Approximation Algorithms for Multiprocessor Scheduling under Uncertainty

(Regular Submission)

Guolong Lin
College of Computer and Information Science
Northeastern University, Boston MA 02115.
lingl@ccs.neu.edu

Rajmohan Rajaraman
College of Computer and Information Science
Northeastern University, Boston MA 02115.
rraj@ccs.neu.edu

Abstract

Motivated by applications in grid computing and projects management, we study multiprocessor scheduling in scenarios where there is uncertainty in the successful execution of jobs when assigned to processors. We consider the problem of *multiprocessor scheduling under uncertainty*, in which we are given n unit-time jobs and m machines, a directed acyclic graph C giving the dependencies among the jobs, and for every job j and machine i, the probability p_{ij} of the successful completion of job j when scheduled on machine i in any given particular step. The goal of the problem is to find a schedule that minimizes the expected makespan, that is, the expected completion time of all the jobs.

The problem of multiprocessor scheduling under uncertainty was introduced in [21] and was shown to be NP-hard even when all the jobs are independent. In this paper, we present polynomial-time approximation algorithms for the problem, for special cases of the dag C. We obtain an $O(\log n)$-approximation for the case of independent jobs, an $O(\log m \log n \log(n+m)/\log\log(n+m))$-approximation when C is a collection of disjoint chains, an $O(\log m \log^2 n)$-approximation when C is a collection of directed out- or in-trees, and an $O(\log m \log^2 n \log(n+m)/\log\log(n+m))$-approximation when C is a directed forest.

Keywords: Approximation Algorithms, Multiprocessor Scheduling.

1 Introduction

We study the problem of multiprocessor scheduling under uncertainty, which was introduced in [21] to study scenarios where there is uncertainty in the successful completion of a job when assigned to a server. One motivating application is in grid computing, where a large collection of computers, often geographically distributed,cooperate to solve complex computational tasks. To make better use of the distributed computers, a task is usually divided into smaller pieces (or jobs) and handed to different computers. For many applications, there could be non-trivial dependencies among these jobs. Due to the possible physical failures, or simply the distributed nature of the computing environment, a machine may not successfully execute the assigned job on time. In this scenario, a natural goal is to determine a schedule of assigning the given jobs to the computers so that the expected completion time of the task is minimized.

A similar example, also discussed in [21], arises while managing a large project in an organization. The project may be broken down into small jobs with dependencies among them, i.e., a job may be executed only after the successful completion of another set of jobs. A group of workers are assigned to this project. Due to practical reasons and different skills, a worker may not be able to finish an assigned job successfully on time. To decrease the chance of the potential delay of some key jobs, the project manager could (and would want to) assign several workers to these jobs at the same time. Based on past experiences and the workers' skill levels, the project manager can estimate the successful probability of any particular worker finishing any particular job. The challenge for the manager is to work out a strategy (or schedule) of assigning the workers to the jobs so that the *expected* completion time of the whole project is as small as possible.

Motivated by the examples above, we study the problem of *multiprocessor scheduling under uncertainty*, henceforth referred to as SUU. We have a set of m machines, a set of n unit-time jobs, and a directed acyclic graph representing precedence constraints on the order of the execution of the jobs. We are also given, for every job j and machine i, the probability p_{ij} of the successful completion of job j when scheduled on machine i in any given particular step. To compensate for this uncertainty, multiple machines can be assigned to one job at the same time. We focus on the problem of computing a schedule to minimize the expected makespan, which is the expected time to complete all the jobs.

1.1 Our results

The multiprocessor scheduling problem SUU is shown to be NP-hard in [21] even when all jobs are independent. In this paper, we present approximation algorithms for SUU, for several special classes of dependency graphs.

- We first consider the case when all the jobs are independent and present an $O(\log n)$-approximation algorithm for the problem (§3).

A crucial component of our approach to the independent jobs case is the formulation of a sub-problem in which we aim to maximize the sum of success probabilities for the jobs. A similar strategy, refined to handle job dependencies, allows us to attack the more general case where the jobs are not independent.

- When the precedence constraints on the jobs form a collection of disjoint chains, we obtain an $O(\log m \log n \frac{\log(n+m)}{\log\log(n+m)})$ approximation algorithm (§4.1). Our results rely on solving a (relaxed) linear program and rounding the fractional solution using results from network flow theory.

- Using the algorithm for disjoint chains and the chain decomposition techniques of [17], we obtain $O(\log m \log^2 n)$ and $O(\log m \log^2 n \frac{\log(n+m)}{\log\log(n+m)})$ approximations for a collection of in- or out-trees and directed forests, respectively (§4.2).

The schedules computed by the algorithms for disjoint chains, trees, and directed forests, are all oblivious in the sense that they specify in advance the assignment of machines to jobs in each time step, independent of the set of unfinished jobs at that step. Oblivious schedules are formally defined in §2, where we also present useful definitions and important properties of schedules that are used in our main results.

To the best of our knowledge, our results are the first approximation algorithms for multiprocessor scheduling under uncertainty problems. Due to space constraints, we have omitted many of the proofs; they may be found in the appendices A through C.

1.2 Related work

The problem studied in our work was first defined in the recent work by Malewicz [21], largely motivated by the application of scheduling complex dags in grid computing [9]. Malewicz characterizes the complexity of the problem in terms of the number of the machines and the *width* of the dependency graph, which is defined as the maximum number of independent jobs. He shows that when the number of machines and the width are both constants, the optimal regimen can be computed in polynomial time using dynamic programming. However, if either parameter is unbounded, the problem is NP-hard. Also, the problem can not be approximated within a factor of $5/4$ unless P=NP. Our work extends that of Malewicz by studying the approximability of the problem when neither the width of the dag nor the number of machines is bounded.

The uncertainty of the scheduling problem we study comes from the possible failure by a machine assigned to a job, as modeled by the p_{ij}'s. There have been different models of uncertainty in the scheduling literature. Most notable is the model where each task has a duration of random length and may require different amount of resources. For related work, see [7, 6, 14, 29, 16, 11].

Scheduling in general has a rich history and a vast literature. There are many variants of scheduling problems, depending on various factors. For example: Are the machines related? Is the execution preemptive? Are there precedence constraints on the execution of the jobs? Are there release dates associated with the jobs? What is the objective function: makespan, weighted completion time, weighted flow time, etc.? See [13] for a survey and [12, 20, 28, 19, 4, 17] for representative work.

Two particular variants of scheduling closely related to our work is job shop scheduling [27] and the scheduling of unrelated machines under precendence constraints. In the job shop scheduling problem, we are given m machines and n jobs, each job consisting of a sequence of operations. Each operation must be processed on a specified machine. A job is executed by processing its operations according to the associated sequence. At most one job can be scheduled on any machine at any time. The goal of the job shop scheduling problem is to find a schedule of the jobs on the machines that minimizes the maximum completion time. This problem is strongly NP-hard and widely studied [10, 18, 1]. Also extensively studied is the the problem of preemptively scheduling jobs with precedence constraints on unrelated parallel machines [19, 27, 17], the processing time of a job depends on the machine to which it is assigned. One common characteristic of this problem and SUU is that in each problem, the capability of a machine i to complete a job j may vary with both i and j. However, while the unrelated parallel machines problem models this nonuniformity using deterministic processing times that vary with i and j, in SUU the jobs are all unit-size but may fail to complete with probabilities that vary with i and j. Owing to the uncertainty in the completion of jobs, SUU schedules appear to be more difficult to specify and analyze. One other technical difference is that in SUU we allow multiple machines to be assigned to the same job at the same time, for the purpose of raising the probability of successfully completing the job. The unrelated parallel machines problem is typically solved by a reduction to instances of the job shop scheduling problem. Some of our SUU algorithms also include similar reductions.

2 Schedules, success probabilities, and mass

In this section, we present formal definitions of a schedule (§ 2.1), introduce the notion of the mass of a job and prove a key technical theorem about the accumulation of mass of a job within the expected makespan of a given schedule (§ 2.2).

2.1 Schedules

In SUU, we are given a set J of n unit-step jobs, and a set M of m machines. There are precedence constraints among the jobs, which form a directed acyclic graph (dag) C. A job j is *eligible* for execution at step t if all the jobs preceding j according to the precedence constraints have been successfully completed before t. For every job j and machine i, we are also given p_{ij}, which is the probability that job j when scheduled on a machine i will be successfully completed, *independent* of the outcome of any other execution. Multiple machines can be assigned to the same job at the same step. Without loss of generality, we assume that for each j, there exists a machine i such that $p_{ij} > 0$.

