Approximation Algorithms for Stochastic Scheduling on Unrelated Machines

by

Jacob Scott

B.S., Electrical Engineering and Computer Science (2005)
University of California, Berkeley

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by .....
David R. Karger
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Approximation Algorithms for Stochastic Scheduling on Unrelated Machines

by

Jacob Scott

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Motivated by problems in distributed computing, this thesis presents the first non-trivial polynomial time approximation algorithms for an important class of machine scheduling problems. We study the family of *preemptive minimum makespan* scheduling problems where jobs have *stochastic* processing requirements and provide the first approximation algorithms for these problems when machines have *unrelated speeds*. We show a series of algorithms that apply given increasingly general classes of *precedence constraints* on jobs. Letting n and m be, respectively, the number of jobs and machines in an instance, when jobs need an *exponentially distributed* amount of processing, we give:

- An $O(\log\log\min\{m,n\})$-approximation algorithm when jobs are independent;
- An $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm when precedence constraints form disjoint chains; and,
- An $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm when precedence constraints form a directed forest.

Very simple modifications allow our algorithms to apply to more general distributions, at the cost of slightly worse approximation ratios. Our $O(\log\log n)$-approximation algorithm for independent jobs holds when we allow *restarting* instead of preemption. Here jobs may switch machines, but lose all previous processing if they do so.

We also consider problems in the framework of *scheduling under uncertainty* [13]. This model considers jobs that require unit processing on machines with identical speeds. However, after processing a job to completion, a machine has an (*unrelated*) probability of failing and leaving the job uncompleted. This difficulty is offset by allowing multiple machines to process a job simultaneously. We prove that this model is equivalent to a slightly modified version of the family of problems described above and provide approximation algorithms for analogous problems with identical ratios.

Thesis Supervisor: David R. Karger
Title: Professor

Acknowledgments

I would like to begin by thanking my advisor, David Karger. His insight and guidance have played an important role in both the research presented here and the course of my exposition. I have learned a great deal from David and I greatly appreciate the time I spent working with him. I would also like to thank Chris Crutchfield, Zoran Dzunic and Jeremy Fineman, with whom I collaborated on early versions of this research. It was a pleasure to work with all of them. I spent two fruitful weeks working closely with Jeremy to write up our work for publication and that process helped shape my exposition in this thesis. I would like to thank Chris for providing feedback on early drafts of this thesis and, especially, for solidarity during many long nights in the Sidney-Pacific study lounges.

Being a part of the CSAIL theory group has been very rewarding and I would like to thank the theory graduate community, in particular, for making my time at MIT so stimulating. That goes double for my officemates and neighbors who, with good cheer, tolerated my occasional diversions into subjects not entirely pertinent. To me, the exciting technical discussions I had with this group largely define the graduate student experience.

Finally, I would like to thank my family for their love and support. This document would certainly not exist without them, not the least because my mother braved its theorems and equations to copyedit it. I owe them a great deal.

Contents

Chapter 1

Introduction

This thesis presents the first nontrivial approximation algorithms for a set of problems in machine scheduling. A classic and well-studied area of combinatorial optimization, machine scheduling encompasses a class of problems in which a set of jobs J are to be processed by a set of machines M. Solutions are given by *schedules* that describe precisely how the jobs should be processed. The goal is to optimize the value of the schedule according to some *objective function*, while maintaining feasibility with respect to a set of *constraints* on jobs and machines. Together, an objective function and a set of constraints define a problem in this space.

Interest in the subject stems in part from its broad applicability to real-world problems, including those found in manufacturing and large-scale computation. As a concrete example, consider a factory that manufactures textiles. Various looms may support the production of certain articles. Their work must be planned so that various contracts, perhaps with different deadlines, are fulfilled. Given a sufficiently large number of looms and articles, manual planning becomes unwieldy and heuristic approaches offer no guarantee of efficiency. Machine scheduling instead offers a rigorous model and has, in fact, been used to handle actual loom scheduling [17].

Motivated by distributed computing, we study a family of *preemptive minimum makespan stochastic* scheduling problems on unrelated machines. Here, each job $j \in J$ requires an amount of processing (equivalently, work) that is set randomly according to a probability distribution provided as input. Each machine $i \in M$ has a speed

$v_{i,j}$ with which it processes job j. The goal is to schedule jobs on machines so as to minimize the expected time by which all jobs complete. Each job can be processed by only one machine at a time and each machine can process only a single job at a time. However, preemption means that jobs may be moved to different machines at any time (without losing prior processing). The amount of work that a job requires is not revealed until it completes (that is, receives sufficient processing); this is at the core of the problem's difficulty.

This setting maps well to distributed computing on commodity hardware, a practice whose growing popularity may be seen in the success of Google's MapReduce [4]. Preemption is more reasonable when considering computational jobs. Also, allowing unrelated machines supports the modeling of heterogeneous machines, like the workstations of volunteers used by projects such as Seti@Home [1].

Consistent with this motivation, we consider problems with precedence constraints. In addition to jobs and machines, these problems take a directed acyclic graph (dag) G on jobs as input. A job j may not be processed until all its predecessors in G have completed. Many sophisticated distributed computations structure their subproblems in this way.

We also extend our algorithms to the model of *multiprocessor scheduling under uncertainty*, introduced by Malewicz [13]. Also motivated by distributed computing, this model considers only jobs requiring unit work, but allows multiple machines to process a single job at once. However, each machine i has probability $q_{i,j}$ of failing to complete job j after processing it for a single timestep.

1.1 Results and previous work

The machine scheduling literature is vast, and covering even a portion of it in depth could easily double the length of this thesis. Instead, we present here our new results in the context of previous work on related problems. For a broad overview of the field, we refer the interested reader to surveys by Karger et. al. [6] and Pinedo [16].

When the amount of processing a job needs is exponentially distributed and there

are no precedence constraints, we show an $O(\log\log\min\{m,n\})$-approximation algorithm (here, and throughout this thesis, m refers to the number of machines and n to the number of jobs). That is, the expected length of a schedule produced by our algorithm is at most $O(\log\log\min\{m,n\})$ times as long as the expected length of the best possible schedule. We know of no previous non-trivial algorithms for this problem. For the special case of *uniform* machines, Weiss and Pinedo [19] show a polynomial time optimal algorithm. In the uniform machine model, machine speeds may be decomposed as $v_{i,j} = s_i$, so that some machines may be faster than others but if so, must then be faster on *all* jobs. Here, the optimal algorithm is to assign faster machines to jobs that are expected to need more processing. This algorithm is known as LEPT, for longest expected processing time, and is very well studied in the stochastic scheduling literature. However, LEPT does not easily generalize to unrelated machines. In this setting, a machine may be faster on one job and slower on another, so there is no clear order (e.g, by speed) by which to rank machines.

Indeed, our algorithms and proof techniques have a very different flavor then those used by Weiss and Pinedo [19]. The latter cast their scheduling problem as a Markov decision process, defining the process state to be the set of uncompleted jobs. In this case, a decision occurs when a job completes and an action is an assignment of machines to jobs. Analysis shows that scheduling according to LEPT at each decision minimizes the expected time until all jobs have completed. Our construction is much closer to that found in deterministic scheduling problems. We schedule jobs in multiple rounds according to the polynomial time optimal algorithm for (deterministic) preemptive minimum makespan scheduling on unrelated machines given by Lawler and Labetoulle [8]. This algorithm takes as input the *fixed* amount of processing that a job requires, and it is our choice of these amounts in each round that ensures our approximation ratio. Our proofs also apply when processing requirements have arbitrary, non-exponential distributions, although with slightly worse approximation ratios that depend on the distributions. We are also able to extend our $O(\log\log n)$-approximation algorithm to apply when *restarting* is allowed instead of preemption. In this case, jobs may be moved to different machines at any time, but they lose all

processing when they do.

When the precedence dag G is composed of disjoint chains and preemption is allowed, we demonstrate an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm (for exponentially distributed job processing, with related bounds for arbitrary distributions). As this class of precedence constraints includes independent jobs as a special case, we again know of no previous algorithms. Even when restricted to uniform or identical machines, previous work on problems allowing disjoint chain constraints is fraught with fragile algorithms that work only in special cases. For example, Papadimitriou and Tsitsiklis [15] consider only identical machines and job processing distributed exponentially, sharing a common rate parameter. This is generalized to allow uniform machines by Liu and Sanlaville [12], but the restriction on processing distributions remains. Both works show polynomial time optimal algorithms for these special cases, but it is not clear whether any algorithms are known when processing requirements are not distributed identically, even when considering approximation algorithms for identical machines. The analysis in these works is like that of Weiss and Pinedo [19], but more involved and not particularly relevant to our algorithms.

As in the independent jobs case, we base our algorithm for the stochastic problem with chain precedence constraints on its deterministic analog. In this case, we use techniques from the $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$-approximation algorithm for preemptive minimum makespan scheduling of jobs with chain-like precedence constraints on unrelated machines shown by Shmoys et al. [18][1]. Again, the fixed amounts of processing that we give as input to this algorithm for each job and the way in which we manipulate the resulting deterministic schedule are key to proving our approximation ratio. We also use a chain decomposition from Kumar et. al. [7] to extend our algorithm to an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm when the precedence constraints in G form a directed forest.

For the model of scheduling under uncertainty, we show approximation algorithms with ratios that match the results above. This contrasts positively with

[1]This paper actually presents an algorithm for the non-preemptive case, but it can be trivially modified to apply when preemption is allowed.

Problem	Approximation ratio shown in thesis	Previous results for SUU
EXP-I, SUU-I	$O(\log\log\min\{m,n\})$	$O(\log n)$
EXP-I (restarting)	$O(\log\log n)$	
EXP-C, SUU-C	$O(\log(n+m)\log\log\min\{m,n\})$	$O\left(\log m\log n\frac{\log(n+m)}{\log\log(n+m)}\right)$
EXP-T, SUU-T	$O(\log n\log(n+m)\log\log\min\{m,n\})$	$O\left(\log m\log^2 n\frac{\log(n+m)}{\log\log(n+m)}\right)$

Table 1.1: A summary of our results. EXP refers to jobs with exponentially distributed processing requirements, and SUU to scheduling under uncertainty. The designators I, C, and T refer to the presence of precedence constraints: independent (none); disjoint chains; and, a directed forest, respectively (see Section 2.1 for a full explanation of this notation). Previous results for SUU are given by Lin and Rajaraman [11].

the first approximation algorithms for the problem shown by Lin and Rajaraman [11]. They provide an $O(\log n)$-approximation algorithm when jobs are independent, an $O\left(\log m\log n\frac{\log(n+m)}{\log\log(n+m)}\right)$-approximation algorithm when precedence constraints form disjoint chains, and an $O\left(\log m\log^2 n\frac{\log(n+m)}{\log\log(n+m)}\right)$-approximation algorithm when they form a directed forest. We approach the problems in this model in two stages. First, we show an equivalence between scheduling under uncertainty and a variant of stochastic scheduling. Next, we construct $O(1)$-approximation algorithms for the deterministic analogs of these (variant) problems. These algorithms fit our previous analysis, which in turn yields our result: approximation algorithms for scheduling under uncertainty. It is instructive to contrast the second stage here with our approach to stochastic scheduling. In the latter, approximation algorithms for the deterministic problems have already been shown in previous work. For the former, we designed novel algorithms. Our techniques here are similar to some used by Lin and Rajaraman [11], primarily in the details of our LP-rounding.

The research in this thesis is based on preliminary work appearing in Crutchfield et. al. [3]. Our presentation focuses much more on results for stochastic scheduling and presents results for scheduling under uncertainty in this framework. Thus, our exposition is quite different. A summary of our results is presented in Table 1.

1.2 Thesis organization

The remainder of the thesis is organized as follows. In Chapter 2, we formalize our notation and give necessary definitions and preliminaries. In Chapter 3, we present approximation algorithms for stochastic scheduling problems with independent jobs. In Chapter 4, we show approximation algorithms for stochastic scheduling problems with precedence constraints between jobs. In Chapter 5, we show an equivalence between scheduling under uncertainty and stochastic scheduling and provide algorithms for this new setting. We conclude in Chapter 6. Certain proofs, which may be more technical or distract from the flow of our presentation, are found in Appendix A.

Chapter 2

Preliminaries

This chapter reviews the definitions and notation that will be used throughout the remainder of the thesis. In Section 2.1, we discuss scheduling problems in the context of Graham's notation and then introduce a revised naming scheme. In Section 2.2, we cover instances of scheduling problems, giving notation and explaining our conventions. In Section 2.3, we do the same for schedules, formally defining how machines are assigned to process jobs. Finally, in Section 2.4, we discuss scheduling algorithms, the study of which forms the bulk of later chapters.

2.1 Problems

As described in Chapter 1, a machine scheduling *problem* is defined by a set of constraints on jobs and machines together with an objective function. The goal of a problem, given an *instance*, is to produce a *schedule*, compatible with the constraints, that optimizes the objective function. In this section, we focus on problem definition and notation. Because of the huge number of possible combinations of constraints and objective functions, problems are typically written according to Graham's notation. This notation takes the form of $\alpha \mid \beta \mid \gamma$, with α describing constraints on machines, β constraints on jobs, and γ the objective function.

In this thesis, we study problems of the form $R \mid \beta \mid E[C_{\max}]$. R indicates that the problem considers *unrelated* machines, where each machine may have a different,

arbitrary speed at which it processes each job. For a machine i and job j, we write this speed as $v_{i,j}$. This is the most general class of machines. The two other common classes are identical and uniform, written P and Q. In the former, all machines share a single speed and in the latter, each machine has the same speed for all jobs. $C_{\max}$ refers to the maximum completion time of any job, so $E[C_{\max}]$ represents the objective function of minimizing the expected makespan (equivalently, length) of the schedule. In our case, this expectation is over the joint probability distribution on job processing requirements as well as any randomness used to construct the schedule. There are many more objective functions than machine classes; one common example is to minimize the weighted sum of job completion times, written $\sum w_j C_j$.

