n/2^k \cdot O(\log n) \frac{p_{k\tau_k}}{m - |J_{k-1}(>\tau_k)|} \cdot \mathsf{Opt} + n/2^k \cdot O(\Delta \log n) \frac{\tau_k}{m - |J_{k-1}|}.$$

By Proposition 5.4.14, we can bound the first term by:

$$\begin{aligned}
n/2^k \cdot O(\log n) \frac{p_{k\tau_k}}{m - |J_{k-1}(>\tau_k)|} \cdot \mathsf{Opt} &= O(\log n) \frac{m - |J_{k-1}(>\tau_k)| + O(\Delta)}{m - |J_{k-1}(>\tau_k)|} \cdot \mathsf{Opt} \\
&= O(\Delta \log n) \cdot \mathsf{Opt}.
\end{aligned}$$

We can bound the second term by:

$$n/2^k \cdot O(\Delta \log n) \frac{\tau_k}{m - |J_{k-1}(>\tau_k)|} = O(\Delta \log n) \cdot n/2^k \cdot \tau_k.$$

Combining both bounds completes the proof. □

We summarize our bounds for the unblocked and blocked jobs by the next lemma, which follows immediately from Lemma 5.4.13 and Lemma 5.4.19.

Lemma 5.4.20. *Suppose Events (5.3) and (5.4) hold. Then the big-in-the-past jobs satisfy:*

$$\sum_k n/2^k \cdot \sum_{k' \le k} \frac{\mathrm{Vol}(I_k \cap I^*_{k'}(>\tau_{k'}, \le \tau_k))}{m - |J_{k-1}(>\tau_k)|} = \widetilde{O}(\sqrt{m}) \cdot \left(\sum_k n/2^k \cdot \tau_k + \mathsf{Opt} \right).$$

5.4.7 Putting it all together

We are ready to prove Theorem 5.3.4 and Theorem 5.4.2, which we restate here for convenience.

Theorem 5.3.4 (Batch Free Time Minimization)**.** *Given Bernoulli jobs, if $m \geq 2$ and Assumption 5.3.2 holds, then* StochFree *outputs a list schedule with expected completion time at most $\widetilde{O}(\sqrt{m}) \cdot \big(\mathbb{E}[\mathsf{Opt}] + O(1)\big)$, where Opt is the optimal adaptive policy.*

Theorem 5.4.2. *If $m = \Omega(1)$ is sufficiently large, then the weighted free time of Alg satisfies:*

$$\mathbb{E}\Big[\sum_k n/2^k \cdot F(J_k)\Big] = \widetilde{O}(\sqrt{m}) \cdot \left(\mathbb{E}\Big[\sum_k n/2^k \cdot F^*(n - n/2^k)\Big] + \mathbb{E}[\mathsf{Opt}]\right) + O(1).$$

Theorem 5.4.2 follows from partitioning Alg's weighted free time into the contribution due to small-in-the-past and big-in-the-past jobs (which we further partitioned into unblocked and blocked jobs.)

Proof of Theorem 5.4.2. We assume $m = \Omega(1)$ is sufficiently large. Then we complete the proof of Theorem 5.4.2 by combining our bounds for the small-in-the-past- and big-in-the-past jobs. We bound Alg's weighted free time by Lemma 5.4.4:

$$\sum_k n/2^k \cdot F(J_k) = O(\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} + \sum_k n/2^k \cdot \tau_k).$$

We recall $\tau_k = 2 \cdot \max(\mathbb{E}F^*(n - n/2^k), F^*(n - n/2^k))$, so $\mathbb{E}\tau_k = O(\mathbb{E}F^*(n - n/2^k))$. Thus, in expectation, the second sum is at most:

$$\mathbb{E}\sum_k n/2^k \cdot \tau_k = O(\mathbb{E}\sum_k n/2^k \cdot F^*(n - n/2^k)).$$

It remains to bound the first sum, which we split into the contribution due to small-in-the-past and big-in-the-past jobs:

$$\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} = \sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(I_k \cap I^*_{k'}(\leq \tau_{k'}))}{m - |J_{k-1}(> \tau_k)|} + \sum_k n/2^k \cdot \sum_{k' \leq k} \frac{\mathrm{Vol}(I_k \cap I^*_{k'}(> \tau_{k'}, \leq \tau_k))}{m - |J_{k-1}(> \tau_k)|}.$$

On Events (5.3) and (5.4), we can apply Lemma 5.4.12 to the first term and Lemma 5.4.20 to the second to obtain:

$$\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} = \widetilde{O}(\sqrt{m})(\sum_k n/2^k \cdot \tau_k + \mathsf{Opt}).$$

Again, in expectation, this contributes $\widetilde{O}(\sqrt{m})(\mathbb{E}\sum_k n/2^k \cdot F^*(n - n/2^k) + \mathbb{E}\mathsf{Opt})$ to Alg's expected weighted free time.

Finally, we consider the event that Event (5.3) or Event (5.4) does not hold. Recall that by Lemma 5.4.10 and Lemma 5.4.17, this happens with probability at most $\frac{1}{\mathrm{poly}(n)}$ because $m = \Omega(1)$ is sufficiently large. Further, on this event, we can trivially upper bound $\sum_k n/2^k \cdot \frac{\mathrm{Vol}(I_k(\leq \tau_k))}{m - |J_{k-1}(> \tau_k)|} = \mathrm{poly}(n)$, because there are n jobs each with size at most $\mathrm{poly}(n)$ almost surely. Thus, the contribution of this event to the overall expectation is $O(1)$. We conclude, Alg's expected weighted free time is at most $\widetilde{O}(\sqrt{m})(\mathbb{E}\sum_k n/2^k \cdot F^*(n - n/2^k) + \mathbb{E}\mathsf{Opt}) + O(1)$. □

To complete the proof of Theorem 5.4.2, we relate the weighted free time to the completion time. We also use our warm-up $\tilde{O}(m)$-approximation when m is too small to apply Theorem 5.3.4.

Proof of Theorem 5.3.4. First, suppose $m = \Omega(1)$ is sufficiently large. Then by Theorem 5.4.2, Alg's weighted free time satisfies:

$$\mathbb{E}\left[\sum_k n/2^k \cdot F(J_k)\right] = \widetilde{O}(\sqrt{m}) \cdot \left(\mathbb{E}\left[\sum_k n/2^k \cdot F^*(n - n/2^k)\right] + \mathbb{E}[\mathsf{Opt}]\right) + O(1).$$

Applying Lemma 5.4.1 to relate weighted free time to completion time gives:

$$\mathbb{E}[\mathsf{Alg}] = \widetilde{O}(1) \cdot \mathbb{E}\left[\sum_k n/2^k \cdot F(J_k)\right] + \widetilde{O}(\mathbb{E}[\mathsf{Opt}]) = \widetilde{O}(\sqrt{m}) \cdot \Big(\mathbb{E}[\mathsf{Opt}] + O(1)\Big).$$

This gives the desired guarantee if $m = \Omega(1)$.

Otherwise, if $m = O(1)$, Lemma 5.4.6 immediately gives that StochFree is $\widetilde{O}(m) = \widetilde{O}(1)$-approximate, so $\mathbb{E}[\mathsf{Alg}] = \widetilde{O}(\mathbb{E}[\mathsf{Opt}])$. □

This completes the analysis of StochFree. Since the proof had several conceptual parts, let us a give a quick summary.

Summary

Recall that our analysis began by passing from completion time to our new proxy objective: weighted free time in § 5.4.1. As we mentioned earlier, a key benefit of working with free times rather than completion times was that we could completely control what jobs we *started* to achieve the ith free time, whereas we have far less control over the first i jobs to *finish*. This allowed us to make the contribution of each job to the weighted free time more modular: either the job contributed to the small volume in a batch, or it contributed to the clogged machines—see (5.2). We then controlled the rate at which Alg and Opt clog up machines in § 5.4.3. Then in § 5.4.4-§ 5.4.6 we compared the times at which Alg and Opt chose to do the same volume of small jobs. Since these were the only two ways in which a job affected the weighted free time, we could combine these two ideas in § 5.4.7 to complete our analysis.

5.5 Conclusion

We gave an improved approximation for stochastic completion times, which does not depend on the job size variances, and has a sublinear dependence on the number of machines m.

Similar to Chapter 4, the key idea enabling our algorithm is a deeper technical understanding of adaptive policies. However, unlike in the case of load balancing, we cannot rely on tried-and-true techniques such as index policies and LP rounding due to known lower bounds for this problem. Instead, we introduce a variety of new ideas such as batched free times and the techniques to analyze them.

The most compelling open question is extending our techniques to more general job size distributions than Bernoulli jobs. Observe that the weighted free time is a valid proxy objective for *any* job size distributions, not just Bernoulli jobs, so extending our result to general stochastic jobs requires us to solve subset selection and batch free time minimization for these settings.

Second, we also do not have a good grasp on the complexity of this problem: is the stochastic problem provably hard to solve/approximate? Typically, for stochastic combinatorial optimization

problems such as load balancing and knapsack, we do not have a toolkit to show hardness other than stating that they generalize their deterministic counterparts, which are already hard. This approach does not tell us anything for minimizing completion times, however, because the deterministic version can be solved optimally in polynomial time.

Chapter 6

Conclusion

In this dissertation, we studied four fundamental problems combinatorial optimization. In the first two chapters, Chapter 2 and Chapter 3, we refined two classical approaches: iterative LP rounding and amortization arguments, leading to improved approximations for k-median-type problems and online throughput maximization. In the final two chapters, Chapter 4 and Chapter 5, we moved our focus to stochastic combinatorial optimization. These problems are not as well understood as their deterministic counterparts, and the respective technical toolkit is less developed. In Chapter 4, with some work, we were able to connect stochastic load balancing with classical LP rounding and potential function approaches by defining an appropriate deterministic proxy for the stochastic problem. However, in Chapter 5, the classical tools provably do not work when considering stochastic completion time minimization. Thus, we had to develop a variety of new ideas such as a new proxy objective and novel probabilistic arguments combining per-realization and concentration guarantees to break through previous barriers.

We already covered concrete open problems regarding each problem in the conclusions of their respective chapters. Instead I want to look ahead to the more speculative directions that inspire my future work. The main question that I am interested in is:

How much information do we really need to solve a particular optimization problem?

We already saw some examples of this question: in online scheduling, how much do we lose by learning about the jobs incrementally over time rather than having access to all of the information up front (competitive analysis)? In stochastic scheduling, how much do we lose by only using the initial distributional information to guide our algorithm rather than adaptively responding to the realized job sizes (adaptivity gaps)? Some other aspects of this question that I am interested in are the following:

- In our stochastic algorithms, we assumed we had oracle access to the given distributions. In particular, our load balancing algorithms required truncated means (due to a guess-and-double argument) for a wide range of truncation values. This effectively means our algorithm needs arbitrary CDF access (or a prohibitively large number of samples to estimate very unlikely truncation values). Algorithms for other stochastic problems often need similarly rich information about the distributions including upper tails and exponential moments [KRT00, IMP15b, Mol19, GKNS21]. How well can algorithms perform with, say very limited sample access to the distributions, or limited statistics about the distributions (means, low degree moments, etc.)?

- We have focused on improved *upper bounds* for stochastic scheduling, but what about lower bounds? Can we give any evidence that, say stochastic load balancing, is any harder than its deterministic counterpart? Perhaps the ideal result would be a hardness of approximation result of the form "Unless *(some complexity assumption)*, stochastic load balancing is hard to approximate within a factor *(some factor larger than* $\frac{3}{2}$*)*[1]". However, this goal seems far away given our current understanding of these problems.

  Note that many adaptivity gap lower bounds and hard examples for stochastic optimization problems rely on very simple discrete distributions (usually Bernoulli or two-point distributions). For such distributions, we can often represent an adaptive policy as a decision tree, where each node corresponds to a decision of the policy and the child edges correspond to the possible realizations of that decision. With this observation, can we show a lower bound on the size of any decision tree representing a (near) optimal adaptive policy? Could we show this indirectly by arguing that such a decision tree must compute some hard-to-compute function? This is just one very speculative approach, but I think the general question is exciting and completely open.

- Another idea that we briefly touched on is *non-clairvoyant algorithms*. For example, we considered load balancing on related machines, where we do not even know the job sizes at all until they have been scheduled. It seems that nothing should be possible in this setting, but surprisingly we can give non-trivial performance guarantees. I am curious how far we can push this idea of algorithms with very limited information or feedback about the problem. Can we study other combinatorial optimization problems where we remove seemingly necessary information about the problem, and still give non-trivial algorithms?

