Journal of Parallel and Distributed Computing 157 (2021) 280–286

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Preemptive scheduling on unrelated machines with fractional precedence constraints

Vaneet Aggarwal *, Tian Lan, Dheeraj Peddireddy

ARTICLE INFO

Article history:
Received 3 May 2019
Received in revised form 25 February 2021
Accepted 13 July 2021
Available online 24 July 2021

Keywords:
Fractional precedence constraints
Preemptive scheduling
Unrelated machines
Birkhoff-von Neumann decomposition

ABSTRACT

Many programming models, e.g., MapReduce, introduce precedence constraints between the jobs. This paper formalizes a notion of precedence constraints, called fractional precedence constraints, where the progress of follower jobs only has to lag behind (fractionally) their leads. For a general set of fractional precedence constraints between the jobs, this paper provides a new class of preemptive scheduling algorithms on unrelated machines that have arbitrary processing speeds. In particular, for a given makespan, we establish both sufficient and necessary conditions on the existence of a feasible job schedule, and then propose an efficient scheduling algorithm based on a novel matrix decomposition method, if the sufficient conditions are satisfied. The algorithm is shown to be a Polynomial-Time Approximation Scheme (PTAS), i.e., its solution is able to achieve any feasible makespan with an approximation bound of $1+\epsilon$, for an arbitrary $\epsilon > 0$.

1. Introduction

Scheduling jobs on distributed servers is a widely studied area in operations research and computer science [17,10]. Many programming models and applications in real world introduce some form of precedence constraints between the jobs, meaning that the processing of some (follower) jobs are required to wait until the completion of other (lead) jobs. An example of such precedence constraints can be found in the popular MapReduce framework [8], which consists of two successive phases – map phase and reduce phase – where a reduce phase cannot begin to process until its related map phase is completed to some extent. Job scheduling under precedence constraints has been studied in [17] and approximation algorithms have been provided under a number of different conditions [11,12,15,6,9,14,3]. However, in the presence of precedence constraints, optimally scheduling jobs on unrelated machines that have arbitrary processing speeds is still an open problem.

In this paper, we consider a new form of precedence constraints, called *fractional precedence constraints*, where the percentage progress of follower jobs must always stay behind (fractionally) lead jobs. We focus on the optimal job scheduling problem on multiple unrelated machines under general fractional precedence constraints. In particular, we represent the precedence constraints by a directed acyclic graph, in which each vertex denotes a job and each arc denotes a fractional precedence constraint between a pair of dependent jobs. The constraint between the jobs j and j', represented as $j \prec j'$, implies that the completion percentage of (lead) job j must be greater than or equal to the completion percentage of (follower) job j' at any given time t. This relaxes the strict precedence constraints considered in prior work such as [15], where job j' cannot begin to process until job j has completely finished. Not only does fractional precedence constraints introduce a new model for scheduling dependent jobs, it is also strongly motivated by the development in practical systems, e.g., the Hadoop MapReduce framework [1] that allows reduce phase to follow map phase by a tunable gap parameterized by the mapred.reduce.slowstart.completed.maps field. More precisely, reduce tasks can process in parallel and utilize the results available from map tasks, while their progress must lag behind to satisfy the causality requirement. The proposed fractional precedence constraint formalizes the notion of such job dependence in the context of scheduling and illuminates new design opportunities beyond the assumption of strict precedence constraints.

Our main contribution is to provide a $1+\epsilon$-approximation algorithm for preemptive scheduling under fractional precedence constraints on unrelated machines, with the aim of minimizing the makespan. By unrelated machine, we do not assume any relation between the processing time of a job on different machines and consider a practical system model, where machines can have arbitrary processing speeds for different jobs. Further, in this paper we consider the preemptive scheduling strategy, since it allows for a job to be interrupted and resumed later on the same or a different machine thus providing additional flexibility as compared to non-preemptive scheduling that forbids the interruption of a job

* Corresponding author.
E-mail address: vaneet@purdue.edu (V. Aggarwal).

https://doi.org/10.1016/j.jpdc.2021.07.010
0743-7315/

execution until it is completed. Without precedence constraints, optimal low-complexity algorithms for preemptive job scheduling have been shown for the makespan as well as the weighted completion time objectives [17]. It is shown in [7] that preempting jobs can reduce the makespan by up to 33% as compared to a non-preemptive two-machine scheduling.

To develop an approximation algorithm for the scheduling problem, we first establish both necessary and sufficient conditions on the existence of a feasible job schedule under the fractional precedence constraints. This paper also assumes that the jobs can be readily moved between machines without any penalty, while this may not be realistic, for instance due to the cost of job migration and data locality. For a problem with m machines and n jobs, we show that the time-slotted scheduling problem is related to the Birkhoff-von Neumann decomposition [4] and develop a $O(\{\max(m,n)\}^{4.5})$-algorithm that finds a feasible job schedule under the fractional precedence constraints, if the sufficient conditions hold. Then, we prove that the gap between the necessary and sufficient conditions can be made sufficiently small, as the time line is divided into smaller time slots. Introducing a new matrix decomposition method, we show that the achieved makespan by our feasible scheduling algorithm is indeed the same as that in the sufficient conditions. These results allow us to provide a bi-section search algorithm to find the optimal makespan, by solving a linear programming Problem for feasibility check. The achieved approximation ratio is proven to be $1+\epsilon$, with an algorithm of complexity $O(poly(1/\epsilon))$ for any $\epsilon > 0$. Thus, the proposed algorithm is approximately optimal for preemptive scheduling with fractional precedence constraints on unrelated machines. The numerical results also demonstrate the improvement of makespan with fractional precedence constraints as compared to strict precedence constraints.

The main contributions of this paper are summarized as follows.

1. We propose fractional precedence constraints to formalize a novel notion of precedence constraints, where the progress of follower jobs has to lag behind (fractionally) their leads.
2. By establishing the sufficient and necessary conditions of the constrained job scheduling, we develop a novel matrix decomposition algorithm to verify the feasibility of a makespan objective.
3. An approximately optimal algorithm is provided for preemptive scheduling with fractional precedence constraints on unrelated machines, with an objective of minimizing makespan.