The diversity of the problems we consider is found in β, their constraints on jobs. All our problems feature either preemption or restarting. The former allows a job to be switched from one machine to another at any time (or to be idled and not run on any machine) without penalty, and is written as *pmtn*. The latter allows the same movement, but at the cost of losing all previous processing at each switch, and is written as *rstrt*. Because restarting does not play a significant role in the rest of our analysis, we postpone its discussion until Section 3.2. Returning to preemption, the simplest problem we encounter is $R \mid pmtn \mid C_{\max}$, for which a polynomial time optimal algorithm is known [8]. The absence of any mention of precedence constraints means that jobs are *independent*, with no such constraints among them. When precedence constraints are present, *prec* is added to β. We restrict ourselves to cases where constraints form disjoint chains or directed forests, written *chains* and *forest*. Likewise, the no mention of job processing implies that the processing required by a job j, written p_j, is an arbitrary fixed value provided as input. A problem with unit jobs would instead have $p_j = 1 \in \beta$.

We consider three different types of processing distributions. The first, already seen, is deterministic (equivalently, fixed) amounts of processing. The next is stochastic, where the amount of processing that a job needs is set randomly according to a probability distribution provided as input. If a job j requires processing distributed according to distribution $\mathcal{D}_j$, we add $\mathcal{D}_j$ to β. We assume that the mean and median

of these distributions are known and write them as μ_j and δ_j, respectively. We focus on exponentially distributed processing, in which case we add λ_j to β (here λ_j is the rate parameter that uniquely defines an exponential distribution). Finally, we consider problems where all that is known about the amount of processing needed by a job is that it lies in a *bounded range*. We add *range* to β in this case, and assume that $p_j \in [p_{j_{\min}}, p_{j_{\max}}]$. Except for the deterministic case, the actual amount of processing needed by a job *remains hidden* until it completes.

For scheduling under uncertainty, the only difference between problems is the class of precedence graphs allowed. All share the following characteristics: jobs require unit processing and machines have identical unit speed. However, after a machine i processes a job j for a single timestep, there is some (unrelated) probability $q_{i,j}$ that it will fail and the job will not be completed. Multiple machines are also allowed to process a single job at the same time. Machines are allowed to switch between jobs only at integral times. Thus, machines repeatedly process jobs until they either complete or fail. The objective function for these problems remains the same as for those above: to minimize the expected time until all jobs complete.

We do not represent scheduling under uncertainty problems using Graham's notation. Instead, we refer to the family of problems as SUU and append a designator when referring to a concrete problem with specific precedence constraints. For independent jobs, we write SUU-I, for disjoint chains SUU-C, and for directed forests, SUU-T (for *tree*).

2.1.1 Revised notation

Unfortunately, many of the problems we consider have very unwieldy representations when written in Graham's notation. For the problem of bounded job processing with chain precedence constraints, we must write $R \mid pmtn, chains, range \mid E[C_{\max}]$. This impacts readability and wastes space, as R and $E[C_{\max}]$ appear in the representation of almost every problem we consider. We solve this issue by adopting a new notation similar to that used above to describe SUU.

Throughout the rest of this thesis, we write scheduling problems in sans serif

font as X-Y. Here X encodes how job processing is distributed and Y indicates which precedence constraints are present. When we discuss the *family* of problems sharing similar processing distribution, we write X alone. We consider four classes of processing distributions and three classes of precedence constraints. When the amounts of required processing are deterministic, we take X as FIXED. When they are from an unknown distribution on a bounded range, we use RANGE; when they are exponentially distributed, we write EXP; and for arbitrary distributions, we use DISTR. Turning to precedence constraints, when jobs are independent we write Y as I, when constraints form disjoint chains we write C, and when they form a directed forest, we write T (for tree, to avoid confusion with the F in FIXED). All problems written in this way assume preemption is allowed, and when we consider restarting instead we note it explicitly.

In this new style, $R \mid pmtn, chains, range \mid E[C_{\max}]$ becomes RANGE-C. In contrast to Graham's notation, this scheme captures the key characteristics of the problem without becoming cumbersome.

2.2 Instances

Given a fixed scheduling problem $\mathcal{P}$, we use I to designate a problem *instance*. As the problems we consider are closely related, the instances we consider share a similar structure. We find it convenient to write an instance as $I = (J, M, G)$, where J describes the set of jobs, M the set of machines, and G the precedence graph. When precedence constraints form disjoint chains, we sometimes write $\{C_k\}$ for G and when jobs are independent, we omit it. As is convention, we refer to the number of jobs in an instance as n, and the number of machines as m. We often refer to a job as "job $j \in J$", and a machine as "machine $i \in M$". When job j_1 is an ancestor of job j_2 in G, we write $j_1 \prec j_2$.

To aid readability, we sometimes replace J, M, and G with a set of variables sufficient to define them. For example, consider an instance $I = (J, M)$ of EXP-I. An equivalent representation is $I = (\{\lambda_j\}, \{v_{i,j}\})$, where λ_j gives the rate parameter

of each exponential distribution and $v_{i,j}$ gives the speed of each machine on each job. An instance $I = (\{p_j\}, \{v_{i,j}\})$ of FIXED-I differs only by replacing λ_j with the explicit amount of processing p_j. This representation is particularly useful when considering two instances that are somehow related. For example, we can easily write two instances I, I' of FIXED-I with rescaled processing requirements as $I = (\{p_j\}, \{v_{i,j}\})$ and $I' = (\{2p_j\}, \{v_{i,j}\})$.

This notation extends easily to scheduling under uncertainty, where we write an instance as $I = (J, \{q_{i,j}\}, G)$. Here $q_{i,j}$ gives the probability that machine i fails to complete job j after processing it for a single timestep. Note that this is a slight departure from the original notation of Malewicz [13], where $p_{i,j}$ represents the probability that machine i *successfully* completes job j (this conflicts with the standard use of p_j as the processing that a job requires).

2.3 Schedules

A *schedule*, which we denote by Σ, describes how machines should process jobs. Fix an instance I of some scheduling problem $\mathcal{P}$. We represent a schedule for this instance as a union of machine *assignments*

$$\Sigma = \bigcup (i, j, t, x)$$

A single assignment (i, j, t, x) instructs machine i to process job j from time t to time $t + x$; we refer to x as the *duration* of the assignment. All schedules must obey the constraints of $\mathcal{P}$. Thus, if I includes a precedence graph G, then for any jobs j_1 and j_2 such that $j_1 \prec j_2$, each assignment to j_2 cannot start until all assignments to j_1 have finished. Similarly, with the exception of scheduling under uncertainty, a schedule may not allow any job to be processed by more than one machine at a time. Conversely, for scheduling under uncertainty all assignment durations must be integral. The final requirement is that a schedule complete all jobs. Formally, this is

achieved when

$$\forall j, \sum_{(i,j',t,x)\in\Sigma | j'=j} x v_{i,j} \geq p_j$$

This definition posses some interesting subtleties for problems in EXP and DISTR where p_j may not be known until after some portion of the schedule has *executed.* The same is true for SUU, with the caveat that the summation is only over assignments that successfully complete. This issue is given a full treatment in Section 2.4.

Our analysis will often use the following two properties of a schedule. The *length* of a schedule is simply the earliest time by which all jobs have completed. The *load* of a machine is the amount of time during which it is assigned to process any job (we say a machine is *idle* when it is not processing any job), and the load of a schedule is the maximum load of any machine. We define these terms formally as

$$\begin{aligned} \text{len}(\Sigma) &= \max_{(i,j,t,x)\in\Sigma} t + x \\ \text{load}(\Sigma) &= \max_{i'} \sum_{(i,j,t,x)\in\Sigma | i=i'} x \end{aligned}$$

Given this definition of length, we can easily define the composition of two schedules Σ_1 and Σ_2 by delaying all assignments in one schedule by the length of the other. If we would like $\Sigma = \Sigma_1 \cdot \Sigma_2$, we may construct it as

$$\begin{aligned} \Sigma_2' &= \bigcup_{(i,j,t,x)\in\Sigma_2} (i, j, t + \text{len}(\Sigma_1), x) \\ \Sigma &= \Sigma_1 \cup \Sigma_2' \end{aligned}$$

We can also rescale the length of a schedule Σ (and the processing it assigns all jobs) by a factor of c by taking

$$\Sigma' = \bigcup_{(i,j,t,x)\in\Sigma} (i, j, ct, cx)$$

2.4 Scheduling algorithms

The results shown in this thesis are scheduling *algorithms*, which we use $\mathcal{A}$ to represent. Given an instance I of a scheduling problem $\mathcal{P}$, a scheduling algorithm $\mathcal{A}$ for $\mathcal{P}$ takes I as input and produces a feasible schedule Σ. That is, $\Sigma = \mathcal{A}(I)$, and we will use $\mathcal{A}(I)$ in the place of Σ. We adopt the convention of writing $\mathcal{A}_{\mathcal{P}}$ to make it clear that $\mathcal{A}$ is an algorithm is for $\mathcal{P}$. To be concise, we abbreviate our problems with the first letter of the class of job processing and the type of precedence constraints involved. For example, we write an algorithm for DISTR-C as $\mathcal{A}_{DC}$.

Of the scheduling problems we will encounter, many of those with fixed job processing requirements have existing scheduling algorithms. For example, consider the following linear program given by Lawler and Labetoulle [8, Section 2], defined on an instance I of FIXED-I.

$$
\begin{aligned}
\textbf{(LP1)} \qquad & \min t \\
\text{s.t.} \quad \sum_{i \in M} v_{i,j} x_{i,j} \;&\geq\; p_j \quad \forall j \in J && (2.1)\\
\sum_{j \in J} x_{i,j} \;&\leq\; t \quad \forall j \in J && (2.2)\\
\sum_{j \in J} x_{i,j} \;&\leq\; t \quad \forall i \in M && (2.3)\\
x_{ij} \;&\geq\; 0 \quad \forall i \in M, j \in J\ . && (2.4)
\end{aligned}
$$

Let $\left(t^*, \left\{x^*_{i,j}\right\}\right)$ represent an optimal solution to (LP1), notation that we adopt throughout the thesis. Here $x^*_{i,j}$ gives the duration (as defined in Section 2.3) for which machine i should process job j and t^* gives a bound on the time required for assignments with these durations to execute. Equation (2.1) ensures that each job receives sufficient processing to complete, while Equations (2.2) and (2.3) ensure that there is enough time for each job to receive (and machine to execute) the assigned processing. Since all feasible schedules must satisfy these constraints, any schedule Σ for I will obey $t^* \leq \text{len}(\Sigma)$. To complete the analysis, Lawler and Labetoulle [8, Section 3] show a polynomial time procedure for constructing a schedule of length

t^* from $\{x^*_{i,j}\}$. Combining (LP1) and this procedure in sequence yields an optimal algorithm, $\mathcal{A}_{FI}$, for FIXED-I that runs in polynomial time.

As alluded to in Section 2.3, there is a fundamental difference between schedules for deterministic problems and those for stochastic problems. In the former, schedules can be fully *computed* before they are *executed*. Deterministic behavior ensures that the act of machines processing jobs changes nothing. This is not the case when the problem is stochastic. Here, as a schedule executes, information about the amount of processing jobs require is revealed. Jobs may complete and even if they do not, the distributions on the amount of processing they need, conditioned on the amount they have already received, may change.

In this context, we consider *dynamic* scheduling algorithms. That is, we give our algorithms the power to pause arbitrarily during the execution of a schedule and compute new assignments. Essentially, we allow schedules to be constructed on the fly. Because the algorithms we present run in polynomial time, the number of pauses is also no more than polynomial. We follow the convention [16, Chapter 9] that an algorithm running in this way is aware of job completions. We assume that an algorithm is paused immediately after each job completes (although it may decide to immediately restart its current schedule).

2.4.1 Approximation and optimal algorithms

As described, the output of the algorithms we present are schedules, and our objective is, uniformly, to minimize their expected length. Unfortunately, we are unable to reach this precise goal. Instead, we provide *polynomial time approximation algorithms*. Let $OPT_{\mathcal{P}}$ be an optimal scheduling algorithm for a fixed scheduling problem $\mathcal{P}$ with objective function $E[C_{\max}]$. By optimal, we mean that for any instance I, and for any algorithm $\mathcal{A}_{\mathcal{P}}$, $\mathrm{E}[\mathrm{len}(OPT_{\mathcal{P}}(I))] \leq \mathrm{E}[\mathrm{len}(\mathcal{A}_{\mathcal{P}}(I))]$. Then, we call an algorithm $\mathcal{A}'_{\mathcal{P}}$ an α-approximation algorithm for $\mathcal{P}$ if, for all instances I, $\mathrm{E}[\mathrm{len}(\mathcal{A}'_{\mathcal{P}}(I))] \leq \alpha \mathrm{E}[\mathrm{len}(OPT_{\mathcal{P}}(I))]$. Here α is the *approximation ratio* achieved by $\mathcal{A}$. Because all the algorithms we present run in time polynomial in the size of their input, from this point forward we take *approximation algorithm* as short for polynomial

time approximation algorithm.

The constraints we place on $OPT_{\mathcal{P}}$ are much lighter than those on the approximation algorithms we construct. To begin with, we allow it unbounded time and space to compute and an unlimited number of pauses. In this context, we could view the problem as some kind of Markov decision process, but this does not facilitate our analysis (and also becomes more complex when job processing distributions are not memoryless). Instead, we give $OPT_{\mathcal{P}}$ even more power; we allow it knowledge of processing needs at the beginning of its computation. These amounts are still distributed stochastically, but $OPT_{\mathcal{P}}$ knows what they are immediately, whereas our algorithms only discover them once jobs complete. Our approximation ratios hold against this more powerful $OPT_{\mathcal{P}}$ and thus clearly do so against any weaker sense of optimality.

Chapter 3

Stochastic Scheduling of Independent Jobs

This chapter covers our approximation algorithms for EXP-I and DISTR-I, where job processing requirements are distributed stochastically and jobs are independent. We begin in Section 3.1 by constructing an $O(\log\log\min\{m,n\})$-approximation algorithm for EXP-I. In Section 3.2, we show that this algorithm can be modified to apply when restarting is allowed instead of preemption. In Section 3.3, we extend these algorithms to work in the DISTR setting, with a slight increase in approximation ratio.