More generally, I'm interested in simple (elegant?) optimization problems with restricted information.

This concludes my dissertation. I hope my more technically-inclined readers learned something interesting and my less technically-inclined readers have a better sense of what I do all day.

[1] This is because $\frac{3}{2}$ is the known hardness for deterministic load balancing on unrelated machines unless $P = NP$ [LST90]

Bibliography

[AAF+97] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[AAKZ22] Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. *Math. Program.*, 192(1):3–27, 2022.

[AGK+04] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

[Alb03] Susanne Albers. Online algorithms: a survey. *Math. Program.*, 97(1-2):3–26, 2003.

[ANR95] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995.

[BCK00] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35(1):108–121, 2000.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[Ben62] George Bennett. Probability inequalities for the sum of independent random variables. *JASA*, 57(297):33–45, 1962.

[BIPV19] Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi R. Varadarajan. A constant approximation for colorful k-center. In *27th Annual European Symposium on Algorithms, ESA*, 2019.

[BJS74] John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974.

[BKM+92] Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real Time Systems*, 4(2):125–144, 1992.

[BPR+17] Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017.

[BPR+18] Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Joachim Spoerhase, Aravind Srinivasan, and Khoa Trinh. An improved approximation algorithm for knapsack median using sparsification. *Algorithmica*, 80(4):1093–1114, 2018.

[CGKT07] Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. Hardness of routing with congestion in directed graphs. In *Proceedings of STOC*, pages 165–178. ACM, 2007.

[CGLS23] Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 LMP approximation barrier for facility location with applications to k-median. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 940–986. SIAM, 2023.

[Che08] Ke Chen. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA*, pages 826–835, 2008.

[CKMN01] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 642–651, 2001.

[Den22] Shichuan Deng. On clustering with discounts. *Inf. Process. Lett.*, 177:106272, 2022.

[DGV08] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Math. Oper. Res.*, 33(4):945–964, 2008.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[EFMM19] Franziska Eberle, Felix A. Fischer, Jannik Matuschke, and Nicole Megow. On index policies for stochastic minsum scheduling. *Oper. Res. Lett.*, 47(3):213–218, 2019.

[EGM+22] Franziska Eberle, Anupam Gupta, Nicole Megow, Benjamin Moseley, and Rudy Zhou. Configuration balancing for stochastic requests. *CoRR*, abs/2208.13702, 2022.

[EMS20] Franziska Eberle, Nicole Megow, and Kevin Schewior. Optimally handling commitment issues in online throughput maximization. In *European Symposium on Algorithms*, 2020.

[FKRS19] Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. *ACM Trans. Algorithms*, 15(2):26:1–26:26, 2019.

[Fre75] David A. Freedman. On tail probabilities for Martingales. *The Annals of Probability*, 3(1):100–118, 1975.

[GI99] Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS*, pages 579–586, 1999.

[GJS23] Paritosh Garg, Linus Jordan, and Ola Svensson. Semi-streaming algorithms for submodular matroid intersection. *Math. Program.*, 197(2):967–990, 2023.

[GKNS21] Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. *Math. Oper. Res.*, 46(1):115–133, 2021.

[GLZ17] Sudipto Guha, Yi Li, and Qin Zhang. Distributed partial clustering. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 143–152. ACM, 2017.

[GMM+03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *TKDE*, 15(3):515–528, 2003.

[GMUX20] Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Greed works - online algorithms for unrelated machine stochastic scheduling. *Math. Oper. Res.*, 45(2):497–516, 2020.

[GMZ21] Anupam Gupta, Benjamin Moseley, and Rudy Zhou. Structural iterative rounding for generalized k-median problems. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, 2021.

[GMZ23] Anupam Gupta, Benjamin Moseley, and Rudy Zhou. Minimizing completion times for stochastic jobs via batched free times. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1905–1930. SIAM, 2023.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[GRSZ14] Fabrizio Grandoni, R. Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Math. Program.*, 146(1-2):525–554, 2014.

[HSSW97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997.

[IKPS18] Sungjin Im, Nathaniel Kell, Debmalya Panigrahi, and Maryam Shadloo. Online load balancing on related machines. In *Proceedings of STOC*, pages 30–43. ACM, 2018.

[IMP15a] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Stochastic scheduling of heavy-tailed jobs. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS*, 2015.

[IMP15b] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Stochastic scheduling of heavy-tailed jobs. In Ernst W. Mayr and Nicolas Ollinger, editors, *Proceedings of STACS*, volume 30, pages 474–486, 2015.

[IQM+20] Sungjin Im, Mahshid Montazer Qaem, Benjamin Moseley, Xiaorui Sun, and Rudy Zhou. Fast noise removal for k-means clustering. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 456–466, 2020.

[JMS02] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 731–740, 2002.

[JV01] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[KKN+15] Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. Facility location with matroid or knapsack constraints. *Math. Oper. Res.*, 40(2):446–459, 2015.

[KLS18] Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 646–659, 2018.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[KP01] Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *Journal of Algorithms*, 38(1):2–24, 2001.

[KP03] Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *Journal of Algorithms*, 49(1):63–85, 2003.

[KRT00] Jon M. Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217, 2000.

[LG18] Shi Li and Xiangyu Guo. Distributed k-clustering for data with heavy noise. In *Advances in Neural Information Processing Systems*, pages 7838–7846, 2018.

[LMNY13] Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 305–314. ACM, 2013.

[LRS98] Tom Leighton, Satish Rao, and Aravind Srinivasan. Multicommodity flow and circuit switching. In *HICSS (7)*, pages 459–465. IEEE Computer Society, 1998.

[LS16] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016.

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

[MKC+15] Gustavo Malkomes, Matt Kusner, Wenlin Chen, Kilian Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *NIPS*, pages 1063–1071, 2015.

[Mol19] Marco Molinaro. Stochastic p load balancing and moment problems via the l-function method. In *Proceedings of SODA*, pages 343–354. SIAM, 2019.

[MPSZ22] Benjamin Moseley, Kirk Pruhs, Clifford Stein, and Rudy Zhou. A competitive algorithm for throughput maximization on identical machines. In *Integer Programming and Combinatorial Optimization, IPCO*, 2022.

[MSU99] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *J. ACM*, 46(6):924–942, 1999.

[Pin08] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[PST04] Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[Rot66] Michael H. Rothkopf. Scheduling with random service times. *Management Science*, 12(9):707–713, 1966.

[Sch08] Andreas S. Schulz. Stochastic online scheduling revisited. In Boting Yang, Ding-Zhu Du, and Cao An Wang, editors, *Proceedings of COCOA*, volume 5165 of *LNCS*, pages 448–457. Springer, 2008.

[SSU16a] Martin Skutella, Maxim Sviridenko, and Marc Uetz. Unrelated machine scheduling with stochastic processing times. *Math. Oper. Res.*, 41(3):851–864, 2016.

[SSU16b] Martin Skutella, Maxim Sviridenko, and Marc Uetz. Unrelated machine scheduling with stochastic processing times. *Math. Oper. Res.*, 41(3):851–864, 2016.

[ST93] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.

[SU05] Martin Skutella and Marc Uetz. Stochastic machine scheduling with precedence constraints. *SIAM J. Comput.*, 34(4):788–802, 2005.

[WP80] Gideon Weiss and Michael Pinedo. Scheduling tasks with exponential service times on non-identical processors to minimize various cost functions. *Journal of Applied Probability*, 17(1):187–202, 1980.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[WVW86] R. R. Weber, P. Varaiya, and J. Walrand. Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *Journal of Applied Probability*, 23(3):841–847, 1986.

Appendix A

Appendix for Generalized k-Median Problems

A.1 Missing Proofs from § 2.2: Construction of LP_{iter}

Proof of Proposition 2.2.1. Let $\mathcal{I}$ be the given instance of GKM and (x^*, y^*) be an optimal solution to LP_1.

Observe that if $x^*_{ij} \in \{0, y^*_i\}$ for all $i \in F, j \in C$, then we can define $F_j = \{i \in F \mid x^*_{ij} > 0\}$ for all $j \in C$. It is easy to verify in this case that y^* is feasible for LP_2 and achieves the same objective value in LP_2 as (x^*, y^*) achieves in LP_1, which completes the proof.

Thus our goal is to duplicate facilities in F and re-allocate the x- and y-values appropriately until $x^*_{ij} \in \{0, y^*_i\}$ for all $i \in F, j \in C$. To prevent confusion, let F denote the original set of facilities, and let F' denote the modified set of facilities, where make $n = |C|$ copies of each facility in F, so for each $i \in F$, we have copies $i_1, \ldots, i_n \in F'$.

Now we define $x' \in [0,1]^{F' \times C}$ and $y' \in [0,1]^{F'}$ with the desired properties. For each $i \in F$, we assume without loss of generality that $0 \le x_{i1} \le x_{i2} \le \cdots \le x_{in} \le y_i$. We define $x'_{i_1 1}, \ldots, x'_{i_n n}$ and $y'_{i_1}, \ldots, y'_{i_n}$ recursively:

Let $y'_{i_1} = x_{i1}$ and $x'_{i_1 j} = x_{ij}$ for all $j \in [n]$.

Now for $k > 1$, let $y'_{i_k} = x_{ik} - x_{i(k-1)}$ and $x'_{i_k j} = \begin{cases} 0 & , j < k \\ y'_{i_k} & , j \ge k \end{cases}$ for all $j \in [n]$.

It is easy to verify that (x', y') is feasible for LP_1 (after duplicating facilities) and $x'_{ij} \in \{0, y'_i\}$ for all $i \in F', j \in C$, as required. Further, it is clear that this algorithm is polynomial time. □

Proof of Proposition 2.2.2. If $d(p,q) = 0$, then the claim is trivial. Suppose $d(p,q) \ge 1$. We can rewrite $d(p,q) = \tau^{\ell+f}$ for some $\ell \in \mathbb{N}, f \in [0,1)$. Also, for convenience we define $\beta = \log_\tau \alpha$. Because $\log_e \alpha$ is uniformly distributed in $[0, \log_e \tau)$, it follows that β is uniformly distributed in $[0,1)$.

It follows, $d(p,q)$ is rounded to $\alpha\tau^\ell = \tau^{\ell+\beta}$ exactly when $\beta \ge f$, and otherwise $d(p,q)$ is rounded to $\tau^{\ell+\beta+1}$ when $\beta < f$. Thus we compute:

$$\mathbb{E}[d'(p,q)] = \int_{\beta=0}^{f} \tau^{\ell+\beta+1}\, d\beta + \int_{\beta=f}^{1} \tau^{\ell+\beta}\, d\beta = \frac{1}{\log_e \tau}(\tau^{\ell+\beta+1}|_{\beta=0}^{f} + \tau^{\ell+\beta}|_{\beta=f}^{1})$$
$$= \frac{1}{\log_e \tau}(\tau^{\ell+f+1} - \tau^{\ell+1} + \tau^{\ell+1} - \tau^{\ell+f}) = \frac{1}{\log_e \tau}(\tau^{\ell+f+1} - \tau^{\ell+f}) = \frac{\tau-1}{\log_e \tau} d(p,q).$$

□

A.2 Missing Proofs from § 2.6

Proof of Lemma 2.6.5. We let $i^* \in S$ be the closest facility to i in S. We show that the cost of connecting C' to i^* is at most $O(\frac{\rho}{\delta})U$. To do so, we partition C' into two sets of clients: those that are far from i relative to $d(i,i^*)$, and those that are close to i. In particular, let $\gamma > 0$ be a constant that we choose later. Then we partition C' into C'_{far} and C'_{close}, where:

$$C'_{far} = \{j \in C' \mid d(j,i) \geq \gamma d(i,i^*)\},$$

and

$$C'_{close} = \{j \in C' \mid d(j,i) > \gamma d(i,i^*)\}.$$

First we bound the connection cost of C'_{far} to i^* using the fact that $i \notin S_0$, so Extra Invariant (2) says that $\sum_{j|i\in F_j} d(i,j) \leq 2\rho U$. Thus we compute:

$$\sum_{j \in C'_{far}} d(j,i^*) \leq (1+\frac{1}{\gamma}) \sum_{j \in C | i \in F_j} d(j,i) \leq (1+\frac{1}{\gamma}) O(\rho) U$$

Now suppose $C'_{close} \neq \emptyset$. Fix any $j^* \in C'_{close}$. Then for all $j \in C'_{close}$, we have $d(j,j^*) \leq d(j,i) + d(j^*,i) \leq 2\gamma d(i,i^*)$. It follows that $C'_{close} \subset B_C(j^*, 2\gamma d(i,i^*))$. Our strategy is to use Extra Invariant (4) $|B_C(j^*,\delta r)|r \leq \rho U$, for all $r \leq R_{j^*}$. Thus we want $2\gamma d(i,i^*) \leq \delta R_{j^*}$. To lower bound R_{j^*} with respect to $d(i,i^*)$, we use our assumption that there exists some $\bar{i} \in S$ such that $d(j^*,\bar{i}) \leq \alpha L(\ell_{j^*})$, where $L(\ell_{j^*}) \leq \tau R_{j^*}$ by Extra Invariant (4). Thus we have:

$$d(j^*,\bar{i}) \leq \alpha\tau R_{j^*}.$$

Further, using the triangle inequality and the fact that i^* is the closest facility to i in S, we have:

$$d(j^*,\bar{i}) \geq d(i,\bar{i}) - d(i,j^*) \geq d(i,i^*) - d(i,j^*) \geq (1-\gamma) d(i,i^*).$$

Combining these two inequalities gives the lower bound $R_{j^*} \geq \frac{1-\gamma}{\alpha\tau} d(i,i^*)$.