Definition 2.1. *A* **schedule** Σ *of* **length** $T \in \mathbb{Z}^+ \cup \{\infty\}$ *is a collection of functions* $\{f_{S,t} : M \to J \cup \{\perp\} \mid S \subseteq J,\ 1 \le t < T+1\}$. *An* **execution** *of the schedule* Σ *means that, at the start of each step* t*, if* S *is the set of unfinished jobs: machine* i *is assigned to job* $f_{S,t}(i)$ *if* $f_{S,t}(i)$ *is eligible and belongs to* S*; otherwise,* i *is idle for that step.*

Our formal definition of a schedule specifies assignment functions $f_{S,t}$ for infinite t. This is because there is a positive probability for a job j to be not completed yet by any given step if $\forall i, p_{ij} < 1$. For the purposes of optimizing expected makespan, however, we can restrict our attention to a restricted class of schedules.

Definition 2.2 ([21]). *A* **regimen** Σ_g *is a schedule in which* $f_{S,t_1}(\cdot) = f_{S,t_2}(\cdot)$ *for any* $S \subseteq J$ *and* $t_1 \neq t_2$*. In other words, the assignment functions* $f_{S,t}$*'s depend only on the unfinished job set* S*. Thus, we can specify* Σ_g *by a complete collection of functions* $\{f_S : M \to S \cup \{\perp\} \mid S \subseteq J\}$.

We denote the minimum expected makespan for a given SUU instance by T^{OPT}, which is finite because for any job j, there exists a machine i, such that $p_{ij} > 0$. It is not hard to see that there exists an optimal schedule which is a regimen because at any step t, one can determine an optimal assignment function, which only depends on the subset of unfinished jobs at step t and is independent of the past execution history or the value t. While a naive specification of an arbitrary regimen uses 2^n different assignment functions, certain regimens can specified succinctly, for instance, by a polynomial-length function that takes S as input and returns f_S. In this paper, we also consider a different restricted class of schedules, called *oblivious schedule*.

Definition 2.3. *An* **oblivious schedule** *is a schedule in which every assignment function* $f_{S,t}$ *is independent of* S*, i.e., for all* t, S, S'*,* $f_{S,t}(\cdot) = f_{S',t}(\cdot)$*. Hence, the assignment functions at any step* t *can be specified by a single function, which we denote by* f_t*.*

Oblivious schedules are appealing for two reasons. First, at any step t, only one assignment function is needed, regardless of the actual unfinished job set S occurring at step t. Recall that there could be many different such S at a given t because of the execution uncertainty. The second benefit is more technical: oblivious schedules allow us to address the *uncertainty* in the SUU problem by solving related *deterministic* optimization problems.

2.2 Success probabilities and mass

When a subset of machines $S \subseteq M$ is assigned to j in any time step, the probability that j is successfully completed is $1 - \prod_{i \in S}(1 - p_{ij})$. For ease of approximation, the following Proposition is useful to us.

Proposition 2.1. *Given $x_1, \cdots, x_k \in [0,1]$, $1-(1-x_1)\cdots(1-x_k) \leq x_1+\cdots+x_k$. Furthermore, if $x_1+\cdots+x_k \leq 1$, then $1-(1-x_1)\cdots(1-x_k) \geq e^{-1}(x_1+\cdots+x_k)$.* □

Proposition 2.1 suggests that we can approximate the success probability with a convenient linear form.

Definition 2.4. *For any schedule Σ, we define the* **mass** *of a job j at the end of step t to be the sum, over all time $t' \in [1,t]$ and over every machine i to which j is assigned at time t', of p_{ij}. Thus, for an arbitrary schedule, the mass of a job j at time t is a random variable. For an* oblivious schedule *Σ_o, the mass of j at the end of any step t is simply*

$$\min\{\sum_{1\leq\tau\leq t}\sum_{i:f_\tau(i)=j} p_{ij}, 1\},$$

where $f_\tau(\cdot)$ is the assignment function of Σ_o at step τ. We say that j accumulates *that mass by step t.*

The following theorem is crucial for our approach to the scheduling problem. We emphasize that it holds for an arbitrary SUU instance. It is used in the proofs of Theorem 3.1 and Lemma 4.2.

Theorem 2.2. *Let Σ be a schedule for an* SUU *instance, whose expected makespan is T. For any job j, in an execution of Σ for $2T$ steps, with probability at least $1/4$, j accumulates a mass of at least $1/4$.*

Proof. Let A be the event that j is finished within step $2T$. Let S_t be the random variable denoting the collection of machines assigned to job j at step t and $P(S_t) = \sum_{i\in S_t} p_{ij}$. Let B be the event that $\sum_{1\leq t\leq 2T} P(S_t) \leq 1/4$. What we want to prove is $\Pr(B^c) \geq 1/4$. Observe that $\Pr(A)$ equals $\Pr(A\cap B)+\Pr(A\cap B^c)$, which is at most $\Pr(A\cap B)+\Pr(B^c)$.

We estimate the value of $\Pr(A\cap B)$ below. Observe that all possible executions of Σ on the jobs form an infinite rooted tree, in which each node represents an intermediate state during an execution (see Figure 1 for an illustration). Each node has an associated set of jobs, representing the unfinished jobs at that state. For a node N, let Jobs(N) be its associated set of unfinished jobs. Note that Jobs(R) for the root node R at level 0 consists of the entire set of jobs. The nodes at level k denote the states after k steps. From each node N at level k to each node Q at level $k+1$, we can compute the corresponding transition probability according to the assignment function $f_{\text{Jobs}(N),k+1}$.

Lemma 2.3. *Consider a tree node N at level k, where $j \in$ Jobs(N). For $1 \leq t \leq k$, let S_t be the machine set assigned to j during step t along the path leading to N from R. Assume that $\sum_{1\leq t\leq k} P(S_t) \leq c$, where $c \leq 1$. And let $P(j,N)$ be the probability that j will be finished by level (step) $2T$ following a tree path through N and $\sum_{1\leq t\leq 2T} P(S_t) \leq c$. Then $P(j,N) \leq c - \sum_{1\leq t\leq k} P(S_t)$.*

Proof of Lemma: We prove the lemma by backward induction on the level number k. Consider the base case: N's level is $2T-1$. We only need to execute the schedule for one more step. Let S_{2T} be the set of machines assigned to j during step $2T$. If $P(S_{2T}) > c - \sum_{1\leq t\leq 2T-1} P(S_t)$, then $P(j,N)=0$. Otherwise, the probability that j is finished within this step is at most $P(S_{2T})$. In either case, the claim is true.

We now assume that the claim is true for any level $k \leq 2T-1$, our aim is to prove that the claim is also true for level $k-1$. Consider a tree node N at level $k-1$. Let S_k be the set of machines assigned to j during step k according to assignment function $f_{\text{Jobs}(N),k}$. A child node of N at level k either does not contain j (j is finished at step k) or contains j (j is not finished at step k). Let the probabilities of the two cases be P_1 and $1-P_1$, respectively. Denote all the children nodes where j is still unfinished as L.

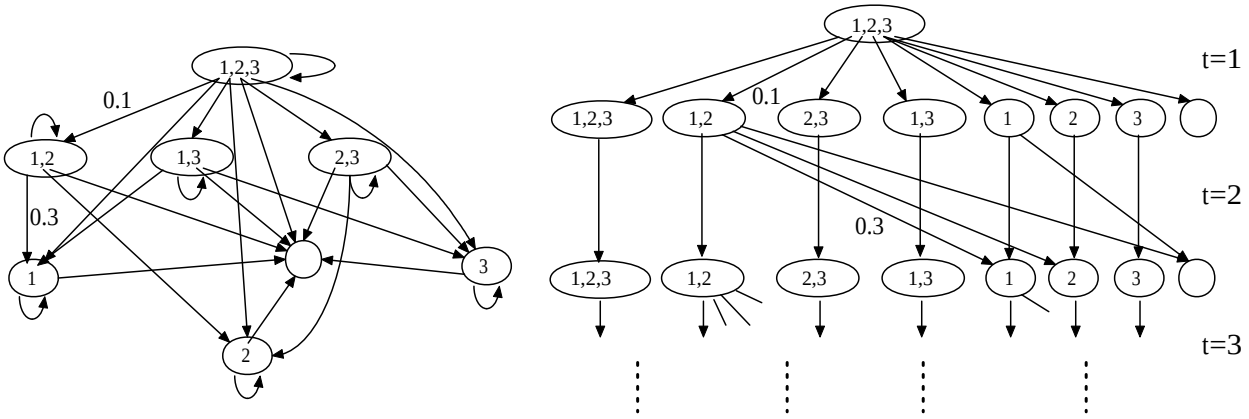

Figure 1: An illustration of the schedule. For simplicity purpose, we only use 3 jobs. Each node represents an intermediate state, with its associated set of unfinished jobs appearing inside. The number close to an edge represents its transition probability. The left graph is a Markov chain representation of a regimen. The right graph is a rooted tree representation of the execution of a schedule. To avoid cluttering, we only show the complete transitions for nodes $\{1, 2\}$ and $\{1\}$ at step 2.

