Improved Approximations for Multiprocessor Scheduling Under Uncertainty

Christopher Y. Crutchfield*
cyc@csail.mit.edu

Zoran Dzunic
zoki@csail.mit.edu

Jeremy T. Fineman†
jfineman@csail.mit.edu

David R. Karger
karger@csail.mit.edu

Jacob H. Scott‡
jhscott@csail.mit.edu

MIT Computer Science and Artificial Intelligence Lab
32 Vassar Street
Cambridge, MA 02139

ABSTRACT

This paper presents improved approximation algorithms for the problem of ***multiprocessor scheduling under uncertainty*** (SUU), in which the execution of each job may fail probabilistically. This problem is motivated by the increasing use of distributed computing to handle large, computationally intensive tasks. In the SUU problem we are given n unit-length jobs and m machines, a directed acyclic graph G of precedence constraints among jobs, and unrelated failure probabilities q_{ij} for each job j when executed on machine i for a single timestep. Our goal is to find a schedule that minimizes the expected makespan.

Lin and Rajaraman gave the first approximations for this NP-hard problem for the special cases of independent jobs, precedence constraints forming disjoint chains, and precedence constraints forming trees. In this paper, we present asymptotically better approximation algorithms. In particular, we improve upon the previously best $O(\log n)$-approximation, giving an $O(\log\log(\min\{m,n\}))$-approximation in the case of independent jobs. We also give an $O(\log(n+m)\log\log(\min\{m,n\}))$-approximation algorithm for precedence constraints that form disjoint chains (improving on the previously best $O\left(\log(n)\log(m)\frac{\log(n+m)}{\log\log(n+m)}\right)$-approximation by a $(\log n/\log\log n)^2$ factor when $n = m^{\Theta(1)}$). Our algorithm for precedence constraints forming chains can also be used as a component for precedence constraints forming trees, yielding a similar improvement over the previously best algorithms for trees.

Our techniques include reducing SUU to a problem in ***stochastic scheduling***, where machines must process a set of jobs with randomly distributed lengths. We show that our algorithms for SUU apply to a standard problem in this setting, giving the first approximation algorithms for preemptive stochastic scheduling on unrelated machines.

*Supported in part by an NSF Graduate Research Fellowship.
†Supported in part by Google, NSF Grant CSR-AES 0615215.
‡Supported by an NDSEG Fellowship.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'08, June 14–16, 2008, Munich, Germany.
Copyright 2008 ACM 978-1-59593-973-9/08/06 ...$5.00.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms

Algorithms, Theory

Keywords

Approximation Algorithms, Multiprocessor Scheduling, Stochastic Scheduling, Scheduling Under Uncertainty

1. INTRODUCTION

Our work concerns approximation algorithms for multiprocessor scheduling under uncertainty, first introduced in [11]. This model extends the classical construction of machine scheduling to handle cases where machines run jobs for discrete timesteps and succeed in processing them only probabilistically. Our motivation stems from the increasing use of distributed computing to handle large, computationally intensive tasks. Projects like Seti@Home [1] divide computations into smaller jobs of relatively uniform length, which are then executed on unreliable machines (e.g., of volunteers).

Scheduling multiple machines to process the same job at once can help overcome the problem of unreliable machines, but too many machines processing a single job can also slow down overall throughput. The situation is exacerbated when precedence constraints among jobs are present, which is often the case for sophisticated computations; here a single job failing may delay the start of many others. Note that the special case of having no precedence constraints retains practical significance. Google's MapReduce architecture [3], for example, generates jobs whose dependencies form a complete bipartite graph, which is equivalent to two phases of independent jobs.

Motivated by these examples, this paper studies the ***multiprocessor scheduling under uncertainty*** (SUU) problem. An SUU instance is comprised of a set of n unit-time jobs and a set of m machines. For each machine i and job j, we are given a failure probability q_{ij}, which is the chance that job j does not complete when run on machine i for a single timestep. Any precedence constraints are modeled as a directed acyclic graph (dag). Our objective is to construct a schedule assigning machines to eligible jobs at each timestep, minimizing the expected time until all jobs have successfully completed. In contrast to many other scheduling problems, SUU allows multiple machines to execute the same job in a single timestep.

When SUU is properly reformulated (in Section 2), it is strikingly similar to minimum makespan problems on unrelated machines in *stochastic* scheduling, where job lengths set according to random variables. Indeed, we demonstrate that our algorithms presented in this paper can apply to this family of problems, and prove similar asymptotic approximation ratios.

Related work

Malewicz's initial presentation of SUU [11] includes a polynomial-time dynamic-programming solution for instances where both the number of machines and the width of the precedence dag are constant. If either of these constraints is relaxed, he proves that the problem becomes NP-hard. Furthermore, when both constraints are removed, there is no polynomial-time approximation algorithm for the problem achieving an approximation ratio lower than 5/4, unless $P = NP$. This work does not include approximation algorithms for the general (NP-hard) problem.

Lin and Rajaraman present the first (and, to date, only) approximation algorithms for SUU [10]. Using a greedy algorithm to maximize the chance of success across all jobs, they give an $O(\log n)$-approximation when all jobs are independent. More sophisticated techniques, including LP-rounding and random delay [9, 15], yield a variety of $O(\operatorname{poly}\log(n + m))$ approximations when precedence graphs are constrained to form only disjoint chains, collections of in- or out-trees, and directed forests. For these settings, our algorithms improve their approximation ratios by a $(\log n/ \log\log n)^2$ factor, when $n = m^{\Theta(1)}$. See Table 1 for a complete comparison.

The wider field of machine scheduling is an established and well-studied area of research with a large number of variations on its core theme (see [6] for a survey). There are three main differences between SUU and problems studied in the literature. First, in SUU, jobs may run on multiple machines in the same timestep. Second, each job has a chance of failing to complete on any machine that processes it. Third, jobs must be scheduled at unit granularity.

We are not aware of many scheduling problems in which jobs may run on multiple machines at the same time. Interestingly, Serafini, motivated by scheduling looms in the textile industry, provides a polynomial-time algorithm [14] for scheduling independent jobs on unrelated machines, given that jobs can be split arbitrarily and run independently on different machines. This problem is quite close to a deterministic analog of our model (with independent jobs), but allows arbitrarily small splits (as opposed to unit-step allocations), yielding a simpler linear-programming solution that is not applicable to our problem.

Of deterministic scheduling problems, SUU most closely resembles the problem of preemptively scheduling jobs with precedence constraints on unrelated parallel machines so as to minimize makespan. In Graham's notation, this problem is written as $R|prec, pmtn|C_{\max}$. Instead of failure probabilities q_{ij}, there is a deterministic processing time p_{ij}, denoting how long it takes for machine i to complete job j. In contrast to SUU, however, machines never fail, and jobs may only run on one machine at a time. As in [10], techniques for this problem [7, 9, 15] play an important role in our approximations. We also borrow techniques from "job-shop scheduling" [5] for our SUU algorithms for precedence constraints, but the particulars of that setting are not very similar to the ones we consider here. In the case of $R|prec, pmtn|C_{\max}$, we are only aware of approximations for precedence constraints forming disjoint chains or trees.

There is also a large body of work on stochastic scheduling (see [13, Part 2] for a representative sample). The majority of the work in this area considers how to schedule jobs whose input lengths are not known, but instead given as random variables distributed according to some probability distribution. In this problem, there are no failure probabilities. Particular attention has been paid to the case when these distributions are restricted to exponential families. We show in Section 2 that SUU is closely related to the problem of preemptively scheduling precedence-constrained jobs whose processing times are given by exponential distributions, on unrelated parallel machines so as to minimize their expected makespan. In stochastic scheduling, this problem is known as $R|pmtn, prec, p_j \sim stoch|\mathrm{E}\left[C_{\max}\right]$. We know of no previous nontrivial approximation algorithms for this problem, although an optimal algorithm exists when the machines are related and jobs are independent [18].

Our results

We give improved approximation algorithms for SUU when jobs are independent (there are no precedence constraints), and when the precedence constraints form disjoint chains. For independent jobs, we give an $O(\log\log(\min\{m, n\}))$-approximation algorithm. One component of this algorithm is based on an LP relaxation.