Now we are ready to choose γ. Recall that we want $2\gamma d(i,i^*) \leq \delta R_{j^*}$, so it suffices to choose γ such that: $2\gamma d(i,i^*) \leq \delta \frac{1-\gamma}{\alpha\tau} d(i,i^*)$. Routine calculations show that we can take $\gamma = \Theta(\delta)$ to satisfy this inequality. Now with this choice of γ, we can bound:

$$\sum_{j \in C'_{close}} d(j,i^*) \leq (1+\gamma)|C'_{close}| d(i,i^*) \leq (1+\gamma)|B_C(j^*, \delta R_{j^*})| d(i,i^*)$$
$$\leq (1+\gamma)\frac{\rho U}{R_{j^*}} d(i,i^*) \leq \frac{\rho U}{R_{j^*}} O(1) R_{j^*} = O(\rho) U$$

To conclude the proof, the connection cost of C'_{far} is at most $(1+\frac{1}{\gamma})O(\rho)U = O(\frac{\rho}{\delta})U$ and the connection cost of C'_{close} is at most $O(\rho)U$. Summing these costs gives the desired result. □

Proof of Lemma 2.6.13. We let $i^* \in S$ be the closest facility to i in S. We show that the cost of connecting C' to i^* is at most $O(\frac{\rho}{\delta})U$. To do so, we partition C' into two sets of clients: those that are far from i relative to $d(i, i^*)$, and those that are close to i. In particular, let $\gamma > 0$ be a constant that we choose later. Then we partition C' into C'_{far} and C'_{close}, where:

$$C'_{far} = \{j \in C' \mid d(j,i) \geq \gamma d(i,i^*)\} \text{ and } C'_{close} = \{j \in C' \mid d(j,i) > \gamma d(i,i^*)\}$$

First we bound the connection cost of C'_{far} to i^* using the fact that $i \notin S_0$, so Extra Invariant (2) says that i is cheap. Thus we compute:

$$\sum_{j \in C'_{far}} d(j,i^*) \leq (1 + \frac{1}{\gamma}) \sum_{j \in C | i \in F_j} d(j,i) \leq (1 + \frac{1}{\gamma}) O(\rho) U$$

Now suppose $C'_{close} \neq \emptyset$. Importantly, all of these clients are within distance $\gamma d(i,i^*)$ of i, so we have $C'_{close} \subset B_C(i, \gamma d(i,i^*))$. Our strategy to bound the connection cost of C'_{close} is to leverage Extra Invariant (4), so in particular we want to use the fact $|\{j \in B_C(i, \frac{\delta t}{4+3\delta}) \mid R_j \geq t\}| \leq \frac{\rho(1+3\delta/4)}{1-\delta/4} \frac{U}{t}$ for any $t > 0$. We want to choose $\gamma, t > 0$ such that $\gamma d(i,i^*) \leq \frac{\delta t}{4+3+\delta}$ and $R_j \geq t$ for all $j \in C'_{close}$. To see why this is useful, for such γ and t, we have $C'_{close} \subset \{j \in B_C(i, \frac{\delta t}{4+3\delta}) \mid R_j \geq t\}$. Then we can bound:

$$\begin{aligned}\sum_{j \in C'_{close}} d(j,i^*) &\leq \sum_{j \in C'_{close}} (1+\gamma) d(i,i^*) = (1+\gamma)|C'_{close}| d(i,i^*) \\ &\leq (1+\gamma)|\{j \in B_C(i, \frac{\delta t}{4+3\delta}) \mid R_j \geq t\}| d(i,i^*) \leq (1+\gamma)(\frac{\rho(1+3\delta/4)}{1-\delta/4}\frac{U}{t}) d(i,i^*)\end{aligned}$$

Now we go back and specify our choice of γ and t, which will allow us to complete the bound of the connection costs. First we lower bound R_j in terms of $d(i,i^*)$ for any $j \in C'_{close}$. We recall that by assumption there exists some $\bar{i} \in S$ such that $d(j,\bar{i}) \leq \alpha L(\ell_j)$, where $L(\ell_j) \leq \tau R_j$ by Extra Invariant (4). Thus we have: $d(j,\bar{i}) \leq \alpha\tau R_j$. Further, using the triangle inequality and the fact that i^* is the closest facility to i in S, we have: $d(j,\bar{i}) \geq d(i,\bar{i}) - d(i,j) \geq d(i,i^*) - d(i,j) \geq (1-\gamma) d(i,i^*)$.

Combining this inequality with the upper bound on $d(j,\bar{i})$ gives that $R_j \geq \frac{1-\gamma}{\alpha\tau} d(i,i^*)$ for all $j \in C'_{close}$. Then we define $t = \frac{1-\gamma}{\alpha\tau} d(i,i^*)$. This gives us $R_j \geq t$ for all $j \in C'_{close}$. Now we can choose $\gamma > 0$ satisfying: $\gamma d(i,i^*) \leq \frac{\delta t}{4+3\delta} \Rightarrow \gamma \leq \frac{\delta}{4+3\delta}\frac{1-\gamma}{\alpha\tau}$. Taking $\gamma = \frac{\delta}{12\alpha\tau} = \Theta(\delta)$ suffices.

Using these choices of γ and t, we can bound $\sum_{j \in C'_{far}} d(j,i^*) = O(\frac{\rho}{\delta})U$, and $\sum_{j \in C'_{close}} d(j,i^*) \leq (1+\gamma)(\frac{\rho(1+3\delta/4)}{1-\delta/4}U)(\frac{\alpha\tau}{1-\gamma}) = O(\rho)U$. Summing these two costs gives the desired result. □

Proof of Theorem 2.1.3. Let $\epsilon' > 0$. We will later choose ϵ' with respect to the given ϵ to obtain the desired approximation ratio and runtime. First, we choose parameters $\rho, \delta \in (0, 1/2)$ and $U \geq 0$ for our pre-processing algorithm guaranteed by Theorem 2.6.3. We take $\rho = \epsilon'^2$ and $\delta = \epsilon'$. We require that U is an upper bound on $\mathsf{Opt}(\mathcal{I})$. Using a standard binary search idea, we can guess $\mathsf{Opt}(\mathcal{I})$ up to a multiplicative $(1+\epsilon')$-factor in time $n^{O(1/\epsilon')}$, so we guess U such that $\mathsf{Opt}(\mathcal{I}) \leq U \leq (1+\epsilon')\mathsf{Opt}(\mathcal{I})$.

With these choices of parameters, we run the algorithm guaranteed by Theorem 2.6.11 to obtain $n^{O(1/\epsilon')}$ many sub-instances such that one such sub-instance is of the form $\mathcal{I}' = (F, C' \subset C, d, k, m' = m - |C^* \setminus C'|)$, where LP_{iter} for $\mathcal{I}'$ satisfies all Basic- and Extra Invariants, and we have:

$$\frac{\log_e \tau}{(\tau-1)(1+\epsilon'/2)}\mathbb{E}[\mathsf{Opt}(LP_{iter})] + \frac{1-\epsilon'}{1+\epsilon'}\sum_{j\in C^*\setminus C'} d(j,S_0) \le U \tag{A.1}$$

Then for each sub-instance output by the pre-processing, we run the algorithm guaranteed by Theorem 2.6.14 to obtain a solution to each sub-instance. Finally, out of these solutions, we output the one that is feasible for the whole instance with smallest cost. This completes the description of our approximation algorithm for k-median with outliers. The runtime is $n^{O(1/\epsilon')}$, so it remains to bound the cost of the output solution and to choose the parameters ϵ' and τ and c.

It suffices to consider the solution output on the instance $\mathcal{I}'$ where LP_{iter} satisfies all Basic- and Extra Invariants and (A.1). By running the algorithm guaranteed by Theorem 2.6.14 on this LP_{iter}, we obtain a feasible solution $S \subset F$ to $\mathcal{I}'$ such that $S_0 \subset S$, and the cost of connecting m' clients from C' to S is at most $(2+\alpha)\mathsf{Opt}(LP_{iter})+O(\epsilon')U$, where $\alpha = \max(3+2\tau^{-c}, 1+\frac{4+2\tau^{-c}}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1})$. To extend this solution on the sub-instance to a solution on the whole instance $\mathcal{I}$, we must connect $m-m' = |C^*\setminus C'|$ clients from $C\setminus C'$ to S. Because $S_0 \subset S$, applying (2.1) allows us to upper bound the expected cost of connecting m clients to S by: $(2+\alpha)\mathbb{E}[\mathsf{Opt}(LP_{iter})]+O(\epsilon')U+\sum_{j\in C^*\setminus C'} d(j,S_0) \le$ $(2+\alpha)\frac{\tau-1}{\log_e \tau}\frac{(1+\epsilon')^2}{1-\epsilon'}U + O(\epsilon')U$.

Now choosing $\tau > 1$ to minimize $\alpha' = (2+\max(3, 1+\frac{4}{\tau}, \frac{\tau^3+2\tau^2+1}{\tau^3-1}))\frac{\tau-1}{\log_e \tau}$ (note that we ignore the $2\tau^{-c}$ terms), we obtain $\tau = 1.2074$ and $\alpha' = 6.947$. We can choose $c \ge 1$ sufficiently large with respect to τ such that $2\tau^{-c}$ is sufficiently small to guarantee $(2+\alpha)\frac{\tau-1}{\log_e \tau} \le 6.947+\epsilon'$

Thus the expected cost of this solution is at most $(6.947+\epsilon')\frac{(1+\epsilon')^2}{1-\epsilon'}U + O(\epsilon')U$, where $U \le (1+\epsilon')\mathsf{Opt}(\mathcal{I})$. Finally, by routine calculations, we can choose $\epsilon' = \theta(\epsilon)$ so that expected cost is at most $(6.947+\epsilon)\mathsf{Opt}(\mathcal{I})$, as required. Note that the runtime of our algorithm is $n^{O(1/\epsilon')} = n^{O(1/\epsilon)}$. □

A.3 Missing Proofs from Analysis of OutliersPostProcess

Proof of Lemma 2.7.2. Without loss of generality, we may assume that no facilities in S_0 are co-located with each other, so $\{F_j \mid j \in C_0\}$ is a disjoint family. This implies that $\{F_j \mid j \in C^*_{<1}\}$ is also a disjoint family. Now we construct a basis for $\bar{y}$. For every integral facility $i \in F_{=1}$, we add the constraint $y_i \le 1$ to our basis. To complete the basis, we need to add $|F_{<1}|$ further linearly independent tight constraints.

We recall that upon termination of PseudoApproximation, no C_{part}-, C_{full}-, or non-negativity constraint is tight for $\bar{y}$, so the only constraints we can choose are the C^*-constraints, the k-constraint, or the coverage constraint. We claim that we cannot add any $C^*_{=1}$-constraint to our basis, because such a constraint is supported only on integral facilities, whose constraints we already added to the basis. However, we can add every $C^*_{<1}$-constraint to our basis, because their supports are disjoint and they contain no integral facilities. Thus, our partial basis consists of all tight integrality constraints and all $C^*_{<1}$-constraints.