The rest of the paper is organized as follows. Section 2 formulates the problem. Section 3 describes the necessary and sufficient conditions for the proposed problems, and provides a feasible algorithm. Further, Section 3 also shows that the proposed sufficient conditions can provide a feasible algorithm, and an algorithm to verify the sufficient conditions. Section 4 proves the approximation ratio of $1+\epsilon$. Section 5 provides a numerical result comparing the approach with strict precedence constraints. Section 6 concludes this paper.

2. Problem formulation

We consider a set of n jobs to be scheduled on m machines. We denote the set of jobs as $\mathcal{J} = \{1, \cdots, n\}$, and the set of machines as $\mathcal{I} = \{1, \cdots, m\}$. The processing time of the job j on machine i is $p_{i,j}$. For the simplicity of notation, let p_j be defined as $p_{1,j}$. Further, we denote the velocity of i-th machine when processing job j by $v_{i,j} = p_{i,j}/p_j$. Thus, machine 1 will have unit velocity 1 for each job, and the velocities of other machines represent how fast or slow they are in terms of processing job j, relative to machine 1. At any given time, we assume that any job j can only be processed by at most one machine. In this paper, we consider a class of fractional precedence constraints that are defined as follows.

Definition 1. For any two jobs j, j', having a fractional precedence constraint $j \prec j'$ implies that the completion percentage of (lead) job j must be greater than or equal to the completion percentage of (follower) job j' at any time.

The fractional precedence constraint $j \prec j'$ implies that the progress of job j' lags behind that of job j at all times. We employ a directed acyclic graph (DAG) $G = (\mathcal{J}, E)$ to represent the fractional precedence constraints of the jobs. More precisely, the set of vertices $\mathcal{J}$ denote the set of jobs, and a fractional precedence constraint $j \prec j'$ is represented as an arc $(j, j') \in E$.

We assume that the scheduling strategy is preemptive. In other words, a job can be paused at any time and resumed later on the same or a different machine. In the presence of general DAG fractional precedence constraints and the preemptive scheduling, this paper aims to minimize the makespan, which is defined as the time taken to complete all the jobs on the unrelated machines.

3. Proposed algorithm

We first establish the necessary conditions for the preemptive job scheduling problem on unrelated machines under fractional precedence constraints. These are followed by a set of sufficient conditions for the problem. In particular, the sufficient conditions are proven by providing a feasible scheduling algorithm that utilizes a new matrix decomposition, related to the Birkhoff-von Neumann decomposition [4]. Then, a low-complexity algorithm is provided to check when the sufficient conditions hold. It enables us to develop an overall scheduling algorithm relying on bisection search of the makespan objective.

In the following subsections, we consider a time-slotted system, where the time is partitioned into equal-size time slots. We denote $x_{i,j,t}$ as the time fraction spent on machine i to process job j, during time slot t. Namely, we process size $x_{i,j,t} \cdot v_{i,j}$ of job j on machine i in time slot t. Let $x_{i,j,0} = 0$ for any job j and machine i, initially at time $t = 0$.

One of the key aspects of the algorithm is to find whether there exists a feasible schedule for a given makespan. Section 3.1 establishes necessary conditions on $x_{i,j,t}$ such that a feasible schedule exists for a given makespan. Section 3.2 provides sufficient conditions such that a valid schedule can be designed to achieve the makespan. In particular, to prove that these conditions are indeed sufficient, a scheduling algorithm is proposed in Section 3.3. This scheduling algorithm is analyzed in Section 3.3. We give a proof that if the sufficient conditions hold, the proposed schedule finishes the jobs by time T, thus achieving the target makespan. Finally, even though sufficient conditions have been provided, we require a computationally efficient algorithm to verify if these conditions are satisfied, which is provided in Section 3.4. This completes the analysis of our proposed algorithm.

Having identified an algorithm, we need to verify if there is a feasible schedule with a given makespan. To this end, a binary search is used to find lowest makespan such that there is a feasible schedule. After finding the minimum makespan (within ϵ gap), the scheduling algorithm in Section 3.3 with that makespan can be used. This overall algorithm is described in Section 3.5.

3.1. Necessary conditions

In this subsection, we will provide a set of necessary conditions on $x_{i,j,t}$, such that all jobs are finished by T time slots.

First, every $x_{i,j,t}$ is non-negative:

$$x_{i,j,t} \geq 0, \quad \forall i \in \{1, \cdots, m\}, \quad \forall j \in \{1, \cdots, n\}, \quad \forall t \in \{1, \cdots, T\}. \tag{1}$$

In any time slot $t \in \{1, \cdots, T\}$, a machine i can process at most 1 fraction of content. Thus, we have

$$\sum_{j=1}^{n} x_{i,j,t} \leq 1, \quad \forall i \in \{1, \cdots, m\}, \quad \forall t \in \{1, \cdots, T\}. \tag{2}$$

In any time slot t, a job j cannot be processed at the same time by different machines. Note that $\sum_{i=1}^{m} x_{i,j,t}$ represents the fraction of time taken for job j in time slot t among all machines. If this exceeds 1, it is impossible to avoid intersection of time where different machines are processing job j. Therefore, we have

$$\sum_{i=1}^{m} x_{i,j,t} \leq 1 \quad \forall j \in \{1, \cdots, n\}, \quad \forall t \in \{1, \cdots, T\}. \tag{3}$$

To ensure that each job is completed, we have

$$\sum_{t=1}^{T} \sum_{i=1}^{m} x_{i,j,t} v_{i,j} = p_j, \quad \forall j \in \{1, \cdots, n\}. \tag{4}$$

By the definition of fractional precedence constraints, if $j \prec j'$, the proportion completed for j' until the end of time t cannot exceed the proportion completed for j until time t.

$$\frac{1}{p_{j'}} \sum_{i=1}^{m} \sum_{s=0}^{t} x_{i,j',s} v_{i,j'} \leq \frac{1}{p_j} \sum_{i=1}^{m} \sum_{s=0}^{t} x_{i,j,s} v_{i,j} \quad \forall j \prec j', \\ \forall t \in \{1, \cdots, T\}. \tag{5}$$

We however note that all these constraints are necessary, but they might not give a feasible scheduling algorithm even if all the conditions are satisfied.