Our analysis for independent jobs relies on a competitive analysis as defined in [17]. Essentially, we show that our algorithm is $O(\log(p_{\max}/p_{\min}))$-competitive for a deterministic scheduling problem (similar to $R|pmtn|C_{\max}$) in which each machine has a deterministic speed, but processing times for jobs are chosen arbitrarily by an adversary, with minimum and maximum values $p_{\min}$ and $p_{\max}$. This competitive result is interesting in its own right.

When the precedence constraints on jobs form a collection of disjoint chains, we have an $O(\log(n + m)\log\log(\min\{m, n\}))$-approximation algorithm. Our disjoint-chains algorithm uses an LP relaxation similar to the one used for independent jobs. We also apply techniques from network-flow theory and prior work on the SUU problem for chains [10]. The $\log\log(\min\{m, n\})$ factor arises from the independent-jobs algorithm — therefore improving that algorithm immediately yields a better algorithm for chains.

Our algorithm for disjoint chains can be extended to yield an $O(\log(n+m)\log(n)\log\log(\min\{m, n\}))$-approximation for directed forests using the chain-decomposition techniques of [7, 10].

We also show how to apply our algorithms to similar variants in the problem of stochastic scheduling, where jobs have stochastic processing times. To the best of our knowledge, these are the first approximation algorithms for stochastic scheduling with the expected-completion-time objective and *unrelated* machines.

Paper organization

In Section 2 we give formal definitions of the SUU problem, the scheduling algorithms that we apply to it, and an equivalent formulation of SUU that plays an important role in our approximation algorithms. Section 3 presents our algorithms for independent jobs, and Section 4 shows how to extend them to handle precedence constraints forming chains. We address tree-like precedence constraints and stochastic scheduling in Sections 5 and 6, respectively.

2. PRELIMINARIES

In this section, we give a formal statement of our problem and then define what we mean by a schedule. Most of our notation is consistent with that of Malewicz [11] or Lin and Rajaraman [10]. One minor difference is that we use "failure probabilities" q_{ij} in lieu of "success probabilities" p_{ij}, for ease of notation. We then present a reformulation of the SUU problem, which we use in subsequent sections to simplify both our algorithms and the analysis involved.

Precedence Constraints	Lin and Rajaraman [10]	This work
Independent	$O(\log(n))$	$O(\log\log(\min\{m,n\}))$
Disjoint Chains	$O\left(\frac{\log(m)\log(n)\log(n+m)}{\log\log(n+m)}\right)$	$O(\log(n+m)\log\log(\min\{m,n\}))$
Directed Forests	$O\left(\frac{\log(m)\log^2(n)\log(n+m)}{\log\log(n+m)}\right)$	$O(\log(n+m)\log(n)\log\log(\min\{m,n\}))$

Table 1: Improved approximation ratios

The ***SUU*** *problem*

An instance $I = (J, M, \{q_{ij}\}, G)$ of the SUU problem includes a set J of unit-step jobs and a set M of machines. Throughout this paper, we let $n = |J|$ be the number of jobs and $m = |M|$ be the number of machines. For each machine i and job j, we are given a ***failure probability*** q_{ij}, which is the probability that job j *does not* complete when run on machine i for one unit step; these probabilities are independent. In addition, without loss of generality, we assume that for each job j, there exists a machine i such that $q_{ij} < 1$.

An SUU instance also includes a set of precedence constraints comprising a directed acyclic graph (dag) G with jobs as vertices. We say that a job j is ***eligible*** for execution at time t if all jobs preceding j (i.e, jobs having a directed path to j) in the dag have successfully completed before time t. If a job j is eligible at time t, a schedule may assign multiple machines $M_{j,t} \subseteq M$ to execute j in parallel. As all machine/job failures are independent, the probability that j *does not* complete in that timestep is $\prod_{i\in M_{j,t}} q_{ij}$.

Failure probabilities are difficult to work with because they multiply. Instead, we define the ***log failure*** of job j on machine i, denoted by ℓ_{ij}, as $\ell_{ij} = -\log q_{ij}$. Here and throughout the paper, we use log to mean a base-2 log. Note that by definition, $q_{ij} = 1/2^{\ell_{ij}}$, and hence $\prod_{i\in M_{j,t}} q_{ij} = 1/2^{\sum_{i\in M_{j,t}} \ell_{ij}}$.

Our work focuses on finding scheduling algorithms that minimize expected makespans for restricted classes of precedence constraints. If there are no precedence constraints, we say that the jobs are ***independent***, and refer to the problem as ***SUU-I*** . When the precedence constraints form a collection of disjoint chains, we call the problem ***SUU-C*** . When the constraints form a collection of disjoint trees, we call the problem ***SUU-T*** . We use sans-serif fonts to refer to problem variants, whereas serif fonts refer to algorithms/schedules for the problem.

Schedules

A ***schedule*** Σ is a policy for assigning machines to (uncompleted) jobs. Jobs must be scheduled at a unit granularity, but the schedule may assign multiple machines to the same job. A schedule may base its decisions on any of its history, but we concern ourselves with only schedules that can be computed in polynomial time. More formally, a schedule is a function $\Sigma : (H \times \mathbb{N}) \rightarrow (M \rightarrow J \cup \{\perp\})$ that, given a history[1] $h \in H$ and time $t \in \mathbb{N}$, returns a function assigning machines to jobs. We use the symbol $\perp$ to indicate that the machine remains idle. To allow for more concise schedules, the assignment function returned by $\Sigma(h, t)$ may map a machine to a job that has already completed.

[1] A full history for a deterministic schedule can be captured by the sets of remaining jobs at each timestep prior to the current timestep t. More formally, let $H_t = \{\langle S_1, S_2, \ldots, S_t\rangle | J = S_1 \supseteq S_2 \supseteq \cdots \supseteq S_t\}$ denote the set of all feasible ordered sets of remaining jobs at timesteps $1, 2, \ldots, t$. Then valid histories are given by the set $H = \bigcup_{t=1}^{\infty} H_t$. Note that compact representations of the history exist, so a polynomially computable schedule may consider the entire history.

We define an ***execution*** of Σ as follows. Suppose that $h \in H$ is the history of the execution up to time t. Then Σ assigns machine i to job $j = \Sigma(h, t)(i)$ at step t. If j has been completed when it is scheduled to run, i is assigned to $\perp$. Since J, M, $\{q_{ij}\}$, and G are invariant over a problem instance, we allow Σ to reference those implicitly.

Whenever the schedule Σ is such that it assigns machines to jobs depending only on the current time and the initial set of jobs, not the jobs that have completed (i.e., for all t, $\Sigma(h, t) = \Sigma(h', t)$ for all $h, h' \in H$), we say that the schedule is ***oblivious***. An oblivious schedule has finite length if it is only defined for $t \leq t_o$, for some t_o.

We say that a schedule is ***semioblivious*** if it can be decomposed into "rounds" such that the assignments within each round are characterized by finite oblivious schedules. Thus, while executing a step contained in a particular round, the assignment of machines to jobs depends only on the initial set of jobs when the round began and the number of steps the round has been running. Note that all oblivious schedules are naturally semioblivious, whereas the converse is not necessarily true. Our schedule for SUU-I is semioblivious.

Malewicz [11] also defines ***regimens*** as schedules having assignment of machines to jobs dependent only on the subset of jobs remaining. Although regimens appear to be the most intuitive form of schedules for the SUU problem, none of the schedules given in this paper are regimens.

We let T_Σ be a random variable denoting the length of the execution of schedule Σ, which is the number of steps before all jobs have completed. Our objective is to minimize $\mathrm{E}[T_\Sigma]$ (denoted by $\mathrm{E}[C_{\max}]$ in much of the scheduling literature). We refer to a schedule that has minimum expected makespan as Σ_{OPT}, and its expected makespan, which is finite [11], as $\mathrm{E}[T_{\mathrm{OPT}}]$. For any SUU instance, Σ_{OPT} exists, and can be computed (inefficiently) by selecting the assignment of jobs to machines on a particular timestep that minimizes the expected makespan of the remaining jobs.

