COMMUNICATION-AWARE SCHEDULING OF PRECEDENCE-CONSTRAINED TASKS ON RELATED MACHINES

Yu Su*[1], Xiaoqi Ren†[2], Shai Vardi‡[3] and Adam Wierman§[1]

[1]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA
[2]Google, Kirkland, WA
[3]Krannert School of Management, Purdue University, West Lafayette, IN

ABSTRACT

Scheduling precedence-constrained tasks is a classical problem that has been studied for more than fifty years. However, little progress has been made in the setting where there are communication delays between tasks. Results for the case of identical machines were derived nearly thirty years ago, and yet no results for related machines have followed. In this work, we propose a new scheduler, Generalized Earliest Time First (GETF), and provide the first provable, worst-case approximation guarantees for the goals of minimizing both the makespan and total weighted completion time of tasks with precedence constraints on related machines with machine-dependent communication times.

1 Introduction

In this paper we study scheduling precedence-constrained tasks onto a set of heterogeneous machines with communication delays between the machines in order to minimize the makespan or the total weighted completion time. Initially, work on this topic was motivated by the goal of scheduling jobs on multi-processor systems, e.g., [1]. Today this problem is timely due to the prominence of large-scale, general-purpose machine learning platforms. For example, in systems such as Google's TensorFlow [2], Facebook's PyTorch [3] and Microsoft's Azure Machine Learning (AzureML) [4], machine learning workflows are expressed via a computational graph, where jobs are made up of tasks, represented as vertices, and precedence relationships between the tasks, represented as edges. This "precedence graph" abstraction allows data scientists to quickly develop and incorporate modular components into their machine learning pipeline (e.g., data preprocessing, model training, and model evaluation) and then easily specify a workflow. The graphs that specify the workflows in platforms such as TensorFlow, PyTorch and AzureML can be made up of hundreds or even thousands of tasks, and the jobs may be run on systems with thousands of machines. As a result, the performance of the platforms depends on how these precedence-constrained tasks are scheduled across machines.

The goal of scheduling jobs composed of precedence-constrained tasks has been studied for more than fifty years, starting with the work of [5]. The simplest version of this scheduling problem focuses on scheduling a single job with n precedence-constrained tasks on m identical parallel machines with the goal of minimizing the *makespan*: the time until the last task completes. More generally, the goal of minimizing the *total weighted completion time* is considered, where the total weighted completion time is a weighted average of the completion time of each task in the job[5]. For the goal of minimizing the makespan, Graham showed that a simple list scheduling algorithm can find a schedule of length within a multiplicative factor of $(2 - 1/m)$ of the optimal. This result is still the best guarantee known for this simple setting. Since then, research has sought to generalize the setting considered in two important ways: (i) to non-identical machines and (ii) to the case where communication is needed between tasks.

* suyu@caltech.edu
† xiaoqiren@google.com
‡ svardi@purdue.edu
§ adamw@caltech.edu

[5] Makespan is a special case of total weighted completion time as a dummy task with weight one can be added as the final task of the job, with all other tasks given weight zero.

Addressing these two issues has been one of the major goals of the field since Graham's initial result fifty years ago. Since that time, considerable progress has mostly been made on generalizations to heterogeneous machines. The focus has been on *(uniformly) related machines*, a model where each machine i has a speed s_i, each task j has a size w_j, and the time to run task j on machine i is w_j/s_i. Under the related machine model, a sequence of results in the 1980s and 1990s culminated in a result that showed how to use list scheduling algorithms in combination with a partitioning of machines into groups with "similar" speeds in order to achieve an $O(\log m)$-approximation algorithm for makespan [6]. This result was also extended in the same work to total weighted completion time by proposing a time-indexed linear programming technique. The extension yields an $O(\log m)$-approximation for total weighted completion time. The idea of using a *group assignment* rule to partition machines into groups of machines with similar speeds and then to assign tasks to a group is a powerful one and has shown up frequently in the years since; it recently led to a breakthrough when the idea of partitioning machines was adapted further and combined with a variation of list scheduling to obtain a $O(\log m/\log\log m)$-approximation algorithm for both makespan and total weighted completion time [7].

Despite the progress made in generalizing from identical machines to heterogeneous machines, there has been little progress toward the goal of incorporating communication delays. Machine-dependent communication delays are crucial for capturing issues such as data locality and the difference between intra-rack and inter-rack communication. We note that if communication delays are machine independent, they can simply be viewed as part of the processing time, making the problem much easier. The state-of-the-art result in the case of communication delays is [8], which studies machine-dependent communication costs in the setting of *identical machines*. In this context, a greedy algorithm called Earliest Time First (ETF) has been shown to produce schedules with a makespan bounded by $(2 - 1/m)\text{OPT}^{(i)} + C$, where $\text{OPT}^{(i)}$ is the optimal schedule length when ignoring communication time and C is the maximum amount of communication of a chain (path) in the precedence graph. However, the analysis for the case of identical machines in [8] is quite complex and it has proven difficult to generalize to the related machines setting. As a result, there has been no progress outside the context of identical machines in the thirty years since [8].

Given the challenge of designing schedulers that are approximately optimal for related machines with machine-dependent communication time, most work studying the design of scheduling policies in this context has relied on developing scheduling heuristics and evaluating these heuristics numerically, e.g., [9, 10, 11, 12, 13, 14]. For a recent survey see [15, 16] and the references therein.