If $P(S_k) > c - \sum_{1 \le t \le k-1} P(S_t)$, then $P(j, N) = 0$, which is $\le c - \sum_{1 \le t \le m-1} P(S_t)$. Otherwise,

$$\begin{aligned} P(j,N) &= P_1 + \sum_{Q \in L} P(j,Q) \\ &\le P_1 + \sum_{Q \in L} (c - \sum_{1 \le t \le k} P(S_t)) \\ &= P_1 + (1 - P_1)(c - \sum_{1 \le t \le k} P(S_t)) \\ &\le P_1 + (c - \sum_{1 \le t \le k} P(S_t)) \\ &\le c - \sum_{1 \le t \le k-1} \Pr(S_t), \end{aligned}$$

where the second inequality follows from the induction hypothesis and the last inequality follows from the fact that $P_1 \le P(S_k)$. This proves the induction step and hence the Lemma. □

By invoking the lemma with $c = 1/4$, we obtain $\Pr(A \cap B) = P(j, R) \le c = 1/4$. Hence $\Pr(A) \le 1/4 + \Pr(B^c)$. And by Markov's inequality, $\Pr(A) \ge 1/2$. We conclude that $\Pr(B^c) \ge 1/4$, completing the proof. □

3 Independent jobs

In this section, we study a special case of the scheduling problem, where the jobs are independent. We refer to this problem as SUU-I. To compute a solution to SUU-I, we first establish that there exists an oblivious schedule in which the total mass accumulated by the jobs in $O(T^{\text{OPT}})$ steps is $\Omega(n)$. To find such a schedule, we formulate a subproblem for maximizing the total sum of masses and then give polynomial-time algorithms to compute an $O(\log n)$-approximate schedule and an $O(\log^2 n)$-approximate oblivious schedule

Algorithm MSM-ALG
INPUT: Jobs J, machines M, p_{ij}'s.

- Set $f(i)$ to nil, $i \in M$.
- For each p_{ij} in nonincreasing order: If $f(i)$ is nil and $\sum_{x:f(x)=j} p_{xj} + p_{ij} \leq 1$, assign i to j, i.e., $f(i) \leftarrow j$.
- For every unused machine i, $f(i) \leftarrow \perp$; output f.

Algorithm SUU-I-ALG
INPUT: Jobs J, machines M, p_{ij}'s.

- Let S_t denote the set of unfinished jobs at the start of step t
- In each step t, schedule according to the assignment determined by MSM-ALG applied to S_t and all machines.

Figure 2: An approximation algorithm for scheduling independent jobs.

for SUU-I. For oblivious schedules, we improve the approximation factor to $O(\log n \cdot \log(\min\{n, m\}))$ when we study the more general case with chain-like precedence constraints in §4.1.

Theorem 3.1. *If there exists a schedule Σ for* SUU-I *with expected makespan T, then there exists an oblivious schedule of length $2T$, in which the total mass accumulated by all jobs is at least $n/16$.*

Proof. Consider an execution E of Σ for $2T$ steps. This execution yields naturally an oblivious schedule Σ_E of length $2T$, whose assignment functions $f_t(\cdot)$'s are defined as follows: $f_t(i) = j$ if machine i is assigned to job j at step t in E. Note that due to execution uncertainty, E, and hence Σ_E are both *random variables*. By Theorem 2.2, for any job j, with probability at least $1/4$, j accumulates a mass of at least $1/4$ by step $2T$ in Σ_E. Thus, the expected mass of j at step $2T$ in Σ_E is at least $1/16$. This implies that the expected total mass of all the jobs at step $2T$ in Σ_E is at least $n/16$. Therefore, there exists an oblivious schedule in which the total mass of the jobs at step $2T$ is at least $n/16$. □

3.1 An $O(\log n)$-approximate schedule for SUU-I

Motivated by Theorem 3.1, we formulate subproblem MaxSumMass for maximizing the sum of masses. In MaxSumMass, we are given a set J of n independent, unit-step jobs, a set M of m machines, and the probabilities p_{ij}, and the goal is to find an assignment $f : M \rightarrow J \cup \{\perp\}$ for a single step that maximizes the sum of masses over the jobs in the step. In Figure 2, we present a $1/3$-approximation algorithm MSM-ALG for MaxSumMass (which can be shown to be NP-hard), and our approximation algorithm for SUU-I, which simply executes, in every step, MSM-ALG on the unfinished jobs.

Theorem 3.2. MSM-ALG *computes a $1/3$-approximate solution to Problem* MaxSumMass. □

Theorem 3.3. *Algorithm* SUU-I-ALG *is an $O(\log n)$-approximation algorithm for* SUU-I.

Proof. Let S_t denote the set of unfinished jobs at the start of step t. Then, by Theorem 3.1, there exists an oblivious schedule of length $2T^{\text{OPT}}$ starting from step t, in which total mass of all jobs in S_t is at least $|S_t|/16$. By averaging over the $2T^{\text{OPT}}$ time steps of this schedule, there exists an assignment of jobs to machines in step t such that the total mass of the jobs in S_t in step t is at least $|S_t|/(32T^{\text{OPT}})$. By Theorem 3.2, in step t of SUU-I-ALG, the total mass of the jobs accumulated in step t is at least $|S_t|/(96T^{\text{OPT}})$. By Proposition 2.1, it follows that the expected number of jobs that complete in step t is at least $|S_t|/(96eT^{\text{OPT}})$.

We thus have a sequence of random variables S_t which satisfy the property $E[|S_{t+1}| \, |S_t] = |S_t|(1 - 1/(96eT^{\text{OPT}}))$. By straightforward Chernoff bound arguments [3, 15], we obtain that with high probability, S_t is empty within $O(T^{\text{OPT}} \log n)$ steps. □

The schedule computed by SUU-I-ALG is adaptive in the sense that the assignment function for each step is dependent on the set of unfinished jobs at the start of the step. Using an extension of MSM-ALG,

we also give a polynomial-time combinatorial algorithm to compute an *oblivious* schedule with expected makespan within an $O(\log^2 n)$ of the optimal. Due to space constraints, we refer the reader to the appendix for details. In §4.1, we improve this bound further to $O(\log n \cdot \log(\min\{n, m\}))$ using an LP-based algorithm.

4 Jobs with precedence constraints

In this section, we study SUU when there are non-trivial precedence constraints on the jobs. We first present in §4.1 a polylogarithmic approximation algorithm for the case when the constraints form disjoint *chains*, and then extend the results in §4.2 to the more general case when the constraints form directed forests. All of the schedules we compute are oblivious.

4.1 Disjoint chains

We consider SUU in the special case where the dependency graph C for the jobs is a collection of disjoint chains $C = \{C_1, \cdots, C_l\}$. We refer to this problem as SUU-C. If job j_1 precedes j_2 according to the constraints, we write $j_1 \prec j_2$.

At a high level, our approach to solve SUU-C is to first compute an oblivious schedule of near-optimal length in which every job has a constant probability of successful completion, then *replicate* this schedule sufficiently many times to conclude that all the jobs are finished with high probability within a desired makespan bound. We first consider the problem of accumulating a constant success probability for each job. As in the independent jobs case, we will use the notion of mass instead of the actual probability. However, we need to take into account the dependencies among the jobs. Therefore, we formulate the following problem AccuMass-C: Given the input for SUU-C, compute an oblivious schedule with minimum length T, subject to two conditions: (i) Every job j accumulates a mass of at least $1/2$ within T; (ii) If $j_1 \prec j_2$, j_1 must already accumulate mass $1/2$ before any machine can be assigned to j_2. Condition (ii) captures the intuition that if j_1 has a low probability of successful completion before step t, then the probability that j_2 is eligible for execution at step t would be small; so it does not make much sense to assign machines to j_2 prior to t in the oblivious schedule.

The following is a relaxed linear program (LP1) for AccuMass-C. Let x_{ij} denote the number of steps during which machine i are assigned to j. Let d_j be the number of steps during which there is some machine assigned to j.

$$
\begin{aligned}
\textbf{(LP1)} \qquad & \min \quad t & & & \\
\text{s.t.} \quad & \sum_{i \in M} p_{ij} x_{ij} & \geq & 1/2 \quad \forall j \in J & (1)\\
& \sum_{j \in J} x_{ij} & \leq & t \quad \forall i \in M & (2)\\
& \sum_{j \in C_k} d_j & \leq & t \quad C_k \in C & (3)\\
& 0 \leq x_{ij} & \leq & d_j \quad \forall i, j & (4)\\
& d_j & \geq & 1 \quad \forall j & (5)
\end{aligned}
$$

Some comments on (LP1) are in order. Equation 1 enforces Condition (i). Equation 2 bounds the *load* on every machine, which we define below. Equation 3 bounds the time length on each chain constraint. Finally Equation 4 ensures that each job accumulates its mass during the d_j steps when there is some machine assigned to it. Let T^* be the optimal value for (LP1) above.

Note that in (LP1) we do not have any condition to prevent two different jobs from two precedence chains to be scheduled on the same machine at the same step. We use the term *pseudo-schedule* to capture such "schedules", in which different jobs from different precedence chains may be scheduled to the same machine simultaneously.