In this paper, we consider schedules that are polynomial-time computable (in the variables n, m, and $\log \mathrm{E}[T_{\mathrm{OPT}}]$) and whose expected makespans approximate $\mathrm{E}[T_{\mathrm{OPT}}]$. We say that Σ is an α-approximation if $\mathrm{E}[T_\Sigma] \leq \alpha \mathrm{E}[T_{\mathrm{OPT}}]$ for all choices of probabilities $\{q_{ij}\}$.

Throughout the remainder of this paper, we use algorithm and schedule interchangeably. Moreover, we generally do not give the schedule explicitly as a function assigning machines to jobs. Instead, we describe it algorithmically.

Problem reformulation

We now describe a new, and equivalent formulation of the SUU problem, which we refer to as SUU*. Because of their equivalence, we refer to both problems as SUU later in the paper.

An SUU* instance $I = (J, M, \{q_{ij}\}, G)$ has the same structure as an SUU instance. The difference is that rather than considering the success or failure of a job as it runs on machines in each timestep, we use the Principle of Deferred Decisions [12] to view the problem as one of deterministically scheduling jobs with randomly distributed lengths.

Instead of failure probability, in SUU*, we view $\ell_{ij} = -\log q_{ij}$ as an amount of ***work*** that a machine does towards a job completion in each unit timestep. As in SUU, machines must be scheduled at a unit granularity. At the start of a schedule's execution, we draw for each job j a single random variable p_j chosen from an exponential distribution with rate parameter $\lambda_j = \ln 2$; specifically, $\Pr[p_j \le c] = 1 - 2^{-c}$. A job j completes when the total work accrued by j exceeds p_j. In other words, j completes at the first step t in which $\sum_{k=1}^{t} \sum_{i \in M_{j,k}} \ell_{ij} \ge p_j$.

Observe that a schedule is oblivious to the random values p_j. Instead, it is only aware of whether a job completes in each timestep. Thus, a schedule must make its decisions for assignments in step t based only on the surviving set of jobs at each previous timestep. Hence, the same schedule may be applied to both SUU and SUU*. In fact, a particular schedule has the same distribution of remaining jobs at each timestep in both SUU and SUU*, and hence both problem formulations are equivalent. Proof of this equivalence is given with Theorem 15 in Appendix A.

3. INDEPENDENT JOBS

This section describes an $O(\log\log n)$-approximation algorithm for SUU-I, the SUU problem with independent jobs. We first give an oblivious $O(\log n)$-approximation algorithm for SUU-I, based on scheduling an (approximation of an) integer linear program. We then modify this algorithm into a semioblivious solution consisting of $O(\log\log n)$ nearly optimal rounds.

An oblivious $O(\log n)$-approximation

We now describe an $O(\log n)$-approximation for SUU-I, which we call SUU-I-OBL. Our approach constructs a schedule of length $O(\mathrm{E}[T_{\mathrm{OPT}}])$, based on an integer linear program, such that each job has no more than a constant probability of failure upon completion. This finite oblivious schedule is repeated until all jobs have completed. Using Chernoff bounds, we conclude that the expected number of repetitions is $O(\log n)$, yielding an $O(\log n)$-approximation.

We use the following integer linear program for SUU-I-OBL. Let x_{ij} denote the number of steps during which machine i is assigned to job j. Recall $\ell_{ij} = -\log q_{ij}$ is the log failure of job j on machine i. Let L_j be a nonnegative real, representing a target work for each job. To understand the integer program, think of L_j as being fixed at $L_j = 1$ for all jobs j. We later assign $L_j = 0$ to jobs that do not need to be scheduled, and increasing values to L_j for the semioblivious $O(\log\log n)$-approximation.

$$
\begin{array}{llcll}
\textbf{(LP1)} & \min t & & & \\
\text{s.t.} & \sum_{i \in M} \ell_{ij} x_{ij} & \ge & L_j \quad \forall j \in J & (1) \\
& \sum_{j \in J} x_{ij} & \le & t \quad \forall i \in M & (2) \\
& x_{ij} & \in & \mathbb{N} \cup \{0\} \quad \forall i \in M, j \in J\,. & (3)
\end{array}
$$

Here Equation (1) enforces that every job in J has a failure probability no greater than 2^{-L_j} (or $1/2$ when $L_j = 1$), and Equation (3) guarantees that all jobs are scheduled for an integral number of steps on each machine. We use (LP1) to refer to this integer linear program generically, and $LP1(J', L)$ to refer to it with $L_j = L$ if $j \in J'$, and $L_j = 0$ otherwise. We denote the optimal value for $LP1(J', L)$ by $t_{LP1(J',L)}$.

A solution for $LP1(J', L)$ naturally generalizes to a finite oblivious schedule, denoted by $\Sigma_{LP1(J',L)}$, with length $t_{LP1(J',L)}$ as follows. Consider a machine i, and consider each job j in arbitrary order. Assign machine i to job j for x_{ij} timesteps. To finish our description of this schedule, we first claim that $t_{LP1(J,1)}$ approximates $\mathrm{E}[T_{\mathrm{OPT}}]$. Then we show how to approximate (LP1) in polynomial time.

LEMMA 1. $t_{LP1(J,1)} = O(\mathrm{E}[T_{\mathrm{OPT}}])$

PROOF. Let t be the optimum solution to $LP1(J,1)$. Consider any subset $U \subseteq J$, and its complement $\overline{U}$. Then $LP1(U,1) + LP1(\overline{U},1) \ge t$, since we can construct a solution to $LP1(J,1)$ by adding a solution to $LP1(U,1)$ and a solution to $LP1(\overline{U},1)$.

Now recall our view of the problem in terms of SUU*: there is a p_j chosen from an exponential distribution for each job j such that job j completes only if $\sum_{i \in M} \ell_{ij} x_{ij} \ge p_j$. For any sample from the event space, let U be the set of jobs j for which $p_j > 1$, and let $\overline{U}$ be the complement set of jobs j for which $p_j < 1$ (note $p_j = 1$ with probability 0 so can be ignored). By definition, each job is in U independently with probability $1/2$. Next observe that whatever Σ_{OPT} is, it must allocate at least 1 unit of work to each job in U; in other words, Equation (1) of (LP1) must hold for every $j \in U$. Thus, the optimum schedule contains a feasible solution to $LP1(U,1)$.

Now observe that by construction, U is a uniformly random subset of J, meaning all subsets are equally likely. Thus,

$$
\begin{aligned}
E[T_{\mathrm{OPT}}] &= 2^{-n} \sum_{U} E[T_{\mathrm{OPT}} \mid U] \\
&= 2^{-n} \cdot \frac{1}{2}\Big(\sum_{U} E[T_{\mathrm{OPT}} \mid U] + \sum_{U} E[T_{\mathrm{OPT}} \mid \overline{U}]\Big) \\
&\ge 2^{-n} \frac{1}{2} \sum_{U} (LP1(U,1) + LP1(\overline{U},1)) \\
&\ge 2^{-n} \frac{1}{2} \sum_{U} LP1(J,1) \\
&= \frac{1}{2} LP1(J,1)
\end{aligned}
$$

Where the second line of this derivation follows from the first paragraph of this proof. $\square$

The following lemma states that, in polynomial time, we can find an integral assignment that approximates (LP1) to within a constant factor. Some aspects of the proof are similar to [10, Theorem 4.1], which solves a slightly different problem useful for precedence constraints that form disjoint chains, but we add several steps that improve the approximation ratio. Our tighter approximation *does* apply to the more general disjoint-chains variant, which we revisit in Section 4.

LEMMA 2. *There exists a polynomial-time algorithm that computes a feasible solution to $LP1(J', L)$ having value $O(t_{LP1(J',L)})$.*

PROOF. We relax our integer linear program to a linear program, and then show that the relaxed LP can be rounded to yield an integral $\{\widehat{x_{ij}}\}$ solution with value $O(T_{LP1(J',L)})$.