Now we consider adding the k-constraint to our basis. Importantly, the k-constraint is linearly independent with the current partial basis only if there exists at least one fractional facility not supported on any F-ball for clients in $C^*_{<1}$. Further, we may assume the k-constraint is tight (otherwise we cannot add it anyways), so there must be at least two fractional facilities not supported on any F-ball for clients in $C^*_{<1}$

However, we note that each F-ball for clients in $C^*_{<1}$ contains at least two fractional facilities. Because these F-balls are disjoint, we have $|F_{<1}| \geq 2|C^*_{<1}|$. If we cannot add the k-constraint to our basis, then we are done. This is because the coverage constraint is the only further constraint we can add the the basis, so we can bound $|F_{<1}| \leq |C^*_{<1}|+1$. This implies implies $|F_{<1}| \leq \frac{1}{2}|F_{<1}|+1 \Rightarrow |F_{<1}| \leq 2$ using the previous inequality.

Otherwise, we add the k-constraint to our basis, which implies $|F_{<1}| \geq 2|C^*_{<1}| + 2$ because of the two fractional facilities outside $F(C^*_{<1})$ and $|F_{<1}| \leq |C^*_{<1}| + 2$ because the k-constraint and coverage constraint contribute are the only further constraints we can add. Again combining these two inequalities gives $|F_{<1}| \leq 2$. □

Proof of Lemma 2.7.3. Let $S = F_{=1}$ be the set of open facilities. It is immediate that $|S| \leq k$. Further, LP_{iter} satisfies all Extra Invariants, so $C_0 \subset C^*$. Because $\bar{y}$ is integral, it is clear that we open S_0. Thus it remains to show that the connecting m clients to S has cost at most $(2+\alpha)\mathsf{Opt}(LP_{iter})$.

It suffices to show that connecting C_{full} and C^* to S is enough clients and achieves the desired cost. Because $\bar{y}$ is integral and by definition of PseudoApproximation, we have that no C_{part}-, C_{full}-, or non-negativity constraint is tight for $\bar{y}$. It follows, $F_j = \emptyset$ for all $j \in C_{part}$ and $B_j = \emptyset$ for all $j \in C_{full}$.

Then the coverage constraint of LP_{iter} implies: $\sum_{j \in C_{part}} \bar{y}(F_j) \geq m - |C_{full} \cup C^*| \Rightarrow |C_{full} \cup C^*| \geq m$, so this solution connects enough clients.

To bound the cost, we compare the cost of each client with its contribution to the objective of LP_{iter}. For all $j \in C^*$ we have $\bar{y}(F_j) = 1$, so $d(j,S) \leq \sum_{i \in F_j} d'(j,i)\bar{y}_i$, which is exactly j's contribution to LP_{iter}.

For all $j \in C_{full}$, we note that $\bar{y}(B_j) = 0$, so j's contribution to LP_{iter} is exactly $L(\ell_j)$. We can apply Theorem 2.5.2 with $\beta = 1$ and set of facilities S to show that $d(j,S) \leq (2+\alpha)L(\ell_j)$ for all $j \in C_{full}$. To conclude, the connection cost of each client is at most $(2+\alpha)$ times its contribution to LP_{iter}, a required. □

Proof of Lemma 2.7.4. Let $S = F_{=1} \cup \{a\}$ be the output solution. First, note that $|S| \leq k$ because $\bar{y}_a + \bar{y}_b = 1$, and those are the only two fractional variables. Second, because $a \notin S_0$, it must be the case that $b \notin S_0$, because a, b are the only fractional facilities, and by Extra Invariant (1), there is one unit of open facility co-located at each $i \in S_0$. Note that this implies that $S_0 \subset F_{=1} \subset S$.

Now there are two cases, either $a, b \in F_j$ for some $j \in C^*$, or $a, b \notin F(C^*)$. Note that in either case, we close b and open a, so we still maintain the property that $\bar{y}(F_j) = 1$ for all $j \in C^*$. Thus, can apply Theorem 2.5.2 with $\beta = 1$ and set of facilities S to show that $d(j,S) \leq (2+\alpha)L(\ell_j)$ for all $j \in C_{full} \cup C^*$.

We consider connecting the clients $C_{part}(a) \cup C_{full} \cup C^*$ to S. First, we show that this is at least m clients. We observe that a and b are the only fractional facilities in $\bar{y}$, and no C_{part}-constraint is tight. It follows that for all $j \in C_{part}$, we have $F_j = \{a\}$, $\{b\}$, or $\emptyset$, so we can rewrite the coverage constraint as:

$$|C_{part}(a)|\bar{y}_a + |C_{part}(b)|\bar{y}_b \geq m - |C_{full} \cup C^*|$$

Then because $\bar{y}_a + \bar{y}_b = 1$ and $|C_{part}(a)| \geq |C_{part}(b)|$ by assumption, we conclude that $|C_{part}(a)| \geq m - |C_{full} \cup C^*|$, as required.

Now it remains to show that the cost of connecting $C_{part}(a)$ to a plus the cost of connecting $C_{full} \cup C^*$ to S is at most $\alpha \mathsf{Opt}(LP_{iter}) + O(\frac{\rho}{\delta})U$. First we handle $C_{part}(a)$. By assumption, $a \notin S_0$, so by Extra Invariant (2), we can bound: $\sum_{j \in C_{part}(a)} d(j,a) \leq \sum_{j \in C | a \in F_j} d(j,a) = O(\rho)U$

For the clients in $C_{full} \cup C^*$ that are not supported on b, closing b does not affect their connection cost; in particular, each such client either has an integral facility in its F-ball to connect to (because we open a and all other facilities are integral), or its F-ball is empty, and there exists an integral facility within $(2+\alpha)L(\ell_j)$ to connect to. In both cases, each client's connection cost is at most $(2+\alpha)$ times its contribution to the objective of LP_{iter}.

The only remaining cost to bound is the clients in $C_{full} \cup C^*$ that are supported on b. Let $C' = \{j \in C_{full} \cup C^* \mid b \in F_j\}$ be these clients. We show that the cost of connecting all of C' to S is at most $O(\frac{\rho}{\delta})U$ using Lemma 2.6.13. Because every client in $j \in C_{full} \cup C^*$ has an open facility in S within distance $(2+\alpha)L(\ell_j)$, Lemma 2.6.13 is applicable to C' with set of facilities S and $i = b \notin S \cup S_0$.

To summarize, the connection costs of $C_{part}(a)$ and C' are at most $O(\frac{\rho}{\delta})U$, and the connection cost of all remaining clients in $C_{full} \cup C^*$ that are not supported on b is at most $(2+\alpha)\mathsf{Opt}(LP_{iter})$, so the total connection cost, which is the sum of these terms, it at most the desired bound. □

A.4 Proof of Theorem 2.6.11: k-Median with Outliers Pre-Processing

The goal of this section is to prove Theorem 2.6.11 using theorems from [KLS18]. Note that we follow exactly the same pre-processing steps; the only difference is that we summarize the results of their pre-processing in a single theorem.

The knapsack pre-processing in Theorem 2.6.3 follows from the pre-processing steps in [KLS18] as well.

A.4.1 Preliminaries

We define the notions of *extended instances* and *sparse extended instances* for k-median with outliers. These definitions are useful to capture the properties of our pre-processing.

Extended instances are used to handle the fact that in our pre-processing, we will guess some facilities to pre-open. Then S_0 is the set of guessed facilities.

Definition A.4.1 (Extended Instance for k-Median with Outliers)**.** An extended instance for k-median with outliers is of the form $\mathcal{I} = (F, C, d, k, m, S_0)$, where F, C, d, k, and m are defined as in a standard k-median with outliers instance (see Definition 2.6.9), and $S_0 \subset F$.

As in k-median with outliers, the goal is to choose a set of at most k open facilities $S \subset F$ and at least m clients $C' \subset C$ to serve to minimize the connection costs of the served clients to the open facilities, so $\sum_{j \in C'} d(j, S)$. However, we add the additional constraint that the set of open facilities must include S_0.

Further, sparse extended instances give our properties for what it means for the facilities and clients to be "cheap" (see the second and third properties in the next definition, respectively.)

Definition A.4.2 (Sparse Extended Instance for k-Median with Outliers)**.** Let $\mathcal{I}' = (F, C', d, k, m', S_0)$ be an extended k-median with outliers instance and $\rho, \delta \in (0, \frac{1}{2})$, $U \geq 0$ be parameters. We say that $\mathcal{I}'$ is (ρ, δ, U)-sparse with respect to solution $(S^*, C^{*\prime})$ if the following three properties hold:

1. the cost of the solution $(S^*, C^{*\prime})$ to $\mathcal{I}'$ is at most U
2. for all $i \in S^* \setminus S_0$, we have $\sum_{j \in C^{*\prime} | d(j,S^*) = d(j,i)} d(j,i) \leq \rho U$
3. for all $p \in F \cup C'$, we have $|B_{C'}(p, \delta d(p, S^*))| d(p, S^*) \leq \rho U$

A.4.2 Sparsification

In this section, we pass from the input k-median with outliers instance to a sparse extended sub-instance by guessing the expensive parts of the input instance. Then on this sparse extended sub-instance, we can strengthen LP_1. The following theorems are directly from [KLS18], so we omit the proofs in this paper. The first theorem states that we can efficiently compute a sparse extended sub-instance at the cost of a small increase in approximation ratio.

Theorem A.4.3. *Let $\mathcal{I} = (F, C, d, m, k)$ be an instance of k-median with outliers with optimal solution (S^*, C^*) and $\rho, \delta \in (0, 1/2)$ be parameters. Then there exists a $n^{O(1/\rho)}$-time algorithm that given $\mathcal{I}$, ρ, δ, and an upper bound U on the cost of the optimal solution (S^*, C^*)[1], outputs $n^{O(1/\rho)}$-many extended k-median with outliers instances of the form $\mathcal{I}' = (F, C', d, m', k, S_0)$ such that $C' \subset C$, $m' = |C^* \cap C'|$, and $S_0 \subset S$. Further, one such instance $\mathcal{I}'$ is (ρ, δ, U)-sparse with respect to the solution $(S^*, C^* \cap C')$ and satisfies:*

$$\frac{1-\delta}{1+\delta} \sum_{j \in C^* \setminus C'} d(j, S_0) + \sum_{j \in C^* \cap C'} d(j, S^*) \leq U \tag{A.2}$$

Once we have our sparse extended sub-instance, say $\mathcal{I}'$, we use these sparsity properties to compute the R-vector, which is needed for our Extra Invariants.

Theorem A.4.4. *Let $\mathcal{I}' = (F, C', d, m', k, S_0)$ be an extended k-median with outliers instance and $\rho, \delta \in (0, 1/2)$ and $U \geq 0$. Suppose $\mathcal{I}'$ is (ρ, δ, U)-sparse instance with respect to solution $(S^*, C^{*\prime})$ to $\mathcal{I}'$ such that $(S^*, C^{*\prime})$ has cost U' on $\mathcal{I}'$. Then there exists a polynomial time algorithm that takes as input $\mathcal{I}'$, ρ, δ, and U and outputs $R \in \mathbb{R}_+^{C'}$ satisfying:*

1. *For every $t > 0$ and $p \in F \cup C'$, we have: $|\{j \in B_{C'}(p, \frac{\delta t}{4+3\delta}) \mid R_j \geq t\}| \leq \frac{\rho(1+3\delta/4)}{1-\delta/4} \frac{U}{t}$*
2. *There exists a solution to $\mathcal{I}'$ of cost at most $(1+\delta/2)U'$ such that if client j is connected to facility i, then $d(j,i) \leq R_j$ and for any facility $i \notin S_0$, the total cost of clients connected to i in this solution is at most $\rho(1+\delta/2)U$*

A.4.3 Putting it all Together: Proving Theorem 2.6.11

Combining the algorithms guaranteed by these above two theorems, we show how to construct LP_{iter} with the desired properties.

Suppose we are given a k-median with outliers instance $\mathcal{I} = (F, C, d, m, k)$, parameters $\rho, \delta \in (0, \frac{1}{2})$, and an upper bound U of $\mathsf{Opt}(\mathcal{I})$. First we run the algorithm guaranteed by Theorem A.4.3 to obtain $n^{O(1/\rho)}$-many extended k-median with outliers instances. Then for each instance, we run the algorithm guaranteed by Theorem A.4.4 to obtain a vector R for each such instance.

[1] Note that we are given U, but not the solution (S^*, C^*)

By Theorem A.4.3, let $\mathcal{I}' = (F, C' \subset C, d, m' = m - |C^* \setminus C'|, k, S_0)$ be the instance output by the first algorithm such that $\mathcal{I}'$ is (ρ, δ, U)-sparse with respect to the solution $(S^*, C^* \cap C')$ and satisfies (A.2). This sub-instance will be the one that is guaranteed by Theorem 2.6.11, so from here we need to compute the R-vector, and construct LP_{iter} with the desired properties.