3.1 Algorithms for EXP-I

The main result of this section is an $O(\log\log\min\{m,n\})$-approximation algorithm for EXP-I. We take a two-step approach. First, we show an $O\left(\log\max_j 2\frac{p_{j\max}}{p_{j\min}}\right)$-approximation algorithm for RANGE-I, where the amount of processing needed by a job is bounded. To take advantage of this result we address the *head* and *tail* of the job processing distributions. That is, we show how to assume that for each job j, $p_j \in \left[\frac{\ln 2}{\lambda_j}, \frac{2\ln\min\{m,n\}}{\lambda_j}\right]$, while only increasing the expected makespan of schedules by a constant factor. As the bounds of this range differ by a factor of $O(\log\min\{m,n\})$, applying our algorithm for RANGE-I yields an $O(\log\log\min\{m,n\})$-approximation

algorithm for EXP-I.

We now show an algorithm, $\mathcal{A}_{RI}$, for scheduling an instance $I = (\{p_{j_{\min}}, p_{j_{\max}}\}, M)$ of RANGE-I. $\mathcal{A}_{RI}$ runs in $R = \left\lceil \log \max_j 2\frac{p_{j_{\max}}}{p_{j_{\min}}} \right\rceil$ rounds. In each round it executes the schedule generated by the optimal, polynomial time algorithm $\mathcal{A}_{FI}$ for FIXED-I presented in Section 2.4. Instances of FIXED-I have jobs with fixed amounts of required processing, so in round r we set this value to be $p_j^r = 2^r p_{j_{\min}}$ for each job j. A job j that has completed by the beginning of round r has its required processing set to $p_j^r = 0$. The schedule executed by $\mathcal{A}_{RI}(I)$ in round r is thus $\mathcal{A}_{FI}\left(\left(\{p_j^r\}, M\right)\right)$. We confirm that $\mathcal{A}_{RI}(I)$ completes all jobs as follows. By our choice of R, in the last round a job j has either completed or has its processing set as $2^{\log \max_j 2\frac{p_{j_{\max}}}{p_{j_{\min}}}} p_{j_{\min}} \geq p_{j_{\max}}$. Thus, all jobs complete by the end of the final round. The following lemma gives a bound on the length of each round.

Lemma 3.1 *Let I be an instance of RANGE-I. Then the length of any round r of $\mathcal{A}_{RI}$ is no more than* $2\text{len}(OPT_{RI}(I))$.

PROOF. By construction, the length of the rth round of $\mathcal{A}_{RI}$ is equal to the length of the schedule produced by $\mathcal{A}_{FI}$ for $\left(\{p_j^r\}, M\right)$. We claim that for all rounds r and all jobs j, $p_j^r \leq 2p_j$, where p_j is the actual (hidden) processing needed by job j. If this is the case, we can upper bound the length of round r in terms of scheduling jobs requiring such processing as

$$\text{len}\left(\mathcal{A}_{FI}\left(\left(\{p_j^r\}, M\right)\right)\right) \leq \text{len}(\mathcal{A}_{FI}\left(\left(\{2p_j\}, M\right)\right))$$

We further claim that

$$\text{len}(\mathcal{A}_F(\left(\{2p_j\}, M\right))) \leq 2\text{len}(OPT_{RI}(I))$$

This can be seen as follows. $OPT_{RI}(I)$ completes all jobs and thus each job j receives at least p_j processing in this schedule. Repeating it twice gives each job at least $2p_j$ processing, and thus completes all jobs in an instance $(\{2p_j\}, M)$ of FIXED-I. Because $\mathcal{A}_{FI}$ is optimal, the length of the schedule produced by $\mathcal{A}_{FI}\left(\left(\{2p_j\}, M\right)\right)$

must thus be no longer than $2\text{len}(OPT_{RI}(I))$.

Taking these two bounds together, proving our claim that $p_j^r \leq 2p_j$ completes the lemma. We give a proof by contradiction. Assume that for some $r > 1$ and j, $p_j^r > 2p_j$ (the first round has $p_j^1 = p_{j_{\min}} \leq p_j$). Then $p_j^{r-1} > p_j$, so job j must have completed by the end of round $r-1$. However, all jobs completed by the start of round r have $p_j^r = 0$, contradicting our assumption. □

$\mathcal{A}_{RI}(I)$ runs $R = \left\lceil \log \max_j 2\frac{p_{j_{\max}}}{p_{j_{\min}}} \right\rceil$ rounds and by Lemma 3.1 each round has length at most $2\text{len}(OPT_{RI}(I))$. Thus, $\mathcal{A}_{RI}$ is an $O(R)$-approximation algorithm for RANGE, which is stated in the following lemma.

Lemma 3.2 *$\mathcal{A}_{RI}$ is an $O\left(\log \max_j 2\frac{p_{j_{\max}}}{p_{j_{\min}}}\right)$-approximation algorithm for RANGE-I.*

Given the nature of the approximation ratio in Lemma 3.2, we would like to reduce the gap between $p_{j_{\min}}$ and $p_{j_{\max}}$ to the extent possible. We thus turn now to handling jobs that require an amount of processing that is either *small* or *large*. We begin by handling small jobs, which we take to mean requiring processing $p_j < \text{E}[Exp(\lambda_j)] = \frac{\ln 2}{\lambda_j}$. The next lemma shows that the processing that a job requires may be assumed to be at least this value at very little cost. If this assumption is violated (e.g., a job completes before it is supposed to), we simply idle any machines that are assigned to it in the future.

Lemma 3.3 *Let $I = I' = (J, M)$ be two identical instances of EXP-I. For I', modify the probability distributions on job processing so that if a job j requires processing $p_j < \frac{\ln 2}{\lambda_j}$, it is increased to $p_j = \frac{\ln 2}{\lambda_j}$. Then $\text{E}[\text{len}(OPT_{EI}(I'))] \leq 2\text{E}[\text{len}(OPT_{EI}(I))]$.*

PROOF. Our analysis of I' proceeds as follows. We partition the jobs in J into two subsets. Those with $p_j > \frac{\ln 2}{\lambda_j}$, we place in L. The remainder (which are small) are placed in $\overline{L}$.[1] We consider processing these two subsets of jobs separately and show that, in expectation, the time spent processing the jobs in L dominates. This remains true even when the jobs in $\overline{L}$ have their required processing increased in I'. Then

[1]We are showing a possible algorithm for OPT_{EI}, and so may assume knowledge of these values.

because the jobs in L have their processing distributed identically in I and I', we are able to bound the expected length of $OPT_{EI}(I')$ in terms of that of $OPT_{EI}(I)$.

Take L and $\overline{L}$ to be defined as above and let $\overline{L}^*$ represent the jobs in $\overline{L}$ with their processing requirements adjusted. We define the instances $I'_L = (L, M)$, $I'_{\overline{L}} = \left(\overline{L}, M\right)$, and $I'_{\overline{L}^*} = \left(\overline{L}^*, M\right)$ of EXP-I to easily express processing these subsets separately. We observe that for all j, $\Pr\left[p_j > \frac{\ln 2}{\lambda_j}\right] = 1/2$ and thus L is a uniform random subset of J. So, by the law of total expectation we have that

$$\begin{aligned}
\mathrm{E}[\mathrm{len}(OPT_{EI}(I'))] &= \frac{1}{2^n}\sum_L \mathrm{E}[\mathrm{len}(OPT_{EI}(I')) \mid L] && (3.1)\\
&\leq \frac{1}{2^n}\sum_L \mathrm{E}[\mathrm{len}(OPT_{EI}(I'_L)) \mid L] \\
&\quad + \frac{1}{2^n}\sum_L \mathrm{E}\left[\mathrm{len}\left(OPT_{EI}\left(I'_{\overline{L}^*}\right)\right) \mid L\right] && (3.2)\\
&\leq \frac{2}{2^n}\sum_L \mathrm{E}[\mathrm{len}(OPT_{EI}(I'_L)) \mid L] && (3.3)\\
&= 2\mathrm{E}[\mathrm{len}(OPT_{EI}(I'_L))] && (3.4)\\
&\leq 2\mathrm{E}[\mathrm{len}(OPT_{EI}(I))] && (3.5)
\end{aligned}$$

This sequence is derived as follows. Equation (3.2) holds because it is legal to schedule by dividing jobs into two subsets and processing them separately (and so upper bounds an optimal algorithm). The key to the proof is Equation (3.3). Here, we note that there is a one-to-one correspondence between each value of $\overline{L}^*$ in the second term and each value of L in the first. Since each job $j \in \overline{L}^*$ has $p_j = \frac{\ln 2}{\lambda_j}$, and each job $j \in L$ has $p_j > \frac{\ln 2}{\lambda_j}$, we conclude that

$$\sum_L \mathrm{E}\left[\mathrm{len}\left(OPT_{EI}\left(I'_{\overline{L}^*}\right)\right) \mid L\right] \leq \sum_L \mathrm{E}[\mathrm{len}(OPT_{EI}(I'_L)) \mid L]$$

Because the jobs in $L \subseteq J$ have identical processing distributions in both I and I', Equations (3.4) and (3.5) complete the proof. □

Lemma 3.3 shows that we can impose a lower bound on the amount of processing required by a job at a very low cost. The obvious next step is to combine Lemmas 3.2

and 3.3 to produce a scheduling algorithm that can handle processing distributions with no lower bound (e.g., $p_j \in [0, p_{j_{\max}}]$). The next lemma does just that.

Lemma 3.4 *$I = (\{\lambda_j\}, M)$ be an instance of EXP-I, and let $I' = \left(\left\{\left[\frac{\ln 2}{\lambda_j}, p_{j_{\max}}\right]\right\}, M\right)$ be an instance of RANGE-I. Let the algorithm $\mathcal{A}'_{EI}$ (for EXP-I) schedule according $\mathcal{A}_{RI}(I')$. Then $\mathcal{A}'_{EI}$ will complete all jobs j such that $p_j \leq p_{j_{\max}}$, and its expected length will obey $\mathrm{E}[\mathrm{len}(\mathcal{A}'_{EI}(I))] \leq O(\max_j \log(2p_{j_{\max}}\lambda_j)\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$.*

PROOF. The proof of this lemma is very similar to those of Lemmas 3.1 and 3.2. As in the proof of Lemma 3.2, any job j uncompleted at the start of the final round receives at least $p_{j_{\max}}$ processing, so any job no longer than this will be completed by $\mathcal{A}'_{EI}$. Lemma 3.1 also holds with the exception of the first round. This exception arises from the possibility that some jobs may only require processing $p_j < p_{j_{\min}} = \frac{\ln 2}{\lambda_j}$, violating lower bound assumed by $\mathcal{A}_{RI}$. Thus, the expected length of $\mathcal{A}'_{EI}(I)$ for all rounds besides the first is

$$O\left(\max_j \log\left(2\frac{p_{j_{\max}}}{p_{j_{\min}}}\right)\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]\right) = O\left(\max_j \log(2p_{j_{\max}}\lambda_j)\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]\right)$$

To bound the expected length of the first round, wherein all jobs have their required processing fixed as $p_j^1 = \frac{\ln 2}{\lambda_j}$, we simply apply Lemma 3.3. We conclude that the expected length of the first round of $\mathcal{A}'_{EI}$ is at most $2\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]$, and that the expected length of $\mathcal{A}'_{EI}$ remains $O(\max_j \log(2p_{j_{\max}}\lambda_j)\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. □

Lemma 3.4 shows that our algorithm $\mathcal{A}_{RI}$ for bounded amounts of job processing also applies when these amounts are exponentially distributed, as long as we ignore large jobs requiring $p_j > p_{j_{\max}}$ processing. Thus, to provide an approximation algorithm for EXP-I, we need to construct a scheduling algorithm that can also handle large jobs. We show how to achieve this in the following lemma.

Lemma 3.5 *Let $I = \{J, M\}$ be an instance of EXP-I. Let $\mathcal{A}_{EI}$ be a polynomial time algorithm for EXP-I that completes only jobs $j \in J$, which satisfy $p_j \leq \frac{2\ln\min\{n,m\}}{\lambda_j}$ and obeys $\mathrm{E}[\mathrm{len}(\mathcal{A}_{EI}(I))] \leq \alpha\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]$. Then there is an $(\alpha+O(1))$-approximation algorithm for EXP-I.*

PROOF. We begin by executing $\mathcal{A}_{EI}(I)$, which completes all jobs with $p_j \leq \frac{2\ln\min\{n,m\}}{\lambda_j}$ with an expected makespan of $\alpha\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]$. We then prove the lemma by providing a schedule that can complete all the remaining *long* jobs and has an expected makespan of only $O(\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. Here we give such a schedule for the case when $n \leq m$. When $m < n$, the problem becomes more involved and so we leave the consideration of this case to Appendix A.

When $n \leq m$, we will schedule jobs one at a time (in arbitrary order), assigning each job to the machine on which it is processed the fastest. It is easy to see that if any jobs remain uncompleted after executing $\mathcal{A}_{EI}(I)$, then the expected makespan of this schedule is $O(n\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. However, the probability that any job requires processing $p_j > \frac{2\ln n}{\lambda_j} \geq \frac{2\ln m}{\lambda_j}$ (that is, that any long jobs exist) is by the union bound at most $\frac{1}{n}$. If there are no such jobs, then the expected makespan of the schedule is zero. Thus, by conditional expectation, the expected makespan of this schedule is $O(\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. □

If we take $p_{j_{\max}} = \frac{2\ln\min\{n,m\}}{\lambda_j}$, Lemma 3.4 shows that $\mathcal{A}_{RI}$ completes all jobs with length bounded by this value, and has an expected makespan bounded by

$$O\left(\max_j \log(2p_{j_{\max}}\lambda_j)\mathrm{E}[\mathrm{len}(OPT_{EI}(I))]\right) = O(\log\log\min\{m,n\}\,\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$$

Applying Lemma 3.5 then yields an $O(\log\log\min\{m,n\})$-approximation algorithm for EXP-I, which we restate in the theorem below.