First, let $\ell'_{ij} = \min\{\ell_{ij}, L\}$. Then we replace each ℓ_{ij} in Equation (1) with ℓ'_{ij}, yielding the constraint $\sum_{i \in M} \ell'_{ij} x_{ij} \ge L, \forall j \in J'$. Note that since assignments are restricted to be integral, this change has no effect on either the feasibility or the value of an assignment. Next we remove Equation (3) and solve the relaxed linear program. Letting $\{x^*_{ij}, t^*\}$ be an optimal solution, we note that $t^* \le t_{LP1(J',L)}$, because integral solutions are feasible.

Our goal now is to round the LP solution to an integral solution, while not increasing its value by very much. We proceed in three steps. First, we group machines having similar ℓ'_{ij} for a job j, yielding a single assignment for the whole group. Then, we round those assignments to integers. Finally, we show that the rounded assignments satisfy (LP1), using an integral flow network.

For each job j, we group machines having ℓ'_{ij} values within a factor of 2, and determine the total assignment to that group. More formally, for each j and integer k, we let

$$D^*_{jk} = \sum_{i:\lfloor \log \ell'_{ij} \rfloor = k} x^*_{ij}$$

be the total assignment of machines with $\ell'_{ij} \in [2^k, 2^{k+1})$ to job j. It should be clear that for all $j \in J'$,

$$\sum_{i \in M} \ell'_{ij} x^*_{ij} \geq \sum_k D^*_{jk} 2^k \geq L/2.$$

We next round the value of D^*_{jk} up to $\lfloor 6D^*_{jk} \rfloor$. We claim that

$$\forall j \in J', \quad \sum_k \lfloor 6D^*_{jk} \rfloor 2^k \geq L.$$

Observe that since $\ell'_{ij} \leq L$, the maximum value of k having nonzero D^*_{jk} is $\lfloor \log L \rfloor$. Thus, the claim follows because

$$\begin{aligned}
\sum_k \lfloor 6D^*_{jk} \rfloor 2^k &\geq 3\left(2\sum_k D^*_{jk} 2^k\right) - \left(\sum_{k \leq \log L} 2^k\right) \\
&\geq 3(L) - \left(\sum_{k=0}^{\infty} L/2^k\right) \\
&\geq 3L - 2L = L.
\end{aligned}$$

In other words, rounding the group assignments down to integers can only cause us to lose at most $2L$ work. We therefore need an assignment giving $3L$ work to the job.

To complete the integral assignment, we construct a network-flow instance as follows. For each job j and integer k, we have a node u_{jk}. For each machine i, we have a node v_i. We also add a source-node s and a sink-node w. For each u_{jk}, we add a directed edge (s, u_{jk}) with capacity $\lfloor 6D^*_{jk} \rfloor$. For each v_i, we add a directed edge (v_i, w) with capacity $\lceil 6t^* \rceil$. Finally, we add a directed edge (u_{jk}, v_i) with infinite capacity, for any j, k, i such that $\lfloor \log \ell'_{ij} \rfloor = k$. Note that for a given j and i, there is exactly one k such that (u_{jk}, v_i) exists. We refer to this edge as edge (j, i).

Note that if we make the capacity of edges (s, u_{jk}) be $6D^*_{jk}$ instead, then a flow of demand $\sum_{jk} 6D^*_{jk}$ exists in this network, and it can be constructed by setting the flow along edge (j, i) to be $6x^*_{ij}$ (and flow along edge (v_i, w) to be $6\sum_{j \in J} x^*_{ij}$). Thus, a flow of capacity $\sum_{jk} \lfloor 6D^*_{jk} \rfloor$ exists when we lower the capacity of edge (s, u_{jk}) to $\lfloor 6D^*_{jk} \rfloor$.

Ford-Fulkerson's theorem [2, 4] states that an integral max flow exists whenever the capacities are integral, as they are here. We therefore take the flow across the edges (j, i) as our integral assignments $\widehat{x_{ij}}$. Moreover, by construction, $\widehat{x_{ij}}$ satisfy

$$\begin{aligned}
&\forall i \in M, \quad \sum_{j \in J} \widehat{x_{ij}} \leq \lceil 6t^* \rceil, \\
&\forall j \in J, \quad \sum_{i \in M} \ell'_{ij} \widehat{x_{ij}} \geq \sum_k \lfloor 6D^*_{jk} \rfloor 2^k \geq L.
\end{aligned}$$

We thus have an integral feasible solution $\{\widehat{x_{ij}}, 6t^*\}$. Noting that $6t^* \leq 6T_{LP1(J',L)}$ completes the proof. □

Recall that Lemma 1 shows that $t_{LP1(J,1)} = O(\mathrm{E}[T_{\mathrm{OPT}}])$. Then Lemmas 1 and 2 in concert state that in polynomial time, we can find a schedule Σ of length $O(\mathrm{E}[T_{\mathrm{OPT}}])$, such that every job has at most a constant probability of failure. Repeating Σ until all jobs complete gives our oblivious schedule SUU-I-OBL.

THEOREM 3. *Let $T_{\text{SUU-I-OBL}}$ denote the random variable corresponding to the time it takes for an execution of SUU-I-OBL to complete all jobs. Then* $\mathrm{E}[T_{\text{SUU-I-OBL}}] = O(\mathrm{E}[T_{\mathrm{OPT}}] \log n)$.

PROOF. From Lemmas 1 and 2, we have a schedule Σ of length $O(\mathrm{E}[T_{\mathrm{OPT}}])$ that gives each job a constant probability of success. Applying a Chernoff bound gives us that a particular job completes in $O(\log n)$ repetitions of Σ, with probability at least $1 - 1/n^{\Theta(1)}$, where the constant exponent appears as a constant factor in the number of repetitions. Taking a union bound over all jobs gives that with probability at least $1 - 1/n^{\Theta(1)}$, *all* jobs complete in $O(\log n)$ repetitions. Since this probability drops off dramatically as the number of repetitions increases, we have $\mathrm{E}[T_{\text{SUU-I-OBL}}] = O(\mathrm{E}[T_{\mathrm{OPT}}] \log n)$. □

An $O(\log\log(\min\{m, n\}))$-approximation

We construct our semioblivious schedule SUU-I as follows. The schedule is divided into "rounds." The first round corresponds to an execution of the schedule suggested by the (rounded) solution to $LP1(J, 1)$. In each following round, (LP1) is applied to all remaining jobs with doubling target work. If $J_k \subseteq J_{k-1} \subseteq J$ are the set of jobs left at the start of round k, then for that round we find an approximate solution to $LP1(J_k, 2^{k-1})$, and schedule obliviously according to it.

SUU-I runs at most $K = \log\log(\min\{m, n\}) + O(1)$ of these rounds. If uncompleted jobs remain after the Kth round, one of two things is done. If $n \leq m$, SUU-I runs each job one at a time on all machines, until all jobs are completed. If $m < n$, SUU-I simply repeats the schedule $\Sigma_{LP1(J_k, 2^{k-1})}$ given by the Kth round until all jobs complete.

We will prove that SUU-I achieves an approximation factor of $O(\log\log\min\{m, n\})$. A key aspect of our analysis is viewing SUU-I as an "online algorithm" to solve the SUU* problem over the hidden input $\{p_j\}$. Essentially, SUU-I "discovers" the values of p_j as jobs complete.

In the proof of the following lemma, we compare the length of rounds $2, 3, \ldots, K$ against the makespan of an optimal offline algorithm, called OFF, that knows the values $\{p_j\}$. In particular, we show that if OFF takes total time t on input $\{p_j\}$, then each round of SUU-I takes time $O(t)$ on the same input. This part of our proof is essentially a competitive analysis [17]. In subsequent lemmas, we bound the time of SUU-I for the later rounds.

LEMMA 4. *The total expected time of rounds $2, 3, \ldots, K$ of SUU-I is $O(K \cdot E[T_{\mathrm{OPT}}])$.*

PROOF. We show that for any fixed set of random values $\{p_j\}$, each round of SUU-I takes time proportional to that of the optimal offline strategy OFF. We then combine all these rounds, taking an expectation over $\{p_j\}$, to get that rounds $2, 3, \ldots, K$ take total *expected* time $O(K \cdot \mathrm{E}[T_{\mathrm{OPT}}])$.