Definition 4.1. *A* **pseudo-schedule** *of* **length** $T \in \mathbb{Z}^+ \cup \infty$ *is a collection of assignment functions,* $\{f_t : M \to 2^J \mid 1 \le t < T+1\}$.

Hence, an assignment function of a pseudo-schedule may map a machine to a set of jobs. In this sense, a pseudo-schedule may not be feasible; we address this issue later when describe how to transform a pseudo-schedule to an appropriate oblivious schedule. An oblivious schedule is a pseudo-schedule in which the value of f_t is a single element.

Definition 4.2. *Given a pseudo-schedule* Σ_g *of (finite) length* T, $\{f_t : M \to 2^J \mid 1 \le t < T+1\}$, *the* **load of a machine** i *is defined as the total number of times that a job is scheduled on* i *in* Σ_g. *Formally, the load of machine* i *is* $\sum_{1 \le t < T+1} |f_t(i)|$. *The* **load** *of* Σ_g *is defined as the maximum load of any machine.*

We remark that a pseudo-schedule of length T may have a load greater than T.

Theorem 4.1. *Within polynomial time one can round an optimal feasible solution to (LP1), and obtain a pseudo-schedule for Problem* AccuMass-C *whose length and load are both* $O(\log m)T^*$.

Proof. Obviously (LP1) is feasible because one can assign machines to each job for a finite steps so that the job can accumulate a mass of $1/2$. Let $\{x_{ij}, d_j, t\}$ be one *optimal* solution to (LP1). (Note that t is equal to T^*.) Our efforts mainly concern the rounding procedure, i.e., obtaining a feasible *integral* solution from the fractional solution without blowing up t too much. We then describe how to get a pseudo-schedule from an integral solution to (LP1). We differentiate between two cases.

The first case is when $t \ge |J| = n$. We round each x_{ij} and d_j up by setting $x^*_{ij} = \lceil x_{ij} \rceil$ and $d^*_j = \lceil d_j \rceil$. We obtain a feasible integral solution with approximation factor 2 since we have

$$\sum_{i \in M} p_{ij} x^*_{ij} \ge 1/2 \quad \forall j \in J, \qquad \sum_{j \in J} x^*_{ij} \le t + n \le 2t \quad \forall i \in M,$$
$$\sum_{j \in C_k} d^*_j \le t + n \le 2t \quad C_k \in C, \qquad x^*_{ij} \le d^*_j \quad \forall i, j.$$

The second case is when $t < |J| = n$. We make use of some results from network flow theory for our rounding in this case. Notice that although we target for a mass of $1/2$, any constant smaller than $1/2$ will do as well because we can always scale every variable up to reach that target, sacrificing only a constant factor. In our presentation below, we use many such scale-up operations. (We haven't tried to optimize the constants.) For a given job j, if $\sum_{i \in M, x_{ij} \ge 1} p_{ij} x_{ij} \ge 1/4$, we can round these x_{ij}'s to the next larger integer. Since $\lceil x_{ij} \rceil \le 2x_{ij}$, this only incurs a factor of 2 blow up in t. Thus, we only need to consider those jobs j such that $\sum_{i \in M, x_{ij} \ge 1} p_{ij} x_{ij} \le 1/4$, which implies that $\sum_{i \in M, x_{ij} < 1} p_{ij} x_{ij} \ge 1/4$. Observe that $\sum_{i \in M, p_{ij} < \frac{1}{8m}, x_{ij} < 1} p_{ij} x_{ij} < 1/8$, which implies $\sum_{i \in M, p_{ij} \ge \frac{1}{8m}, x_{ij} < 1} p_{ij} x_{ij} \ge 1/8$.

We bucket these p_{ij}'s into at most $B = \lceil \log(8m) \rceil$ intervals $(2^{-(k+1)}, 2^{-k}]$ $(k = 0, 1, \ldots)$. For a bucket $b : (2^{-(b+1)}, 2^{-b}]$, if $\sum_{p_{ij} \in \text{bucket } b} x_{ij} < 1/32$, we remove this bucket from further consideration. Note that the sum of $p_{ij} x_{ij}$ over all removed buckets is at most $1/16$. Hence for the p_{ij}'s in the remaining buckets, we still have $\sum_{i \in M, p_{ij} \ge \frac{1}{8m}, x_{ij} < 1} p_{ij} x_{ij} \ge 1/16$.

For each job j, there is a bucket $b_j : (2^{-(b_j+1)}, 2^{-b_j}]$ such that $\sum_{p_{ij} \in \text{bucket } b_j} x_{ij} \ge \frac{2^{b_j}}{16B}$. Denote the sum on the left side of the above inequality by D_j. If necessary, we scale all the x_{ij}'s (and other variables) up by a factor of 32, so that all $D_j \ge 1$. We then round D_j down to $\lfloor D_j \rfloor$. These operations only cost us a

constant factor in terms of approximation. Thus for the ease of the presentation below, we assume that the D_j's are integral and let $D = \sum_{j \in J} D_j$.

We now construct a *network-flow* instance as follows (see Figure 3). We have one node for each job j, one node for each machine i, a source node u, and a destination node v. We add an edge (i, j) for each x_{ij} contributing to the computation of D_j's. We orient the edge (i, j) from j to i, with edge capacity $\lceil d_j \rceil$. From each machine node i, add an edge toward v, with capacity $\lceil 2t \rceil$. For each job node j, add an edge from u to j, with capacity D_j.

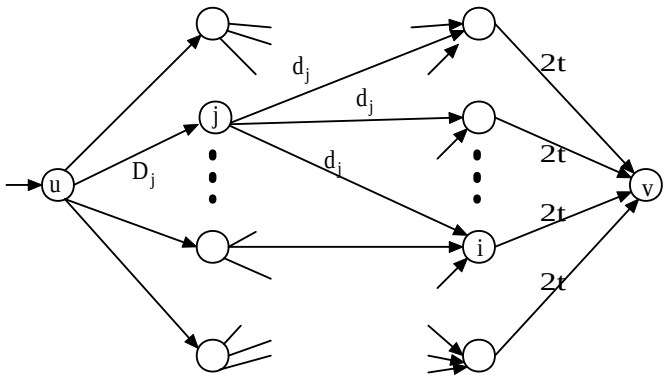

Figure 3: A network flow instance for the rounding of an optimal solution to (LP1)

The argument before the construction shows that a flow of demand D at u can be pushed through the network, where the x_{ij}'s specify such a feasible flow. D is actually the maximum flow of the network (consider the cut where one side consists of u alone). From Ford-Fulkerson's theorem [8, 5], we know that there exists an *integral* feasible flow when the parameters are integral, as in our instance. We take such an integral flow value on edge (j, i) as our rounded solution x^*_{ij}. Furthermore, the integral solution obtained observes the following identities.

$$\sum_{i \in M} p_{ij} x^*_{ij} \geq \frac{1}{16\lceil \log(8m) \rceil} \quad \forall j \in J, \qquad \sum_{j \in J} x^*_{ij} \leq \lceil 2t \rceil \quad \forall i \in M$$

$$\sum_{j \in C_k} \lceil d_j \rceil \leq \lceil 2t \rceil \quad C_k \in C, \qquad x^*_{ij} \leq \lceil d_j \rceil \quad \forall i, j.$$

Raising all the values by a factor of $O(\log m)$, we obtain an integral feasible solution $\{\hat{x}_{ij}, \hat{d}_j, \hat{t}\}$, where $\hat{t} = O(\log m) T^*$.

We now describe how to construct from the integral solution a pseudo-schedule Σ_s whose length and load are both bounded by $\hat{t} = O(\log m) T^*$. Consider a job j in a chain $C_k \in C$. Given the $\hat{x}_{ij}$'s, let $L_j = \max_i \hat{x}_{ij}$. Let $\psi_j = \sum_{j_0 : j_0 \prec j} L_{j_0}$. We assign the machines to j within a step interval of length L_j from step $\psi_j + 1$ to $\psi_j + L_j$, using each machine i $\hat{x}_{ij}$ times. In other words, the assignment functions for chain C_k are specified as follows. For any job j and machine i, if $\hat{x}_{ij} > 0$, $f^k_t(i) = \{j\}$ for $t \in [\psi_j + 1, \psi_j + \hat{x}_{ij}]$. This can be done because each machine is assigned to j at most L_j times and different machines can be assigned to j at the same step. After we define the $f^k_t(\cdot)$ for every chain $C_k \in C$, we define the assignment functions for Σ_s as

$$f_t(i) = \cup_{k : C_k \in C} f^k_t(i) \quad \text{for } i \in M, t \in [1, \hat{t}].$$

Recall that the range of the assignment functions for a pseudo-schedule is a set of jobs. This completes the proof of the theorem. □

We now relate AccuMass-C to SUU-C. Recall that T^* is the optimal value of (LP1) we write for Problem AccuMass-C, and T^{OPT} is the expected makespan of an optimum schedule Σ for Problem SUU-C.