3.2. Sufficient conditions

In this subsection, we will provide a set of sufficient conditions on $x_{i,j,t}$, such that the all jobs are finished by time T. The proof of sufficiency of these conditions will be shown in the following subsection.

The problem of minimizing makespan with fractional precedence constraints can be reduced to a set of decision problems $\left\{\mathcal{D}(T) : T = 1, \cdots, \lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil\right\}$, where $\mathcal{D}(T)$ returns whether there exists a feasible preemptive scheduling which completes processing in time T. Note that a simple feasible scheduling is to assign all jobs to a single machine according to a topological ordering based on their precedence-DAG. The makespan for the scheduling is upper bounded by $\lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil$. Therefore, the optimal makespan is at most $\lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil$.

Suppose for a given T, we want to know if we are able to complete all jobs in T time slots.

The necessary condition (5) can have a situation where the delayed job may be running on faster machine and thus there may not exist a feasible strategy where we can split the times within the time-slot to satisfy the fractional precedence constraint at all times. In order to avoid such scenarios, we assume that if $j \prec j'$, the proportion completed for j' until the end of time t cannot exceed the proportion completed for j until time $t-1$. Namely, we have

$$\frac{1}{p_{j'}} \sum_{i=1}^{m} \sum_{s=0}^{t} x_{i,j',s} v_{i,j'} \leq \frac{1}{p_j} \sum_{i=1}^{m} \sum_{s=0}^{t-1} x_{i,j,s} v_{i,j} \quad \forall j \prec j', \\ \forall t \in \{1, \cdots, T\}. \tag{6}$$

Thus, we let the sufficient conditions be (1)-(4), (6). Since the sufficient conditions are weaker than the necessary conditions, there will be a gap. However, note that the gap is small if the time-slot becomes small, which will be the key to prove approximate optimality of the proposed algorithm.

3.3. Proof of sufficiency

In this subsection, we will show that (1)-(4), (6) establish the sufficient conditions on the existence of a feasible schedule. We prove this by constructing a feasible schedule below. Suppose that for a given T, the conditions (1)-(4), (6) are satisfied. Further, let $\bar{x}_{i,j,t}$ be a feasible solution that satisfies the conditions. We note that based on our system model, a feasible schedule cannot have multiple machines processing the same job at the same time, and each machine can process only one job at a time.

We can represent the information in $\bar{x}_{i,j,t}$ by a set of non-negative matrices X^t of size $m \times n$ (for each t), where $(X^t)_{i,j}$ is $\bar{x}_{i,j,t}$. From our model, matching constraints (that any machine can process at most one job at a time, and any job can be processed by at most one machine at a time) require that jobs to be processed simultaneously cannot locate on the same row or on the same column of matrix X^t. Therefore, a decomposition of X^t to pseudo-permutation matrices (i.e., non-negative matrices having at most one positive entry each row and each column) is needed. Note that by (2)-(3), the row sum and column sum of matrix X^t is less than or equal to 1. We want to find $d_1^t, \cdots, d_{l^t}^t \geq 0$, $\sum_{i=1}^{l^t} d_i^t \leq 1$, and a set of pseudo-permutation matrices $\{\Pi_1, \cdots, \Pi_{l^t}\}$, such that we can decompose the matrix X^t into $X^t \leq \sum_{i=1}^{l^t} d_i^t \Pi_i$ (We denote $\leq$ in matrices as the element-wise inequality). If we find such a decomposition, we can take d_i^t time fraction to process the job-machine pairs represented in Π_i simultaneously for the i-th iteration. Since $\sum_{i=1}^{l^t} d_i \leq 1$, and by the definition of pseudo-permutation matrix, we can schedule the jobs from X^t in a unit time slot satisfying the required matching constraints.

It is easy to see that such a problem is similar to Birkhoff-von Neumann decomposition. Note that by Birkhoff–von Neumann theorem [4], the class of $r \times r$ doubly stochastic matrices (defined as non-negative square matrices whose rows and columns sums to 1) is a convex polytope. For any $r \times r$ doubly stochastic matrix M, there exists $\theta_1, \cdots, \theta_k \geq 0$, $\sum_{i=1}^{k} \theta_i = 1$ and a set of permutation matrices $\{\Pi_1, \cdots, \Pi_k\}$ such that $M = \sum_{i=1}^{k} \theta_i \Pi_i$.

Note that X^t may have row sum or column sum less than one, and X^t may not be a square matrix. Thus, Birkhoff-von Neumann decomposition cannot be applied directly. We note that a non-square matrix cannot be doubly stochastic, limiting the direct use of Birkhoff-von Neumann decomposition. The proposed decomposition algorithm is summarized in Algorithm 1. In line 3, we augment X^t to a square matrix $\hat{X}^t$ with size $r \times r$, where r is the maximum of m and n. We let the entries on the first m rows and n columns of $\hat{X}^t$ to be the same as the entries in X^t, and let the rest of the entries be zero.

Since (2)-(3) guarantee that both the row sum and column sum of matrix X^t are less or equal to 1, we use a while loop in line 4-7 to modify $\hat{X}^t$ into a square matrix with each row sum and column sum equals to one. We achieve the goal by finding out the entry in the location with smallest row sum and column sum in each iteration. After each search, we add a number to the entry to make sure that at least one of its row sum or column sum reaches 1, and both modified sums are no greater than 1. Since we turn at least one row (or column) to reach a sum of 1 each time, and keep

all row sums and column sums less or equal to 1, we can modify $\hat{X}^t$ to a square matrix with each row sum and column sum equals to one on convergence.