Contributions. In this paper we propose a new scheduler, Generalized Earliest Time First (GETF), and prove that it computes a makespan that is at most of length $O(\log m/\log\log m)\text{OPT}^{(i)} + C$ in the case of related machines and machine-dependent communication times, where C is the amount of communication time in a chain (path) in the precedence graph. Additionally, we generalize our result to the objective of total weighted completion time and show that GETF produces a schedule $\mathcal{S}$ whose total weighted completion time is at most $O(\log m/\log\log m)$ $\text{wOPT}^{(i)} + \sum_j \omega_j C(\mathcal{S}, j)$, where $\text{wOPT}^{(i)}$ is the optimal total weighted completion time, ω_j is the weight in the objective, and $C(\mathcal{S}, j)$ is the communication requirement in a chain in the precedence graph. These two results address long-standing open problems. Note that the makespan result matches state-of-the-art bounds for the special cases (i) when there is zero communication time and (ii) when the machines are identical. In the case of total weighted completion time, no previous result exists for the case of identical machines with communication time, but the result matches the best known bound for the case with related machines and zero communication time.

The key technical advance that enables our new result is a dramatically simplified analysis of ETF in the setting of identical machines. The state-of-the-art result in this setting is [8], which is established using a long, complex argument. In contrast, the core idea in our proof of Theorem 4.1 is a short, simple proof of a *Separation Principle* which can be used to provide a novel proof of the approximation ratio for ETF in the case of identical machines. The proof is simple and general enough that it can be extended from identical machines to related machines by adapting recent advances from [7].

Related literature. In recent years, the design and optimization of large-scale general-purpose machine learning platforms has been an overarching goal, bridging many communities in both industry and academia. The emergence of platforms such as TensorFlow, PyTorch and AzureML illustrate the power of such systems to democratize tools from machine learning, making them accessible and scalable for anyone.

Since the emergence of such systems, there has been a torrent of work that seeks to optimize the scheduling and assignment of the precedence-constrained graphs in such systems. Heuristics have emerged for managing straggler tasks, e.g., [10, 17, 9, 18]; scheduling tasks with different computational properties, e.g., jobs with MapReduce-type structures [19, 20, 21, 22, 23, 24], scheduling approximation jobs [9, 25, 18], and managing communication times [16, 26]. Many of these heuristics have led to system designs that have had a significant industrial impact.

Such designs typically address the challenges associated with precedence constraints in ad hoc ways based on simplifying assumptions about the structures of the graphs. In contrast, there is a long history of analytic work seeking to design schedulers for precedence-constrained tasks with provable worst-case guarantees. As we have already mentioned, the initial results on this topic for makespan were provided by Graham, who gave a $(2 - 1/m)$-approximation algorithm based on list scheduling for $P|prec|C_{max}$ [5]. A decade later, it was shown by [27] that it is NP-hard to approximate $P|prec|C_{max}$ within a factor of $4/3$. This left a gap which has been essentially closed recently, when [28] proved that it is NP-hard to achieve an approximation factor less than 2, given the assumption of a new variant of the Unique Game Conjecture introduced by [29]. In the case of total weighted completion time objective $P|prec|\sum_j \omega_j C_j$, the negative results carry over from the makespan objective since makespan objective can be viewed as a special case of total weighted completion time objective. Moreover, under the assumption of the stronger version of the Unique Game Conjecture, it is shown in [29] that it is even hard to approximate within a factor of $2 - \epsilon$ for the problem with one machine. On the positive side, a 7-approximation was given in [30], and [31, 32] later improved it to a 4-approximation. The current best known result is a $(2 + 2\ln 2 + \epsilon)$-approximation by [7] via a time-indexed linear programming relaxation technique.

The results mentioned above all focus on identical machines with zero communication delays. When related machines are considered, the problem becomes more challenging. An early result on this topic is [6], which proposed a Speed-based List Scheduling (SLS) algorithm that obtains an approximation of $O(\log m)$ for $Q|prec|C_{max}$. A time-indexed linear programming technique has been proposed in the same work that gives a $O(\log m)$ bound for $Q|prec|\sum_j \omega_j C_j$. Recently, an improvement to $O(\log m/\log\log m)$ for both objectives was proven in [7]. The best known lower bound for the problem of related machines is from [33], which shows that it is impossible for a polynomial time algorithm to approximate the minimal makespan to any constant factor assuming the hardness of an optimization problem on k-partite graphs.

In contrast, when communication delay is considered, much less is known. To our knowledge, no approximation ratio is known for $P|prec, c_{i,j}|C_{max}$, and this open problem was noted by [34]. The only algorithm with a guaranteed worst-case performance bound in this setting is ETF [8], which provides a bound of $(2 - 1/m)\text{OPT}^{(i)} + C$ on the makespan in the case of identical machines. Prior to our paper, no algorithm with a worst-case approximation guarantee for either makespan or total weighted completion time is known for the case of related machines with communication delays, i.e., $Q|prec, c_{i,j}|C_{max}$ and $Q|prec, c_{i,j}|\sum_j \omega_j C_j$.

2 Problem formulation

We study a model that generalizes $Q|prec, c_{i,j}|\sum_j \omega_j C_j$ by including machine-dependent communication times. Our goal is to derive bounds on the total weighted completion time and the makespan, which is an important special case of the total weighted completion time that uses a particular choice of ω_j.