Consider an optimal offline strategy OFF that knows the random values of $\{p_j\}$, and let $T_{\mathrm{OFF}}(\{p_j\})$ denote OFF's makespan given the values $\{p_j\}$ (i.e., $T_{\mathrm{OFF}}(\{p_j\})$ is the minimum over all strategies for a fixed $\{p_j\}$). For each $k \in \{2, 3, \ldots, K\}$, let $J_k \subseteq J$ be the subset of jobs such that $p_j > 2^{k-2}$. Thus, OFF must assign machines to job $j \in J_k$ such that $\sum_{i \in M} x_{ij} \ell_{ij} \geq 2^{k-2}$. And hence we have $T_{\mathrm{OFF}}(\{p_j\}) \geq T_{LP1(J_k, 2^{k-2})}$.

Now consider an execution of SUU-I for the same $\{p_j\}$. We note that if a job j remains uncompleted at the start of the kth round, for $k \in \{2, 3, \ldots K\}$, then $p_j > 2^{k-2}$. This inequality follows from the fact that in the $(k-1)$th round, every job receives work that exceeds 2^{k-2}. Hence, the jobs executed in the kth round are a subset of those defined by J_k above. By Lemma 2, the kth round takes time $O(T_{LP1(J_k, 2^{k-1})})$. Observing that $T_{LP1(J_k, 2^{k-1})} \leq 2T_{LP1(J_k, 2^{k-2})}$, we conclude that SUU-I's kth round takes time $O(T_{\mathrm{OFF}}(\{p_j\}))$.

We thus have that for any particular $\{p_j\}$, rounds $2, 3, \ldots, K$ of SUU-I take total time $O(K \cdot T_{\mathrm{OFF}}(\{p_j\}))$. Since OFF is the optimal offline strategy, it follows that $T_{\mathrm{OPT}}(\{p_j\}) \geq T_{\mathrm{OFF}}(\{p_j\})$ on the same $\{p_j\}$. Thus, SUU-I takes time at most $O(k \cdot T_{\mathrm{OPT}}(\{p_j\}))$. Taking the expectation over all $\{p_j\}$, we conclude that these rounds take in expectation total time

$$O\Big(K \cdot E_{\{p_j\}}[T_{\mathrm{OPT}}(\{p_j\})]\Big) = O(K \cdot \mathrm{E}[T_{\mathrm{OPT}}]). \qquad \Box$$

In fact, Lemma 4 shows that this doubling technique yields a deterministic $O(\log(p_{\max}/p_{\min}))$-competitive algorithm for the problem of hidden but arbitrarily chosen $p_j \in [p_{\min}, p_{\max}]$. SUU-I, however, changes strategies after K rounds. We take advantage of the fact that when $p_{\max}$ is chosen from an exponential distribution, the probability that $p_{\max}$ exceeds $\Theta(\log n)$ is small.

The following two lemmas bound the expected time of the late rounds of SUU-I when $m > n$ or $n > m$, respectively.

LEMMA 5. *Suppose $m \geq n$. Then rounds $K+1, K+2, \ldots$ of SUU-I take total expected time $O(E[T_{\mathrm{OPT}}])$.*

PROOF. Recall that after the Kth round, SUU-I runs jobs one after the other. Running jobs one at a time on all machines is trivially an $O(n)$-approximation. It remains to bound the probability that SUU-I proceeds to the Kth round.

Recall also that jobs remaining after round K must have $p_j \geq 2^{K-1} \geq 2^{\log\log n + O(1)} \geq 2\log n$ (for appropriate choice of constant in $O(1)$), which occurs with probability at most $2^{-2\log n} = 1/n^2$. Applying a union bound over all j, we get probability at most $1/n$ that any job survives to round $K+1$. And hence we conclude that the expected time for rounds $K+1, K+2, \ldots$ is at most $(1/n)O(n\mathrm{E}[T_{\mathrm{OPT}}]) = O(\mathrm{E}[T_{\mathrm{OPT}}])$. $\Box$

LEMMA 6. *Suppose $n > m$. Then rounds $K+1, K+2, \ldots$ of SUU-I take total expected time $O(E[T_{\mathrm{OPT}}])$.*

PROOF. Let $\Sigma_K = \Sigma_{LP1(J_K, 2^{k-1})}$ be the schedule computed for the the Kth round. Here, SUU-I repeats Σ_K until all jobs complete. Define the load H of a finite schedule to be the maximum number of timesteps during which any machine is assigned to an uncompleted job (i.e., $H = \max_i \sum_j x_{ij}$); our schedule Σ_K can be easily "compressed" to run in exactly a number of a timesteps equal to its load. We will analyze how long it takes for the load of our instance to drop from its initial expected value $O(\mathrm{E}[T_{\mathrm{OPT}}])$ (at the end of the Kth round) to 0 (when all jobs have completed).

Let T_K be the (random variable) denoting the length of Σ_K. Let X_i be the random variable representing a number of repetitions of Σ_k necessary to drop the its load from $T_K/2^i$ to $T_K/2^{i+1}$. We define X to be the random variable denoting the number of timesteps until the compressed load of Σ_K drops below 1 (and hence reaches 0). Then we have

$$X \leq \sum_{i=0}^{\log T_K} \frac{X_i T_K}{2^i} = T_K \sum_{i=0}^{\log T_K} \frac{X_i}{2^i},$$

By construction, a single execution of Σ_K ensures that each job remains with probability at most $1/m^2$, and thus the expected load of each machine shrinks by at least a (multiplicative) factor of m^2. Markov's inequality gives us that each machines load decreases by a factor of $m/2$. with probability at least $1 - 1/(2m)$. Taking a union bound over all machines gives us that the load of *all* machines decreases by a factor of $m/2$. with probability at least $1/2$.

For $m > 4$, we now have that each repetition of Σ_K decreases the remaining load by a factor of 2 with probability at least $1/2$, and hence $\mathrm{E}[X_i] \leq 2$ for all i—requiring only an expected constant number of repetitions to reduce the load by a constant factor. We now have that

$$E\left[\sum_{i=0}^{\log T_K} \frac{X_i}{2^i}\right] \leq \sum_{i=0}^{\infty} \frac{\mathrm{E}[X_i]}{2^i} \leq O(1).$$

Since T_k and each X_i are independent, it follows that

$$\mathrm{E}[X] \leq \mathrm{E}[T_K] \sum_{i=0}^{\infty} \frac{\mathrm{E}[X_i]}{2^i} = O(\mathrm{E}[T_K]).$$

Having $\mathrm{E}[T_K] \leq O(\mathrm{E}[T_{\mathrm{OPT}}])$ (as exhibited by Lemma 4) completes the proof. $\Box$

THEOREM 7. *Let $K = \log\log(\min\{m, n\}) + O(1)$ and let the random variable $T_{\text{SUU-I}}$ be defined as the time it takes for an execution of SUU-I to complete all of the jobs. Then $\mathrm{E}[T_{\text{SUU-I}}] = O(K \cdot \mathrm{E}[T_{\mathrm{OPT}}])$.*

PROOF. We apply Lemma 1 to bound the expected length of the first round, Lemma 4 to bound the expected length of rounds $2, 3, \ldots, K$, and Lemma 5 or Lemma 6 as appropriate to bound the expected length of rounds $K+1, K+2, \ldots$. We conclude that $\mathrm{E}[T_{\text{SUU-I}}] = O(\mathrm{E}[T_{\mathrm{OPT}}] + K \cdot \mathrm{E}[T_{\mathrm{OPT}}] + \mathrm{E}[T_{\mathrm{OPT}}]) = O(K \cdot \mathrm{E}[T_{\mathrm{OPT}}])$. $\Box$

4. JOBS WITH CHAIN-LIKE PRECEDENCE CONSTRAINTS

In this section we give our $O(\log(n+m)\log\log(\min\{m, n\}))$-approximation for SUU-C, the case when precedence constraints form a collection of disjoint chains. Our algorithm may be used as a subroutine for SUU-T, the more general case where precedence constraints form disjoint trees (see Section 5).

In SUU-C, the dependency graph G is a collection disjoint chains $G = \{C_1, C_2, \ldots, C_z\}$, where each C_k gives a total order on a subset of jobs.