Theorem 3.6 *There is an $O(\log\log\min\{m,n\})$-approximation algorithm for EXP-I.*

3.2 Restarting instead of preemption

In this section we show how to modify our previous algorithms to yield an $O(\log\log n)$-approximation algorithm for EXP-I when restarting is allowed instead of preemption. Recall from Section 2.1 that under restarting, whenever a job switches machines its

processing is reset to zero. To complete all jobs, a schedule Σ must then satisfy

$$\forall j \in J \quad \exists (i,j,t,x) \in \Sigma \text{ s.t. } xv_{i,j} \geq p_j$$

Restarting is a more realistic restriction than preemption because of the penalty it enforces on job movement. It is also a middle ground. Because our algorithms from Section 3.1 rely on composing schedules constructed independently, none of our techniques would apply in the non-preemptive case, where each job must run uninterrupted on one machine until completion. Instead we take the middle ground, requiring a job to receive all necessary processing from a single machine, but allowing earlier abortive attempts.

We begin by modifying our algorithm for RANGE-I to work with restarting. We do this by having it schedule according to the 2-approximation algorithm for $R \mid\mid C_{\max}$ (in which each job must run uninterrupted on exactly one machine), shown by Lenstra et. al. [10], instead of the optimal algorithm for FIXED-I ($R \mid pmtn \mid C_{\max}$) described in Section 2.4. The next lemma shows that this modification does not affect our approximation ratio.

Lemma 3.7 *There is an* $O\left(\log max_j 2^{\frac{p_{j\max}}{p_{j\min}}}\right)$ *for RANGE-I when restarting is allowed instead of preemption.*

PROOF. Let $\mathcal{A}'_{FI}$ be the 2-approximation algorithm for $R \mid\mid C_{\max}$ shown by Lenstra et. al. [10] and I be an instance of RANGE-I with restarting. We follow the construction of $\mathcal{A}_{RI}$ from Section 3.1, scheduling multiple rounds with fixed processing requirements that double each round. However, we schedule each round according to $\mathcal{A}'_{FI}(\{p_j^r\}, M))$ instead of $\mathcal{A}_{FI}(\{p_j^r\}, M))$. We assume that jobs will be switched every round, and so reset the processing of all uncompleted jobs at the end of a round to zero.

We claim that when applied to I, each round of this new algorithm takes time at most $4\text{len}(OPT_{RI}(I))$ (note that in throughout section we take OPT to imply restarting rather than preemption), and that all jobs complete by the final round.

The first claim follows directly from the analysis in Lemma 3.1, substituting $\mathcal{A}'_{FI}$ for $\mathcal{A}_{FI}$ and restarting for preemption. The constant here increases from 2 to 4 because $\mathcal{A}'_{FI}$ is a 2-approximation, whereas $\mathcal{A}_{FI}$ is optimal. The second claim is true because each job that is uncompleted by the start of the final round receives at least $p_{j_{\max}}$ processing *in that round.* Thus, all jobs will complete by the end of the final round even though their processing is reset each round.

$\mathcal{A}'_{FI}$ is non-preemptive and, thus, during the round in which a job j completed, it must have received sufficient processing from a single machine without interruption. This ensures that the completion condition from the beginning of the section is met. □

We turn this result into an $O(\log\log n)$-approximation algorithm for EXP-I with restarting by following the analysis in Section 3.1. First we apply Lemma 3.3, showing that for an instance I of EXP-I, the algorithm given by Lemma 3.7 completes all jobs having $p_j \leq \frac{2\ln n}{\lambda_j}$ with an expected makespan of $O(\log\log n \cdot \mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. Any jobs that remain uncompleted are then run with one-at-a-time scheduling, again adding a factor of only $O(\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$ to our expected makespan. Crucially, neither the proof of Lemma 3.3 nor one-at-a-time scheduling relies on EXP-I being preemptive. Thus, we can apply them without change, yielding the following theorem.

Theorem 3.8 *There is an $O(\log\log n)$-approximation algorithm for EXP-I when restarting is allowed instead of preemption.*

Unfortunately, the analysis of our $O(\log\log m)$-approximation algorithm for EXP-I does depend on allowing preemption. In particular, it relies on executing multiple independent schedules and applying the resulting cumulative processing to jobs. Thus, we are unable to generalize this result to apply with restarting instead of preemption.

3.3 Generalizing to DISTR

In EXP-I, all jobs have their required processing distributed exponentially. However, our approximation algorithms have used only a few basic facts about the exponential

distribution and are easily adapted to arbitrary distributions (although our approximation ratios suffer). In this section, we give approximation algorithms for problems in DISTR. Recall from Section 2.1 that for problems in this class, the processing needed by a job j is set according to an arbitrary probability distribution $\mathcal{D}_j$ with mean μ_j and median δ_j.

There are two places where previous analysis relied explicitly on the distribution of processing amounts. First, the median of the distribution was frequently used as a "lower bound" on the amount of processing required by a job. For the exponential distribution, this value is $\frac{\ln 2}{\lambda_j}$ and it appears very frequently throughout the previous two sections. It should be changed to δ_j for DISTR; the analysis itself is unchanged beyond this substitution. As an example, consider the following restatement of Lemma 3.3.

Lemma 3.9 *Let $I_1 = I_2 = (\{\mathcal{D}_j\}, M)$ be two identical instances of DISTR-I. For I_2, modify the probability distributions on job processing such that if a job j is assigned an amount of processing $p_j < \delta_j$, it is increased to $p_j = \delta_j$. Then $\mathrm{E}[\mathrm{len}(OPT_{DI}(I_2))] \leq 2\mathrm{E}[\mathrm{len}(OPT_{DI}(I_1))]$.*

Second, the tail of the exponential distribution is used in Lemma 3.5 to ensure that after allocating $\frac{2\ln\min\{n,m\}}{\lambda_j}$ to a job, it completes with probability at least $1-\frac{1}{\min\{n^2,m^2\}}$. For DISTR this upper bound should be changed. For each job j, let b_j be the minimum value such that $\Pr[p_j > b_j] \leq \max\{1/n^2, 1/m^2\}$. Then b_j should replace uses of $\frac{2\ln\min\{n,m\}}{\lambda_j}$. Again, the analysis remains the same after this replacement. By Markov's Inequality, b_j is at most $2n\mu_j$, but tighter upper bounds may be possible depending on the $\{\mathcal{D}_j\}$ provided. The following lemma restates Lemma 3.5 using b_j.

Lemma 3.10 *Let $I = (J, M)$ be an instance of DISTR-I. Let $\mathcal{A}_{DI}$ be a polynomial time algorithm for DISTR-I such that $\mathrm{E}[\mathrm{len}(\mathcal{A}_{DI}(I))] \leq \alpha\mathrm{E}[\mathrm{len}(OPT_{DI}(I))]$, that completes only jobs $j \in J$ which satisfy $p_j \leq b_j$. Then there is an $(\alpha+O(1))$-approximation algorithm for DISTR-I.*

We now state our approximation algorithms for DISTR-I. This algorithm may be constructed as in the previous sections, with the modifications explained above.

Theorem 3.11 *There is an* $O\left(\max_j \log \frac{b_j}{\mu_j}\right)$*-approximation algorithm for DISTR-I.*

Let b'_j be the minimum value such that $\Pr\left[p_j > b'_j\right] \leq 1/n^2$. Then our approximation algorithm for DISTR-I can be extended to an $O\left(\max_j \log \frac{b'_j}{\mu_j}\right)$-approximation algorithm when restarting is allowed instead of preemption, as shown in Section 3.2.

Chapter 4

Stochastic Scheduling of Jobs with Precedence Constraints

This chapter covers our approximation algorithms for EXP and DISTR when there are precedence constraints between jobs. We begin in Section 4.1, where we show an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-C. In Section 4.2, we generalize this result to an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-T. Finally, in Section 4.3, we show that these results apply in the DISTR setting with a slight increase in approximation ratio (as in Section 3.3). None of the algorithms in this chapter apply when restarts are allowed instead of preemption.

4.1 Algorithms for EXP-C

Unfortunately, our results in Chapter 3 do not hold for EXP-C, the variant of EXP with precedence constraints composed of disjoint chains. The failure occurs in Lemma 3.2, which does not hold when RANGE-I is generalized to RANGE-C. At issue is the lemma's assumption that all jobs are eligible for processing from the start of the algorithm. This is not true for instances of RANGE-C, where jobs are eligible only after their predecessors complete.

Because these algorithms do not apply beyond EXP-I, in this section we take a

different tack. We are eventually able to show an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-C, and using a chain decomposition developed by Kumar et. al. [7], an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-T. Following the approach of Shmoys et. al. [18] and Lin and Rajaraman [11], we begin by relaxing EXP-C to allow *pseudoschedules.*

In a pseudoschedule for an instance $I=(J,M,G)$ of EXP-C, a machine may process two jobs $j_1,j_2\in J$ simultaneously if they are independent of each other (that is, neither $j_1\prec j_2$ nor $j_2\prec j_1$). The *congestion* of a pseudoschedule is then the maximum number of jobs that any machine ever simultaneously processes (a pseudoschedule with congestion one is thus a schedule). Pseudoschedules are otherwise identical to schedules; in particular, they must still obey precedence constraints. The length and load of a pseudoschedule are defined identically to those of a schedule (these definitions were presented in Section 2.3). We write pseudoschedules with Σ, and when we consider a pseudoschedule as opposed to a strict schedule, we mention this explicitly.

This relaxation allows us to treat the jobs lying on each chain separately, which will prove crucial to our analysis. We will show a "pseudoscheduling" algorithm for FIXED-C, and then prove that it produces pseudoschedules of length $O(OPT_{EC}(I))$ when job lengths are set as $p_j=\frac{\ln 2}{\lambda_j}$. Then, we will adapt this algorithm to apply to EXP-C. This technique is similar to our analysis in Section 3.1, but much more involved. Finally, we will reduce the congestion of the pseudoschedules produced to one, yielding the desired approximation algorithm for EXP-C.

Consider the following linear program for an instance $I=(J,M,\{C_k\})$ of FIXED-C, which is essentially a restatement of one presented by Shmoys et. al. [18, Section 4.3].

$$\begin{aligned}
\textbf{(LP2)}\qquad & \min t \\
\text{s.t.}\quad & \sum_{i\in M} x_{i,j}v_{i,j} \;\geq\; p_j \quad \forall j\in J && (4.1)\\
& \sum_{i\in M} x_{i,j} \;\leq\; d_j \quad \forall j\in J && (4.2)
\end{aligned}$$

$$\sum_{j \in C_g} d_j \leq t \quad \forall g \in \{1, 2, \ldots, k\} \tag{4.3}$$

$$\sum_{j \in J} x_{i,j} \leq t \quad \forall i \in M \tag{4.4}$$

$$x_{i,j} \geq 0 \quad \forall i \in M, j \in J \tag{4.5}$$

Here $x_{i,j}$ gives the amount of time that machine i should spend processing job j, and d_j gives the length of the window during which j should be processed. Equation (4.1) ensures that all jobs receive sufficient processing. Equations (4.2) and (4.3) ensure that there is enough time to execute all assignments while still obeying precedence constraints. Finally, Equation (4.4) bounds the load of the machines. We let $\left(d_j^*, x_{i,j}^*, t^*\right)$ be an optimal solution to the linear program. Let $\prec$ be the partial order on jobs defined by $\{C_k\}$, and define an arbitrary total order on the machines $i \in M$. We define the time at which machine i should start to process job j as $\text{start}(i,j) = \sum_{j' \prec j} d_{j'}^* + \sum_{i' < i} x_{i',j}^*$. The pseudoschedule produced by our algorithm is then given by

$$\mathcal{A}_{FC}(I) = \bigcup_{i,j} \left\{ (i, j, \text{start}(i,j), \text{start}(i,j) + x_{i,j}^*) \right\}$$

By construction, in this pseudoschedule, a job j is processed only during a window of length d_j^*. Further, this window starts after all of j's predecessors in G have completed, and ends before any of j's successors have received any processing. This guarantees that no machine will process two jobs lying along the same chain at the same time. Each machine i processes for a total of $\sum_j x_{i,j}^*$ time so machine loads are bounded.

To simplify later analysis, we convert $\mathcal{A}_{FC}$ to allocate assignments of discrete duration. Let $f = \min_{i,j} \frac{p_j}{2v_{i,j}m}$. We round $x_{i,j}^*$ down to the nearest multiple of f and then double it (hence all assignments are integral multiples of f). By our choice of f, $\max_j \sum_i \frac{v_{i,j}}{f} \leq p_j$. This implies that our rounding reduced the processing of each job at most by half, so that doubling ensures that all jobs receive sufficient processing. Note that this procedure at most doubles t^*.

We now move from FIXED-C to EXP-C. We begin by showing an upper bound on

the length of $\mathcal{A}_{FC}(I)$ in terms of the length of the optimal pseudoschedule $OPT_{EC}(I')$ for a closely related instance I' of EXP-C.

Lemma 4.1 *Let $I = (\{p_j\}, M, G)$ be an instance of FIXED-C, and $I' = (\{\lambda_j\}, M, G)$ be an instance of EXP-C, such that for each job j, $p_j = \frac{\ln 2}{\lambda_j}$. Then* $\text{len}(\mathcal{A}_{FC}(I)) \leq O(\text{E}[\text{len}(OPT_{EC}(I'))])$.

PROOF. The value of t^* (before rounding) is shown to be a lower bound on $\text{E}[\text{len}(OPT_{FC}(I'))]$ by Shmoys et. al. [18], and thus $\text{len}(\mathcal{A}_{FC}(I)) \leq O(\text{len}(OPT_{FC}(I)))$. To complete the proof, we show that for an instance of EXP-C, setting the minimum processing p_j required by a job j to be $\frac{\ln 2}{\lambda_j}$ only increases the expected length of the optimal pseudoschedule by a constant factor. That is, we show that Lemma 3.3 still holds for EXP-C with pseudoschedules. Precedence constraints complicate proving this claim, so we present the remainder of this proof in Appendix A. □

Our next step is to transform $\mathcal{A}_{FC}$ into a pseudoscheduling algorithm for EXP-C. Thus, we need to handle jobs that require processing $p_j \geq \frac{\ln 2}{\lambda_j}$. We will construct a *schedule* for each chain C_g independently, and then show good bounds on the length and load of the *pseudoschedule* that executes all of them simultaneously[1]. We will find it helpful to look at the projection of $\mathcal{A}_{FC}$ onto certain jobs. Given an instance I, we will write $\mathcal{A}(I)|_Q$ to refer to the projection of $\mathcal{A}(I)$ onto the jobs $Q \subseteq J$ (that is, to ignore all assignments to jobs not in Q).