We now bound the value T^* in terms of T^{OPT} in Lemma 4.2. This lemma, together with Theorem 4.1 immediately yields a pseudo-schedule that solves AccuMass-C with load and length within $O(\log n)$ factor of T^{OPT}.

Lemma 4.2. $T^* \leq 16T^{OPT}$. □

Theorem 4.3. *A pseudo-schedule with length and load bounded by $O(\log m) \cdot T^{OPT}$ can be computed within polynomial time, such that: (i) Every job j accumulates at least $1/2$ mass. (ii) If $j_1 \prec j_2$, j_2 can only begin the accumulation after j_1 accumulates $1/2$ mass.* □

In the remainder of this section, we describe how to convert a pseudo-schedule obtained from Theorem 4.3 to a feasible schedule. According to Theorem 4.3, we can compute a pseudo-schedule Σ_s of length $O(\log m) \cdot T^{\text{OPT}}$ in which every job accumulates a mass of at least $1/2$, and hence a success probability of at least $\frac{1}{2e}$. Moreover, if $j_1 \prec j_2$, no machine is assigned to j_2 until j_1 has accumulated $1/2$ such mass. We now convert Σ_s to a (feasible) oblivious schedule Σ_o in two steps. We describe these two steps briefly and refer the reader to the appendix for details.

1. We use the elegant random delay technique of [19, 27] to delay the start step of the execution for each chain appropriately and obtain a new pseudo-schedule $\Sigma_{s,1}$ in which the number of jobs scheduled on any machine at any step is $O(\frac{\log(n+m)}{\log\log(n+m)})$. The randomized schedule can also be derandomized using techniques from [22, 25, 27]. We then "flatten" $\Sigma_{s,1}$ to obtain an oblivious schedule $\Sigma_{o,1}$, sacrificing a factor of $O(\frac{\log(n+m)}{\log\log(n+m)})$ in the schedule's length.

2. To obtain the final oblivious schedule Σ_o, we to take the oblivious schedule $\Sigma_{o,1}$ from above and replicate each step's machine assignment $O(\log n)$ times, so that all jobs will be finished with high probability.

Theorem 4.4. *For Problem SUU-C, there exists a polynomial-time algorithm to compute an oblivious schedule schedule with expected makespan within a factor of $O(\log m \log n \frac{\log(n+m)}{\log\log(n+m)})$ of the optimal.* □

For independent jobs, i.e., when the constraints C in Problem SUU-C is empty, we can prove a bound for oblivious schedules that slightly improves over the result stated at the end of §3.

Theorem 4.5. *For Problem SUU-I, there exists a polynomial-time algorithm to compute an oblivious schedule schedule with expected makespan within a factor of $O(\log n \cdot \log(\min\{n, m\}))$ of the optimal.* □

4.2 Tree-like precedence constraints

Our algorithm for tree-like precedence constraints uses techniques from [17], who extend the work of [27] on scheduling unrelated parallel machines with chain precedence constraints to the case where there are tree-like precedence constraints by decomposing the directed forests into $O(\log n)$ collection of chains. To state their result, we first introduce some notations used in [17]. Given a dag $G(V, E)$, let $d_{in}(u)$ and $d_{out}(u)$ denote the in-degree and out-degree, respectively, of u in G. A *chain decomposition* of G is a partition of its vertex set into subsets $B_1, \ldots, B_\lambda$ (called blocks) such that: (i) The subgraph induced by each block B_i is a collection of vertex-disjoint directed chains; (ii) For any $u, v \in V$, let $u \in B_i$ be an ancestor of $v \in B_j$. Then, either $i < j$, or $i = j$ and u and v belong to the same directed chain of B_i; (iii) If $d_{out}(u) > 1$, then none of u's out-neighbors are in the same blocks as u. The *chain-width* of a dag is the minimum value λ such that there is a chain decomposition of the dag into λ blocks. We now state the decomposition result.

Lemma 4.6 ([17], Lemma 1). *Every dag whose underlying undirected graph is a forest has a chain decomposition of width γ, where $\gamma \leq 2(\lceil \log n \rceil + 1)$. The decomposition can be computed within polynomial time.*

Using Lemma 4.6, we simply decompose a given directed forest into at most $\gamma = O(\log n)$ blocks, and within each block, apply our algorithm for the chain case (Theorem 4.4). Since the optimal expected makespan on any subgraph (subset of jobs) is a lower bound for that of the whole graph (whole set of jobs), this approach gives up another factor of $\log n$. We have thus obtained

Theorem 4.7. *For Problem* SUU*, if the dependency graph C is a directed forest, there exists a polynomial-time algorithm to compute an oblivious schedule schedule with expected makespan within a factor of $O(\log m \log^2 n \frac{\log(n+m)}{\log\log(n+m)})$ of the optimal.*

When the precedence constraints form a collection of *out trees* (rooted trees with edges directed away from the root) or *in trees* (defined analogously), we can obtain an improved approximation algorithm by again following the ideas of [17]. More specifically, we decompose the out/in trees into $O(\log n)$ blocks; then randomly delay each chain by an amount of steps chosen uniformly from $[0, O(\Pi_{max}/\log n)]$ (this step can be derandomized in polynomial time); and prove that with high probability, at most $O(\log n)$ jobs can be scheduled on any machine simultaneously. We defer the details to the full version.

Theorem 4.8. *For Problem* SUU*, if the dependency graph C is a collection of out/in trees, there exists a polynomial-time algorithm to compute an oblivious schedule schedule with expected makespan within a factor of $O(\log m \log^2 n)$ of the optimal.*

5 Open problems

In this paper, we have presented polylogarithmic approximation algorithms for the problem of multiprocessor scheduling under uncertainty, for special classes of dependency graphs. We believe that our bounds are not tight; in particular, we conjecture that a more careful analysis will improve the approximation ratios by an $O(\log n)$ factor in each case. It will also be interesting to obtain approximations for more general classes of dependencies, and to consider online versions of our scheduling problem.

References

[1] D. Applegate and B. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3(2):149–156, 1991.

[2] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[3] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[4] F. Chudak and D. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *ALGORITHMS: Journal of Algorithms*, 30, 1999.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Cambridge, MA, second edition, 2001.

[6] A. Fernandez, R. Armacost, and J. Pet-Edwards. A model for the resource constrained project scheduling problem with stochastic task durations. In *7th Industrial Engineering Research Conference Proceedings*, 1998.

[7] A. Fernandez, R. Armacost, and J. Pet-Edwards. Understanding simulation solutions to resource constrained project scheduling problems with stochastic task durations. *Engineering Management Journal*, 10(4):5–13, 1998.

[8] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, 1962.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 2004.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[11] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, 1999.

[12] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal (BSTJ)*, 45:1563–1581, 1966.

[13] L. Hall. Approximation algorithms for scheduling. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems, PWS Publishing Company*. 1997.

[14] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[15] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.

[16] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *SICOMP: SIAM Journal on Computing*, 30, 2000.

[17] V. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. In *APPROX: International Workshop on Approximation Algorithms for Combinatorial Optimization*, 2005.

[18] E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical Report BS-R8909, Centre for Mathematics and Computer Science., Amsterdam, 1991.

[19] F. T. Leighton, B. M. Maggs, and S. Rao. Packet routing and job-shop scheduling in O (congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[20] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *MATHPROG: Mathematical Programming*, 46, 1990.

[21] G. Malewicz. Parallel scheduling of complex dags under uncertainty. In *Proceedings of the 17th annual ACM symposium on Parallelism in algorithms and architectures*, pages 66 – 75, Las Vegas, Nevada, USA, 2005.

[22] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *JCSS: Journal of Computer and System Sciences*, 37, 1988.

[23] P. Raghavan and C. Thompson. Provably good routing in graphs: Regular arrays. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1985.

[24] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *COMBINAT: Combinatorica*, 7, 1987.

[25] J. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIJDM: SIAM Journal on Discrete Mathematics*, 8, 1995.

[26] A. Schrijver. *Theory of linear and integer programming*. Interscience Series in Discrete Mathematics and Optimization. Wiley, 1986.

[27] D. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SICOMP: SIAM Journal on Computing*, 23, 1994.

[28] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the Association for Computing Machinery (JACM)*, 48(2):206–242, 2001.

[29] M. Skutella and M. Uetz. Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 589–590, Washington, D.C., US, 2001.