In lines 8-18, we build up a sequence of sets $\mathcal{A}_e^t : e = 1, \cdots, l^t$ to show the scheduling detail within time slot t. Each set contains the time fraction d_e^t planned to take for each job j_k assigned on each machine i_q. We construct the sets $\mathcal{A}_e^t : e = 1, \cdots, l^t$ by decomposing $\hat{X}^t$ into a set of permutation matrices. By definition, a permutation matrix is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. Each permutation matrix can be interpreted as an assignment of jobs if we only focus on its upper left $m \times n$ submatrix. For e-th iteration, we find the permutation matrices by building up corresponding bipartite graphs $\mathcal{I} \cup \mathcal{J}$ to find perfect matchings. If $\hat{X}^t$ has positive entry $(\hat{X}^t)_{i,j}$, we add an edge $(i, j) : i \in \mathcal{I}, j \in \mathcal{J}$ with capacity 1 to the bipartite graph. Every time we find a perfect matching, we denote the smallest time fraction $(\hat{X}^t)_{i,j}$ as d_e^t, add the corresponding (i, j) pairs (only for $i = 1, \cdots, m$ and $j = 1, \cdots, n$) and time fraction d_e^t to $\mathcal{A}_e^t$. After that, we update $\hat{X}^t$ by deducting d_e^t to the corresponding entries in $\hat{X}^t$. We note that lines 8-18 follow the Birkhoff-von Neumann decomposition, since we obtain a square matrix which is decomposed into a convex combination of permutation matrices. It can be seen that at least one entry of $\hat{X}^t$ is changed to 0 after lines 13-15, so the loop presented in lines 9-18 terminates after a maximum of $O((\max(m, n))^2)$ iterations.

Note that we have modified X^t to $\hat{X}^t$ in line 3-7 by adding entries and expanding the size, thus the actual fraction of time used is less than or equal to that given by the algorithm. Thus, the actual algorithm will use no larger fraction of time as compared to that given by the decomposition.

Algorithm 1 Scheduling within time slot t.

1: **Input:** A $m \times n$ matrix X^t where $(X^t)_{i,j}$ is $\bar{x}_{i,j,t}$.
2: **Output:** A list of assignments $\{\mathcal{A}_e^t : e = 1, 2, \cdots, l^t\}$ with data size $\{d_e^t : e = 1, 2, \cdots, l^t\}$ to be scheduled in turn within time slot t.
3: Denote $r = \max\{m, n\}$. Let $\hat{X}^t$ be a $r \times r$ matrix where $(\hat{X}^t)_{i,j} = X_{i,j}^t$ for all $i \in \mathcal{I}$, and for all $j \in \mathcal{J}$, and let $(\hat{X}^t)_{i,j} = 0$ elsewhere.
4: **while** $\min\{\min_j\{\sum_{i=1}^r (\hat{X}^t)_{i,j}, \min_i\{\sum_{j=1}^r (\hat{X}^t)_{i,j}\}\} < 1$ **do**
5: Let $i^* = arg\min_i \sum_{j=1}^r (\hat{X}^t)_{i,j}$, and $j^* = arg\min_j \sum_{i=1}^r (\hat{X}^t)_{i,j}$
6: Update $(\hat{X}^t)_{i^*,j^*} = (\hat{X}^t)_{i^*,j^*} + \min\{1 - \sum_{j=1}^r (\hat{X}^t)_{i^*,j}, 1 - \sum_{i=1}^r (\hat{X}^t)_{i,j^*}\}$.
7: **end while**
8: Let e denotes the iteration number. Initialize $e = 1$.
9: **while** $\hat{X}^t$ has non zero element **do**
10: Build a bipartite graph $G = (\mathcal{I}, \mathcal{J}, E)$ where $(i, j) \in E$ if $(\hat{X}^t)_{i,j} > 0$. We assign capacity 1 to each edge.
11: Find the maximum matching (perfect matching) in graph G. Without loss of generality, suppose the maximum matching contains $(i_1, j_1), \cdots, (i_r, j_r)$.
12: $d_e^t = \min\{(\hat{X}^t)_{i_1,j_1}, \cdots, (\hat{X}^t)_{i_r,j_r}\}$.
13: **for** $q = 1$ to r **do**
14: Update $(\hat{X}^t)_{i_q,j_q} = (\hat{X}^t)_{i_q,j_q} - d_e^t$.
15: **end for**
16: Update $e = e + 1$.
17: $\mathcal{A}_e^t = \{(i_k, j_k) : k \in \{1, \cdots, r\}, i_k \in \mathcal{I}, j_k \in \mathcal{J}\}$
18: **end while**
19: $l^t = e$
20: **for** $e = 1$ to l^t **do**
21: For $(i_k, j_k) \in \mathcal{A}_e^t$ assign time fraction d_e of job j_k to machine i_k simultaneously, if the size left for job j_k is less than d_e, finish what is left.
22: **end for**

By the properties of Birkhoff-von Neumann decomposition from Theorem 1 in [4], we have the following lemma.

Lemma 1. *For the d_e^t obtained as the result of Algorithm 1, $\sum_{e=1}^{l^t} d_e^t = 1$ for any time slot t. Further, the Algorithm 1 is a $O(\max\{m, n\}^{4.5})$ algorithm.*

Proof. We first show that the augmentation of matrix X given by the lines 3-7 of Algorithm 1 results in a doubly stochastic matrix. At the end of the while loop, we note that all the row and column sums will be 1. So, the only possibility for the augmented matrix to not be a doubly stochastic matrix is if the loop never terminates. If the augmented matrix is not doubly stochastic, there is at least a row as well as a column which does not sum to 1. Thus, we can augment the corresponding element on the said row and column to make the higher of the sums as 1. Since in each run, at least one of the row or the column sums becomes 1, the while loop runs at most $2 \times \max(m, n)$ times. Thus, the while loop runs finitely many times and will result in a doubly stochastic matrix. Further, we can see that the arg-min in line 5 takes $O(\max\{m, n\})$ time. So, the overall complexity for augmentation is $O(\max\{m, n\}^2)$.