The intuition underlying the algorithm we present is as follows. For a single chain, if we consider only jobs that are short enough (e.g., have d_j^* small compared to t^*), then we can schedule them by repeatedly executing $\mathcal{A}_{FC}(I)|_{\{j\}}$ until j completes. By standard Chernoff bound arguments, the time it takes for all short jobs along a chain C_g to complete in this scheme will be $O(t^*)$ with high probability. The core of the problem is then jobs that are long with respect to t^*. Here we show how to schedule these jobs with the algorithm for independent jobs shown in Section 3.1.

[1]Because we are creating a pseudoschedule, we can consider jobs on different chains independently of each other.

We now describe a schedule $\mathcal{A}^g_{EC}(I)$, which processes the jobs in chain C_g. For each job $j \in C_g$, we check whether $d^*_j \leq \ell_{good} = \frac{t^*}{\log(n+m)}$. If so, we call j *good*. $\mathcal{A}^g_{EC}$ processes good jobs by repeatedly executing $\mathcal{A}_{FC}(I)|_{\{j\}}$ until j completes. If $d_j > \ell_{good}$, we then we say j is *bad*. When $\mathcal{A}^g_{EC}(I)$ encounters a bad job, it inserts a *blank* of length ℓ_{good} into its schedule. That is, it delays the processing of j's successor in C_g (if it exists) by ℓ_{good} timesteps. Of course, bad jobs must be processed somehow. We describe how this is done below.

Our pseudoscheduling algorithm $\mathcal{A}_{EC}$ executes all $\mathcal{A}^g_{EC}$ in parallel. Every ℓ_{good} timesteps, we pause $\mathcal{A}_{EC}(I)$ (and thus all $\mathcal{A}^g_{EC}(I)$). At each pause, we check every $\mathcal{A}^g_{EC}(I)$ and see if it is currently blanked on a bad job. Let $B \subseteq J$ be the set of such jobs. $\mathcal{A}_{EC}(I)$ then schedules these jobs according to the scheduling algorithm for independent jobs shown in Theorem 3.6 and unpauses the $\mathcal{A}^g_{EC}(I)$. This technique is feasible because all jobs in B are from distinct chains and so are independent of each other. This completes our description of $\mathcal{A}_{EC}$.

Our next step should clearly be to transform $\mathcal{A}_{EC}$ from a pseudoschedule into a schedule. To do this, we will need bounds on both the length and load of $\mathcal{A}_{EC}(I)$ (the reason for this second requirement becomes apparent once we present Lemma 4.4). For ease of analysis, we break these quantities into two parts: that which accrues during the *good part* of $\mathcal{A}_{EC}$, when it is processing good jobs, and that which accrues in the *bad part* of $\mathcal{A}_{EC}$, when it is processing bad jobs. Because we pause every ℓ_{good} timesteps during the good part of $\mathcal{A}_{EC}(I)$ to handle bad jobs, it makes sense to bound the length of the good portion first. We use the following technical lemma to prove our bounds; its proof appears in Appendix A.

Lemma 4.2 *For each $j \in \{1, 2, \ldots, n\}$, let y_j be a positive integer drawn from the geometric distribution $\Pr[y_j = k] = (1/2)^k$ (where k is a positive integer), and let $b_j \geq 1$ be a weight associated with each j. Let W and η be chosen such that $W/\log\eta \geq b_j$ for all j, $W \geq \sum_j 2b_j$, and $\log\eta \leq W$. Then $\sum_j b_j y_j \leq O(cW)$ with probability at least $1 - 1/\eta^c$, for any positive constant c.*

We can use Lemma 4.2 to obtain a high probability bound on both the length

and load of the bad portion of $\mathcal{A}_{EC}(I)$. Recall that $(d_j^*, x_{i,j}^*, t^*)$ represents an optimal solution to (LP2). Let $W = t^*$, $\eta = n + m$, and y_j be the number of times $\mathcal{A}_{FC}(I)|_{\{j\}}$ is repeated before job j completes. If we set b_j according to $d_j^* = \text{len}(\mathcal{A}_{FC}(I)|_{\{j\}})$, the lemma implies that with high probability the good portion of $\mathcal{A}_{EC}(I)$ has length $O(\text{E}[\text{len}(OPT_{EC}(I))])$. If we instead fix i and set $b_j = x_{i,j}^*$, then applying Lemmas 4.1 and 4.2 with a union bound over machines shows that the load of the good portion is $O(\text{E}[\text{len}(OPT_{EC}(I))])$, again with high probability.

We now turn to the bad portion of $\mathcal{A}_{EC}(I)$. We concern ourselves only with its length. Given the results in the previous paragraph, the number of times $\mathcal{A}_{EC}(I)$ pauses is bounded by $O(\log(n+m))$ with high probability, because with this probability the length of the good portion is bounded by $O(\text{E}[\text{len}(OPT_{EC}(I))])$, and we pause every $\ell_{good} = O\left(\frac{\text{E}[\text{len}(OPT_{EC}(I))]}{\log(n+m)}\right)$ timesteps as it executes. By Theorem 3.6 the expected time required to handle each pause is $O(\log\log\min\{n,m\}\,\text{E}[\text{len}(OPT_{EC}(I))])$. Thus, we see that with high probability, the total length of this portion is bounded by $O(\log(n+m)\log\log\min\{n,m\}\,\text{E}[\text{len}(OPT_{EI}(I))])$ as well. We restate these results in the lemma below.

Lemma 4.3 *With probability at least* $1 - \frac{1}{(n+m)}$, *both the load and length of the good portion of* $\mathcal{A}_{EC}(I)$ *are bounded by* $O(\text{E}[\text{len}(OPT_{EC}(I))])$. *The length of the bad portion of* $\mathcal{A}_{EC}(I)$ *is bounded by* $O(\log(n+m)\log\log\min\{n,m\}\,\text{E}[\text{len}(OPT_{EC}(I))])$ *with the same probability.*

To convert $\mathcal{A}_{EC}(I)$ to a schedule, we need to eliminate the *congestion* in the good part of the pseudoschedule (since the bad part of $\mathcal{A}_{EC}(I)$ is a schedule, we do not need to modify it).[2] Recall that the congestion of a pseudoschedule is defined as the maximum number of jobs any machine is assigned to process simultaneously. Thus, to produce a feasible schedule, we must reduce the congestion of $\mathcal{A}_{EC}(I)$ to one. This ensures that each machine is processing only a single job at a time. We take a first step towards eliminating congestion by applying the random delay technique

[2]We note, ironically, that what we previously considered to be the good portion of our pseudoschedule is now problematic.

developed by Leighton et. al. [9] (and later used by Shmoys et. al. [18]), described in the lemma below.

Lemma 4.4 *Let $I = (J, M, \{C_k\})$ be an instance of EXP-C, and Σ be a pseudoschedule for I that satisfies* $\text{load}(\Sigma) = \text{poly}(n+m)$. *Let the pseudoschedule Σ' idle each chain $C_k \in G$ for a random integral time chosen from $[0, \text{load}(\Sigma)]$, and then schedule according to Σ. Then the congestion of Σ' is $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$ with probability at least $1 - 1/n$.*

Note that to use Lemma 4.4, $\mathcal{A}_{EC}(I)$ must have load that is polynomial in $(n+m)$. We claim that this can be assumed and provide a proof in Appendix A.

Claim 4.5 *The pseudoschedule $\mathcal{A}_{EC}(I)$ may be treated as though its load is polynomial in $(n+m)$.*

We apply Lemma 4.4 to the good portion of $\mathcal{A}_{EC}(I)$. With high probability, we may take the congestion of the result to be $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$. Because of the bound we showed in Lemma 4.3 on the load of this part of $\mathcal{A}_{EC}(I)$, applying Lemma 4.4 does not (asymptotically) increase its length.

At this point, all that remains is to remove the last bit of congestion left by Lemma 4.4. We do this by *rescaling time* when processing good jobs. Recall that the good portion of $\mathcal{A}_{EC}(I)$ consists of repeatedly executing projections of $\mathcal{A}_{FC}(I)$. Because $\mathcal{A}_{FC}(I)$ was modified to allocate assignment durations as integral multiples of $f = \min_{i,j} \frac{p_j}{2v_{i,j}m}$, all machines will be processing the same set of jobs throughout an interval of this length. Thus, if we slow down time by a factor of $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$, there will be enough time in a single interval for each machine to *serially* process all jobs it was assigned. With high probability, rescaling time in this way eliminates all congestion from $\mathcal{A}_{EC}(I)$ and yields a *schedule.*

The length of the good portion of the newly "schedulized" $\mathcal{A}_{EC}(I)$ increases by a factor of $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$ but (by Lemma 4.3) is still dominated by that of the bad portion. Finally, to show our result in expectation we revert to one-at-a-time scheduling if any of the events we require fail to take place. Since this occurs with

probability $O(1/n)$, it adds only $O(\mathrm{E}[\mathrm{len}(OPT_{EC}(I))])$ to our expected makespan. We now state the section's main theorem.

Theorem 4.6 *There is an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-C.*

Unfortunately, this approximation algorithm does not extend to the case in which restarting is allowed instead of preemption. Specifically, in the good portion of $\mathcal{A}_{EC}(I)$, we depend on the the cumulative processing of the repeated (and delayed) projections of $\mathcal{A}_{FC}(I)$. Thus we are unable to restart jobs when these "sub-schedules" complete.

4.2 Algorithms for EXP-T

In this section, we extend the approximation algorithm shown in Theorem 4.6 to apply to EXP-T, where precedence constraints form a directed forest. We achieve this by using a *chain decomposition*, which allows us to use this algorithm essentially unmodified for instances of EXP-T. However, we do pay an $O(\log n)$ penalty in the resulting approximation ratio.

Given an instance $I = (J, M, G)$ of EXP-T, a chain decomposition as defined by Kumar et. al. [7] is a partition of jobs into disjoint subsets (called *blocks*) so that each block, along with the subgraph of G it induces, forms a valid instance of EXP-C. That is, the precedence constraints between the jobs in any block must form vertex-disjoint chains. These blocks also support an ordering that is consistent with G. Thus, if job j_1 precedes job j_2, and the two are placed in distinct blocks, j_1 will be in the earlier block.

Given an instance I of EXP-T and a chain decomposition of I into w blocks, we consider blocks in topological order, scheduling each one according to the algorithm given in Theorem 4.6. Clearly, this yields an $O(w\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-T. This algorithm also does not extend to the case where restarting is allowed instead of preemption, as it uses Theorem 4.6 as a subroutine. We use the following lemma shown by Kumar et. al. [7] to upper bound

the number of blocks in the chain decomposition of any directed forest. Theorem 4.8 follows immediately.

Lemma 4.7 *Every dag G whose underlying undirected graph is a forest has a (polynomial time computable) chain decomposition into $O(\log n)$ blocks.*

Theorem 4.8 *There is an $O(\log n \log(n+m) \log\log \min\{m,n\})$-approximation algorithm for EXP-T.*

4.3 Generalizing to DISTR

The same arguments made in Section 3.3 to show that EXP-I generalizes to DISTR-I apply to both EXP-C and EXP-T generalizing to DISTR-C and DISTR-T (respectively). Thus, when the processing that a job j requires is set according to an arbitrary probability distribution $\mathcal{D}_j$ with mean μ_j and median δ_j, and b_j is defined as the minimum value such that $\Pr[p_j > b_j] \leq \max\{1/n^2, 1/m^2\}$, we have the following theorem.

Theorem 4.9 *There is an $O\left(\log(n+m) \max_j \log \frac{b_j}{\mu_j}\right)$-approximation algorithm for DISTR-C. There is an, $O\left(\log n \log(n+m) \max_j \log \frac{b_j}{\mu_j}\right)$-approximation algorithm for DISTR-T.*

Chapter 5

Scheduling Under Uncertainty

This chapter presents approximation algorithms for problems in SUU, where jobs require unit processing and machines probabilistically fail. In Section 5.1, we show an equivalence between problems in SUU and those in a variant of EXP, EXP*, and then develop $O(1)$-approximation algorithms for analogous deterministic problems in FIXED*. These fit with our techniques from Chapters 3 and 4 and yield approximation algorithms for problems EXP* with ratios matching those achieved for EXP.

Following this outline, in Section 5.2 we show an $O(1)$-approximation algorithm for FIXED*-I, which yields an $O(\log\log\min\{m,n\})$-approximation algorithm for SUU-I. In Section 5.3 we show an $O(1)$-approximation algorithm for FIXED*-C. This yields both an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for SUU-C and an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for SUU-T.

5.1 Transforming SUU to EXP*

The overall goal of this chapter is to construct approximation algorithms for SUU. It will be convenient for us to rely, to the extent possible, on the techniques developed in Chapters 3 and 4. We begin by reducing from problems in SUU to problems in (a variant of) EXP. It will be instructive to review the key similarities and differences between SUU and EXP. This will help us understand intuitively why such a reduction is possible.

Recall from Section 2.1 that the problems in SUU all consider unit jobs and machines with identical speed, allowing preemption but requiring integral assignment duration, while problems in EXP have the amount of processing required by a job exponentially distributed and machine speeds unrelated. Both sets of problems share the objective of minimizing expected schedule length. Beyond these superficial differences, SUU differs in two key ways from typical machine scheduling problems in general, including EXP in particular. First, after processing a job j for a unit timestep, a machine i may fail to complete it with some independent, unrelated probability $q_{i,j}$. Second, SUU allows multiprocessing, so a job may be processed by multiple machines simultaneously.