A Success probability and mass

Proof of Proposition 2.1: The first assertion follows from the identity $(1-x_1)\cdots(1-x_k) \geq 1-(x_1+\cdots+x_k)$, which can be proved using a simple induction argument. The base case of $k=1$ is trivial. Suppose the identity holds for $k-1$. If $x_1+\cdots+x_{k-1} > 1$, then the identity holds for k; Otherwise, according to the induction hypothesis,

$$\begin{aligned} & (1-x_1)\cdots(1-x_{k-1})(1-x_k) \\ \geq\ & [1-(x_1+\cdots+x_{k-1})](1-x_k) \\ \geq\ & 1-(x_1+\cdots+x_k). \end{aligned}$$

For the second assertion, notice that if $0 \leq x \leq 1$, $1-x \leq e^{-x} \leq 1-\frac{x}{e}$. Since $1-x \leq e^{-x}$, $(1-x_1)\cdots(1-x_k) \leq e^{-x_1}\cdots e^{-x_k}$, we have

$$\begin{aligned} & 1-(1-x_1)\cdots(1-x_k) \\ \geq\ & 1-e^{-x_1}\cdots e^{-x_k} \\ =\ & 1-e^{-(x_1+\cdots+x_k)} \\ \geq\ & \frac{x_1+\cdots+x_k}{e}, \end{aligned}$$

where the last inequality follows because $e^{-x} \leq 1-\frac{x}{e}$ for $x \in [0,1]$ and the assumption that $x_1+\cdots+x_k \leq 1$. □

B Independent jobs

Proof of Theorem 3.2: Consider a bi-partite graph, where one side of the graph lie the nodes for jobs J and the other side lie the nodes for machines M. There is an edge (i,j) between machine i and job j for any $p_{ij} > 0$. MSM-ALG can be viewed as picking and orienting the edges. Let Opt = $\{(i,j)\}$ be the collection of edges of picked by the optimum assignment f^*. Let SOL be the solution computed by MSM-ALG. We use a charging argument below. Consider any edge $(i,j) \in$ Opt.

1. $(i,j) \in$ SOL, charge p_{ij} to itself.
2. $(i,j) \notin$ SOL:

(a) (i, j) is not added because in step 2, $f(i) \neq nil$. Let $j' = f(i)$. Charge p_{ij} to $p_{ij'}$ where $(i, j') \in$ SOL. Notice that $p_{ij} \leq p_{ij'}$, and $p_{ij'}$ will be charged at most once due to this situation because each machine i in Opt is used at most once.

(b) (i, j) is not added because in step 2, $f(i) = nil$ yet $\sum_{x:f(x)=j} p_{xj} + p_{ij} > 1$. Since p_{ij}'s are processed in decreasing order, we conclude that in SOL, $\sum_{x:f(x)=j} p_{xj} \geq 1/2$. Charge p_{ij} to $2\sum_{x:f(x)=j} p_{xj}$.

Observe that one copy of SOL is sufficient to cover the charges of types 1 and 2(a). Two copies of SOL are sufficient to cover the charges of type 2(b) because, by definition, the mass of any job is at most 1 in any assignment.

We conclude that MSM-ALG computes a solution with an approximation factor $1/3$. □

In the remainder of this section, we present a combinatorial algorithm for computing an oblivious schedule with expected makespan within $O(\log^2 n)$ of the optimal. According to Theorem 3.1, there exists an oblivious schedule of length $2T^{\text{OPT}}$, in which total mass of all jobs is at least $n/16$. Intuitively, if one computes an oblivous schedule Σ_1 of length $2T^{\text{OPT}}$ with the aim of maximizing the *total sum* of masses over the jobs, there should be *many* jobs accumulating constant masses in Σ_1. One can then remove those jobs and compute a second oblivious schedule Σ_2 of length $2T^{\text{OPT}}$ to maximize the total sum of masses for the remaining jobs, to remove some additional jobs which have accumulated constant masses. Since each computation of the oblivious schedule removes *many* jobs, this process should terminate quickly. By concatenating the $\Sigma_1, \Sigma_2, \ldots$ together, one obtains an oblivious schedule Σ in which *every* job accumulates constant mass.

By Theorem 3.2, we have a $1/3$ approximation algorithm for Problem MaxSumMass. However, MaxSumMass only considers oblivious schedules of length 1, i.e., each machine is assigned to at most one job. What we need is a procedure of finding an oblivous schedule of length $2T^{\text{OPT}}$, which maximizes the sum of masses over jobs. It turns out that one can extend MSM-ALG easily to take into account the schedule length, which can be *arbitrary*, and still obtain the same aproximation factor of $1/3$. We now formalize our discussion.

Problem (MaxSumMass-Ext): We are given a set J of n independent, unit-step jobs and a set M of m machines. Let p_{ij} denote the probability that job j is successfully completed if assigned to machine i. We are also given a parameter $t \in \mathbb{Z}^+$. The goal of the problem is to find an oblivious schedule Σ_o of length t such that the total sum of masses accumulated by the jobs by step t is maximized.

We show below Algorithm MSM-E-ALG, which outputs an oblivious schedule Σ_o of length $t \in \mathbb{Z}^+$ that is a $1/3$ approximate solution to Problem MaxSumMass-Ext. Algorithm MSM-E-ALG is a simple modification from MSM-ALG as follows. Since the schedule is of length t, each machine can be assigned t times. We maintain a *remaining capacity* parameter for each machine, t_i, initialized to the value t, to keep track of how many steps machine i is still available to be assigned. We also use x_{ij} to keep track of how many steps machines i is assigned to job j. In Step 2(a) of MSM-E-ALG, as long as t_i is positive, assign i to j for as many steps as necessary. In Step 2(b), we update t_i accordingly. In Step 3, we output an oblivious schedule $\Sigma_o = \{f_\tau(\cdot) : 1 \leq \tau \leq t\}$, which can be specified by x_{ij}'s as follows. Let $j_1, \ldots, j_n$ be an ordering of the jobs. $f_\tau(i) = j_k$ for $\sum_{1 \leq l < k} x_{ij_l} + 1 \leq \tau \leq \sum_{1 \leq l \leq k} x_{ij_l}$ and $1 \leq k \leq n$. Observe that the running time of MSM-E-ALG is independent of the value t because each p_{ij}, hence each pair (i, j), is processed exactly once in Step 2. It is not hard to see that MSM-E-ALG outputs a $1/3$ approximate solution to Problem MaxSumMass-Ext because similar analysis for MSM-ALG from Theorem 3.2 can be applied.

Lemma B.1. MSM-E-ALG *computes a solution to Problem* MaxSumMass-Ext *with an approximation factor* $1/3$.

We now present an approximation algorithm SUU-I-OBL for Problem SUU-I.

Algorithm 1 MSM-E-ALG

INPUT: Jobs J, machines M, p_{ij}'s and t.

1. Sort p_{ij}'s in decreasing order. Initialize: $\forall i, t_i \leftarrow t$; $\forall i, j, x_{ij} \leftarrow 0$.
2. For each p_{ij} according to the order:
   (a) $x_{ij} \leftarrow \min\left\{t_i, \left\lfloor \frac{1-\sum_{k \in M} x_{kj} \cdot p_{kj}}{p_{ij}} \right\rfloor\right\}$.
   (b) $t_i \leftarrow t_i - x_{ij}$.
3. Output Σ_o specified by x_{ij}'s.

Algorithm 2 SUU-I-OBL

INPUT: Jobs J, machines M, p_{ij}'s.

1. $t \leftarrow 1$.
2. $I \leftarrow 1$. $R \leftarrow J$. $\Sigma \leftarrow$ "empty schedule".
3. While ($|R| > 0$) and ($I \leq 66 \log n$)
   (a) Let Σ_I be the output of invoking MSM-E-ALG on R, M with the current t value. $\Sigma \leftarrow \Sigma \circ \Sigma_I$.
   (b) Remove jobs that accumulate at least $1/96$ mass from R.
   (c) $I \leftarrow I + 1$.
4. If $|R| > 0$, then $t \leftarrow 2t$, GOTO step 2; Otherwise, return Σ.

A few comments on SUU-I-OBL are in order. We use MSM-E-ALG repeatedly to accumulate constant masses for a good fraction of the jobs each round, until all jobs accumulate constant masses. There is still one obstacle though. Since we don't know the value of T^{OPT}, we have to "guess" a value of t for MSM-E-ALG, which must be large enough, e.g., at least $2T^{\text{OPT}}$, to ensure that there *exists* an oblivious schedule of length t in which the total mass is at least $n/16$, as proved in Theorem 3.1. In summary, in the loop of SUU-I-OBL (Step 3), we repeatedly invoke MSM-E-ALG to accumulate $1/96$ mass for the jobs, for at most $66 \log n$ rounds (we will explain the reason shortly). At the end of the loop (Step 4), if there are some remaining jobs, that means our t value is not large enough, we hence double the value of t and try the new t again by resetting the other parameters. Note that during each invocation of MSM-E-ALG, we start from scratch by ignoring any mass that the jobs may have accumulated in the previous rounds. We now analyze the performance of SUU-I-OBL.