Our algorithm for disjoint chains is similar to Lin and Rajaraman's algorithm [10], but we achieve a better approximation ratio through various improvements. We first give an overview of the algorithm. We provide more details later in the section.

To construct our schedule, we first find assignment $\{x_{ij}\}$ of machines to jobs (where x_{ij} is an integral number of steps for which machine i is assigned to job j), giving each job one unit of work, such that the "length" and "load" of the assignment are bounded by $O(\mathrm{E}[T_{\mathrm{OPT}}])$. The ***load*** of a machine is the number of timesteps for which any job is assigned to it (i.e, $\sum_j x_{ij}$), and the load of the assignment is the maximum across all machines. The ***length*** of a chain is the sum of the length of the jobs in the chain. The length of a job j, denoted by d_j, is the maximum number of steps for which j is assigned to a single machines (i.e., $d_j = \max_i x_{ij}$). Clearly, a schedule taking time T must have a length and load no more than T.

We use an LP relaxation (similar to (LP1) in Section 3) to generate our assignment. Details appear later in the section. As in Section 3, our LP relaxation achieves an $O(1)$-approximation. Note that this assignment does not immediately yield a schedule.

As we transform our assignment into an adaptive schedule, we treat long and short jobs differently. We say that a job is ***short*** if the length of its assignment is at most some value γ, to be defined later, and the job is ***long*** otherwise. To simplify presentation, suppose for now that all jobs are short. We later describe how to deal with long jobs.

We then transform the assignment into an *adaptive* schedule Σ_k for each chain C_k. The schedule Σ_k considers the next eligible (uncompleted) job j in C_k, and (obliviously) schedules the next d_j timesteps according to the assignment $\{x_{ij}\}$. Specifically, if Σ_k begins executing job j at time t, then it schedules j from time t to $t + x_{ij}$ on machine i. (Machine i remains idle from time $t + x_{ij}$ to $t + d_j$.) After the d_j timesteps, Σ_k again considers the next eligible job in the chain (which may be the same job if it failed). We note that each time job j is obliviously scheduled, it has a constant probability of success.

We then combine all the Σ_k in a straightforward manner, yielding a "pseudoschedule" for the SUU-C instance, denoted by $\{\Sigma_k\}$. In particular, a ***pseudoschedule*** runs all Σ_k "in parallel," possibly assigning multiple jobs to the same machine in each timestep. To avoid confusion, we call each of the timesteps of a pseudoschedule a ***superstep***, and we call the number of jobs assigned to a single machine during a superstep t the ***congestion*** at that superstep, denoted by $c(t)$. We "flatten" each superstep to $c(t)$ timesteps by arbitrarily ordering the jobs assigned to each machine, thus yielding a schedule called SUU-C. If $c_{\max}$ is the maximum congestion over all supersteps, and Z is the maximum length of any chain, then SUU-C comprises $O(c_{\max}Z)$ timesteps.

To reduce congestion, we apply a random-delay technique [9, 16], also used by Lin and Rajaraman [10]. We also utilize the fact that when chains consist of sufficiently many (short) jobs, the number of supersteps spanned by Σ_k is near the expected length of Σ_k, with high probability. To deal with long jobs, we run SUU-I $O(\log(n+m))$ times, which dominates the runtime, yielding the $O(\log(n+m)\log\log(\min\{m,n\}))$-approximation.

Finding an assignment with low load and length.

As in Section 3, we use an integer linear program to optimize for the constraints. This integer linear program for chains matches that used in [10, LP1].

$$
\begin{array}{llcll}
\textbf{(LP2)} & \min t & & & \\
\text{s.t.} & \sum_{i\in M} \ell_{ij}x_{ij} & \geq & 1 \quad \forall j \in J & (4)\\
 & \sum_{j\in J} x_{ij} & \leq & t \quad \forall i \in M & (5)\\
 & \sum_{j\in C_k} d_j & \leq & t \quad \forall C_k \in G & (6)\\
 & 0 \leq x_{ij} & \leq & d_j \quad \forall i \in M, j \in J & (7)\\
 & d_j & \geq & 1 \quad \forall j \in J & (8)\\
 & x_{ij} & \in & \mathbb{N}\cup\{0\} \quad \forall i \in M, j \in J\,. & (9)
\end{array}
$$

Equations (4), (5), and (9) correspond to Equations (1), (2), and (3), respectively, in (LP1). Equation (5) bounds the load of each machine. Equation (6) bounds the length of each chain, and Equations (7) and (8) determines the length of each job.

The following lemma, proven in [10, Lemma 4.2], states that the optimal value for (LP2) is a lower bound on $\mathrm{E}[T_{\mathrm{OPT}}]$.

LEMMA 8. *Let $t_{(LP2)}$ be the optimal value for (LP2). Then $t_{(LP2)} = O(\mathrm{E}[T_{\mathrm{OPT}}])$.* □

The next lemma exhibits an $O(1)$-approximation to (LP2). Lemmas 8 and 9 together imply a polynomial-time algorithm giving an integral assignment $\{x_{ij}\}$ of machines to jobs, such that the load and length are both $O(\mathrm{E}[T_{\mathrm{OPT}}])$.

LEMMA 9. *Let $t_{(LP2)}$ be the optimal value for (LP2). There exists a polynomial-time algorithm that computes a feasible solution to (LP2) having value $O(t_{(LP2)})$.*

PROOF. The rounding proceeds as in Lemma 2, starting by removing Equation (9) and replacing Equation (4) by $\sum_{i\in M}\ell'_{ij}x_{ij} \geq 1$, for $\ell'_{ij} = \min\{\ell_{ij}, 1\}$. The only major difference is in the capacity of some edges in the flow network. Instead of giving edge (j,i) an infinite capacity, we restrict the capacity of edge (j,i) to $\lceil 6d_j^*\rceil$, where d_j^* is the assignment given by the optimal solution to the relaxed linear program. We note that the length of a chain C_k may increase up to at most $6\sum_{j\in C_k} d_j^* + |C_k| \leq 7\sum_{j\in C_k} d_j^*$. The capacity of edges (v_i, w) from machine nodes to the sink node remains $\lceil 6t^*\rceil$, so machine loads are also bounded. □

Reducing congestion of SUU-C

As described thus far, SUU-C may have $\Theta(n)$ congestion. We take advantage of a random-delay technique [9,16] to reduce congestion to $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$, with high probability. Essentially, we modify SUU-C to simply delay the start time of each chain by a value chosen uniformly at random from $\{0,1,\ldots,H\}$, where H is the load of SUU-C.

The delay technique is summed up by the following theorem, proof omitted (as similar theorems appear elsewhere). It originates in [9], and Lin and Rajaraman [10, Section 4.1] outline the necessary proof as applied to SUU-C.

THEOREM 10. *Consider a pseudoschedule $\{\Sigma_k\}$ with total load H, where H is polynomially bounded in n and m. Consider the pseudoschedule $\{\Sigma_{k'}\}$ generated by randomly shifting, or "delaying," the start time of each chain schedule Σ_k by a value chosen uniformly at random from $\{0,1,\ldots,H\}$. Then $\{\Sigma_{k'}\}$ has congestion at most $O\left(\frac{\log(n+m)}{\log\log(n+m)}\right)$, with high probability with respect to n and m.* □

Notice that whenever the load of and length of $\{\Sigma_k\}$ are bounded by $O(\mathrm{E}[T_{\mathrm{OPT}}])$, it follows that the length of $\{\Sigma_{k'}\}$ is at most $O(\mathrm{E}[T_{\mathrm{OPT}}])$ supersteps, with high probability.

Since Σ_k repeats the assignment for some jobs, the load and length of the pseudoschedules $\{\Sigma_{k'}\}$ are random variables. We note, however, that the random successes and failures of jobs (and hence load and length) are independent of the initial random delay selected. Suppose that our random execution yields a load and length of at most $O(\mathrm{E}[T_{\mathrm{OPT}}])$. Suppose also that T_{OPT} is polynomially bounded in n and m. Then we can apply Theorem 10 to conclude that each superstep has low congestion, and thus the makespan is $O(c_{\max}\mathrm{E}[T_{\mathrm{OPT}}]) = O\left(\frac{\log(n+m)}{\log\log(n+m)}\mathrm{E}[T_{\mathrm{OPT}}]\right)$ steps.