Specifically, we consider the task of scheduling a job made up of a set V of n tasks on a heterogeneous system composed of a set M of m machines with potentially different processing speeds and communication speeds. The tasks form a directed acyclic graph (DAG) $G = (V, E)$, in which each node j represents a task and an edge (j', j) between task j and task j' represents a precedence constraint. We interchangeably use node or task, as convenient. Precedence constraints are denoted by a partial order $\prec$ between two nodes of any edge, where $j' \prec j$ means that task j can only be scheduled after task j' completes. Let w_j represent the processing demand of task j. The amount of data to be transmitted between task j' and task j is represented by the edge weight $w_{j',j}$ of (j', j).

The system is heterogeneous in two aspects: processing speed and communication speed. For processing speed, we consider the classical *related machines* model: a machine i has speed s_i, and it takes w_j/s_i uninterrupted time units for task j to complete on machine i. Specifically, computer resources such as CPUs and GPUs have varying speeds; hence schedulers must be able to handle heterogeneous servers. The communication speed $s_{i',i}$ between any two machines i', i is heterogeneous across different machine pairs. We index the machine to which task j is assigned by $h(j)$. If $i = h(j)$ and $i' = h(j')$, then communication time between task j' and j in the DAG is $w_{j',j}/s_{i',i}$.

For simplicity, we consider a setting where the machines are fully connected to each other, so any machine can communicate with any other machine. This is without loss of generality as one can simply set the communication speed between any two disconnected machines to 0. We also assume that the DAG is connected. Again, this is without loss of generality because, otherwise, the DAG can be viewed as multiple DAGs and the same results can be applied to each. As a result, our results trivially apply to the case of multiple jobs. Additionally, our model assumes that each machine (processing unit) can process at most one task at a time, i.e., there is no *time-sharing*, and the machines are assumed to be *non-preemptive*, i.e., once a task starts on a machine, the scheduler must wait for the task to complete before assigning any new task to this machine. This is a natural assumption in many settings, as interrupting a task and transferring it to another machine can cause significant processing overhead and communication delays due to data locality, e.g., [35].

Algorithm 1 Generalized Earliest Time First (GETF)

INPUT: group assignment rule $f(\cdot)$, tie-breaking rule
OUTPUT: schedule $\mathcal{S}$ with machine assignment mapping $h(\cdot)$ and starting time mapping $t(\cdot)$

```
$R \leftarrow \{1, 2, \ldots, n\}$
while $R \neq \emptyset$ do
    $A = \{j : j \in R, \nexists j'$ s.t. $j' \in R$ and $j' \prec j\}$
    For $j \in A, t'_j =$ earliest starting time on machine $m'_j$ s.t. $m'_j \in f(j)$
    $B = \{j : j = \arg\min_{j' \in A} t(j')\}$
    Choose $j$ from $B$ to start on machine $m'_j$ with a starting time $t'_j$ based on the given tie-breaking rule
    $h(j) = m'_j, t(j) = t'_j$
    $R \leftarrow R \setminus \{j\}$
end while
```

The goal of the scheduler in our model is to minimize the *total weighted completion time* of the job, denoted by $\sum_j \omega_j C_j$, where C_j is the completion time of task j and ω_j is the weight associated with task j. We also consider the *makespan*, denoted by C_{max}, which is the time when the the final task in the DAG completes. Note that the problem we consider is an offline scheduling problem. This is a classical problem with relevance to modern ML platforms, which use batch scheduling of precedence constrained tasks in their pipelines, e.g. [2]. It is also known to be challenging. Specifically, minimizing the makespan (and hence also minimizing the total weighted completion time) of jobs with precedence constraints is known to be NP-complete [36]. Thus, we aim to design a polynomial-time algorithm that computes an *approximately* optimal schedule. We say that an algorithm is a ρ-approximation algorithm if it always produces a solution with an objective value within a factor of ρ of optimal in polynomial time.

Our main results use three important concepts. First, our results provide bounds in terms of $\text{OPT}^{(i)}$ and $\text{wOPT}^{(i)}$, which are the optimal makespan and the optimal total weighted completion time if the communication delays were zero, respectively. Note that $\text{OPT}^{(i)}$ and $\text{wOPT}^{(i)}$ are a lower bound of the corresponding objectives of the problem when communication delays are not included. Second, we provide bounds in terms of the communication time of a *terminal chain* of the schedule. A *chain* in the DAG is a sequence of immediate predecessor-successor pairs, whose first node is a node with no predecessor and last node is a leaf node with no successors. Third, we provide bounds in terms of the communication time of a terminal chain of a subset of the DAG that is naturally formed in the scheduling process. Formally, for any given schedule, a terminal chain $\mathbb{C}$ of length N can be constructed in the following fashion. We start with one of the tasks that ends last in the given schedule, denoted as c_N. Among all the immediate predecessors of node c_N, we pick one of the tasks that finishes last and define it as c_{N-1}. In such a way, we can construct a chain of tasks $c_1 \prec c_2 \prec \ldots \prec c_N$ until the first node c_1 in the chain does not have a predecessor. There may be many such terminal chains, and our results apply to any arbitrary terminal chain for the given schedule.

3 Generalized Earliest Time First (GETF) Scheduling

In this section, we introduce a new algorithm – Generalized Earliest Time First (GETF) – for scheduling tasks with precedence constraints in settings where servers have heterogeneous service rates and communication times. For GETF, we provide provable worst-case approximation guarantees for both the goal of minimizing the makespan and minimizing the total weighted completion time.