Note that the cost of solution $(S^*, C^* \cap C')$ to $\mathcal{I}'$ is exactly $U' = \sum_{j \in C^* \cap C'} d(j, S^*)$. It follows, on this instance $\mathcal{I}'$, the algorithm guaranteed by Theorem A.4.4 outputs a vector $R \in \mathbb{R}_+^{C'}$ such that for every $t > 0$ and $p \in F \cup C'$, we have: $|\{j \in B_{C'}(p, \frac{\delta t}{4+3\delta}) \mid R_j \geq t\}| \leq \frac{\rho(1+3\delta/4)}{1-\delta/4} \frac{U}{t}$, and there exists a solution, say $(\bar{S}, \bar{C})$ to $\mathcal{I}'$ of cost at most $(1+\delta/2)U'$ such that if j is connected to facility i, then $d(j,i) \leq R_j$ and for any facility $i \in \bar{S} \setminus S_0$, the total cost of clients connected to i in this solution is at most $\rho(1+\delta/2)U$.

It remains to construct LP_{iter}. To do so, first we construct a strengthened LP for the instance $\mathcal{I}'$ such that $(\bar{S}, \bar{C})$ is feasible for the strengthened LP, which we call LP_1':

$$
\begin{aligned}
\min_{x,y} \quad & \sum_{i \in F, j \in C'} d(i,j) x_{ij} && && (LP_1')\\
\text{s.t.} \quad & x_{ij} \leq y_i \quad \forall i \in F, j \in C' && y_i = 1 \quad \forall i \in S_0 \\
& \sum_{i \in F} x_{ij} \leq 1 \quad \forall j \in C' && x_{ij} = 0 \quad \forall i \in F, j \in C' \text{ s.t. } d(i,j) > R_j \\
& \sum_{i \in F} y_i \leq k && \sum_{j \in C'} d(i,j) x_{ij} \leq \rho(1+\delta/2) U y_i \quad \forall i \notin S_0 \\
& \sum_{j \in C', i \in F} x_{ij} \geq m && x_{ij} = 0 \quad \forall i \notin S_0, j \in C' \text{ s.t. } d(i,j) > \rho(1+\delta/2)U \\
& 0 \leq x, y \leq 1
\end{aligned}
$$

The left column of constraints are the same as LP_1 and the right column of constraints are extra constraints that are valid for the solution $(\bar{S}, \bar{C})$ to our sub-instance $\mathcal{I}'$. Because these constraints are valid for the solution $(\bar{S}, \bar{C})$, the following proposition is immediate.

Proposition A.4.5. *$Opt(LP_1') \leq (1+\delta/2)U'$.*

We want to give a similar construction as in § 2.2, where we construct LP_{iter} satisfying all Basic Invariants from LP_1. We note that the main difference in our procedure here when compared to § 2.2 is how we eliminate the x-variables. To compute the F-balls for LP_1', we must carefully duplicate facilities to capture the constraints: $\sum_{j \in C'} d(i,j) x_{ij} \leq \rho(1+\delta/2) U y_i \quad \forall i \notin S_0$. We pass from LP_1' to LP_2 with the next lemma.

Lemma A.4.6. *There exists a polynomial time algorithm that takes as input LP_1', duplicates facilities in F, and outputs a vector $\bar{y} \in [0,1]^F$ and sets $F_j \subset B_F(j, R_j)$ for all $j \in C'$ such that:*

1. *$\bar{y}(F_j) \leq 1$ for all $j \in C'$*
2. *$\bar{y}(F) \leq k$*

3. $\sum_{j \in C'} \bar{y}(F_j) \geq m$

4. $\sum_{j \in C} \sum_{i \in F_j} d(i,j)\bar{y}_i \leq \mathit{Opt}(LP_1')$

5. *For all $i \in S_0$, there is one unit of open facility co-located with i in $\bar{y}$*

6. *For every facility i not co-located with a facility in S_0, we have* $\sum_{j \in C' | i \in F_j} d(i,j) \leq 2\rho(1+\delta/2)U$

Applying the algorithm guaranteed by the above lemma to LP_1', we can obtain the F_j-sets. Using these F-balls, we proceed similarly as in § 2.2. Thus, next we randomly discretize the distances to powers of $\tau > 1$ (up to a random offset) to obtain $d'(p,q)$ for all $p, q \in F \cup C$. Again, the possible discretized distances are $L(-2) = -1, L(-1) = 0, \ldots, L(\ell) = \alpha\tau^\ell$ for all $\ell \in \mathbb{N}$, and d' satisfies Proposition 2.2.2.

Then we define the radius levels and inner balls in the exact same way, so: $\ell_j = \min_{\ell \geq -1}\{\ell \mid d'(j,i) \leq L(\ell) \quad \forall i \in F_j\}$ and $B_j = \{i \in F_j \mid d'(j,i) \leq L(\ell_j - 1)\}$

To complete the data of LP_{iter} for $\mathcal{I}'$, we need to define the sets C_{part}, C_{full}, and C^*. Here we must slightly modify the construction of § 2.2 to accommodate the set of pre-opened facilities, S_0. To satisfy Extra Invariant (1), we create a set C_0 of dummy clients such that for each $i \in S_0$, there exists a dummy client $j(i) \in C_0$ that is co-located with i such that $F_{j(i)}$ has radius level -1 and consists of all co-located copies of i. Thus, we define $C_{part} = C'$, $C_{full} = \emptyset$, and $C^* = C^0$.

This completes the description of LP_{iter} for sub-instance $\mathcal{I}'$. To complete our algorithm, we output each $\mathcal{I}'$ along with LP_{iter}, S_0, and R.

To summarize, our algorithm is to first run the algorithm guaranteed by Theorem A.4.3 to obtain $n^{O(1/\rho)}$-many sub-instances. For each sub-instance, we compute R using Theorem A.4.4, construct LP_1', construct the F-balls using Lemma A.4.6, and define the rest of the data of LP_{iter} as in § 2.2. The runtime of our algorithm is immediate, so it suffices to show that one of the outputs has the desired properties.

In particular, we consider the sub-instance $\mathcal{I}' = (F, C' \subset C, d, m' = m - |C^* \setminus C'|, k, S_0)$ output by the algorithm guaranteed by Theorem A.4.3 such that $\mathcal{I}'$ is (ρ, δ, U)-sparse with respect to the solution $(S^*, C^* \cap C')$ and satisfies (A.2). For the remainder of this section, we consider the LP_{iter} constructed for this specific sub-instance. To complete the proof, we verify that LP_{iter} satisfies the two desired properties.

Proposition A.4.7. *LP_{iter} satisfies all Basic and Extra Invariants.*

Proof. It is easy to verify that LP_{iter} satisfies all Basic Invariants by construction. For the Extra Invariants, we handle them one-by-one.

Extra Invariant (1) holds by construction of LP_{iter}. To show Extra Invariant (2), we again apply Lemma A.4.6, which states that the F-balls have the desired property.

To show Extra Invariant (3), we note that in Lemma A.4.6, the F-balls are constructed such that $F_j \subset B_F(j, R_j)$, for all $j \in C'$. Thus, for all $j \in C'$ and $i \in F_j$, we have $d(i,j) \leq R_j$. Then by definition of the radius levels, we have $L(\ell_j) \leq \tau R_j$, as required. Finally, Extra Invariant (4) follows from the guarantee of Theorem A.4.4. □

Proposition A.4.8. $\frac{\log_e \tau}{(\tau-1)(1+\delta/2)}\mathbb{E}[\mathit{Opt}(LP_{iter})] + \frac{1-\delta}{1+\delta}\sum_{j \in C^* \setminus C'} d(j, S_0) \leq U$

Proof. We first show that $\mathbb{E}[\mathsf{Opt}(LP_{iter})] \le \frac{\tau-1}{\log_e \tau}\mathsf{Opt}(LP_1')$. We have $C_{part} = C'$, $C_{full} = \emptyset$, and $C^* = C_0$. Note that the dummy clients in C^* contribute zero to the objective of LP_{iter}, because they are co-located with one unit of open facility in their F-balls. Thus Lemma A.4.6 implies that there exists a feasible solution $\bar{y}$ to LP_{iter} of cost at most $\mathsf{Opt}(LP_1')$ up to the discretization of the distances. The cost of discretization is bounded by Proposition 2.2.2, and immediately gives the extra $\frac{\tau-1}{\log_e \tau}$-factor. The factor of $\frac{\tau-1}{\log_e \tau}$ is due to the cost of discretization, which is bounded by Proposition 2.2.2.

Now we relate $\mathsf{Opt}(LP_1')$ to $U' = \sum_{j \in C^* \cap C'} d(j, S^*)$, which satisfies (A.2) by the guarantees of Theorem A.4.3. Combining (A.2) with Proposition A.4.5, we can obtain our final bound:

$$\begin{aligned}
\frac{\log_e \tau}{(\tau-1)(1+\delta/2)}\mathbb{E}[\mathsf{Opt}(LP_{iter})] + \frac{1-\delta}{1+\delta}\sum_{j \in C^* \setminus C'} d(j, S_0) &\le \frac{1}{1+\delta/2}\mathsf{Opt}(LP_1') + \frac{1-\delta}{1+\delta}\sum_{j \in C^* \setminus C'} d(j, S_0) \\
&\le \sum_{j \in C^* \cap C'} d(j, S^*) + \frac{1-\delta}{1+\delta}\sum_{j \in C^* \setminus C'} d(j, S_0) \\
&\le U
\end{aligned}$$

□

Appendix B

Appendix for Stochastic Load Balancing

B.1 Maximal Inequalities

In this section, we state some useful probabilistic inequalities. Further, we give the proofs of Lemmas 4.2.2, 4.2.5, and 4.4.2.

Lemma B.1.1 (Bernstein's inequality)**.** *Let the random variables $X_1, \ldots, X_n$ be independent with $X_i - \mathbb{E}[X_j] \leq b$ for each $1 \leq j \leq n$. Let $S = \sum_{j=1}^{n} X_j$ and let $\sigma^2 = \sum_{j=1}^{n} \sigma_j^2$ be the variance of S. Then, for any $t > 0$,*

$$\mathbb{P}[S > \mathbb{E}[S] + t] \leq \exp\left(- \frac{t^2}{2\sigma^2(1 + bt/3\sigma^2)} \right).$$

Lemma B.1.2 (Bennett's inequality [Ben62])**.** *Let $X_1, \ldots, X_n$ be independent random variables with zero mean such that $X_j \leq a$ for all j. Let $S = \sum_j X_j$ and $\sigma^2 = \sum_j \mathbb{E}[X_j^2]$. Then, for all $t \geq 0$,*

$$\mathbb{P}(S > t) \leq \exp\left(\frac{-\sigma^2}{a^2} \left(\left(1 + \frac{at}{\sigma^2} \right) \log\left(1 + \frac{at}{\sigma^2} \right) - \frac{at}{\sigma^2} \right) \right).$$

Lemma B.1.3 (Chernoff-Hoeffding type inequality; Theorem 1.1 in [DP09])**.** *Let $S = \sum_{j=1}^{n} X_j$ where X_j, $1 \leq j \leq n$, are independently distributed in $[0, 1]$. If $t > 2e\mathbb{E}[S]$, then*

$$\mathbb{P}[S > t] \leq 2^{-t}.$$

Having these inequalities at hand, we are now ready to prove Lemmas 4.2.2, 4.2.5, and 4.4.2, that bound the expected maximum of sums of independent variables.