With the above in mind we modify EXP, moving it closer to SUU. To this end we introduce two new problem constraints. First, let *int* represent the requirement that machines must process jobs for integral duration. Next, let *mp* (for multi-processing) represent a relaxation allowing multiple machines to process a single job simultaneously. Adding these constraints to EXP, we write the result as $R \mid pmtn, int, mp, p_j \sim Exp(\lambda_j) \mid E[C_{\max}]$ according to Graham's notation. Henceforth, we will refer to this class of problems as EXP*, although we note that this is a slight abuse of our notation if we consider *mp* to be a constraint on machines. We represent the analogous family of problems with fixed processing requirements by FIXED*.

Using the Principle of Deferred Decisions [14], we now argue that an instance $I = (J, \{q_{i,j}\}, G)$ of SUU is equivalent to an instance $I' = (\{\lambda_j = 1\}, \{-\ln q_{i,j}\}, G)$ of EXP*, for any precedence graph G.

Lemma 5.1 *Let Y be an arbitrary class of precedence constraints and* $I = (J, \{q_{i,j}\}, G)$ *and* $I' = (\{\lambda_j = 1\}, \{-\ln q_{i,j}\}, G)$ *be instances of SUU-Y and EXP*-Y. Then any algorithm* $\mathcal{A}_{E^*Y}$ *for EXP*-Y can be used to schedule both I and I' and furthermore,* $\mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I))] = \mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I'))]$.

PROOF. We view the processing of a job in I as a sequence of coin flips. From this perspective, when machine i processes job j for a single timestep, it flips a coin with

bias $q_{i,j}$, and completes j if it heads. Using the Principle of Deferred Decisions, for each job j and machine i, we flip these coins in advance, before the schedule for I begins executing. We then reveal them as necessary when machines process jobs. If we have revealed a set of coins S for job j, each with bias $q_{i,j}$ for some $i \in M$. Then the probability that all coins are tails (and j does not complete) is $1 - \prod_S q_{i,j}$.

We consider the act of revealing a coin with bias $q_{i,j}$ to require $-\ln q_{i,j}$ processing. Thus, if we let X_j be the amount of processing required before heads is revealed for job j, then for any set of coins revealed S, $\Pr[X_j \leq -\sum_S \ln q_{i,j}] = 1 - e^{-\sum_S \ln q_{i,j}}$. Thus, X_j is exponentially distributed with $\lambda = 1$ for all $j \in J$. Because EXP* allows multiprocessing and schedules only in integral durations, we see that I and I' are simply two different views of the same problem instance. Finally, we note that our analysis is independent of precedence constraints (because jobs become eligible in the same way in EXP* and SUU), so this result holds for arbitrary precedence graphs.

□

Given this equivalence between instances of problems in SUU and EXP*, we will show that SUU and EXP* are equivalent with regards to approximation algorithms. This allows us to construct approximation algorithms for EXP* and have them apply to SUU, and we build on this technique in later sections.

Theorem 5.2 *There is a (polynomial time) reduction from α-approximation algorithms for problems EXP* to α-approximation algorithms for problems SUU.*

PROOF. Let I and I' be two equivalent instances of SUU-Y and EXP*-Y as defined in Lemma 5.1. Take $\mathcal{A}_{E^*Y}$ to be an α-approximation for EXP*-Y. By the same lemma, we have that $\mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I))] = \mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I'))]$. This also applies to optimal algorithms, implying that $OPT_{SY}(I)$ cannot be larger than $OPT_{E^*Y}(I')$. Combining these results, we see that

$$\frac{\mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I))]}{\mathrm{E}[\mathrm{len}(OPT_{SY}(I))]} \leq \frac{\mathrm{E}[\mathrm{len}(\mathcal{A}_{E^*Y}(I'))]}{\mathrm{E}[\mathrm{len}(OPT_{E^*Y}(I'))]} \leq \alpha$$

Thus $\mathcal{A}_{E^*Y}$ is an α-approximation algorithm for SUU-Y.

□

As mentioned at the outset of this chapter, we take advantage of Theorem 5.2 by applying the techniques of Chapters 3 and 4 to EXP*. Specifically, for our earlier results to apply to EXP*, we need only replace algorithms for FIXED with ones for FIXED*. This is because our change to requiring integer assignment durations and allowing multiprocessing does not effect the parts of our algorithms that handle the stochastic nature of job processing requirements. We codify this idea in the claim below; it may be verified by inspection.

Claim 5.3 *An $O(\alpha)$-approximation algorithm for EXP-I, which operates according to the framework shown in Section 3.1, and an $O(1)$-approximation algorithm for FIXED*-I are sufficient to show an $O(\alpha)$-approximation algorithm for EXP*-I. By Theorem 5.2, this in turn yields an $O(\alpha)$-approximation algorithm for SUU-I as well. An analogous result holds for EXP-C when the $O(1)$-approximation algorithm for FIXED*-C is allowed to produce a pseudoschedule.*

5.2 Algorithms for SUU-I

In this section, we construct an $O(1)$-approximation algorithm for FIXED*-I. As defined in Section 5.1, this problem concerns jobs requiring fixed amounts of work and integral assignment durations that may be multiprocessed. By Claim 5.3 and Theorem 3.6, such a result shows an $O(\log\log\min\{m,n\})$-approximation algorithm for SUU-I.

Multiprocessing is relatively easy to accommodate, so we first present a straightforward optimal algorithm when it is allowed. This initial algorithm may generate schedules whose assignments have non-integral duration. Then as is standard, we demonstrate a rounding procedure that produces integral assignments. Finally, we prove that our rounding only increases the lengths of schedules by a constant factor and conclude that we have shown an $O(1)$-approximation algorithm for FIXED*-I.

Consider the following linear program, defined on an instance $I = (\{p_j\}, \{v_{i,j}\})$ of FIXED*-I. It is identical to (LP1), save for the removal of Equation (2.2). We delete this inequality because in this setting a job may receive processing from multiple machines at the same time, and so a feasible schedule for I need not obey it.

$$\begin{aligned} & \textbf{(LP3)} \quad \min \quad t \\ \text{s.t.} \quad & \sum_{i \in M} v_{i,j} x_{i,j} \geq p_j \quad \forall j \in J & (5.1) \\ & \sum_{j \in J} x_{i,j} \leq t \quad \forall i \in M & (5.2) \\ & x_{i,j} \geq 0 \quad \forall i \in M, j \in J\,. & (5.3) \end{aligned}$$

As in (LP1), we let $\left(x^*_{i,j}, t^*\right)$ represent an optimal solution to the linear program, with $x^*_{i,j}$ giving the duration for which machine i should process job j and t^* a bound on the length of the schedule. Equation (5.1), like Equation (2.1), ensures that every job receives sufficient processing. Equation (5.2), like Equation (2.3), guarantees that all machines have enough time to complete their assignments. Clearly, $t^* \leq OPT_{F^*I}(I)$. If we define an arbitrary order on jobs and define $\text{start}(i,j) = \sum_{j'<j} x^*_{i,j}$ as the time that machine i should start to process job j, then given integral $\left\{x^*_{i,j}\right\}$ we can easily create a schedule Σ for I as

$$\Sigma = \bigcup_{i,j} \left(i, j, \text{start}(i,j), \text{start}(i,j) + x^*_{i,j}\right)$$

Note that this schedule does nothing to prevent multiple machines from processing a job j simultaneously (as opposed to $\mathcal{A}_{FI}$ for FIXED-I, which does). We now show how to round the $\left\{x^*_{i,j}\right\}$ to integers while only increasing the makespan of the schedule by a constant factor. We begin our argument by slowing down some of the machines.

Lemma 5.4 *Let $v'_{i,j} = \min\{v_{i,j}, p_j\}$. Then we can replace Equation (5.1) with*

$$\sum_{i \in M} v'_{i,j} x_{i,j} \geq p_j \quad \forall j \in J \qquad (5.4)$$

PROOF. If machine i processes job j at all, then (if $\{x^*_{i,j}\}$ are integral) it will process it for at least one timestep, so $x^*_{i,j}$ will be at least one. In this case, when $v_{i,j} > p_j$, the job j receives more processing than it requires. We ignore this "surplus" processing by setting $v'_{i,j} = \min\{v_{i,j}, p_j\}$, without affecting the length or feasibility of the resulting schedule. □

Next, we round down all speeds to the nearest power of two, so that $v'_{i,j} = 2^k$ for some k. This is safe to do because it can only make our schedule longer. We assume that $\min_{i,j} v'_{i,j} = 2$, which can easily be achieved by scaling up both machine speeds and job processing needs. Thus k will range from 1 to $\max_j \lfloor \log p_j \rfloor$.

We consider the total processing that each group of machines sharing a rounded speed performs on a job j. For each j and integer k, we define this value to be $d^*_{j,k} = \sum_{i:\ v'_{i,j}=k} x^*_{i,j}$. Then for all $j \in J$

$$p_j \leq v'_{i,j} x^*_{i,j} \leq 2 \sum_{i \in M} \sum_{k} d^*_{j,k} 2^k \tag{5.5}$$

Now we show that a schedule of length $\lceil 2t^* \rceil$ can be generated from $\{d^*_{j,k}\}$ alone, using maximum network flow. This approach (computing a scheduling from group assignments using network flow) is also used by Lin and Rajaraman [11, Section 4], but our construction is more general, and will achieve a better approximation ratio.

Lemma 5.5 *Let $I = (J, M, G)$ be an instance of FIXED*-I. Given $(\{d^*_{j,k}\}, t^*)$, it is possible to construct a schedule Σ, such that $\text{len}(\Sigma) \leq \lceil 2t^* \rceil$, which completes all jobs in J.*

PROOF.

The network flow instance we describe is an augmented bipartite graph. On the left, we have a vertex $u_{j,k}$, for each job j and speed 2^k. On the right, we have vertex w_i for each machine i. We add an edge with infinite capacity between $u_{j,k}$ and w_i when $v'_{i,j} = 2^k$; for each pair i, j there is exactly one such k for which this is true. We refer to this edge as (j, i). [1] Later, we take the flow across edge (j, i) to represent the

[1] By adding this edge, we signify that it is possible for machine i to process job j with speed 2^k.

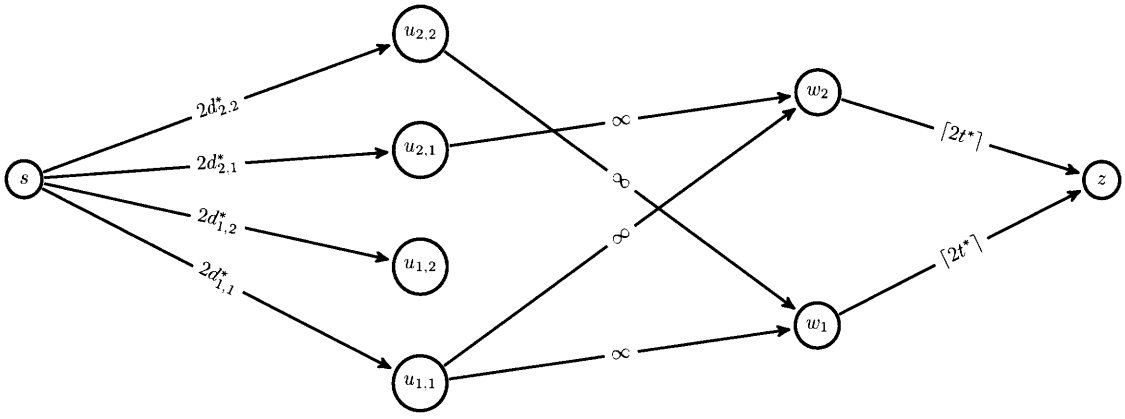

Figure 5-1: An example network flow instance from Lemma 5.5 with two jobs, two machines, and two rounded speeds. The rounded speeds here are $\{v'_{1,1} = 2, v'_{1,2} = 4, v'_{2,1} = 2, v'_{2,2} = 2\}$. There is no outgoing edge from $u_{1,2}$ because no machine processes j_1 with speed 4 (thus, $d^*_{1,2}$ should be zero).

duration with which machine i should process job j.

We augment our graph by adding a source node s and an edge $(s, u_{j,k})$ to each $u_{j,k}$. As we are interpreting flow as duration of processing, we give each of these edges capacity $2d^*_{j,k}$ (recall that we rounded speeds down to the nearest power of two, so we must double the length of assignments). This ensures that each job receives the correct amount of processing from machines with each rounded speed. We also add a sink node z and an edge (w_i, z) from each w_i to z. We set the capacity of these edges to be uniformly $\lceil 2t^* \rceil$ and thus no machine will have a load larger than this. An example is presented in Figure 5.1.

We compute a maximum flow on this graph and take the flow across edge (j, i) to represent the duration for which machine i should process job j. Thus, saturating all $(s, u_{j,k})$ edges is sufficient to yield a feasible schedule (because this gives each job $2d^*_{j,k}$ processing on machines i with $v'_{i,j} = 2^k$ as required). We prove that this is the case by showing that the minimum cut (and hence the maximum flow) is equal to $2\sum_k \sum_j d^*_{j,k}$, which forces the saturation we seek.

Consider a minimum cut of this graph where the vertices have been partitioned into two disjoint sets X_1, X_2 such that $s \in X_1$ and $w \in X_2$. If there exists any edge

(j,i) with one vertex in X_1 and one vertex in X_2, then the cut has infinite value. Thus either $X_1 = \{s\}$ or $X_2 = \{z\}$, in which case the value of the minimum cut is clearly $2\sum_k \sum_j d^*_{j,k} \leq 2t^*$. By duality, this is also the value of the maximum flow.

Finally, we construct our schedule from the set of assignments $\left\{x'_{i,j} = \text{flow}((j,i))\right\}$, as we did with the assignments given by (LP3).

□

Ford-Fulkerson's theorem [2, 5] states that an integral max flow exists whenever capacities are integral. In the network constructed above, all edges have integral capacity, save, possibly, those for edges $(s, u_{j,k})$ to which we assigned capacity $d^*_{j,k}$. [2] Thus if all $\left\{d^*_{j,k}\right\}$ are integral, then the assignments in Σ will be, too. We now show that rounding $d^*_{j,k}$ to $\lfloor 3d^*_{j,k} \rfloor$ maintains feasibility.