At its core, GETF is a greedy algorithm. Like ETF, it seeks to run tasks that can be started earliest, thus minimizing the idle time created by the precedence constraints in a greedy way. However, this simple heuristic does not take into account the potential difference between the service rates of different machines. For this, GETF is similar to SLS. It uses a group assignment function $f(\cdot)$ to determine sets of "similar" machines and then assigns tasks to different groups of machines. Within the groups of similar machines, GETF uses the ETF greedy allocation rule.

GETF is parameterized by a group assignment function $f(\cdot)$ and a tie-breaking rule, and proceeds in two stages. At every iteration, GETF finds a set A of all the tasks that are ready to process and are not yet scheduled. For every task in A, GETF calculates the earliest starting time if it was only allowed to schedule on machines in the assigned group. Then, GETF computes B, the set of tasks in A with the earliest starting times, and chooses one of the tasks to process on a machine based on the tie-breaking rule. The pseudocode for GETF is presented in Algorithm 1 and Figure 1 in section 3.3 illustrates the operation of GETF on a simple example (Example 1).

GETF can be instantiated with different group assignment and tie-breaking rules. To understand how these rules work, consider a situation where the m machines are divided into K groups $M_1, M_2, \ldots, M_K$ by a group assignment rule.

Let $f(j)$ denote the group of machines to which task j can be assigned, $j = 1, \dots, n$. Given this notation, a schedule under GETF consists of two mappings: a mapping $h(\cdot)$ from each task to its assigned machine and a mapping $t(\cdot)$ from each task to its starting time. Further, for any schedule with $h(\cdot)$ produced by GETF, $h(\cdot)$ of the produced schedule should be consistent with group assignment function $f(\cdot)$, i.e., $h(j) \in f(j)$ for each task j.

The choice of the group assignment rule has a significant impact on the performance of GETF. Indeed, different group assignment functions are used for the goals of minimizing the makespan and total weighted completion time. While our results hold for any tie-breaking rule, different tie-breaking rules could provide meaningful improvements in real-world workloads. As it could be helpful to keep a specific tie-breaking rule in mind while considering the algorithm and proofs, the reader may find it helpful to consider random tie-breaking. Our technical results are based on the specific group assignment functions described in the following subsections.

3.1 A Group Assignment Rule for Makespan

The group assignment rule $f_{\mathsf{mksp}}(\cdot)$ for the goal of minimizing the makespan that we focus on is adapted from SLS, which is designed for the setting *without* communication time. Specifically, machines of similar speeds are grouped together as follows.

First, all the machines with speed less than a $\frac{1}{m}$ fraction of the speed of the fastest machine are discarded. Then, the remaining machines are divided into K groups $M_1, M_2, \dots, M_K$ where $K = \lceil \log_\gamma m \rceil$, $\gamma = \log m / \log \log m$. Note that $K = O(\log m / \log \log m)$. Given the removal of the slowest machines, we can assume that any remaining machine has speed within a factor of $\frac{1}{m}$ of the fastest machine. Without loss of generality, we assume the speed of the fastest machine is m and the group M_k contains machines with speeds in range $[\gamma^{k-1}, \gamma^k)$.

It may seem strange that some machines are discarded, but note that the total speed of discarded machines is not bigger than the speed of the fastest machine. So, if we consider the scheduling problem with zero communication time, removing these machines at most doubles the makespan in the worst case.

After dividing machines into K groups in the preprocessing step, we need to assign the machines. This step is more involved than the division. The design of the group assignment rule $f_{\mathsf{mksp}}(\cdot)$ is based on the solution of a linear program (LP), which is a relaxed version of the following mixed integer linear program (MILP).

$$\min_{x_{i,j}, C_j, T} \quad T$$

$$\sum_i x_{i,j} = 1 \qquad \forall j \tag{1a}$$

$$w_j \sum_i \frac{x_{i,j}}{s_i} \le C_j \qquad \forall j \tag{1b}$$

$$C_{j'} + w_j \sum_i \frac{x_{i,j}}{s_i} \le C_j \qquad j' \prec j \tag{1c}$$

$$\frac{1}{s_i} \sum_j w_j x_{i,j} \le T \qquad \forall i \tag{1d}$$

$$C_j \le T \qquad \forall j \tag{1e}$$

$$x_{i,j} \in \{0, 1\} \quad \forall i, j \tag{1f}$$

While the MILP is only designed to produce a group assignment rule, its optimal solution does not necessarily provide a feasible schedule. In the MILP, $x_{i,j} = 1$ if task j is assigned to machine i; otherwise $x_{i,j} = 0$. For each task j, C_j denotes the completion time of task j. Constraint (1a) ensures that every task is processed on some machine. For any task j, processing time $w_j \sum_i \frac{x_{i,j}}{s_i}$ is bounded by its completion time as in constraint (1b). Constraint (1c) enforces the precedence constraints between any predecessor-successor pair (j', j). Constraint (1d) guarantees that the total load assigned to machine i is $w_j \sum_i \frac{x_{i,j}}{s_i}$ and it should not be greater than the makespan. Finally, constraint (1e) states that the makespan should not be smaller than the completion time of any task.

Since we cannot solve the MILP efficiently, we relax it to form an LP by replacing constraint (1f) with $x_{i,j} \ge 0$. Let x^*, C^*, T^* denote the optimal solution of this LP. Note that T^* provides a lower bound on $\mathrm{OPT}^{(i)}$, the optimal makespan for the same problem with zero communication time.