Lemma 4.2.2. *Let $S_1, \ldots, S_m$ be sums of independent random variables, that are bounded in $[0, \tau]$ for some $\tau > 0$, such that $\mathbb{E}[S_i] \leq \tau$ for all $1 \leq i \leq m$. Then, $\mathbb{E}[\max_i S_i] = O\left(\frac{\log m}{\log \log m}\right)\tau$.*

Proof. By re-scaling, it suffices to prove the lemma for $\tau = 1$. Consider one such sum, say $S = \sum_j X_j$, where the X_js are independent and bounded in $[0, 1]$. We use Bennett's inequality (Lemma B.1.2) to bound the upper tail of S. Applying Bennett's inequality to $S - \mathbb{E}[S]$ with $a = 1$ and $\sigma^2 = \sum_j \mathbb{E}\left[(X_j - \mathbb{E}[X_j])^2\right] \leq \sum_j \mathbb{E}[X_j^2] \leq \sum_j \mathbb{E}[X_j] \leq 1$ gives for any $t \geq 0$:

$$\mathbb{P}[S > 1 + t] \leq \mathbb{P}[S > \mathbb{E}[S] + t]$$

$$\leq \exp\left(-(\sigma^2 + t)\log\left(1 + \frac{t}{\sigma^2}\right) + t\right)$$
$$\leq \exp\big(-t\log(1+t) + t\big) = \frac{e^t}{(1+t)^t},$$

where we use $0 \leq \sigma^2 \leq 1$ in the third inequality. In particular, for $\ell = O\big(\frac{\log m}{\log\log m}\big)$ large enough, we have for any $t \geq 0$:

$$\mathbb{P}[S > 1 + \ell + t] \leq \frac{e^{\ell+t}}{(1+\ell+t)^{\ell+t}} \leq \frac{e^\ell}{\ell^\ell} \cdot \frac{e^t}{\ell^t} = O\left(\frac{1}{m}\right) \cdot e^{-t}.$$

This tail bound holds for all $S_1, \ldots, S_m$. Union-bounding over all m sums gives:

$$\begin{aligned}
\mathbb{E}\big[\max_i S_i\big] &= \int_0^\infty \mathbb{P}\big[\max_i S_i > t\big] \cdot dt \\
&\leq (1+\ell) + \int_0^\infty \mathbb{P}\big[\max_i S_i > 1 + \ell + t\big] \cdot dt \\
&\leq (1+\ell) + \sum_i \int_0^\infty \mathbb{P}[S_i > 1 + \ell + t] \cdot dt \\
&= (1+\ell) + m \cdot O\left(\frac{1}{m}\right) \int_0^\infty e^{-t} \cdot dt = O(\ell) = O\left(\frac{\log m}{\log\log m}\right).
\end{aligned}$$

□

Lemma 4.2.5. *Let $S_1, \ldots, S_m$ be sums of independent random variables bounded in $[0, \tau]$ such that $\mathbb{E}[S_i] \leq O(\log m)\tau$ for all $1 \leq i \leq m$. Then, $\mathbb{E}[\max_i S_i] \leq O(\log m)\tau$.*

Proof. Let $C \geq 0$ be a constant such that $\mathbb{E}[S_i] \leq C\tau \log m$ for all $1 \leq i \leq m$ and fix $t > 2eC\tau \log m$, where e is the Euler constant. Fixing $i \in [m]$, we know that the random variables constituting S_i are independent and bounded by τ. Hence, using the Chernoff-Hoeffding bound in Lemma B.1.3, we have $\mathbb{P}[S_i > t] < 2^{-t}$. Therefore,

$$\begin{aligned}
\mathbb{E}\Big[\max_{1\leq i\leq m} S_i\Big] &= \int_0^\infty \mathbb{P}\Big[\max_{1\leq i\leq m} S_i > t\Big] dt \\
&\leq O(\log m)\tau + \int_{2eC\tau\log m}^\infty \mathbb{P}\Big[\max_{1\leq i\leq m} S_i > t\Big] dt \\
&\leq O(\log m)\tau + \sum_{i=1}^m \int_{2eC\tau\log m}^\infty \mathbb{P}\Big[S_i > t\Big] dt \\
&\leq O(\log m)\tau + m\int_{2eC\tau\log m}^\infty 2^{-t} dt \\
&= O(\log m)\tau,
\end{aligned}$$

where the third line uses a union bound over all $i \in [m]$. □

Lemma 4.4.2. *Let $c_1, \ldots, c_m \in \mathbb{N}_{\geq 1}$ be constants such that $c_i \geq \frac{3}{2}c_{i+1}$ for all $1 \leq i \leq m$. Let $S_1, \ldots, S_m$ be sums of independent random variables bounded in $[0, \tau]$ such that $\mathbb{E}[S_i] \leq c_i\tau$ for all $1 \leq i \leq m$. Then, $\mathbb{E}\big[\max_i \frac{S_i}{c_i}\big] \leq O(\tau)$.*

Proof. By re-scaling, we can assume $\tau = 1$. Fix one sum, say $S = \sum_j X_j$. The X_js are independent with $X_j/c \le \frac{1}{c}$ and $X_j/c - \mathbb{E}\big[X_j/c\big] \le \frac{1}{c}$. Therefore,

$$\mathrm{Var}\big[S/c\big] = \sum_j \mathrm{Var}\big[X_j/c\big] \le /c \sum_j \mathbb{E}\big[X_j/c\big] \le \frac{1}{c},$$

where we used the independence of the X_j and the fact that $\sum_j \mathbb{E}[X_j] \le c$. We fix some $t \ge 1$. Bernstein's inequality (Lemma B.1.1) guarantees

$$\mathbb{P}\left[\frac{S}{c} \ge 2 + t\right] \le \exp\left(-\frac{t^2}{2/c + 2t/3}\right) \le \exp\left(-c\frac{t^2}{2 + 2t/3}\right) \le \exp\left(-\frac{c}{3}t\right),$$

where we used that $c \ge 1$. Using a union bound, we obtain

$$\begin{aligned}
\mathbb{P}\Big[\max_{1\le i\le m} S_i/c_i \ge 2 + t\Big] &\le \sum_{i=1}^{m} \exp\left(-\frac{c_i}{3}t\right) \\
&\le \sum_{i=1}^{m} \exp\left(-(3/2)^{m-i} c_m \frac{t}{3}\right) \\
&\le \sum_{i=1}^{m} \exp\left(-(3/2)^{m-i} \frac{t}{3}\right) \\
&\le \sum_{i=1}^{m} \left(\frac{2}{3}\right)^{m-i} e^{-t/3} \\
&\le 3e^{-t/3},
\end{aligned}$$

where we used $c_i \ge \frac{3}{2}c_{i+1}$ in the second inequality and $c_m \ge 1$ in the third inequality. For the fourth inequality, we fix $t \ge 3$ and use that, for $s \ge 1$ and $x \ge 0$, we have $\exp\big(-s\big(\frac{3}{2}\big)^x\big) \le \big(\frac{2}{3}\big)^x e^{-s}$.

Hence, we conclude the proof with

$$\mathbb{E}\Big[\max_{1\le i\le m} S_i/c_i\Big] \le (2+6) + \int_5^\infty \mathbb{P}\Big[\max_{1\le i\le m} S_i/c_i > 2 + t\Big]dt \le 8 + 3\int_5^\infty e^{-t/5}dt = 8 + 15/e = O(1).$$

□

B.2 Machine smoothing analysis

Lemma 4.4.1. *Given an instance $\mathcal{I}$ of load balancing with m related machines and stochastic jobs, Algorithm 12 efficiently computes an instance $\mathcal{I}_s$ of smoothed machines with the same set of jobs satisfying Properties (i) to (iii).*

Proof. It is clear that the algorithm is efficient and outputs $\mathcal{I}_s$ satisfying (i) and (ii). It remains to show (iii). To do so, we show that each step increases Opt by only a constant factor.

Recall that Opt is an adaptive policy, so when Opt decides to schedule job j on machine i, it immediately learns the realized value of X_{ij}, or equivalently $X_j = s_i \cdot X_{ij}$ on related machines. Hence, we may assume that the next scheduling decision of Opt is completely determined by the previously realized X_js. In particular, if Opt decides to schedule j on machine i, then we can modify Opt by scheduling j on some other machine i' and leaving the subsequent decisions (which in general depend on X_j) unchanged. Similarly, if we modify the speed of machine i, then this does not affect subsequent scheduling decisions.

(12.6) Let $\mathsf{Opt} = \mathsf{Opt}(\mathcal{I})$ denote the initial optimal policy. Consider the following adaptive policy for the instance after step 12.6: If Opt schedules j on i such that i is not deleted, then we also schedule j on i. Otherwise, Opt schedules j on i such that i is deleted for being too slow. Then we schedule j on the fastest machine.

For every realization of job sizes, the modified policy only increases the load of the fastest machine. We delete at most $m-1$ machines each having speed at most $\frac{1}{m}$. Thus, we schedule all jobs assigned to these machines on a machine that is at least m times faster. The increase in load on the fastest machine is thus at most $(m-1) \cdot \frac{\mathsf{Opt}}{m} \leq \mathsf{Opt}$.

After deleting all slow machines, all machine speeds are in $\left(\frac{1}{m}, 1\right]$.

(12.9) We decrease the speed of each machine by at most a factor 2, so the makespan of the optimal policy increases by at most a factor 2.

Further, after rounding down the machine speeds, there are at most $\lceil \log m \rceil$ distinct speeds and thus groups.

(12.15) First, we note that we keep at least one group, namely the fastest one. Consider any group k that is deleted, and let $k' > k$ be the fastest subsequent group that is kept. Because we delete all groups between k and k', we have $m_k < \left(\frac{3}{2}\right)^{k'-k} \cdot m_{k'}$. Further, because all groups have distinct speeds that differ by at least a factor 2, we also have $s_{k'} \geq 2^{k'-k} \cdot s_k$.

Let Opt be the optimal policy after step 12.9. We re-assign the jobs that Opt schedules on group k to group k' as follows. Because $m_k < \left(\frac{3}{2}\right)^{k'-k} \cdot m_{k'}$, we fix a mapping from the machines in group k to those of k' such that each machine in k' is mapped to by at most $\left(\frac{3}{2}\right)^{k'-k}$ machines in k. Then, when Opt schedules a job on a machine in group k, we instead schedule it on the machine it maps to in group k'. This completes the description of our modified policy.

To bound the makespan, consider any machine i in a kept group k'. We upper-bound the increase in load on i due to re-assignments from slower deleted groups. For any deleted group $k < k'$, at most $\left(\frac{3}{2}\right)^{k'-k}$ machines from group k map to i. Each such machine in group k under policy Opt has load at most Opt. However, recall that i is at least a $2^{k'-k}$-factor faster than any machine in group k, so the increase in load on machine i due to deleted machines from group $k < k'$ is at most $\left(\frac{3}{2}\right)^{k'-k} \cdot 2^{-(k'-k)} \cdot \mathsf{Opt} = \left(\frac{3}{4}\right)^{k'-k} \cdot \mathsf{Opt}$. Summing over all $k < k'$, the total increase in load on a machine in group k' is at most $\sum_{k<k'} \left(\frac{3}{4}\right)^{k'-k} \cdot \mathsf{Opt} = O(\mathsf{Opt})$. □

Appendix C

Appendix for Stochastic Completion Time Minimization

C.1 Sensitivity of number of machines

For a fixed collection of jobs, and any number of machines m, we let $\mathsf{Opt}(m)$ be the optimal completion time for these jobs on m machines.

Lemma C.1.1. *For any number of machines m sufficiently large, there exists a collection of* identical *Bernoulli jobs with $\mathbb{E}\mathit{Opt}(\frac{m}{2}) = e^{\Omega(m)} \cdot \mathbb{E}\mathit{Opt}(m)$.*

Proof. We fix a number of machines m. Define $L = e^{cm}$ for a constant $c > 0$. Then consider the collection of $\frac{7}{8}mL$ Bernoulli jobs distributed as $\mathrm{Ber}(\frac{1}{L})$. Note that because jobs are identically distributed, we may assume Opt list schedules jobs in arbitrary order.

We first claim that $\mathbb{E}\mathsf{Opt}(m) = O(m)$. To see this, let $H \sim \mathrm{Binom}(\frac{7}{8}mL, \frac{1}{L})$ be the number of jobs that come up heads. On the event $H \leq m$, each machine schedules some number of jobs with realized size zero and then at most one job with realized size 1. Thus, on this event we have $\mathsf{Opt}(m) \leq H$. Further, by Chernoff (Proposition C.4.1), we have:

$$\mathbb{P}(H > m) \leq \mathbb{P}(H \geq \mathbb{E}[H] + \frac{m}{8}) = e^{-\Theta(m)}.$$

We conclude, the contribution of the event $H \leq m$ to $\mathbb{E}\mathsf{Opt}(m)$ is at most $\mathbb{E}H = \frac{7}{8}m$, and the contribution of the event $H < m$ is at most $\mathrm{poly}(mL) \cdot \mathbb{P}(H > m) = \mathrm{poly}(me^{cm}) \cdot e^{-\Theta(m)} = O(1)$ for c sufficiently small. This gives $\mathbb{E}\mathsf{Opt}(m) \leq \frac{7}{8}m + O(1) = O(m)$.