Having shown that the augmentation results in a doubly stochastic matrix, the rest of the proof is based on the well-known Birkhoff-von Neumann theorem [13], which states that any $r \times r$ doubly stochastic matrix is a convex hull of the set of $r \times r$ permutation matrices. Thus, for any $r \times r$ doubly stochastic matrices M, there exist $\theta_1, \cdots, \theta_k \geq 0$, $\sum_{i=1}^k \theta_i = 1$ and a set of permutation matrices $\{\Pi_1, \cdots, \Pi_k\}$ such that $M = \sum_{i=1}^k \theta_i \Pi_i$. Since, lines 8-18 of Algorithm 1 follow Birkhoff-von Neumann decomposition of the augmented matrix with d_e^t being the coefficients of decomposition, $\sum_{e=1}^{l^t} d_e^t = 1$ for all t.

To understand the Birkhoff-von Neumann decomposition result, it can be seen that the loop (line 9) terminates when all the entries of the matrix $\hat{X}^t$ are 0 resulting in row and column sums being 0 after l^t iterations. Following the lines 13-15 of Algorithm 1, we note that sum of each row and column is decreased by d_e^t at each iteration e. Given that $\hat{X}^t$ is a doubly stochastic matrix, it follows that $\sum_{e=1}^{l^t} d_e^t = 1$. The complexity of the Birkhoff-von Neumann decomposition is $O(\max\{m, n\}^{4.5})$, by mapping the matching problem to the maximum flow problem. The details of this complexity can also be seen in [5]. □

Similar statements can be found in Theorem 1 of [13], Theorem 4.3 of [19], Fact 2 of [16], and Lemma 4 of [18]. Due to augmentation and the addition of values in the terms (lines 3-7, Algorithm 1), the actual fraction of time taken in any machine will be at most 1, thus guaranteeing that the schedule can be completed within a timeslot.

3.4. Algorithm for verifying sufficient conditions

In this subsection, we give a polynomial time oracle $\mathcal{D}(T)$ to decide whether or not there exists a feasible solution satisfying (1)-(4), (6) for a given T. In order to do that, we form a linear program called LP-FeasibilityChecking, whose solution will determine whether the conditions are feasible or not.

Definition 2. For any T, we define the following linear optimization as LP-FeasibilityChecking.

$$\min \quad \sum_{j=1}^{n} y_j \tag{7}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{i,j,t} \leq 1, \quad \forall i \in \{1, \cdots, m\}, \quad \forall t \in \{1, \cdots, T\}; \tag{8}$$

$$\sum_{i=1}^{m} x_{i,j,t} \leq 1 \quad \forall j \in \{1, \cdots, n\}, \quad \forall t \in \{1, \cdots, T\}; \tag{9}$$

$$\frac{1}{p_{j'}} \sum_{i=1}^{m} \sum_{s=0}^{t} x_{i,j',s} v_{i,j'} \leq \frac{1}{p_j} \sum_{i=1}^{m} \sum_{s=0}^{t-1} x_{i,j,s} v_{i,j} \quad \forall j \prec j', \quad \forall t \in \{1, \cdots, T\} \tag{10}$$

$$\sum_{t=1}^{T}\sum_{i=1}^{m} x_{i,j,t}v_{i,j} + y_j = p_j, \quad \forall j \in \{1,\cdots,n\}, \tag{11}$$

$$y_j \geq 0, \quad \forall j \in \{1,\cdots,n\} \tag{12}$$

$$x_{i,j,t} \geq 0, \quad \forall i \in \{1,\cdots,m\}, \quad \forall j \in \{1,\cdots,n\}, \quad \forall t \in \{1,\cdots,T\} \tag{13}$$

For any T, the above constraints (8)-(13) are always feasible - since all $x_{i,j,t} = 0$ and $y_j = p_j$ is a solution. We solve the linear programming problem with T and consider the optimal value for a given T.

Definition 3. For any given T, we define $f(T)$ as the optimal $\sum_{j=1}^{n} y_j$ achieved by solving the linear optimization problem (7)-(13).

If $f(T) = 0$, then the constraints (8)-(13) have $\{x_{i,j,t}\}$ that can be satisfied with $y_j = 0$. In other words, $\{x_{i,j,t}\}$ is a solution for the feasibility problem (1)-(4), and (6). If $f(T) > 0$, then the problem ((1)-(4), (6)) has no feasibility. Thus, we obtain the following result.

Lemma 2. *We have $f(T) = 0$ if and only if (1)-(4) and (6) have a feasible solution with parameter T.*

Proof. Note that for any fixed T, (1)-(3) and (6) are the same as (8)-(10), and (13). If (7)-(13) has an optimal solution with value 0, by (12), $y_j = 0 : j = 1,\cdots,n$. Therefore (11) is equivalent to (4). In other words, if $\{x_{j,i,t} : t = 1,\cdots,T\}$ is a solution to (7)-(13) with zero optimal value, it also satisfies (1)-(4), (6), with parameter T.

Similarly, if $\{x_{i,j,t}\}$ satisfies (1)-(4), (6), $\{x_{i,j,t}\}$ is a solution for (7)-(13) with optimal value 0. Since zero valued objective is feasible and is lowest possible due to (12), zero will be the optimal solution. □

3.5. Proposed algorithm

In this section, we provide Algorithm 2 for the overall scheduling. We assume $f(0) \neq 0$. In lines 4-13, we use binary search to find the minimum makespan with the help of our defined linear programming problem. A simple feasible scheduling is to assign all jobs to a single machine according to a topological ordering based on their precedence-DAG. The makespan for the scheduling is upper bounded by $\lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil$. Therefore, the optimal makespan is at most $\lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil$. Note that if $f(T_1) = 0$, then $f(T_2) = 0$ for any $T_2 \geq T_1$ by keeping all of the decision variables the same and setting additional variables equal to zero. Based on our definition of polynomial time oracle based on $f(T)$, we find a minimum makespan by binary search starting from interval $[0, \lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil]$. The interval length is decreased to half of its previous length. If the intermediate point is feasible, we take the left interval, else take the right interval. Once the interval shrinks to length 1, we obtain the optimal choice of T for which a schedule will be generated.