For a set $M_k \subseteq M$ of machines, let $s(M_k)$ denote the total speed of machines in M_k, i.e.,

$$s(M_k) = \sum_{i \in M_k} s_i.$$

Define $x^*_{M_k,j}$ as the total fraction of task j assigned to machines in set M_k:

$$x^*_{M_k,j} = \sum_{i \in M_k} x^*_{i,j}.$$

For any task j, define ℓ_j as the largest group index such that at least half of the tasks are fractionally assigned to machines in groups $M_\ell, \ldots, M_K$:

$$\ell_j = \max_{\ell} \ell \quad \text{s.t.} \quad \sum_{k=\ell}^{K} x^*_{M_k,j} \geq \frac{1}{2}.$$

We note that any choice of constant above works for the purpose of our worst case analysis of GETF, but the choice can potentially have an impact on its empirical performance. Thus the choice of the parameter should be further optimized when applied in practice. Each task j is assigned to the group $f_{\mathsf{mksp}}(j)$ that maximizes the total speed of machines in that group among candidates $M_{l_j}, \ldots, M_K$, i.e.,

$$f_{\mathsf{mksp}}(j) = \underset{M_k : \ell_j \leq k \leq K}{\arg\max} \quad s(M_k).$$

3.2 A Group Assignment Rule for Total Weighted Completion Time

The group assignment rule $f_{\mathsf{twct}}(\cdot)$ for the goal of minimizing the total weighted completion time is similar in spirit to $f_{\mathsf{mksp}}(\cdot)$ but is based on modified solutions of a different LP. We divide machines into groups in the same way as in Section 3.1. Without loss of generality, we assume that $\frac{w_j}{s_i} \geq 1$ for any task j to be processed on any machine i. Thus, we can divide the time horizon into the following time-indexed intervals of possible task completion times: $[1, 2], (2, 4], (4, 8], \ldots, (\tau_{Q-1}, \tau_Q]$ where $Q = \log\left(\sum_j \frac{w_j}{\min_i s_i}\right)$ and $\tau_q = 2^q$ for $0 \leq q \leq Q$. Then, the MILP that forms the basis for the group assignment rule can be formulated as follows:

$$\min_{x_{i,j,q}, C_j} \quad \sum_j \omega_j C_j$$

$$\sum_i \sum_q x_{i,j,q} = 1 \qquad \forall j \tag{2a}$$

$$w_j \sum_i \frac{1}{s_i} \sum_q x_{i,j,q} \leq C_j \qquad \forall j \tag{2b}$$

$$C_{j'} + w_j \sum_i \frac{1}{s_i} \sum_q x_{i,j,q} \leq C_j \qquad j' \prec j \tag{2c}$$

$$\sum_{t=1}^{q} \sum_i x_{i,j,t} - \sum_{t=1}^{q} \sum_i x_{i,j',t} \leq 0 \qquad \forall q, j' \prec j \tag{2d}$$

$$\sum_q \tau_{q-1} \sum_i x_{i,j,q} < C_j \qquad \forall j \tag{2e}$$

$$\frac{1}{s_i} \sum_j w_j \sum_{t=1}^{q} x_{i,j,t} \leq \tau_q \qquad \forall i, q \tag{2f}$$

$$x_{i,j,q} \in \{0, 1\} \qquad \forall i, j, q \tag{2g}$$

Again, the MILP is only designed to find a group assignment rule and thus its optimal solution does not necessarily produce a feasible schedule. Here, $x_{i,j,q} = 1$ if task j is assigned to machine i and it completes in the qth interval $(\tau_{q-1}, \tau_q]$. For each task j, C_j denotes the completion time of task j and ω_j represents its weight in the objective of total weighted completion time. Constraint (2a) enforces that each task will be assigned to some machine. Constraint (2b) guarantees that the completion time of a task is not smaller than its processing time. Constraints (2c) and (2d) together enforce the precedence constraint for every predecessor-successor pair. Constraint (2e) guarantees that the completion time of task j is not smaller than the left boundary of the qth interval $(\tau_{q-1}, \tau_q]$. The total load assigned to machine i up to qth interval is $\frac{1}{s_i}\sum_j w_j \sum_{t=1}^{q} x_{i,j,t}$, and it should not be greater than the upper bound τ_q as enforced in constraint (2f).

To define the group allocation rule, we relax constraint (2g) to form an LP. As in the previous section, let x^*, C^* denote the optimal solution for this LP. Note that $\sum_j \omega_j C_j^*$ provides a lower bound for wOPT$^{(i)}$. For any task j, define $q(j)$ as the the minimum value of q such that both $\sum_{t=1}^{q}\sum_i x^*_{i,j,t} \geq \frac{1}{2}$ and $C_j^* \leq 2^q$ are satisfied. Intuitively, $q(j)$ can be viewed as a rough estimate of the completion time of task j. Define $\alpha(j)$ as the total fraction of task j over any machine in the first $q(j)$ intervals with respect to solution x^*:

$$\alpha_j = \sum_{t=1}^{q(j)}\sum_i x^*_{i,j,t}.$$

We construct a set of feasible solutions $\tilde{x}$ based on the optimal solution x^* for the LP:

$$\tilde{x}_{i,j} = \sum_{q=1}^{q(j)} \frac{x^*_{i,j,q}}{\alpha_j} \quad \forall i,j. \tag{3}$$