Lemma 5.6 *Applying Lemma 5.5 with* $\left(\left\{\lfloor 3d^*_{j,k} \rfloor\right\}, 3t^*\right)$ *yields a feasible schedule.*

PROOF. The arguments made in the proof of Lemma 5.5 imply that for the network flow instance constructed with these parameters, a maximum flow saturates all edges outgoing from the source s. This means that each job j will be processed for $\lfloor 3d^*_{j,k} \rfloor$ time by machines i with speed $v'_{i,j} = 2^k$. What we must show is that this much processing completes all jobs. More formally, we claim

$$\forall j \in J, \sum_{i \in M} \sum_k \lfloor 3d^*_{j,k} \rfloor 2^k \geq p_j$$

Consider an arbitrary job j. Because $v'_{i,j} \leq p_j$ for all i, the maximum value of k having nonzero $d^*_{j,k}$ is $\lfloor \log p_j \rfloor$. Thus we have that

$$\begin{aligned}
\sum_k \lfloor 3d^*_{j,k} \rfloor 2^k &\geq \sum_k 3d^*_{j,k} 2^k - \sum_{k \leq \log p_j} 2^k \\
&\geq 3p_j - \sum_{k=0}^{\infty} \frac{p_j}{2^k} \\
&\geq 3p_j - 2p_j = p_j
\end{aligned}$$

[2] This is why we used $\lceil 2t^* \rceil$ instead of simply $2t^*$.

Intuitively, this tells us that rounding the group assignments $\{d^*_{j,k}\}$ down to integers can cause a job j to lose at most $2p_j$ processing. Giving an assignment where each job j is processed as though it has length $3p_j$ then guarantees that j will receive p_j processing after being rounded down.

□

We can now state the section's main theorem.

Theorem 5.7 *There is a $O(\log\log\min\{m,n\})$-approximation algorithm for SUU-I.*

PROOF. For an instance I of FIXED*-I, we have described a polynomial time algorithm, which produces a feasible schedule of length $\lceil 6t^* \rceil \leq 7t^* = O(OPT_{F^*I}(I))$, and is thus an $O(1)$-approximation algorithm for FIXED*-I. The theorem then follows from Claim 5.3 and Theorem 3.6.

□

5.3 Algorithms for SUU-{C, T}

In this section we show an $O(1)$-approximation algorithm for FIXED*-C (integral duration assignments, multiprocessing, chain precedence constraints) when pseudoschedules are allowed. We follow the outline of Section 5.2, applying results from Chapter 4 to show an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for SUU-C and an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for SUU-T.

We begin by presenting a linear program for an instance $I = (\{p_j\},\{v_{i,j}\},\{C_k\})$ of FIXED*-C, which produces an optimal pseudoschedule with assignments that violate our integrality constraint. Then we perform a rounding that yields integral assignments while increasing the length of the schedule by only a constant factor. The linear program is very similar to (LP2) from Section 4.1.

(LP4) $\min t$

$$
\begin{aligned}
\text{s.t.} \quad \sum_{i \in M} x_{i,j} v_{i,j} &\geq p_j \quad \forall j \in J && (5.6)\\
x_{i,j} &\leq d_j \quad \forall j \in J && (5.7)\\
\sum_{j \in C_g} d_j &\leq t \quad \forall g \in \{1, 2, \ldots, k\} && (5.8)\\
\sum_{j \in J} x_{i,j} &\leq t \quad \forall i \in M && (5.9)\\
d_j &\geq 1 \quad \forall j \in J && (5.10)\\
x_{i,j} &\geq 0 \quad \forall i \in M, j \in J && (5.11)
\end{aligned}
$$

Equations (5.6) and (5.9) are identical to Equations (4.1) and (4.4), and ensure that sufficient processing for jobs and bounded load for machines. Equations (5.7) and (5.8) reprise Equations (4.2) and (4.3), and ensure that there is enough time to execute all assignments. Note that Equation (5.7) removes the summation over $i \in M$ from Equation (4.2), thereby allowing multiprocessing. Equation (5.10) has been added to force every job to be processed for at least one timestep.

Again we let $\left(x^*_{i,j}, d^*_j, t^*\right)$ represent an optimal solution to (LP3), and note that $t^* \leq OPT_{F^*C}(I)$. Define $\text{start}(j) = \sum_{j' \prec j} d^*_j$ to be the start of the window during which job j will be processed. Then, as in Section 5.2, we can generate a feasible pseudoschedule Σ from an integral solution to (LP4) as

$$\Sigma = \bigcup_{i,j} (i, j, \text{start}(j), \text{start}(j) + x^*_{i,j})$$

Thus, we would like to show a rounding procedure that results in a pseudoschedule with integral assignment durations and makespan $O(t^*)$. We follow the rounding shown in Section 5.2, with one modification. In Lemma 5.5, we set the edge capacity for each edge $(u_{j,k}, w_i)$ to be $\lfloor 6d^*_j \rfloor$, rather than infinite. The same arguments made in Lemma 5.6 show that this modification does not change the maximum flow or prevent jobs from receiving sufficient processing. The time to process any chain C_g increases to at most $\left\lceil \sum_{j \in C_g} 6d^*_j \right\rceil$, which is upper bounded by $\sum_{j \in C_g} 7d^*_j \leq 7t^* = O(OPT_{F^*C}(I))$. This construction yields the main result of the section.

Theorem 5.8 *There is an* $O(\log(n+m)\log\log\min\{m,n\})$*-approximation algorithm for* SUU-C*, and an* $O(\log n\log(n+m)\log\log\min\{m,n\})$*-approximation algorithm for* SUU-T*.*

PROOF. We have shown an $O(1)$-approximation algorithm for FIXED*-C when pseudoschedules are allowed. Applying Claim 5.3 and Theorem 4.6 yields the algorithm for SUU-C. Applying Lemma 4.7 to this algorithm produces the algorithm for SUU-T. □

Chapter 6

Conclusion

In previous chapters, we presented a series of polylogarithmic approximation algorithms for a diverse family of stochastic scheduling problems. We summarize our results in terms of two key contributions. First, in Chapters 3 and 4 we showed the first non-trivial approximation algorithms for the EXP (exponentially distributed job processing) family of preemptive minimum makespan scheduling problems. These included an $O(\log\log\min\{m,n\})$-approximation algorithm for EXP-I, shown in Section 3.1; an $O(\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-C, shown in Section 4.1; and an $O(\log n\log(n+m)\log\log\min\{m,n\})$-approximation algorithm for EXP-T, shown in Section 4.2. In Sections 3.3 and 4.3, we demonstrated how simple changes to our analysis allowed the algorithms above to apply to problems in DISTR (arbitrarily distributed job processing) with a small increase in approximation ratio. In Section 3.2, we gave matching approximation algorithms for EXP-I and DISTR-I when restarting is allowed instead of preemption.

Second, in Chapter 5 we extended our techniques to cover problems in SUU (where jobs require unit processing and machines probabilistically fail). In Section 5.1, we showed a reduction from problems in SUU to problems in EXP*, a variant of EXP. Then, in Sections 5.2 and 5.3, we constructed $O(1)$-approximation algorithms for problems in FIXED*, the fixed processing analog of EXP*. Coupled with analysis from Chapters 3 and 4, this allowed us to show approximation algorithms for SUU-I, SUU-C, and SUU-T with ratios matching those for EXP-I, EXP-C, and EXP-T.

There are two clear avenues for extending our work. Improved approximations algorithms with better ratios might be possible for the problems we have explored. We do not know of any lower bounds on achievable approximation ratios for these problems and, indeed, are not convinced that our bounds are tight. Also, an improvement in our algorithm for EXP-I would immediately yield improvements to our algorithms for EXP-C and EXP-T, where it is used as a subroutine. However, it does not seem likely that our current techniques will easily yield further improvements as they depend on the approximation algorithm shown for RANGE-I, where the optimal algorithms we consider have a very large advantage.

We have also not presented any approximation algorithms for precedence graphs more general than directed forests. This is the second logical place to consider further work. Unfortunately, the picture here is less rosy. Any more general stochastic scheduling result would likely also yield an approximation algorithm for the deterministic case. No such approximation algorithms are currently known, which leads us to believe that generalizing our algorithms to wider classes of precedence graphs will pose more difficulty than finding, for example, an $O(1)$-approximation algorithm for EXP-I.

Bibliography

[1] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill, second edition, 2001.

[3] Christopher Crutchfield, Zoran Dzunic, Jeremy T. Fineman, David R. Karger, and Jacob Scott. Improved approximations for multiprocessor scheduling under uncertainty. In *Proceedings of the twentieth annual ACM symposium on parallel algorithms and architectures*, 2008.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[5] L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton University Press Princeton, NJ, 1962.

[6] David R. Karger, Clifford Stein, and Joel Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[7] V.S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Scheduling on Unrelated Machines Under Tree-Like Precedence Constraints. *Proceedings of the eighth international workshop on approximation algorithms for combinatorial optimization problems*, 2005.

[8] Eugene L. Lawler and Jacques Labetoulle. On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *Journal of the ACM*, 25(4):612–619, 1978.

[9] F.T. Leighton, Bruce M. Maggs, and Satish B. Rao. Packet routing and job-shop scheduling in o(Congestion + Dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[10] Jan K. Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, 1990.

[11] Guolong Lin and Rajmohan Rajaraman. Approximation algorithms for multiprocessor scheduling under uncertainty. In *Proceedings of the nineteenth annual ACM symposium on parallel algorithms and architectures*, pages 25–34, San Diego, California, USA, 2007.

[12] Zhen Liu and Eric Sanlaville. Stochastic Scheduling with Variable Profile and Precedence Constraints. *SIAM Journal on Computing*, 26(1):173–187, 1997.

[13] Grzegorz Malewicz. Parallel scheduling of complex dags under uncertainty. In *Proceedings of the seventeenth annual ACM symposium on parallel algorithms and architectures*, pages 66–75, Las Vegas, Nevada, USA, 2008.

[14] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[15] Christos H. Papadimitriou and John N. Tsitsiklis. On Stochastic Scheduling with In-Tree Precedence Constraints. *SIAM Journal on Computing*, 16(1):1–6, 1987.

[16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall Englewood Cliffs, NJ, 2002.

[17] Paolo Serafini. Scheduling Jobs on Several Machines with the Job Splitting Property. *Operations Research*, 44(4):617–628, 1996.

[18] David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, June 1994.

[19] Gideon Weiss and Michael L. 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.

Appendix A

Proofs

This appendix presents all necessary proofs that were not shown in the main body of the thesis.

A.1 Proof of Lemma 3.5

Lemma 3.5 *Let $I = \{J, M\}$ be an instance of EXP-I. Let $\mathcal{A}_{EI}$ be a polynomial time algorithm for EXP-I that completes only jobs $j \in J$, which satisfy $p_j \leq \frac{2 \ln \min\{n,m\}}{\lambda_j}$ and obeys $\mathrm{E}[\mathrm{len}(\mathcal{A}_{EI}(I))] \leq \alpha \mathrm{E}[\mathrm{len}(OPT_{EI}(I))]$. Then there is an $(\alpha + O(1))$-approximation algorithm for EXP-I.*

PROOF.

We finish the proof of the lemma given in Section 3.1 by showing how to construct a schedule to handle jobs with length $p_j \geq \frac{2 \ln m}{\lambda_j}$, given that $m < n$. Our schedule will have expected makespan $O(\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$, and thus be sufficient to complete the proof.

We begin by considering the schedule $\Sigma_{tail} = \mathcal{A}_{FI}\left(\left(\left\{\frac{2 \ln m}{\lambda_j}\right\}, M\right)\right)$. Our scheduling algorithm will repeatedly execute Σ_{tail}. As we do this, jobs will complete and we will be able to execute Σ_{tail} in less time (by ignoring assignments to completed jobs). We can derive a tight bound on how much time will be saved from analysis by Lawler and Labetoulle [8, Section 3], who show that for FIXED-I, a set of assignments $\{x_{i,j}\}$ can be turned into a feasible schedule Σ' such that

$$\text{len}(\Sigma') = \max\left\{\max_{i\in M}\sum_{j\in J} x_{i,j}, \max_{j\in J}\sum_{i\in M} x_{i,j}\right\}$$

Recall that the projection of a schedule onto a set of jobs $J' \subseteq J$, written $\Sigma|_{J'}$, ignores assignments in Σ to jobs not in J'. Let $J_q \subseteq J$ be the set of jobs that remain uncompleted at the start of the qth repetition of Σ_{tail}, and $\{x^*_{i,j}\}$ be the set of assignment durations in Σ_{tail}. Then on its qth repetition, Σ_{tail} can be executed in length

$$\text{len}\left(\Sigma_{tail}|_{J_q}\right) = \max\left\{\max_{i\in M}\sum_{j\in J_q} x^*_{i,j}, \max_{j\in J_q}\sum_{i\in M} x^*_{i,j}\right\}$$

Let $L = \text{len}(\Sigma_{tail}) = \text{len}(\Sigma_{tail}|_J)$. Then once $\text{len}\left(\Sigma_{tail}|_{J_q}\right) \leq \frac{\ln m}{\ln n} L$, we can scale all assignments by a factor of $\frac{\ln m}{\ln n}$. This rescaled schedule then has length L, and executing it will complete all jobs j with $p_j \leq \frac{2\ln n}{\lambda_j}$. At this point, we can schedule remaining jobs with one-at-a-time scheduling, at a cost of $O(\text{E}[\text{len}(OPT_{EI}(I))])$. Thus, if we let $q_{\max}$ be the smallest q such that $\text{len}\left(\Sigma_{tail}|_{J_q}\right) \leq \frac{\ln m}{\ln n} L$, then we can bound the length of our algorithm, $\mathcal{A}_{tail}$, as

$$\text{len}(\mathcal{A}_{tail}(I)) \leq L + \sum_{k=1}^{q_{\max}} \text{len}\left(\Sigma_{tail}|_{J_q}\right)$$

Where the initial L cost comes from our execution of the rescaled schedule. Thus, we can complete the proof by showing that

$$\text{E}\left[L + \sum_{k=1}^{q_{\max}} \text{len}\left(\Sigma_{tail}|_{J_q}\right)\right] \leq O(\text{E}[\text{len}(OPT_{EI}(I))])$$