If $t \geq 2T^{\text{OPT}}$, with one invocation of MSM-E-ALG using t, let x be the number of jobs that get at least $1/96$ mass. The total sum of masses over the jobs is at most $x \cdot 1 + (n - x) \cdot 1/96$ because the mass that any job accumulates is at most 1. From Theorem 3.1, we know that there exists an oblivious schedule of length t, with a total sum of mass at least $n/16$. Now according to Lemma B.1, MSM-E-ALG has an approximation ratio of $1/3$. Thus,

$$x \cdot 1 + (n - x) \cdot 1/96 \geq 1/3 \cdot n/16.$$

It follows that $x \geq n/95$. Since each invocation of MSM-E-ALG makes at least $1/95$ of the jobs accumulate $1/96$ mass, it is sufficient to invoke MSM-E-ALG at most $66 \log n$ times until all jobs accumulate at least $1/96$ mass.

To prove that SUU-I-OBL terminates in polynomial time, we first bound the value of T^{OPT}. Let $p_{min} = \min_{i,j} p_{ij}$. Obviously, if we let the jobs accumulate sufficient mass one by one by assigning all machines to a single job at any step, then every job accumulates a mass of at least 1 within a time interval of $\lceil \frac{n}{p_{min}} \rceil$. This implies that $T^{\text{OPT}} = O(\frac{n}{p_{min}} \log n)$. Since t is doubling every iteration in SUU-I-OBL, $O(\log n + \log \frac{1}{p_{min}})$ different t values will be "probed" before the algorithm terminates. With each t value, we invoke MSM-E-ALG at most $66 \log n$ times, and each such invocation runs in polynomial time. We conclude that algorithm SUU-I-OBL terminates within time polynomial in the size of the input. We have thus proved:

Lemma B.2. *For Problem* SUU-I*, one can compute in polynomial time an oblivious schedule of length $O(\log n)T^{\text{OPT}}$ in which every job accumulates a mass of at least $1/96$.*

Theorem B.3. *For Problem* SUU-I*, within polynomial time, we can compute an oblivious schedule whose expected makespan is within a factor of $O(\log^2 n)$ of the optimal.*

Proof. Using Lemma B.2, we first compute an oblivious schedule Σ_o of length $T = O(\log^2 n) \cdot T^{\text{OPT}}$ in which every job accumulates a mass of at least $1/96$. The infinite repetition of Σ_o, Σ_o^∞, is the oblivious schedule we want. Treating the execution of Σ_o^∞ during each step interval of $[k \cdot T + 1, (k + 1) \cdot T]$, where $k = 0, 1, \ldots$, as one iteration, by Proposition 2.1 we know that every job has a success probability of at least $\frac{1}{24e}$ during each iteration. Within $O(\log n)$ iterations, all jobs are finished with high probability. Thus, the expected makespan of Σ_o^∞ is within $O(\log^2 n)$ of T^{OPT}. We now formalize this argument.

Let random variable X be the iteration number when all jobs are finished. We bound the expected value

of X below.

$$\begin{aligned}
E[X] &= \sum_{i=0}^{\infty} \Pr(X > i) \\
&= \sum_{i=0}^{362\log n - 1} \Pr(X > i) + \sum_{i=362\log n}^{\infty} \Pr(X > i) \\
&\leq 362\log n \cdot 1 + \sum_{i=362\log n}^{\infty} n \cdot (1 - \frac{1}{96e})^i \\
&= 362\log n + n \cdot (1 - \frac{1}{96e})^{362\log n} \cdot \sum_{i=0}^{\infty} (1 - \frac{1}{96e})^i \\
&\leq 362\log n + \frac{96e}{n},
\end{aligned}$$

where the third inequality follows because every job has a probability $\frac{1}{96e}$ of success within each iteration, and the last inequality follows by summing the geometric series and the fact that $(1 - \frac{1}{96e})^{181} < 1/2$. This completes the proof of the theorem. □

C Jobs with dependencies

Proof of Lemma 4.2: The following linear program is the same as (LP1), except that $1/2$ is replaced by $1/16$ and t is replaced by $2T^{\text{OPT}}$. We argue that this linear program is feasible.

$$\begin{aligned}
\sum_{i \in M} p_{ij} x_{ij} &\geq 1/16 \quad \forall j \in J \\
\sum_{j \in J} x_{ij} &\leq 2T^{\text{OPT}} \quad \forall i \in M \\
\sum_{j \in C_k} d_j &\leq 2T^{\text{OPT}} \quad C_k \in C \\
x_{ij} &\leq d_j \quad \forall i, j \\
d_j &\geq 1 \quad \forall j \\
x_{ij} &\geq 0 \quad \forall i, j
\end{aligned}$$

Consider the first $2T^{\text{OPT}}$ execution steps using an optimal schedule Σ. Let random variable X_{ij} be the number of steps in which i is assigned to j. Let random variable Y_j be the total number of steps when there is some machine assigned to j. We know from Theorem 2.2 that with probability at least $1/4$, j accumulates at least $1/4$ mass within $2T^{\text{OPT}}$ steps. This amounts to the fact that the expected accumulated mass for j is at least $1/16$. Thus

$$\sum_{i \in M} p_{ij} \cdot E[X_{ij}] \geq 1/16.$$

Since in Σ a machine is assigned to at most a job at any step, $\sum_{j \in J} X_{ij} \leq 2T^{\text{OPT}}$. So

$$\sum_{j \in J} E[X_{ij}] \leq 2T^{\text{OPT}}.$$

Since we are considering only $2T^{\text{OPT}}$ steps of Σ, we have $\sum_{j \in C_k} Y_j \leq 2T^{\text{OPT}}$. Obviously, $X_{ij} \leq Y_j$. Taking the expectation, we have

$$\sum_{j \in C_k} E[Y_j] \leq 2T^{\text{OPT}}$$

and

$$E[X_{ij}] \leq E[Y_j].$$

We conclude that $x_{ij} = E[X_{ij}]$ for $i \in M, j \in J$ and $d_j = E[Y_j]$ for $j \in J$ form a solution to the linear program. Raising this solution by a factor of 8, we obtain a solution to (LP1). This means that a t of value $16T^{\text{OPT}}$ is achievable in (LP1). We have thus proved that $T^* \leq 16T^{\text{OPT}}$. This completes the proof of the lemma. □

We next describe in detail the two steps that convert a pseudo-schedule to a feasible oblivious schedule. Since the second step is simpler, we describe it first.

Schedule replication: We first replicate $\Sigma_{o,1}$ at each step by a factor of $\sigma = 16 \log n$ to get another oblivious schedule $\Sigma_{o,2}$. More precisely, let T denote $\Sigma_{o,1}$'s length and let $g_t(\cdot)$'s be the assignment functions of $\Sigma_{o,1}$. We define the assignment functions $f_t(\cdot)$'s of $\Sigma_{o,2}$ as follows. For any $t \in [1, \sigma \cdot T]$, $f_t(\cdot) = g_\tau(\cdot)$, where $\tau = \lfloor \frac{t-1}{\sigma} \rfloor + 1$. Note that if $\Sigma_{o,1}$ can be specified in space polynomial in the size of the input, as we will show in the "delay" step, so can $\Sigma_{o,2}$.

We define yet another oblivious schedule $\Sigma_{o,3}$ of length n as follows. Topologically sort the jobs according to the precedence constraints, e.g., appending the precedence chains one after another, and let $j_1, \ldots, j_n$ be the jobs in the sorted order. The assignment functions $h_t(\cdot)$'s for $\Sigma_{o,3}$ are specified as follows. $\forall i \in M, h_t(i) = j_t$, where $1 \leq t \leq n$. Now the final oblivious schedule we want is $\Sigma_o = \Sigma_{o,2} \circ \Sigma_{o,3}^{\infty}$. In other words, oblivious schedule Σ_o is simply the replicated $\Sigma_{o,1}$ followed by assigning all the machines to some job at each step.

We now analyze the expected makespan of Σ_o. If all jobs are successfully completed within step σT, the expected makespan is at most σT. The probability that this does not happen is at most $n(1 - \frac{1}{2e})^{\sigma} < 1/n^2$. Notice also that from step $\sigma T + 1$ on, Σ_o assigns all the machines to a single job at each step periodically (due to $\Sigma_{o,3}$, with a period length of n). The expected number of steps for a job to be completed is at most T^{OPT} if all the machines are assigned to it. Since we periodically assign the machines to any fixed job, on average, it takes at most (nT^{OPT}) steps to complete any fixed job. Hence, on average, it takes at most $n^2 T^{\text{OPT}}$ steps to complete all the jobs using the assignment functions beyond step σT. The expected makespan of Σ_o is thus at most

$$(1 - 1/n^2)\sigma \cdot T + 1/n^2 \cdot (\sigma \cdot T + n^2 T^{\text{OPT}}).$$

As we will prove shortly, $T = O(\log m \frac{\log(n+m)}{\log \log(n+m)}) \cdot T^{\text{OPT}}$ and $\sigma = 16 \log n$. We conclude that the expected makespan of Σ_o is $O(\log n \log m \frac{\log(n+m)}{\log \log(n+m)}) \cdot T^{\text{OPT}}$.