Notice that the group assignment rule $f_{\mathsf{twct}}(\cdot)$ is of the same form as $f_{\mathsf{mksp}}(\cdot)$, with $\tilde{x}$ replacing x^*. For task j, define $\tilde{\ell}_j$ as before but with respect to $\tilde{x}$ instead of x^*:

$$\tilde{\ell}_j = \max_{\ell} \ell \quad \text{s.t.} \quad \sum_{k=\ell}^{K} \tilde{x}_{M_k,j} \geq \frac{1}{2}.$$

The group assignment rule $f_{\mathsf{twct}}(\cdot)$ for the goal of minimizing the total weighted completion time follows as below:

$$f_{\mathsf{twct}}(j) = \mathop{\arg\max}_{M_k:\tilde{\ell}_j \leq k \leq K} \quad s(M_k).$$

3.3 A Comparison of GETF and SLS

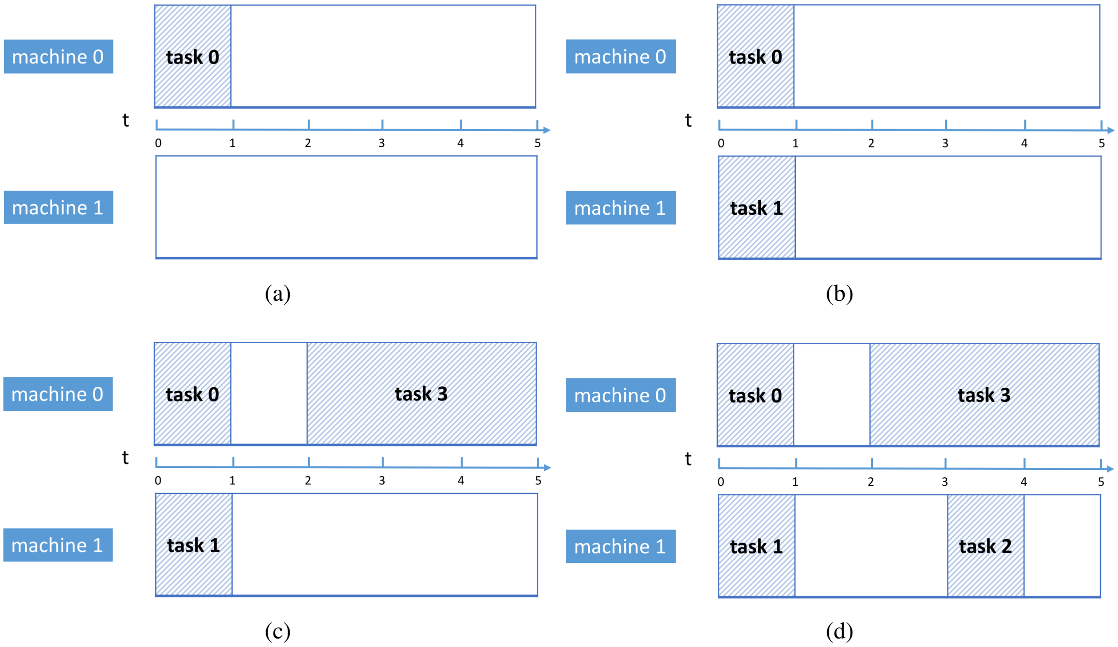

Figure 1: An illustration of GETF running on Example 1. (a)-(d) show the first four iterations.

The description of GETF above highlights that it combines the greedy heuristic of ETF with the speed-based assignment heuristic of SLS. This enables GETF to provide guarantees for settings with both heterogeneous processing rates and

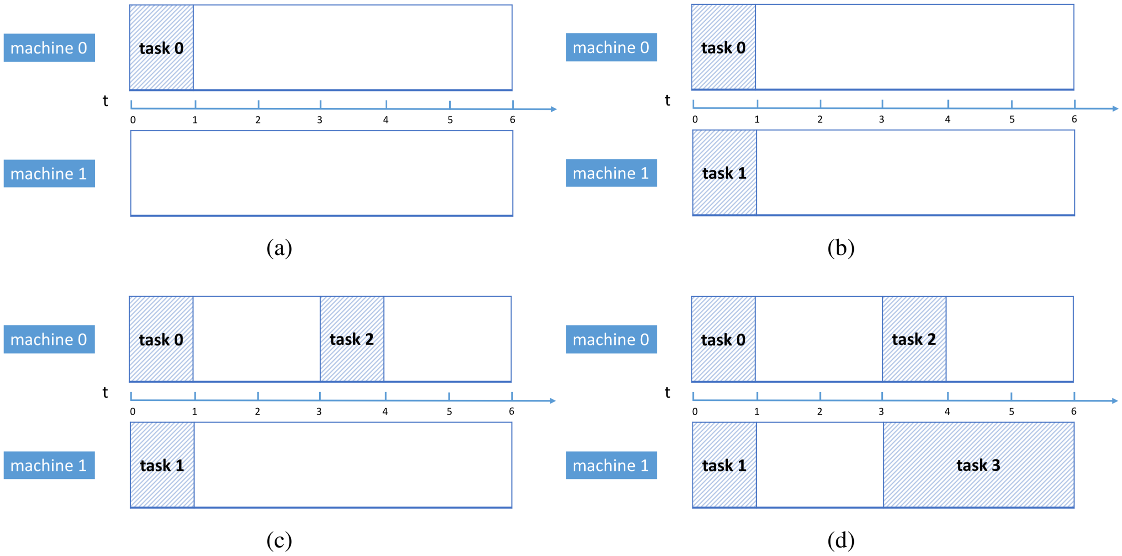

Figure 2: An illustration of SLS running on Example 1. (a)-(d) show the first four iterations.

communication delays. In contrast, SLS does not provide guarantees in settings with communication time. This is a result of the fact that SLS is based on list scheduling and does not always schedule the earliest task first, thus making it impossible to bound the overall idle time in between tasks.