In line 14, we use the makespan T achieved by the binary search above to solve the Linear Programming Problem (LP-T) defined by (7)-(13). The optimal solution of LP-T, $\{\bar{x}_{i,j,t} : t = 1,\cdots,T\}$ make up the matrices X^t which provide a scheduling outline (time fraction to process each job on each machine within a time slot). Lines 15-17 provide a detailed schedule within each time slot using Algorithm 1. At the end of the cycle, we get a sequence of sets $\{\mathcal{A}_e^t : t = 1,\cdots,T, e = 1,\cdots,l^t\}$ by decomposing the X^t into a convex combination of permutation matrices. The coefficients of decomposition represent the fraction of time within the time slot t and the permutation matrices represent the job-machine pair to be processed in the time fraction. For the overall scheduling, we process these sets in the increasing order of e for all time slots.

Algorithm 2 Overall Scheduling Algorithm.

1: **Input:** n jobs with sizes p_j for $j \in \{1,\cdots,n\}$, m machines with velocity $v_{i,j}$ for machine $i \in \{1,\cdots,m\}$, job $j \in \{1,\cdots,n\}$.
2: **Output:** Makespan T, and a list of assignments $\{\mathcal{A}_e^t : p = 1,2,\cdots,l^t, t\}$ with data size $\{d_e^t : e = 1,2,\cdots,l^t, t\}$ to be scheduled in turn within each time slot $t = 1,\cdots,T$.
3: Let $T_L = \lceil \sum_j p_j / \min_{i,j}\{v_{i,j}\} \rceil$
4: **Binary Search for makespan** T**:** Initialize $a = 0, b = T_L$.
5: **while** $b - a > 1$ **do**
6: Let $T = \lceil (a+b)/2 \rceil$, solve LP-T and find the value of $f(\lceil (a+b)/2 \rceil)$
7: **if** $f(\lceil (a+b)/2 \rceil) = 0$ **then**
8: $b = \lceil (a+b)/2 \rceil$
9: **else**
10: $a = \lfloor (a+b)/2 \rfloor$
11: **end if**
12: **end while**
13: $T = b$
14: **Feasible Scheduling:** Solve LP-T, and suppose the optimal solution is $\bar{x}_{i,j,t}$: for all $i \in \mathcal{I}$, $j \in \mathcal{J}$, and $t = 1,\cdots,T$.
15: **for** $t = 1$ to T **do**
16: Use Algorithm 1 to schedule jobs assigned within time slot t
17: **end for**

The following theorem shows the feasibility that if the sufficient conditions are satisfied for a makespan of T, the proposed algorithm achieves a makespan of T.

Lemma 3. *If the sufficient conditions are satisfied, with T as the makespan, the total makespan achieved by Algorithm 2 does not exceed T.*

Proof. Note that $\hat{X}^t$ is augmented from X^t, namely each element of $\hat{X}^t$ is greater or equal to the one in X^t. Within any time slot t, we assign at most d_e^t time fraction from job j_k to machine i_k simultaneously for iteration e, the processing time for job assigned on any time slot is less than or equal to 1 (from Lemma 1). Adding the processing time for all time slots, we conclude that the total completion time does not exceed T. □

For time complexity, note that linear programming problems are solvable in polynomial time, (The runtime of using Karmarkar's algorithm is $O((mnT)^{3.5}(mnT)^2 \cdot \log(mnT) \cdot \log\log(mnT)))$ [2]. Thus, finding solution for $f(T)$ is polynomial time. The number of times that the binary search is used is $O(\log(\sum_j p_j / \min_{i,j}\{v_{i,j}\}))$. Let T in line 13 be denoted as T_0. By Lemma 1, the time complexity of scheduling jobs within one time slot is $O(\max\{m,n\}^{4.5})$, the complexity of scheduling on T_0 time slots in lines 14-17 is $O(\max\{m,n\}^{4.5}T_0)$. Since T in each binary search loop is $O(T_L)$ and so is T_0, and $T_L = O(\sum_j p_j / \min_{i,j}\{v_{i,j}\})$, we have the following result.

Lemma 4. *Algorithm 2 has a complexity of $O(poly(\max(m,n), \sum_j p_j / \min_{i,j}\{v_{i,j}\}))$.*

4. Approximation guarantee

We note that the constraints (1)-(5) establish necessary conditions and the constraints (1)-(4), (6) establish sufficient conditions for the existence of a feasible schedule. Given Lemma 3, we know that with the feasible conditions being satisfied for a given T, we can find an algorithm with a makespan of T. To prove the approximation bound of our proposed algorithm, in the following Lemma, we characterize the gap in the number of time-slots taken by the optimal algorithm as compared to that taken by the proposed algorithm.

Lemma 5. *Suppose P is the number of jobs in precedence-DAG. Suppose the optimal makespan is T_1. Then, the makespan of the proposed algorithm is at-most $T_1 + P$.*

Proof. We note that since T_1 is the makespan for the optimal schedule, the corresponding schedule satisfies (1)-(5) with $T = T_1$. Let $\{x_{i,j,t} : t = 1, \cdots, T_1\}$ be the above optimal schedule.

By the topological ordering of the precedence-DAG, suppose job j is the $\sigma(j)$-th job in the sequence. We modify $\{x_{i,j,t} : t = 1, \cdots, T_1\}$ to $\{\tilde{x}_{i,j,t} : t = 1, \cdots, T_1 + P\}$ as the following:

$$\tilde{x}_{i,j,t} = \begin{cases} 0, & t = 1, \cdots, \sigma(j), \\ x_{i,j,t-\sigma(j)+1}, & t = \sigma(j), \cdots, T_1 + \sigma(j), \\ 0, & t = T_1 + \sigma(j), \cdots, T_1 + P. \end{cases}$$

$\{\tilde{x}_{i,j,t} : t = 1, \cdots, T_1 + P\}$ definitely satisfies (1)-(4), and (6) with parameter $T = T_1 + P$. □

Note that the difference between (5) and (6) is one time unit in the summation for each precedence constraint. If we change the size of a time slot unit, we can make the ratio $(T + P)/T$ arbitrarily close to 1. Thus, we have the following result.