The following lemma implies that most executions of SUU-C result in low load and length. (We deal with the fact that T_{OPT} may not be polynomially bounded later in the section.) In this lemma, y_j is the random variable indicating the number of repetitions of job j's assignment used to complete j, and d_j denotes the length of job j's assignment. In the SUU-C context, $\eta = n + m$. The value W here represents the load or the length of the assignment. The lemma states that whenever a job has length (or causes load) that is logarithmically smaller than the total, then the length of the chain or (load on a machine) is close to the expectation, with high probability. Union bounding over all $O(n)$ chains (or m machines) implies that the total length (and load) of SUU-C schedule is close to the expectation, with high probability. The proof appears in Appendix B.

LEMMA 11. *For each $j \in \{1, 2, \ldots, n\}$, let y_j be a positive integer drawn from the geometric distribution $\Pr[y_j = k] = (1/2)^k$ (for k a positive integer), and let $d_j \geq 1$ be a weight associated with each j. Let W and η be chosen such that $W/\log \eta \geq d_j$ for all j, $W \geq \sum_j 2d_j$, and $\log \eta \leq W$. Then $\sum_j d_j y_j \leq O(cW)$ with probability at least $1 - 1/\eta^c$, for any positive constant c.*

We conclude that if jobs are short, where short jobs have length at most $\gamma = t_{LP2}/\log(n+m)$, and if t_{LP2} is polynomial in n and m, then SUU-C takes time $O\left(\frac{\log(n+m)}{\log\log(n+m)} \mathrm{E}[T_{\mathrm{OPT}}]\right)$ with high probability. To get this bound in expectation, we simply modify SUU-C to run the $O(n)$-approximation (as for SUU-I) whenever congestion, load, or length exceed the desired bounds, which occurs with probability at most $1/n$.

Handling long jobs

We now extend SUU-C to handle jobs having length greater than $t_{LP2}/\log(n+m)$. In the chain schedule Σ_k, we replace each longer job by a "pause" of length $t_{LP2}/\log(n+m)$. Specifically, no job from the chain is scheduled until $t_{LP2}/\log(n+m)$ supersteps later. We then divide our schedule SUU-C into $O(\log(n+m))$ segments of length $t_{LP2}/\log(n+m)$ supersteps. Note that by construction, there is at most one pause per chain per segment. After executing each segment, SUU-C executes SUU-I on the jobs corresponding to the pauses starting in that segment (suspending the rest of the chains until completion). Once those long jobs complete, SUU-I continues to the next segment.

All of our previous analyses (that assume jobs are short) still hold. In particular, we satisfy the requirement that all relevant long jobs complete before the short jobs are scheduled again. Since there are $O(\log(n+m))$ executions of SUU-I, it follows that the total expected time increases to

$$O(\log(n+m)\log\log(\min\{m,n\}) \cdot \mathrm{E}[T_{\mathrm{OPT}}]),$$

giving an $O(\log(n+m)\log\log(\min\{m,n\}))$-approximation.

Extending to nonpolynomial t_{LP2}

We now address the requirement in Theorem 10 that load and length be polynomially bounded in n and m. We make use of a trick from [16, Section 3.1], also used in [10]. Consider the chain schedule Σ_k (having length $O(t_{LP2})$, with high probability) before the random delay is applied. We round each assignment x_{ij} down to the nearest multiple of t_{LP2}/nm. We thus treat the assignments as integers in the range $\{0, 1, \ldots, O(nm)\}$. We can then apply the random-delay technique (from Theorem 10) to these rounded assignments.

The issue now is that the rounding may have decreased many assignments, so we reinsert steps into the schedule. In particular, whenever executing job j, we reinsert steps (*not* supersteps) into the execution, executing only job j during those steps. Specifically, the execution of job j may result in reinserting at most an expected $2t_{LP2}/nm$ steps for each machine, and hence $2t_{LP2}/n$ steps in total. Summing across all n jobs gives an expected $2t_{LP2}$ steps, thereby increasing the total length of SUU-C by $O(\mathrm{E}[T_{\mathrm{OPT}}])$.

THEOREM 12. *Let $T_{\text{SUU-C}}$ be the random variable denoting the time at which an execution of SUU-C completes all jobs. Then* $\mathrm{E}[T_{\text{SUU-C}}] = O(\log(n+m)\log\log(\min\{m,n\})\mathrm{E}[T_{\mathrm{OPT}}])$. □

5. JOBS WITH TREE-LIKE PRECEDENCE CONSTRAINTS

We can obtain algorithms for tree-like precedence constraints by trivially applying techniques from [7], as done in [10]. We state the bound here without proof.

When precedence constraints form a directed forest, the technique from [7] decomposes the graph into $O(\log n)$ blocks, each consisting of disjoint chains. We then apply SUU-C $O(\log n)$ times.

THEOREM 13. *If precedence constraints form a directed forest, there exists a polynomially computable schedule with expected makespan $O(\log(n)\log(n+m)\log\log(\min\{m,n\})\mathrm{E}[T_{\mathrm{OPT}}])$.*

6. STOCHASTIC SCHEDULING

This section shows how our algorithms from Sections 3 and 4 apply to the problem of preemptively scheduling, on unrelated parallel machines, jobs whose lengths are given by exponentially distributed random variables. In Graham notation, these problems are of the form $R|pmtn, prec, p_j{\sim}stoch|\mathrm{E}[C_{\max}]$, and we refer to the group as STOCH. We show an $O(\log\log n)$-approximation for the special case of STOCH-I, when all jobs are independent. Then we discuss briefly how the rest of our algorithms for SUU generalize to the STOCH context.

Preliminaries

An instance $I_{\text{stoch}} = (J, M, \{\lambda_j\}, \{v_{ij}\}, G)$ of STOCH contains a set of jobs J, a set of machines M, and a dependency graph G just as in SUU. However, the length p_j of each job j, instead of identically one, is set randomly according to the exponential distribution with rate parameter λ_j. That is, $\Pr[p_j \leq c] = 1 - e^{-c\lambda_j}$. The actual value of p_j is not revealed until the job completes. Finally, for each machine i and job j, v_{ij} specifies the speed with which machine i processes job j. Thus, if x_{ij} is the amount of time during which machine i processes job j, then j completes once $\sum_i x_{ij}v_{ij} \geq p_j$. For STOCH, x_{ij} need not be integral, but we do require that every job be processed by at most one machine at any point in time.

An $O(\log\log n)$-approximation for STOCH-I

We now show how to provide a $O(\log\log n)$-approximation for STOCH-I. Our techniques are analogous to those in Section 3, and our algorithm STC-I operates similarly to SUU-I.

STC-I runs in $K = \log\log n + O(1)$ rounds, each corresponding to an oblivious schedule Σ_k. We construct the oblivious Σ_k such that any job having $p_j \leq 2^{k-2}/\lambda_j$ completes by the end of the round. Specifically, Σ_k corresponds to solving the *deterministic* problem $R|pmtn|C_{\max}$, with the length each job j fixed as $2^{k-2}/\lambda_j$. Jobs remaining after the end of the Kth round are run one at a time on the fastest possible machine. We use the algorithm from Lawler and Labetoulle [8] to compute a schedule for $R|pmtn|C_{\max}$ in polynomial time, giving us each of our Σ_k.

The following theorem states that STC-I approximates STOCH-I. The proof sketch is similar to Theorem 3 and Lemma 1

THEOREM 14. *Let $T_{\text{STC-I}}$ be the random variable denoting the time it takes for an execution of STC-I to complete all jobs. Then* $\mathrm{E}[T_{\text{STC-I}}] = O(\log\log n \cdot \mathrm{E}[T_{\mathrm{OPT}}])$.