We approach this problem as follows. Let X_p be a random variable taking as its value the smallest number of repetitions such that $\text{len}\left(\Sigma_{tail}|_{J_{X_p}}\right) \leq \frac{\text{len}(\Sigma_{tail})}{2^p}$. If we let $p_{\max} = \log\ln n - \log\ln m + 1$ be the number of times $\text{len}\left(\Sigma_{tail}|_{J_q}\right)$ must halve before we can switch to one-at-a-time scheduling, then we can express the expected length

of $\mathcal{A}_{tail}$ as

$$\begin{aligned}
\mathrm{E}[\mathrm{len}(\mathcal{A}_{tail}(I))] &\leq \mathrm{E}[L] + \sum_{p=1}^{p_{\max}} \mathrm{E}\left[(X_p - X_{p-1})\frac{L}{2^{p-1}}\right] && \text{(A.1)}\\
&= \mathrm{E}[L] + \sum_{i=p}^{p_{\max}} \mathrm{E}[(X_p - X_{i-p})]\,\mathrm{E}\left[\frac{L}{2^{p-1}}\right] && \text{(A.2)}\\
&= \mathrm{E}[L] + \sum_{p=1}^{p_{\max}} \mathrm{E}[(X_p - X_{i-p})]\,\mathrm{E}\left[\frac{\mathrm{len}(OPT_{EI}(I))}{2^{p-1}}\right] && \text{(A.3)}
\end{aligned}$$

Here, Equation (A.1) holds because $\frac{L}{2^{p-1}}$ is an upper bound on the length of all repetitions of Σ_{tail} after X_{p-1}. Equation (A.2) holds because L is independent of $\{X_p\}$; that is, the length of the schedule does not effect how quickly it shrinks. Finally, because all jobs remaining at the first execution of Σ_{tail} satisfied $p_j \geq \frac{2\ln m}{\lambda_j}$, and $\mathcal{A}_{FI}$ is optimal,

$$L = \mathrm{len}(\Sigma_{tail}) = \mathrm{len}\left(\mathcal{A}_{FI}\left(\left(\left\{\frac{2\ln m}{\lambda_j}\right\}, M\right)\right)\right) \leq \mathrm{len}(OPT_{EI}(I))\,,$$

which yields Equation (A.3).We will show that for all p, $\mathrm{E}[(X_p - X_{p-1})] = O(1)$. Combining this assertion with Equation (A.3) shows that the expected length of our schedule obeys $\mathrm{E}[\mathrm{len}(\mathcal{A}_{tail}(I))] \leq O(\mathrm{E}[\mathrm{len}(OPT_{EI}(I))])$. As stated at the outset of the proof, this is sufficient to prove the lemma.

Consider an arbitrary repetition q of Σ_{tail}. Let $h_M^q = \max_{i\in M}\sum_{j\in J_q} x_{i,j}^*$ be the maximum load (across remaining jobs) of any machine, and $h_J^q = \max_{j\in J'}\sum_{i\in M} x_{i,j}^*$ be the maximum time during which any job is processed. We will show that with probability at least $\frac{1}{2}$, the larger of these values will drop by a factor of two after a single repetition of Σ_{tail}. Thus, with probability at least $\frac{1}{4}$, we will have that $\max\left\{h_J^{q+2}, h_M^{q+2}\right\} \leq \frac{1}{2}\max\left\{h_J^q, h_M^q\right\}$. Since $\mathrm{len}\left(\Sigma_{tail}\big|_{J_q}\right) = \max\left\{h_M^q, h_J^q\right\}$, this yields with probability at least 1/4, $\mathrm{len}\left(\Sigma_{tail}\big|_{J_{q+2}}\right) \leq \frac{1}{2}\mathrm{len}\left(\Sigma_{tail}\big|_{J_q}\right)$. Since we took q to be arbitrary, we can take $q = X_{p-1}$, thus showing that for all p, $\mathrm{E}[(X_p - X_{p-1})] = O(1)$. In the analysis below, we assume that $m \geq 4$. If this is not the case, we can create four "virtual" machines by slowing time by a factor of four, and assigning a different

virtual machine at every fourth instant, only increasing the above expectation by a constant factor.

Assume $h_J^q \geq h_M^q$. Call a job $j \in J_q$ *long* if $\sum_{i \in M} x_{i,j}^* \geq \frac{h_M^q}{2}$. By our bound on h_M^q, the number of bad jobs is at most $4m$. Because the exponential distribution is memoryless and each job j receives $\frac{2 \ln m}{\lambda_j}$ processing according to Σ_{tail}, the probability that a job does not complete after a single repetition is at most $\frac{1}{m^2}$. Thus, the expected number of long jobs remaining after one repetition of Σ_{tail} is at most $\frac{2}{m}$. By assumption, $m \geq 4$, so Markov's Inequality yields that the probability any long jobs remain is at most $\frac{1}{2}$. It is easy to confirm that $\min h_J^q - \frac{h_M^q}{2} \geq \frac{h_J^q}{2}$, so that eliminating all jobs halves h_J^q.

Now consider the remaining case, when $h_M^q > h_J^q$. As above, the probability that any job survives a repetition of Σ_{tail} is $\leq \frac{1}{m^2}$. Thus, in expectation, the load of each machine decreases by this factor after one repetition. By Markov's Inequality, the probability that a single machine's load does not drop by a factor of at least $\frac{1}{2}$ is at most $\frac{2}{m^2}$. Taking a union bound over all machines, the probability that *any* machine does not halve its load is at most $\frac{2}{m}$, by assumption less than $\frac{1}{2}$.

□

A.2 Proof of Lemma 4.1

Lemma 3.5 *Let $I = (\{p_j\}, M, G)$ be an instance of FIXED-C, and $I' = (\{\lambda_j\}, M, G)$ be an instance of EXP-C, such that for each job j, $p_j = \frac{\ln 2}{\lambda_j}$. Then* $\text{len}(\mathcal{A}_{FC}(I)) \leq O(\text{E}[\text{len}(OPT_{EC}(I'))])$.

PROOF. Let $I_1 = I_2 = \{J, M, \{C_k\}\}$ be two identical instances of EXP-C. For I_2, modify the probability distributions on job processing such that if a job j is assigned an amount of processing $p_j < \frac{\ln 2}{\lambda_j}$, it is increased to $p_j = \frac{\ln 2}{\lambda_j}$. We finish the proof of the lemma given in Section 4.1 by showing that when pseudoschedules are allowed, $\text{E}[\text{len}(OPT_{EC}(I_2))] \leq 2\text{E}[\text{len}(OPT_{EC}(I_1))]$.

We would like to follow the proof of Lemma 3.3, but the addition of precedence con-

straints invalidates Equation (3.2). This is because dividing jobs into subsets and processing them separately may violate these constraints. Thus, we show how to modify the proof of Lemma 3.3 to address this problem. Recall that in Lemma 3.3, jobs with $p_j > \frac{\ln 2}{\lambda_j}$ are placed in the set L, and the remainder (which pose a problem because they are too short) are placed in $\overline{L}$. $\overline{L}^*$ then represents the jobs in $\overline{L}$ with their lengths increased to $\frac{\ln 2}{\lambda_j}$. The instances $I'_L = (L, M), I'_{\overline{L}} = \left(\overline{L}, M\right)$, and $I'_{\overline{L}^*} = \left(\overline{L}^*, M\right)$ of EXP-I were then defined and scheduled separately. We modify these instances in the following straightforward way: let $I'_L = (L, M, z((C_k))), I'_{\overline{L}} = \left(\overline{L}, M, z(\{C_k\})\right)$, and $I'_{\overline{L}^*} = \left(\overline{L}^*, M, z(\{C_k\})\right)$. Here $z()$ shortcuts the jobs in each chain that are not present, as it does not make sense to consider the original chains after jobs have been partitioned.

We show now that Equation (3.2) holds for these redefined instances. Let Σ_L be an optimal pseudoschedule for I'_L and $\Sigma_{\overline{L}^*}$ for $I'_{\overline{L}^*}$. We construct a pseudoschedule Σ_J for I' as follows. For each chain C_g, consider the jobs it contains in topological order. Schedule each such job j according to $\Sigma_L|_j \cdot \Sigma_{\overline{L}^*}|_j$. As defined, Σ_J allocates sufficient processing to all jobs, and has length and load bounded by the sum of those of Σ_L and $\Sigma_{\overline{L}^*}$. Thus, Equation (3.2) holds.

The proof of the lemma thus follows from the remainder of the proof for Lemma 3.3. We note that relaxing to pseudoschedules is crucial, because it allows us to consider the the partition of jobs separately for each chains.

□

A.3 Proof of Lemma 4.2

Note that this proof is essentially verbatim from Crutchfield et. al. [3].

Lemma 4.2 *For each $j \in \{1, 2, \ldots, n\}$, let y_j be a positive integer drawn from the geometric distribution $\Pr[y_j = k] = (1/2)^k$ (where k is a positive integer), and let $b_j \geq 1$ be a weight associated with each j. Let W and η be chosen such that $W/\log\eta \geq b_j$ for all j, $W \geq \sum_j 2b_j$, and $\log\eta \leq W$. Then $\sum_j b_j y_j \leq O(cW)$ with probability at least $1 - 1/\eta^c$, for any positive constant c.*

PROOF. First, we round all the b_j up to the next power of 2, grouping them by value. We let Z_k be the set of j such that b_j obeys

$$W/(2^k \log \eta) < b_j \leq W/(2^{k-1} \log \eta)$$

for each $k \in \{1, 2, \ldots, \lceil \log(W/\log \eta) \rceil\}$. We observe that

$$\sum_j b_j y_j \leq \sum_k \frac{W}{2^k \log \eta} \sum_{j \in Z_k} y_j$$

and thus our goal is first to bound $\sum_{j \in Z_k} y_j$ near its expectation.

To bound $\sum_{j \in Z_k} y_j$, we view each y_j as the length of a sequence of fair-coin flips that terminates when flipping the first "head." Thus, their sum exceeds some value q_k (to be assigned later) when q_k fair-coin flips do not yield $|Z_k|$ heads. Let Q_k be the sum of q_k Bernoulli random variables with expectation 1/2 (i.e., Q_k is the number of "heads" in a size-q_k set of fair-coin flips). Then we have $\Pr[Q_k < |Z_k|] = \Pr\left[\sum_{j \in Z_k} y_j > q_k\right]$.

We will bound $\Pr[\exists k \text{ s.t. } Q_k < |Z_k|] \leq \eta^{-c}$ by taking a union bound over all k. Note, however, that k ranges over roughly $\log W$ values and W is not a function of a η, so we need a stronger bound than $\Pr[Q_k < |Z_k|] \leq \eta^{-c}$. Instead, we set q_k such that $\Pr[Q_k < |Z_k|] \leq \eta^{-c} 2^{-k}$. Then taking a union bound over all k gives probability at most

$$\sum_{k=1}^{\log(W/\log \eta)} \eta^{-c} 2^{-k} \leq \eta^{-c} \sum_{k=1}^{\infty} 2^{-k} \leq \eta^{-c}$$

that there exists a k such that $\sum_{j \in Z_k} y_j > q_k$. Thus if we set $q_k = \Theta(c(|Z_k| + \log \eta + k))$ and apply a Chernoff bound for $\Pr[Q_k < |Z_k|]$, we achieve the desired probability bound.

Thus, with probability at least $1 - 1/\eta^c$, we are left with

$$\sum_j b_j y_j \leq \sum_k \frac{W}{2^k \log \eta} \Theta(c(|Z_k| + \log \eta + k))$$

$$
\begin{aligned}
&= O\left(c\sum_k \frac{W\,|Z_k|}{2^k \log \eta}\right) + O\left(cW\sum_k \frac{\log\eta + k}{2^k \log \eta}\right) \\
&\leq O\left(c\sum_j b_j\right) + O\left(cW\sum_{k=1}^{\infty} \frac{1+k}{2^k}\right) \\
&= O(cW)
\end{aligned}
$$

where the third line of the derivation follows from the restriction that $W \geq \sum_j 2b_j$. □

A.4 Proof of Claim 4.5

Claim 4.5 *The pseudoschedule $\mathcal{A}_{EC}(I)$ may be treated as though its load is polynomial in $(n+m)$.*

PROOF. Lemma 4.4 applies only to pseudoschedules with bounded load. Here, we would like to apply it to the good portion of $\mathcal{A}_{EC}(I)$. Letting $(x^*_{i,j}, d^*_j, t^*)$ be an optimal solution to (LP2), we have from Lemma 4.3 that $\text{load}(\mathcal{A}_{EC}(I)) = O(t^*) = O(\text{E}[\text{len}(OPT_{EC}(I))])$. We ignore the case when this is not true, as we revert to one at a time scheduling there. Thus, if $t^* > nm$, we adopt the following procedure, originally given by Lin and Rajaraman [11].

Round down $x^*_{i,j}$ to the nearest multiple of $\frac{t^*}{nm}$. Call this value $x'_{i,j}$. We construct $\mathcal{A}_{FC}(I)$ from these revised assignments and, viewing them as integers in the range $\{0, 1, \ldots, O(nm)\}$, apply Lemma 4.4. However, each job j is now missing $x^*_{i,j} - x'_{i,j} \leq \frac{t^*}{nm}$ processing from machine i. Thus, whenever we execute $\mathcal{A}_{FC}(I)|_{\{j\}}$ while processing good jobs, we add back in $\frac{t^*}{nm}$ processing for j on each machine i. During this special additional processing, we allow only one machine to be active at a time (and thus do not impact the congestion of $\mathcal{A}_{EC}(I)$). Each time we execute $\mathcal{A}_{FC}(I)|_{\{j\}}$ (for any j) we pay a total cost of $\frac{t^*}{n}$ in length. Since every such execution completes j with probability $\frac{1}{2}$, the total cost paid by all jobs in expectation is only $2t^*$. Thus, we have successfully applied Lemma 4.4 while increasing the expected makespan of our schedule by only $O(\text{E}[\text{len}(OPT_{EC}(I))])$, which is negligible in our later analysis. □