Theorem 1. *Algorithm 2 provides a feasible scheduling having $1 + \epsilon$ approximation rate for any $\epsilon > 0$, with a complexity of $O(poly(1/\epsilon))$.*

Proof. We note that the ratio gap between the optimal makespan and the makespan for the proposed feasible solution can be reduced to be arbitrarily close to 1 by expanding the number of time-slots. Further, when sufficient conditions are satisfied for any T, there is a feasible solution achieving makespan T. For the computational complexity, the loop for binary search runs $O(\log(1/\epsilon))$ times since the number of time-slots are $O(1/\epsilon)$. Solving the linear optimization for verifying the feasibility is $O(poly(1/\epsilon))$, which proves the result as in the statement of the theorem. □

5. Numerical results

In order to see the performance of fractional precedence constraints, we compare the approach with two baselines; a policy assuming strict precedence constraints over full jobs and fractional jobs. In this case of fractional jobs, we break down each job into d equal fractions and strict precedence constraints are assumed over the fractions i.e. if $j \prec j'$, the number of completed (processed) fractions of job j must be greater than that of job j'. This will numerically demonstrate the advantage of using fractional precedence constraints over strict precedence constraints in scheduling problems. We note that for the strict precedence constraints, it has been shown that the scheduling is hard to approximate within a factor of $2 - \epsilon$ for any $\epsilon > 0$, even for identical machines [20]. Thus, there are no known efficient solutions for scheduling problems with strict precedence constraints, and thus we implement a heuristic strategy for the baselines.

The comparable strategy assigns the incomplete jobs with the least processing time (LPT) on the next available machine. The LPT rule orders the jobs in the order of increasing processing times. Whenever a machine is freed, the shortest job ready at the time will begin processing. This algorithm mainly adapts the least processing time with precedence constraints and is described in Algorithm 3.

In lines 4-9, we select the first job on the topologically ordered list of all the incomplete jobs and assign it to the available machine with the shortest processing time. In lines 10-14, we iterate over the rest of the incomplete jobs without any precedence constraints with the first job and assign them to available machines based on their least processing times.

Algorithm 3 A heuristic preemptive scheduling algorithm.

1: **Input:** n jobs with precedence constraints, represented by a graph $G = (V, E)$, where $|V| = n$. If $(j_1, j_2) \in E$, the two jobs j_1 and j_2 has a precedence constraint $j_1 \prec j_2$.
2: **Output:** A feasible, preemptive scheduling under strict precedence constraints.
3: **while** There exists at least one incomplete job **do**
4: List all of the incomplete jobs (we only consider the incomplete part of jobs as the whole jobs from now on).
5: Label the k incomplete jobs with topological ordering and denote the list as $\sigma(1), \cdots, \sigma(k)$.
6: Label all of the machines as available. Label all of the jobs in the list as unselected.
7: Select the machine having least processing time of job $\sigma(1)$. Label the selected machine as unavailable, and label $\sigma(1)$ as selected.
8: Denote t as the time needed for $\sigma(1)$ to be completed on the selected machine.
9: Assign job $\sigma(1)$ on the machine for time t.
10: **for** $j = 2, \cdots, k$ **do**
11: **if** There is no $(\sigma(1), \sigma(j)) \in E$, for all *job* selected **then**
12: Select the available machine having least processing time of job $\sigma(j)$. Label the selected machine as unavailable, and label $\sigma(j)$ as selected.
13: **end if**
14: **end for**
15: **end while**

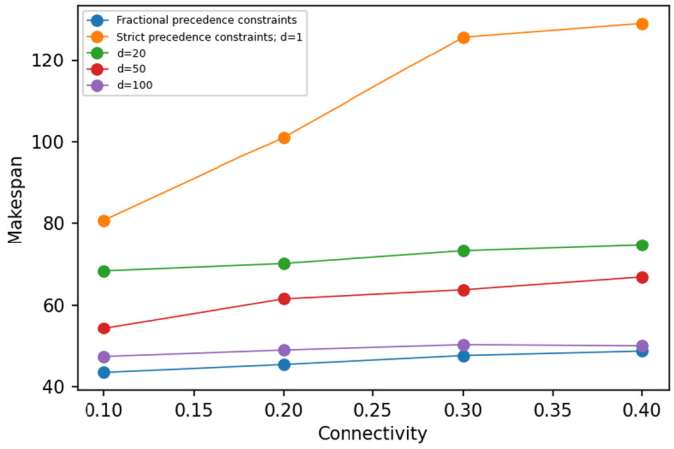

Fig. 1. Makespan as a function of connectivity in the job graph.

We assume that there are 30 jobs to be processed on eight machines. The processing task size of each job is taken uniform in $(0, 10)$, and the velocities of the machines are assumed to be uniform in $(0, 1)$. Once the task sizes and the velocities are generated, they are kept fixed for the experiment. To model the constraints between the jobs, we assume that the graph is formed as a random graph, where any two jobs are connected with probability p. Further, the jobs are assumed to be ordered from 1 to 30, and the direction is from the lower numbered job to the higher numbered job indicating that the higher numbered job needs the lower numbered job to be finished if there is a connection between them. This is done to avoid cyclic characteristics in the generated job graph.

Fig. 1 shows the improved makespan for the proposed fractional precedence constraint algorithm and compares it to the processing time that is resulted from existing heuristic scheduling algorithms under strict precedence constraints over full jobs [20] and fractional jobs. It can be observed that the proposed algorithm performs significantly better than the heuristic algorithms. However the gap in performance is reduced as the number of fractions d increases. Despite the performance, it should be noted that the heuristic algorithms do not provide any formal notions of convergence guarantees. We also note that as p increases (i.e., when jobs have higher precedence dependence), the proposed approach does

not result in a large increase of makespan since the proposed algorithm taking into account fractional precedence constraints can efficiently schedule the jobs and eliminate server idle times, while this is not the case with the strict precedence algorithm where the makespan increases significantly with p.