PROOF SKETCH. We show that the length of the first round approximates $\mathrm{E}[T_{\mathrm{OPT}}]$, using a slight modification to Lemma 1, where L_j are set according to $1/(2\lambda_j))$. Then we an use an competitive-analysis argument to prove that the $K-1$ subsequent rounds take expected time $O(\mathrm{E}[T_{\mathrm{OPT}}] \cdot K)$, along the lines of Lemma 4.

To complete the proof, we note that when $\max_j p_j$ is bounded above by $2 \log n/\lambda_j$, all jobs complete by the end of round K. Since the p_j are exponentially distributed,

$$\Pr\left[\exists j \text{ s.t. } p_j > 2 \log n/\lambda_j\right] \leq 1/n$$

Thus, running jobs sequentially is trivially an n-approximation, but we see that it occurs only with probability at most $1/n$. The expected completion time is thus dominated by the first K rounds, which take $O(\mathrm{E}\left[T_{\mathrm{OPT}}\right] \cdot K) = O(\log \log n \cdot \mathrm{E}\left[T_{\mathrm{OPT}}\right])$. □

A slight modification to this algorithm gives an $O(\log \log(n))$-approximation to the slightly weaker setting of STOCH-R-I. Here we allow *restarts* instead of preemption. In this setting, a job may be moved from one machine to another, but it loses all previous progress when this migration occurs. In contrast, recall that with preemption a job accrues work from all machines that process it. The only necessary change to Theorem 14 is setting Σ_k according to $R||C_{\max}$ instead of $R|pmtn|C_{\max}$.

Other results

We can apply the remaining algorithms from Sections 3 and 4 to STOCH, with identical approximation ratios. Thus, we have algorithms achieving an $O(\log \log(\min\{m, n\}))$-approximation for STOCH-I, an $O(\log(n+m) \log \log(\min\{m, n\}))$-approximation for STOCH-C, and an $O(\log(n) \log(n+m) \log \log(\min\{m, n\}))$-approximation for STOCH-T. Here problems have been named according to their SUU analogs. These approximation ratios also hold when we allow restarts instead of preemption.

7. CONCLUSION

In this paper, we have presented improved approximation algorithms for multiprocessor scheduling under uncertainty. We believe that our bounds are not tight. In particular, we believe that a fully adaptive schedule should be able to trim an $O(\log \log(\min\{m, n\}))$ factor from our bounds. It would also be interesting if a greedy heuristic could achieve the same bounds. Finally, we would be interested in developing nontrivial approximations for more general precedence constraints. At first glance, however, it seems like any technique for SUU and arbitrary precedence constraints may generalize to $R|pmtn, prec|C_{\max}$, which remains unsolved.

8. REFERENCES

[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] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

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

[5] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*. Ellis Horwood, 1982.

[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] E.L. Lawler and J. Labetoulle. On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *Journal of the ACM (JACM)*, 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] 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.

[11] 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, 2005.

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

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

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

[15] David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. In *Proceedings of the second annual ACM-SIAM symposium on discrete algorithms*, pages 148–157, San Francisco, California, United States, 1991.

[16] 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.

[17] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, February 1985.

[18] Gideon Weiss and Michael Pinedo. Scheduling Tasks with Exponential Service Times on Non-Identical Processors to Minimize Various Cost Functions. *Journal of Applied Probability*, 17(1):187–202, 1980.

APPENDIX

A. PROBLEM REFORMULATION

Here, we prove that SUU and SUU* are equivalent. First, we define a history of an execution. Then we prove the main theorem, which shows that SUU and SUU* have the same distribution over histories.

Consider an execution of a schedule Σ on SUU or SUU*. We define the ***history*** of the execution after $t-1$ steps by a sequence of job subsets $\langle S_1, S_2, \ldots, S_t \rangle$, with $J = S_1 \supseteq S_2 \supseteq \cdots \supseteq S_t$, where S_k is the set of jobs remaining at the start of step t.

THEOREM 15. *Consider $(t-1)$-step execution histories h_t and h_t^* of schedule Σ on SUU-instance $I = (J, M, \{q_{ij}\}, G)$ and corresponding SUU*-instance $I^* = (J, M, \{q_{ij}\}, G)$, respectively. For any particular $(t-1)$-step history x_t, we have $\Pr[h_t = x_t] = \Pr[h_t^* = x_t]$.*

PROOF. By induction on time t. Initially, $\Pr[h_1 = \langle J \rangle] = \Pr[h_1^* = \langle J \rangle] = 1$.

Suppose that both executions have the same $(t-1)$-step history x_t, and let $M_{j,t}$ be the set of machines assigned to job j at step t by the schedule Σ given the history x_t. We need only show that both SUU and SUU* have the same distribution over remaining jobs after step t.

Let S_{t+1} and S_{t+1}^* be the random variables denoting the set of remaining jobs at the start of step t, for SUU and SUU* executions, respectively. For the SUU execution, the probability that job j *does not* complete in step t is given by $\Pr[j \in S_{t+1} | x_t] = \prod_{i \in M_{j,t}} q_{ij}$.

We now consider the probability that the SUU* execution *does not* complete a remaining job j in step t. For job j, let $z_{j,k} = \sum_{i \in M_{j,k}} \ell_{ij}$ be the amount of work j receives at step k. By definition, j does not complete if $\sum_{k=1}^{t} z_{j,k} < p_j$. By assumption, since j remains at step t, we have $\sum_{k=1}^{t-1} z_{j,k} < p_j$. Moreover, since the exponential distribution of p_j is memoryless, we have

$$\Pr\left[\sum_{k=1}^{t} z_{k,j} < p_j \middle| \sum_{k=1}^{t-1} z_{k,j} < p_j\right] = \Pr\left[\sum_{i \in M_{j,t}} \ell_{ij} < p_j\right]$$

Thus, we have

$$\begin{aligned} \Pr[j \in S_{t+1}^* | x_t] &= \Pr\left[\sum_{i \in M_{j,t}} \ell_{ij} < p_j\right] \\ &= 2^{-\sum_{i \in M_{j,t}} \ell_{ij}} \\ &= \prod_{i \in M_{j,t}} q_{ij}, \end{aligned}$$

and hence $\Pr[j \in S_{t+1} | x_t] = \Pr[j \in S_{t+1}^* | x_t]$.

Since all jobs have the same distribution given the history, we conclude $\Pr[S_{t+1} = S | x_t] = \Pr[S_{t+1}^* = S | x_t]$. Applying the inductive hypothesis (i.e., $\Pr[h_t = x_t] = \Pr[h_t^* = x_t]$) completes the proof. □

B. PROOFS

LEMMA 11. *For each $j \in \{1, 2, \ldots, n\}$, let y_j be a positive integer drawn from the geometric distribution $\Pr[y_j = k] = (1/2)^k$ (for k a positive integer), and let $d_j \geq 1$ be a weight associated with each j. Let W and η be chosen such that $W/\log\eta \geq d_j$ for all j, $W \geq \sum_j 2d_j$, and $\log\eta \leq W$. Then $\sum_j d_j y_j \leq O(cW)$ with probability at least $1 - 1/\eta^c$, for any positive constant c.*

PROOF. First, we round all the d_j up to the next power of 2, grouping the d_j by value. Specifically, let Z_k be the set of j such that d_j is bounded in the following way

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

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

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

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 b_k (to be assigned later) when b_k fair-coin flips do not yield $|Z_k|$ heads. Let B_k be the sum of b_k Bernoulli random variables with expectation $1/2$ (i.e., B_k is the number of "heads" in a size-b_k set of fair-coin flips). Then we have $\Pr[B_k < |Z_k|] = \Pr\left[\sum_{j \in Z_k} y_j > b_k\right]$.

We will bound $\Pr[\exists k \text{ s.t. } B_k < |Z_k|] \leq \eta^{-c}$ 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 $\Pr[B_k < |Z_k|] \leq \eta^{-c}$. Instead, we set b_k such that $\Pr[B_k < |Z_k|] \leq \eta^{-c} 2^{-k}$. 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 $\sum_{j \in Z_k} y_j > b_k$. Thus if we set $b_k = \Theta(c(|Z_k| + \log\eta + k))$ and apply a Chernoff bound for $\Pr[B_k < |Z_k|]$, we achieve the desired probability bound.

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

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

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