To illustrate the difference between GETF and SLS, we provide a simple example of scheduling a job made up of four tasks.

Example 1. *We consider a job made up of four tasks,* $0, 1, 2, 3$ *with processing demands* $1, 1, 1,$ *and* 3 *that are to be scheduled on a set of two identical machines with the same processing speed equal to* 1*. The weight for the edges in the graph are listed as below:* $w_{0,2} = w_{0,3} = w_{1,2} = 2, w_{1,3} = 1$*. We assume* $s_{i,j} = 1$ *for* $i \neq j$*; otherwise* $s_{i,i} = 2$ *for* $i = 0, 1$*.*

The schedules of GETF and SLS are illustrated in Figures 1 and 2. Note that, since the servers are identical, the group assignment rule does not play a role in these examples. Given a priority list $(0, 1, 2, 3)$*, a possible schedule produced by SLS puts tasks* 0 *and* 2 *on machine* 0 *and assigns the rest of tasks to machine* 1 *as demonstrated in Figure 2. A terminal chain for the given schedule is task* 1 *followed by task* 3*, and the idle time of length* 2 *between the end of task* 1 *and the start of task* 3 *on machine* 1 *is not bounded by the communication time between task* 1 *and* 3*. In contrast, task* 3 *starts earlier on machine* 0 *in a schedule produced by GETF, see Figure 1. List scheduling does not always schedule the earliest task at each step, thus making the idle time on machine* 1 *not necessarily bounded by communication time between task* 1 *and task* 3*. Our proofs in Section 4.1 highlight that maintaining a tight bound on the communication time between tasks is crucial to achieving a good approximation ratio in settings with machine-dependent communication time.*

4 Results

Our main results bound the approximation ratio of GETF in settings with related machines and heterogeneous communication time for the goals of minimizing the makespan and minimizing the total weighted completion time.

4.1 Makespan

In the case of minimizing the makespan, our main result provides a bound in terms of the communication time of a terminal chain of the schedule. Specifically, let $\mathbb{C} : c_1 \prec c_2 \prec \ldots \prec c_N$ be a terminal chain for the schedule and define C as the communication time over such a chain in the worst case, i.e.

$$C = \sum_{j=2}^{N} \frac{w_{c_{j-1},c_j}}{\bar{s}(c_{j-1}, c_j)},$$

where $\bar{s}(c_{j-1}, c_j)$ is defined as the slowest speed between $h(c_{j-1})$, the machine assigned to c_{j-1} and any machine in the group $f(c_j)$, i.e.,

$$\bar{s}(c_{j-1}, c_j) = \min_{i \in f(c_j)} s_{h(c_{j-1}),i}.$$

Note that C can be computed efficiently and minimized over all the terminal chains using dynamic programming and that the tie-breaking rule can have an impact on C due to its impact on terminal chains.

Theorem 4.1. *For any schedule $\mathcal{S}$ produced by GETF with group assignment rule, $f_{\mathsf{mksp}}(\cdot)$*

$$C_{max}(\mathcal{S}) \leq O(\log m / \log\log m)\textit{OPT}^{(i)} + C,$$

where OPT$^{(i)}$ is the optimal schedule length obtained if communication time for all pairs were zero.

Theorem 4.1 represents the first result for makespan in the setting of related machines and heterogeneous communication time, addressing a problem that has been open since ETF was introduced for identical machines thirty years ago. Additionally, it matches the state-of-the art results for the case without communication time, where the best known approximation ratio is $O(\log m / \log\log m)$ [7], and the case with communication time but identical machines, where the best known approximation ratio is $(2 - \frac{1}{m})\text{OPT}^{(i)} + C$ [8].

Concretely, in the special case of identical machines, the group assignment rule $f_{\mathsf{mksp}}(\cdot)$ is no longer required when implementing GETF since all machines share the same speed and so there is only one group of machines. Thus, GETF reduces to ETF. The theorem makes use of C' which is defined as

$$C' = \frac{1}{m} \sum_{j=2}^{N} \sum_{i=1}^{m} \frac{w_{c_{j-1},c_j}}{s_{h(c_{j-1}),i}}.$$

Note that C' differs from C since it is an average over the terminal chain. The result we obtain in this case is the following, which matches the current state-of-the-art result of [8].

Proposition 4.2. *Consider a setting with m identical machines. For any schedule $\mathcal{S}$ produced by GETF,*

$$C_{max}(\mathcal{S}) \leq \left(2 - \frac{1}{m}\right) \textit{OPT}^{(i)} + C',$$

where OPT$^{(i)}$ is the optimal schedule length obtained if communication time for all pairs were zero.