6. Conclusions

This paper formalizes the notion of fractional precedence constraints, where the progress of follower jobs has to lag behind lead jobs. Approximation algorithms are developed to solve the preemptive scheduling problem under fractional precedence constraints on unrelated machines, with an objective of minimizing makespan. The proposed algorithm achieves an approximation bound of $1+\epsilon$, for any $\epsilon > 0$, and is thus approximately optimal. This paper gives novel results for minimizing makespan with the fractional precedence constraints assuming certain ideal conditions such as ignoring some realistic constraints on data migration. Extending the proposed model and algorithm with such practical considerations is an open problem and will be considered in our future work. There has been limited literature on the approximate algorithms for weighted completion time objective with general fractional precedence constraints. Considering the objective of weighted completion time would be an interesting open problem.

CRediT authorship contribution statement

Vaneet Aggarwal: Formal analysis; Funding acquisition; Investigation; Methodology; Project administration; Writing – original draft; Writing – review & editing. **Tian Lan:** Problem discussion; Project administration; Writing – review & editing. **Dheeraj Peddireddy:** Writing – review & editing.

Declaration of competing interest

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

References

[1] Apache Software Foundation, Hadoop, https://hadoop.apache.org.
[2] I. Adler, M.G. Resende, G. Veiga, N. Karmarkar, An implementation of Karmarkar's algorithm for linear programming, Math. Program. 44 (1–3) (1989) 297–335.
[3] V. Aggarwal, M. Xu, T. Lan, S. Subramaniam, On the optimality of scheduling dependent mapreduce tasks on heterogeneous machines, preprint, arXiv:1711.09964, 2017.
[4] G. Birkhoff, Tres observaciones sobre el algebra lineal, Rev. Univ. Nac. Tucumn, Ser. A 5 (1946) 147–150.
[5] C.-S. Chang, W.-J. Chen, H.-Y. Huang, On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann, in: 1999 Seventh International Workshop on Quality of Service, IWQoS'99 (Cat. No. 98EX354), IEEE, 1999, pp. 79–86.
[6] F.A. Chudak, D.B. Shmoys, Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds, J. Algorithms 30 (2) (1999) 323–343.
[7] E. Coffman Jr, M. Garey, Proof of the 4/3 conjecture for preemptive versus nonpreemptive two-processor scheduling, Report Bell Laboratories, Murray Hill 166, 1991.
[8] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[9] K. Djellab, Scheduling preemptive jobs with precedence constraints on parallel machines, Eur. J. Oper. Res. 117 (2) (1999) 355–367.
[10] M. Drozdowski, Scheduling for Parallel Processing, Springer, 2009.
[11] J. Han, R. Sadykov, F. Vanderbeck, Parallel machine scheduling with generalized precedence relations, in: Multidisciplinary International Scheduling Conference: Theory & Applications, 2013.
[12] K. Jansen, R. Solis-Oba, Approximation algorithms for scheduling jobs with chain precedence constraints, in: International Conference on Parallel Processing and Applied Mathematics, Springer, 2003, pp. 105–112.
[13] E.L. Lawler, J. Labetoulle, On preemptive scheduling of unrelated parallel processors by linear programming, J. ACM 25 (4) (1978) 612–619.
[14] S. Li, Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations, preprint, arXiv:1707.08039, 2017.
[15] K. Makarychev, D. Panigrahi, Precedence-constrained scheduling of malleable jobs with preemption, in: International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 823–834.
[16] M.J. Neely, E. Modiano, Y.-S. Cheng, Logarithmic delay for $n \times n$ packet switches under the crossbar constraint, IEEE/ACM Trans. Netw. 15 (3) (2007) 657–668.
[17] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, Springer, 2016.
[18] Z. Qiu, C. Stein, Y. Zhong, Minimizing the total weighted completion time of coflows in datacenter networks, in: Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2015, pp. 294–303.
[19] D. Shah, J.N. Tsitsiklis, Y. Zhong, On queue-size scaling for input-queued switches, Stoch. Syst. 6 (1) (2016) 1–25.
[20] O. Svensson, Conditional hardness of precedence constrained scheduling on identical machines, in: Proceedings of the Forty-Second ACM Symposium on Theory of Computing, 2010, pp. 745–754.

Vaneet Aggarwal received the BTech degree from the Indian Institute of Technology Kanpur, Kanpur, India, in 2005 and the MA and PhD degrees from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively, all in Electrical Engineering. He is currently an Associate Professor in the School of Industrial Engineering, and the School of Electrical and Computer Engineering at Purdue University. Prior to that, he was a Senior Member of the Technical Staff - Research for five years with AT&T Labs Research, Bedminster, NJ, USA. Dr. Aggarwal has been an adjunct faculty at Columbia University for two years. His research interests are in the applications of statistical, algebraic and optimization techniques to wireless systems, machine learning, and distributed storage systems. Dr. Aggarwal received the Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009 and IEEE Jack Neubauer Memorial Award in 2017. He is currently an Associate Editor for IEEE Transactions on Communications, and IEEE Transactions on Green Communications and Networking.

Tian Lan is an associate professor in the Department of Electrical and Computer Engineering at the George Washington University, which I joined in 2010. I received my Ph.D. from the Department of Electrical Engineering at the Princeton University in 2010, M.S. from the Department of Electrical and Computer Engineering at the University of Toronto in 2005, and B.A.Sc. in Electrical Engineering from the Tsinghua University in 2003. My research areas include network optimization and algorithms, cyber security, network protocols, cloud and edge (fog) computing, distributed storage, and wireless networks.

Dheeraj Peddireddy is a PhD student in the School of IE at Purdue University. His research interests are in machine learning and distributed computing.