Converting pseudo-schedule Σ_s to an oblivious schedule: We now address the issue when the computed pseudo-schedule Σ_s from Theorem 4.3 is not yet feasible, that is, when some machine is assigned to more than one job at the same step. We claim that we can convert Σ_s to an oblivious schedule $\Sigma_{o,1}$ by sacrificing a factor of $O(\frac{\log(n+m)}{\log \log(n+m)})$.

Let Π_{max} be the load of Σ_s, i.e., the maximum number of jobs assigned to any machine. A result by Shmoys, Stein and Wein on job shop scheduling problem [27, Lemma 2.1] states that if we *delay* the starting step of each chain by an integral amount independently and uniformly chosen from $[0, \Pi_{max}]$, the resulting pseudo-schedule has no more than $O(\frac{\log(n+m)}{\log \log(n+m)})$ jobs scheduled on any machine during any step. We now explain what we mean by the term *delay*. Recall that in the last paragraph of the proof for Theorem 4.1, we first specify a function f_t^k for each constraint chain $C_k \in C$, and then define assignment function for Σ_s as $f_t = \cup_k f_t^k$. Suppose that a chain C_k is delayed by an amount of ϕ_k, the assignment function g_t^k for chain C_k is modified as follows. $\forall i \in M$, if $t \leq \phi_k, g_t^k(i) = \emptyset$; otherwise, $g_t^k(i) = f_{t-\phi_k}^k(i)$. And the assignment

function for the schedule is defined as $f_t = \cup_k g_t^k$. To make our presentation self-contained, we now outline the argument for the bound of $O(\frac{\log(n+m)}{\log\log(n+m)})$ below.

Fix a step t and a machine i. Let $p = \Pr[$at least τ units of processing are scheduled on machine i at step $t]$. Note that a job j could be scheduled in multiple steps, and each job is unit-step, it is equivalent to say that there are multiple processing units of job j. There are at most $\binom{\Pi_{max}}{\tau}$ ways to choose those τ processing units. Focus on a particular choice of τ units. If these units are from different chains, the probability that they are all scheduled at step t is at most $(\frac{1}{\Pi_{max}})^\tau$ since we choose the delay independently and uniformly from $[0, \Pi_{max}]$. Otherwise, the probability is 0 because our pseudo-schedule can never assign two units from the same chain to the same machine at the same step. Therefore,

$$\begin{aligned} p &\leq \binom{\Pi_{max}}{\tau}\left(\frac{1}{\Pi_{max}}\right)^\tau \\ &\leq \left(\frac{e\Pi_{max}}{\tau}\right)^\tau \left(\frac{1}{\Pi_{max}}\right)^\tau \\ &\leq \left(\frac{e}{\tau}\right)^\tau \end{aligned}$$

If $\tau = \alpha\frac{\log(n+m)}{\log\log(n+m)}$, then $p < (n+m)^{-(\alpha-1)}$. Let L_{max} be the length of the longest chain according to Σ_s. The probability that *any* machine at *any* step is assigned at least $\alpha\frac{\log(n+m)}{\log\log(n+m)}$ jobs is bounded by $m(\Pi_{max}+L_{max})(n+m)^{-(\alpha-1)}$. With the assumption, which we will remove shortly, that T^{OPT} is bounded by a polynomial in $(n+m)$, $\Pi_{max}+L_{max}$ is bounded by a polynomial in $(n+m)$ as well. If we choose α to be sufficiently large, then with high probability, no more than $\alpha\frac{\log(n+m)}{\log\log(n+m)}$ jobs are scheduled on any machine at any step.

Shmoys, Stein and Wein [27] also derandomize the algorithm so that $O(\log(n+m))$ jobs can be scheduled on any machine simultaneously, based on results by [23, 24, 22]. Schmdit, Siegel and Srinivasan [25] give a different derandomization strategy and obtain a collision bound matching the randomized algorithm, i.e., $O(\frac{\log(n+m)}{\log\log(n+m)})$ machines simultaneously for any machine. We denote this (derandomized) pseudo-schedule by $\Sigma_{s,1}$, whose length is at most twice that of Σ_s. According to Theorem 4.3, Σ_s's length is $O(\log m)\cdot T^{\text{OPT}}$, it follows that we can "flatten" $\Sigma_{s,1}$ out to obtain an oblivious schedule $\Sigma_{o,1}$ whose length is $O(\log m\frac{\log(n+m)}{\log\log(n+m)})\cdot T^{\text{OPT}}$, in which each machine is assigned to one job at any step. We comment that the *random delay* technique originates in [19] when they study the job shop scheduling problem.

Reducing T^{OPT}: We now address the issue that T^{OPT} is not always bounded by a polynomial in $(n+m)$. We make use of a trick from [27, Section 3.1]. Consider the pseudo-schedule Σ_s computed in Theorem 4.3. For each job j, let l_{ij} be the number of steps in which machine i is assigned to j and L_j be $\max_i l_{ij}$. Denote $\max_j L_j$ by L. We know that all machines are assigned to j within a window of length L_j. Let $\beta = nm$. Round each l_{ij} down to the nearest multiple of $\frac{L}{\beta}$, and denote this value by l'_{ij}. We therefore can treat the l'_{ij} as integers in $\{0,\ldots,\beta\}$. A schedule for this new problem can be trivially rescaled to one with the real values l'_{ij}. Since $\beta = nm$, the schedule now *effectively* has a length (and load) bounded by a polynomial in $(n+m)$. Hence our discussions of the random delay and derandomization hold now. Let Σ' be the resulting feasible oblivious schedule, with length bounded by $O(\log m\frac{\log(n+m)}{\log\log(n+m)})T^{\text{OPT}}$ and load bounded by $O(\log m)T^{\text{OPT}}$. To get a feasible oblivious schedule $\Sigma_{o,1}$ so that every job accumulates $1/2$ mass, we *insert* $(l_{ij}-l'_{ij})$ units of processing to Σ'. The insertion can be done in a way that preserves the precedence constraints, i.e., if $j_1 \prec j_2$, then no machine can be assigned to j_2 before j_1 accumulates $1/2$ mass. Since each insertion lengthens Σ' by an amount $\leq \frac{L}{nm}$ and we have at most nm such insertions, the length of the schedule is increased by at most L. The loads on the machines are the same as before the rounding. Note that L is bounded by Π_{max}, which is $O(\log m)T^{\text{OPT}}$. We thus have obtained a feasible oblivious schedule

$\Sigma_{o,1}$ whose length is $O(\log m \frac{\log(n+m)}{\log\log(n+m)})T^{\text{OPT}}$, in which every job accumulates a *constant* mass. Finally, we use the *replication* technique discussed earlier in this section to obtain the desired schedule.

Proof of Theorem 4.5: Let (LP2) be the linear program obtained from (LP1) by removing constraints 3, 4, 5, and T_2^* be (LP2)'s optimal value. We first show that one can round an optimal feasible solution to (LP2), and obtain an oblivious schedule for Problem AccuMass-C, whose length, and hence load, are both $O(\log(\min\{n,m\})) \cdot T_2^*$.

For Problem SUU-I, Condition (ii) of AccuMass-C is void. We thus don't need constraints 3, 4, 5 when writing the linear program. The rounding in the proof of Theorem 4.1 gives an $O(\log m)$ blow-up. If $m \geq n$, we can do a better analysis for the rounding procedure. Since there are $n+m$ non-trivial constraints in (LP2), there are at most $n+m$ nonzero values in any basic feasible solution [2, 26]. In an optimal solution $\{x_{ij}, t\}$ (which is basic feasible), we may assume without loss of generality that for any machine i, there exists a j such that $x_{ij} > 0$. Otherwise, we may remove that machine from consideration in (LP2). From here, we conclude that the number of machines i that have at least two $x_{ij} > 0$ is at most n. When we round x_{ij}'s, we only need to consider these machines i with at least two $x_{ij} > 0$. Then the same rounding procedure in the proof of Theorem 4.1 gives a factor $O(\log n)$ blow-up because for each job, we only need to consider $O(\log n)$ buckets.

We conclude that one can obtain an integral feasible solution $\{\hat{x}_{ij}, \hat{t}\}$ where $\hat{t} = O(\log(\min\{n,m\})) \cdot T_2^*$. Furthermore, from $\{\hat{x}_{ij}, \hat{t}\}$, one can construct a (feasible) oblivious schedule for Problem AccuMass-C, whose length, and hence load, are $\hat{t} = O(\log(\min\{n,m\})) \cdot T_2^*$. This is because the load on each machine is bounded by $\hat{t}$ according to Equation 2 and the jobs are independent. Hence the machine assignment can be done in such a way that no more than one job is scheduled on any machine at any step.

We thus have an oblivious schedule in which every job accumulates a *constant* mass within time that is at most $O(\log(\min\{n,m\})$ times optimal. We now apply the schedule replication step and obtain the desired bound. □