On the other hand, we have $\mathbb{E}\mathsf{Opt}(\frac{m}{2}) = \Omega(mL)$. Let $H' \sim \mathrm{Binom}(\frac{3}{4}mL, \frac{1}{L})$ be the number of heads among the first $\frac{3}{4}mL$ jobs. Analogously by Chernoff we have $\mathbb{P}(H' < \frac{m}{2}) \leq e^{-\Theta(m)}$. Thus, on the event $H' \geq \frac{m}{2}$ (which happens with probability $(1 - o(1))$, after scheduling the first $\frac{3}{4}mL$ jobs, each of the $\frac{m}{2}$ machines has a job of realized size 1. It follows, the remaining $\Omega(mL)$ unscheduled jobs all have completion times at least 1. Thus, we can lower bound $\mathbb{E}\mathsf{Opt}(\frac{m}{2}) = \Omega(mL) \cdot (1 - o(1))$. Taking m sufficiently large gives the desired gap. □

Lemma C.1.2. *For any collection of deterministic jobs and any $m \geq 2$, we have $\mathit{Opt}(\frac{m}{2}) \leq 3 \cdot \mathit{Opt}(m)$.*

Proof. Consider the schedule achieving $\mathsf{Opt}(m)$, and let C_j^m be the completion time of job j in this schedule. We construct a schedule on $\frac{m}{2}$ machines with completion time at most $3 \cdot \mathsf{Opt}(m)$. Our algorithm is to list schedule the jobs on $\frac{m}{2}$ machines in increasing order of C_j^m.

Let C_j be the completion time of job j in this schedule. We claim that $C_j \leq 3C_j^m$ for all jobs j, which gives the desired result. Assume for contradiction that this is not the case, so let j be the first job with $C_j > 3C_j^m$. It must be the case that up until time $2C_j^m$, all $\frac{m}{2}$ machines are busy running jobs j' with $C_{j'}^m \leq C_j^m$. The total size of such j' jobs is strictly larger than $\frac{m}{2} \cdot 2C_j^m = m \cdot C_j^m$. However, $\mathsf{Opt}(m)$ must complete all such j' jobs by time C_j^m. This is a contradiction. □

C.2 Exchange Argument

Lemma 5.1.4. *Consider a collection of Bernoulli jobs. Then for each possible size parameter, the optimal adaptive completion time schedule for these jobs starts the jobs with this size parameter in increasing order of their probabilities for all realizations of the job sizes.*

Proof. We suppose there exist jobs a, b such that $X_a \sim s \cdot Ber(p_a)$ and $X_b \sim s \cdot Ber(p_b)$ with $p_a \leq p_b$ such that the optimal completion time policy schedules b before a in some realization. Consider the decision tree corresponding to this policy (described in § 5.2.) Thus, we assume this tree schedules b before a is some realization (i.e. some root-leaf path.) It follows, there exists a subtree rooted at b such that a is scheduled on each root-leaf path of this subtree. We denote this subtree by T. Entering this subtree, the machines have some fixed initial loads and T schedules a fixed set of jobs J.

We will modify the subtree T so that we start a before b on each root-leaf path. Further, this will not increase the expected completion time of the overall schedule. We construct the modified subtree T' as follows. Let the left- and right subtrees (corresponding to the root job b coming up size 0 or s) of T be T_L and T_R, respectively. T' is rooted at job a. In T', the right subtree of a is T_R, but with job a replaced by job b. We denote this modifed subtree by $T_R(a \to b)$. On the left subtree of a, first, independently of all jobs, we flip a coin that comes up heads with probability q. We will choose q later. If the coin is tails, then we schedule subtree T_L with job a replaced by job b, so $T_L(a \to b)$. Otherwise, if the coin is heads, then we schedule b. The left- and right subtrees of b are T_L and T_R, except the job a is replaced by a dummy job that is always zero. In particular, this job does not contribute to the completion time, but upon reaching this node we will always follows the left subtree. We denote these subtrees by $T_L(-a), T_R(-a)$, respectively. This completes the description of T'. See Figure C.1 for the modified tree T'.

Note that T' schedules the same jobs as T and always starts a before b. It remains to choose q such that the expected completion time of T' is the same as T. We choose q such that the probability of entering $T_R(a \to b)$ or $T_R(-a)$ is exactly p_b. We conflate the name of a subtree (e.g. $T_R(a \to b)$) with the event that we enter the subtree. Thus, we want $\mathbb{P}(T_R(a \to b) \vee T_R(-a)) = p_b$. The former probability is exactly $p_a + \bar{p}_a q p_b$, where we define $\bar{p} = 1 - p$ for a probability p. This gives $q = \frac{p_b - p_a}{\bar{p}_a p_b}$. Thus, we have chosen q such that $\mathbb{P}(T_R(a \to b) \vee T_R(-a)) = p_b$ and $\mathbb{P}(T_L(a \to b) \vee T_L(-a)) = \bar{p}_b$. One should imagine that these two events are our replacements for the original tree T entering T_R and T_L.

In both subtrees $T_L(a \to b)$ and $T_L(-a)$, we replace the original job a from T_L with b and a zero job, respectively. Let $\tilde{X}_a$ denote the size of the the replacement job, which is supported on $\{0, s\}$. We compute the distribution of $\tilde{X}_a$:

$$\mathbb{P}(\tilde{X}_a = s \mid T_L(a \to b) \vee T_L(-a)) = \frac{\mathbb{P}(T_L(a \to b))}{\mathbb{P}(T_L(a \to b) \vee T_L(-a))} p_b = \frac{\bar{p}_a \bar{q}}{\bar{p}_b} p_b = p_a.$$

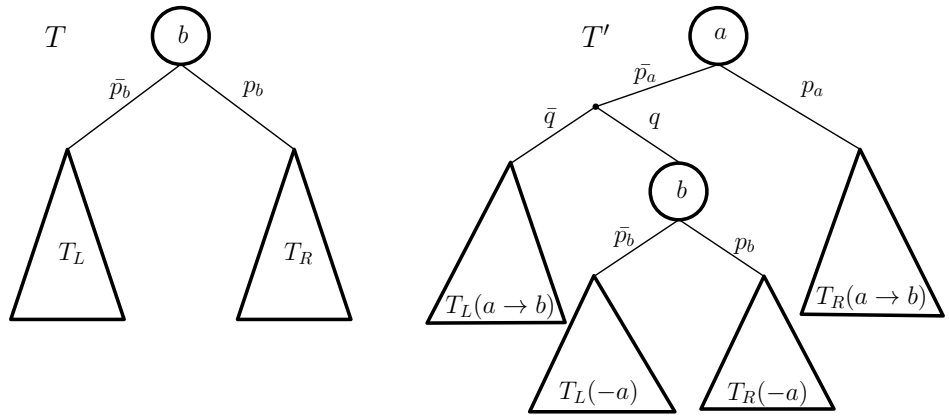

Figure C.1: Original and modified decision trees

It follows, conditioned on $T_L(a \to b) \vee T_L(-a)$, our replacement job for a has the same distribution as a. An analogous computation for the right subtree gives:

$$\mathbb{P}(\tilde{X}_a = s \mid T_R(a \to b) \vee T_R(-a)) = \frac{\mathbb{P}(T_R(a \to b))}{\mathbb{P}(T_R(a \to b) \vee T_L(-a))} p_b = \frac{p_a}{p_b} p_b = p_a,$$

so the distribution of our replacement job conditioned on $T_R(a \to b) \vee T_R(-a)$ has the same distribution as a as well.

To summarize, we have constructed a tree T' that starts a before b. T' enters $T_L(a \to b)$ or $T_L(-a)$ with probability $\bar{p}_b$: exactly the same as the probability that T enters T_L. Further, T' enters $T_L(a \to b)$ or $T_L(-a)$ with the same initial loads as T entering T_L, because both correspond to all previous jobs in the subtree having size 0. Finally, upon entering $T_L(a \to b)$ or $T_L(-a)$, the job we replace a with has the same distribution as a. The analogous properties hold for the right subtree as well. We conclude, for any job $j \in J \setminus \{a, b\}$, the expected completion time of j in T' is the same as in T (subject to the same initial loads.)

It remains to show that the expected completion time of a and b weakly decreases from T to T'. We define $\ell' \sim T_L$ to be the load of the least-loaded machine upon reaching the node a in subtree T_L (we define $\ell' \sim T_R$ analogously.) This is well-defined, because a is scheduled on every root-leaf path in T_L. Note that ℓ' does not depend on the job scheduled at node a. It follows, the expected completion time of a and b in T' are:

$$\mathbb{E}_{T'} C_a = \ell + s p_a$$

$$\mathbb{E}_{T'} C_b = \mathbb{P}(T_L(a \to b))(\mathbb{E}_{\ell' \sim T_L} \ell' + s p_b) + \mathbb{P}(T_L(-a))\ell + \mathbb{P}(T_R(-a))(\ell + s) + \mathbb{P}(T_R(a \to b))(\mathbb{E}_{\ell' \sim T_R} \ell' + s p_b).$$

Now we simplify the completion time of b. First, we consider the terms corresponding to the left subtree. We have $\mathbb{P}(T_L(a \to b)) = \bar{p}_b \frac{p_a}{p_b}$, $\mathbb{P}(T_L(a \to b)) + \mathbb{P}(T_L(-a)) = \bar{p}_b$, and $\ell' \geq \ell$ for $\ell' \sim T_L$. Combining these three observations:

$$\begin{aligned}\mathbb{P}(T_L(a \to b))(\mathbb{E}_{\ell' \sim T_L} \ell' + s p_b) + \mathbb{P}(T_L(-a))\ell &= \mathbb{P}(T_L(a \to b))\mathbb{E}_{\ell' \sim T_L} \ell' + \bar{p}_b s p_a + \mathbb{P}(T_L(-a))\ell \\ &\leq \bar{p}_b(\mathbb{E}_{\ell' \sim T_L} \ell' + s p_a).\end{aligned}$$

Now we consider the right subtree. Analogously, we have $\mathbb{P}(T_R(a \to b)) = p_a$, $\mathbb{P}(T_R(a \to b)) + \mathbb{P}(T_R(-a)) = p_b$, and $\ell' \geq \ell$ for $\ell' \sim T_R$. We compute:

$$\mathbb{P}(T_R(-a))(\ell + s) + \mathbb{P}(T_R(a \to b))(\mathbb{E}_{\ell' \sim T_R} \ell' + s p_b) = (p_b - p_a)(\ell + s) + p_a \mathbb{E}_{\ell' \sim T_R} \ell' + s p_b + p_a s p_b$$

$$\leq p_b(\mathbb{E}_{\ell' \sim T_R} \ell' + p_a s) + (p_b - p_a)s.$$

Combining our expressions for the left- and right-subtrees gives our final bound on the completion time of a and b:

$$\mathbb{E}_{T'} C_a + \mathbb{E}_{T'} C_b \leq \ell + sp_b + \bar{p}_b(\mathbb{E}_{\ell' \sim T_L} \ell' + sp_a) + p_b(\mathbb{E}_{\ell' \sim T_R} \ell' + p_a s) = \mathbb{E}_T C_b + \mathbb{E}_T C_a.$$

□

C.3 Justification for Assumption 5.3.2

Lemma 5.3.3. *Let $m \geq 2$. Suppose there exists an algorithm for completion time minimization for Bernoulli jobs on m machines satisfying Assumption 5.3.2 that outputs a list schedule with expected completion time at most $\alpha(\mathbb{E}\mathsf{Opt} + O(1))$. Then there exists a $O(\alpha)$-approximate algorithm for the same problem without the assumption. Further, the resulting algorithm is also a list schedule, and it preserves efficiency and determinism.*

Proof. Let $\mathcal{A}$ be the algorithm assumed by the lemma. We will run $\mathcal{A}$ on a subinstance of jobs satisfying Assumption 5.3.2. Suppose we have a collection J of Bernoulli jobs of the form $X_j \sim s_j \cdot Ber(p_j)$ for arbitrary size parameters s_j.

First, we round up all size parameters to the nearest power of 2. This at most doubles Opt. Then, we rescale all s_j's uniformly so that $\sum_j \mathbb{E}X_j = 1$. Note that now we have $\mathbb{E}\mathsf{Opt} \geq \sum_j \mathbb{E}X_j = \Omega(1)$. Finally, we partition $J = S \cup M \cup L$ into small, medium, and large jobs, respectively such that S consists of the jobs j with $s_j < \frac{1}{n^2}$, M the jobs j with $\frac{1}{n^2} \leq s_j < n^8$, and L the jobs j with $s_j \geq n^8$. Thus, M is a collection of Bernoulli jobs satisfying Assumption 5.3.2.

Our algorithm to schedule J is the following:

---

i. List-schedule all large jobs L in arbitrary order.
ii. List-schedule all small jobs S in arbitrary order.
iii. Run $\mathcal{A}$ to schedule the medium jobs M.

It is clear that this algorithm is efficient, deterministic, and outputs a list schedule as long as $\mathcal{A}$ does as well. It remains to bound the total completion time of this schedule, which we denote by Alg. We let B be the event that some large job comes up heads (i.e. has realized size at least n^8.)

On the event $\bar{B}$, every large job comes up tails, so they contribute 0 to Alg. Then we list-schedule the small jobs with initial load 0 on every machine. The total completion time of all jobs in S can be crudely upper-bounded by the max load after S times the number of jobs, which is at most $\frac{1}{n} \cdot n = O(\mathbb{E}\mathsf{Opt})$.

After this, we schedule the medium jobs using $\mathcal{A}$. After scheduling S, all machines are free by time $\frac{1}{n}$. Let $\mathcal{A}$ be the total completion time of running $\mathcal{A}$ on jobs M starting at time 0. We need the following monotonicity property of list schedules, which is analogous to Lemma 5.4.5

Lemma C.3.1. *Consider a set of deterministic jobs and a fixed list schedule of those jobs. Then increasing the initial load or decreasing the number of machines weakly increase the total completion time of the schedule.*

Proof. Let J be the set of jobs. Consider initial load vectors $\ell, \ell' \in \mathbb{R}^m$, where the ith entry of each vector denotes the initial load on machine i. Now suppose $\ell \leq \ell'$, entry-wise. It suffices to show

that $C(J, \ell) \le C(J, \ell')$, where $C(J, \ell)$ is the total completion time achieved by our list-schedule with initial load ℓ. This suffices, because we can decrease the number of machines by making the initial loads of some machines arbitrarily large so that they will never be used.

We prove $C(J, \ell) \le C(J, \ell')$ by induction on the number of jobs, $|J|$. In the base case, $|J| = 0$, so the claim is trivial because $C(J, \ell) = 0$ and $C(J, \ell') = 0$. For $|J| > 0$, let j be the first job in the list, which is scheduled, without loss of generality, on the first machine for both initial loads ℓ and ℓ'. Then:

$$C(J, \ell) = (\ell_1 + s_j) + C(J \setminus \{j\}, \ell + s_j e_1) \le (\ell'_1 + s_j) + C(J \setminus \{j\}, \ell' + s_j e_1) = C(J, \ell'),$$

where e_1 is the first standard basis vector, so we have $\ell + s_j e_1 \le \ell' + s_j e_i 1$ entry-wise. Then we assumed inductively that $C(J \setminus \{j\}, \ell + s_j e_1) \le C(J \setminus \{j\}, \ell' + s_j e_1)$. □

By the above lemma, we can upper-bound the total completion time of $\mathcal{A}$ on jobs M by starting once all machines are free after scheduling S, so at time $\frac{1}{n}$. This increases the completion time of each job by $\frac{1}{n}$. To summarize, on the event that every large job comes up tails, we have:

$$\mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{\bar{B}} \le \frac{1}{n} \cdot n + \frac{1}{n} \cdot n + \mathbb{E}\mathcal{A} = O(\mathbb{E}\mathsf{Opt}) + \alpha(\mathbb{E}\mathsf{Opt} + O(1)) = O(\alpha) \cdot \mathbb{E}\mathsf{Opt},$$

where we used the guarantee of $\mathcal{A}$ and $\mathbb{E}\mathsf{Opt} = \Omega(1)$.

It remains to consider the event where some large job comes up heads. In this case, we will not use the guarantee of $\mathcal{A}$. Instead, we will upper bound Alg by the cost of an arbitrary list schedule. We define $S_1 = \max_{j \in J} X_j$ and S_2 to be the size of the second-largest job in J. On the event $B \cap \{S_2 \le \frac{1}{n^2} S_1\}$, we note that no job is scheduled after the largest job with size S_1 on the same machine (using $m \ge 2$.) Noting all other jobs have size at most $\frac{1}{n^2} S_1$, we can upper bound Alg by:

$$\mathsf{Alg} \le S_1 + n \cdot \frac{1}{n} S_1 \le 2S_1 \le 2\mathsf{Opt},$$

so we have $\mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B, S_2 \le \frac{1}{n^2} S_1} = O(\mathbb{E}\mathsf{Opt})$.

Finally, we bound $\mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B, S_2 > \frac{1}{n^2} S_1}$. We partition $B = \cup_{k=0}^{\infty} B_k$, where $B_k = \{\max_{j \in J} X_j \in [2^k n^8, 2^{k+1} n^8)\}$. On the event $B_k \cap \{S_2 > \frac{1}{n^2} S_1\}$, there are at least two jobs of size at least $2^k n^6$. Recall that $\sum_{j \in J} \mathbb{E} X_j = 1$, so in particular $\mathbb{E} X_j \le 1$ for all $j \in J$. Thus by Markov's inequality, $\mathbb{P}(X_j \ge 2^k n^6) \le 2^{-k} n^{-6}$ for all $j \in J$. By union-bounding over all pairs of jobs in J:

$$\mathbb{P}(B_k, S_2 > \frac{1}{n^2} S_1) \le \mathbb{P}(\exists \text{ two jobs in } J \text{ with size at least } 2^k n^6) \le O(n^2)(2^{-k} n^{-6})^2.$$

Further, on the event $B_k \cap \{B, S_2 > \frac{1}{n^2} S_1\}$, every job has size at most $2^{k+1} n^8$, so we have $\mathsf{Alg} \le n \cdot n 2^{k+1} n^8 = 2^{k+1} n^{10}$. Thus, for each k, we have:

$$\begin{aligned} \mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B_k, S_2 > \frac{1}{n^2} S_1} &\le 2^{k+1} n^{10} \cdot \mathbb{P}(B_k, S_2 > \frac{1}{n^2} S_1) \\ &= 2^{k+1} n^{10} \cdot O(n^2)(2^{-k} n^{-6})^2 = O(2^{-k}). \end{aligned}$$

To complete the proof, we partition $B = \cup_{k=0}^{\infty} B_k$ to bound $\mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B, S_2 > \frac{1}{n^2} S_1}$:

$$\mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B, S_2 > \frac{1}{n^2} S_1} = \sum_{k=0}^{\infty} \mathbb{E}\mathsf{Alg} \cdot \mathbb{1}_{B_k, S_2 > \frac{1}{n^2} S_1} = O(\sum_{k=0}^{\infty} 2^{-k}) = O(\mathbb{E}\mathsf{Opt}).$$

□

C.4 Concentration arguments

We need the following standard Chernoff bound.

Proposition C.4.1 (Chernoff bound)**.** *Let $X = X_1 + \dots X_n$ be a sum of independent, $\{0,1\}$-valued random variables and $\mu = \mathbb{E}X$. Then we have:*

- $\mathbb{P}(X \le (1-\delta)\mu) \le \exp\big(-\frac{\delta^2\mu}{2}\big)$ *for all* $0 \le \delta \le 1$.
- $\mathbb{P}(X \ge (1+\delta)\mu) \le \exp\big(-\frac{\delta^2\mu}{2+\delta}\big)$ *for all* $0 \le \delta$.

Lemma 5.4.9. *Let $m = \Omega(1)$ be sufficiently large. Then there exists a constant $c \ge 0$ such that for all batches k and thresholds $\tau > 2\mathbb{E}F^*(n - n/2^k)$, we have $\mathbb{E}|J_k(> \tau)| \le m + c\sqrt{m}$.*

Proof. Fix $c \ge 0$ which we will choose sufficiently large later. Then assume for contradiction that there exists a batch k and threshold $\tau > 2\mathbb{E}F^*(n - n/2^k)$ such that $\mathbb{E}|J_k(> \tau)| > m + c\sqrt{m}$.

To reach a contradiction, it suffices to show that $\mathbb{P}(|J_k(> \tau)| \le m) < \frac{1}{2}$. This is because on the complement event $|J_k(> \tau)| > m$ (which we assume happens with probability strictly larger than $\frac{1}{2}$), we also have $|J_k^*(> \tau)| > m$ by Theorem 5.2.1. This implies $F^*(n - n/2^k) > \tau \ge 2\mathbb{E}F^*(n - n/2^k)$. This would contradict the definition of $\mathbb{E}F^*(n - n/2^k)$.

For convenience, let $\mu = \mathbb{E}|J_k(> \tau)|$. By Chernoff, we have:

$$\mathbb{P}(|J_k(> \tau)| \le m) = \mathbb{P}(|J_k(> \tau)| \le \mu(1 - \frac{\mu - m}{\mu})) \le \exp\big(-\frac{(\mu - m)^2}{2\mu}\big).$$

There are two cases to consider. Recall that by assumption, we have $\mu > m + c\sqrt{m}$. If $\mu \ge 2m$, then $\mathbb{P}(|J_k(> \tau)| \le m) \le \exp(-\frac{\mu}{8}) \le \exp(-\frac{m}{4}) < \frac{1}{2}$ for $m = \Omega(1)$ sufficiently large. Otherwise, $m + c\sqrt{m} < \mu < 2m$. Then $\mathbb{P}(|J_k(> \tau)| \le m) \le \exp(-\frac{c^2 m}{2m}) < \frac{1}{2}$ for $c = O(1)$ sufficiently large. □

Lemma 5.4.10. *Let $\Delta = O(\sqrt{m}\log n)$ and $m = \Omega(1)$ be sufficiently large. Then with probability at least $1 - \frac{1}{\mathrm{poly}(n)}$, the following events hold:*

$$\{|J_k(> \tau)| \overset{\pm\Delta}{\approx} \mathbb{E}|J_k(> \tau)| \quad \forall \text{ batches } k \text{ and thresholds } \tau > 2\mathbb{E}F^*(n - n/2^k)\}. \tag{5.3}$$

Proof. Note that there are $O(\log n)$ choices for k and $L = O(\log n)$ relevant choices for τ. Thus, by a standard union bound argument it suffices to show that for fixed k and $\tau > 2\mathbb{E}F^*(n - n/2^k)$, we have:

$$\mathbb{P}(||J_k(> \tau)| - \mathbb{E}|J_k(> \tau)|| > \Delta) = \frac{1}{\mathrm{poly}(n)}.$$

Now we may assume m is large enough so that $\mathbb{E}|J_k(> \tau)| \le m + c\sqrt{m} \le (c+1)m$ for sufficiently large constant $c \ge 0$ (guaranteed by Lemma 5.4.9.) Then we can bound the deviation of $|J_k(> \tau)|$ again with a Chernoff bound. Let $\mu = \mathbb{E}|J_k(> \tau)|$. We take $\Delta = O(\sqrt{\mu}\log n) = O(\sqrt{m}\log n)$.

There are two cases to consider. If $\mu < \Delta$, then the lower tail is trivial:

$$\mathbb{P}(|J_k(> \tau)| \le \mu - \Delta) \le \mathbb{P}(|J_k(> \tau)| < 0) = 0.$$

For the upper tail we use Chernoff:

$$\mathbb{P}(|J_k(> \tau)| \ge \mu + \Delta) = \mathbb{P}(|J_k(> \tau)| \ge (1 + \frac{\Delta}{\mu})\mu) \le \exp(-\frac{\Delta^2}{2\mu + \Delta}) \le \exp(-\frac{\Delta^2}{3\Delta}) = \frac{1}{\mathrm{poly}(n)}.$$

Otherwise, $\mu \geq \Delta$, so in particular $\frac{\Delta}{\mu} \leq 1$. Then we use Chernoff for both the lower- and upper tails:

$$\mathbb{P}(|J_k(>\tau)| \leq \mu + \Delta) = \mathbb{P}(|J_k(>\tau)| \leq (1 + \frac{\Delta}{\mu})\mu) \leq \exp\left(-\frac{\Delta^2}{2\mu}\right) = \frac{1}{\text{poly}(n)}.$$

$$\mathbb{P}(|J_k(>\tau)| \geq \mu + \Delta) = \mathbb{P}(|J_k(>\tau)| \geq (1 + \frac{\Delta}{\mu})\mu) \leq \exp\left(-\frac{\Delta^2}{2\mu + \Delta}\right) \leq \exp\left(-\frac{\Delta^2}{3\mu}\right) = \frac{1}{\text{poly}(n)}.$$

□