4.2 Total Weighted Completion Time

Similarly to the makespan case, we provide a bound with respect to the communication time of chains. However, since total weighted completion time depends on the completion time of every task (instead of just one task as in the case of makespan), the communication time of terminal chains of many subsets of the DAG show up in the bound. More formally, assume that the tasks are indexed with respect to their order in the schedule determined by GETF, denoted by $\mathcal{S}$. At iteration j, task j is to be scheduled. Let $G(\mathcal{S}, j)$ denote a DAG formed by a set of the tasks that have been scheduled so far and the corresponding edges within these tasks. Define $\mathcal{S}(j)$ to be a subset of the given schedule $\mathcal{S}$ up to iteration j, i.e., it is a schedule for DAG $G(\mathcal{S}, j)$. This definition ensures that task j is one of the tasks that ends last in the schedule $\mathcal{S}(j)$. Now, let $\mathbb{C}(\mathcal{S}, j) : c_1 \prec c_2 \prec \cdots \prec c_{N_j}$ be a terminal chain that ends with task $j = c_{N_j}$ in the schedule $\mathcal{S}(j)$, and define $C(\mathcal{S}, j)$ as the communication time over such a chain in the worst case, i.e.,

$$C(\mathcal{S}, j) = \sum_{j'=2}^{N_j} \frac{w_{c_{j'-1},c_{j'}}}{\bar{s}(c_{j'-1}, c_{j'})}.$$

This definition of $C(\mathcal{S}, j)$ generalizes the notion of C used in Theorem 4.1 for makespan and plays a similar role in the theorem below.

Theorem 4.3. *For any schedule $\mathcal{S}$ produced by GETF with group assignment rule $f_{\mathsf{twct}}(\cdot)$,*

$$\sum_j \omega_j C_j \leq O(\log m / \log\log m) w\textit{OPT}^{(i)} + \sum_j \omega_j C(\mathcal{S}, j),$$

where wOPT$^{(i)}$ is the optimal total weighted completion time obtained if communication time for all pairs was zero.

Theorem 4.3 is the first result on total weighted completion time for the setting of related machines with heterogeneous communication time and it matches the bounds in cases where previous results exist. In particular, if the weights are chosen so as to recover makespan, then the bound matches that of Theorem 4.1. Similarly, results for identical machines can be recovered as done in the case of makespan. However, note that the group assignment rule used for GETF here is different than that in Theorem 4.1. The rule used in Theorem 4.3 applies more generally but, while both group assignment rules yield the same worst-case performance bound for makespan, we expect that the rule used in Theorem 4.1 will lead to a smaller makespan in most practical settings as it is designed for the purpose of minimizing the makespan.

5 Proofs

In this section, we present our proofs of Theorems 4.1 and 4.3. The general form of both arguments is similar; however, the case of total weighted completion time is more involved. The first step of our argument is to show a general upper bound, which is valid for GETF regardless of choices of group assignment function $f(\cdot)$, and tie-breaking rule. This *Separation Principle* can be used to easily establish the result for makespan in the case of identical machines (Proposition 4.2), and represents a significant simplification compared to existing proofs of that result in the literature. We then tighten the general bound by taking advantage of the choices of $f(\cdot)$ described in Section 3 for makespan and total weighted completion time. Finally, we establish a connection between the makespan and total weighted completion time in the same settings by introducing a time-indexed LP that enables us to bound the total weighted completion time.

5.1 A Separation Principle

The Separation Principle presented here is a key component of our proof of Theorem 4.1. The core of nearly all proofs in this area is the construction of a chain, which is then used to bound the overall makespan. This idea goes back to the first list scheduling algorithms proposed by [5]. The key to our argument is to bound the amount of communication time between any predecessor-successor pairs in a terminal chain. However, as we discuss in Section 3, it is not possible to do this under list scheduling algorithms.

Our approach also differs considerably from the approach used to study ETF in [8], where the authors divide $[0, C_{max}]$ into two sets of time intervals, one for the time when all the machines are busy and the other that one chain covers. Extending this approach to related machines does not appear possible. In contrast, in our argument, the construction of a terminal chain is simple and so we can identify the set of time intervals between tasks in the terminal chain and take advantage of the greedy nature of GETF to bound these times directly.

A key feature of the the Separation Principle below is that it separates the analysis of the terminal chain from the analysis of the group assignment rule, which provides another valuable simplification of the previous proof approaches.

Theorem 5.1 (Separation Principle)**.** *For any choice of group assignment function $f(\cdot)$ and tie-breaking rule, GETF produces a schedule $\mathcal{S}$ of makespan*

$$C_{max}(\mathcal{S}) \leq P + \sum_{k=1}^{K} D_k + C,$$

where

$$P = \sum_{c_j \in \mathbb{C}} \frac{w_{c_j}}{s_{h(c_j)}},$$

$$D_k = \frac{\sum_{j:k \in f(j)} w_j}{s(M_k)},$$

$$C = \sum_{j=1}^{N-1} \frac{w_{c_j, c_{j+1}}}{\bar{s}(c_j, c_{j+1})}.$$

Note that the upper bound in this result is valid regardless of the choice of group assignment rule and tie-breaking rule. P is the sum of processing times along a terminal chain and D_k can be viewed as total load assigned to machines in group M_k. Both P and $D_k, k = 1, 2, \ldots, K$, are not dependent on the communication constraint, which enables us to take advantage of any good choice of group assignment rule $f(\cdot)$ for general DAG scheduling, even in the case of zero communication time.

Proof. Our proof proceeds in four steps:

(i) Define a terminal chain $\mathbb{C}$. Recall that a chain $\mathbb{C}$, $c_1 \prec c_2 \prec \ldots \prec c_N$ is a terminal chain when task c_N completes at the end of the overall schedule.

(ii) Partition the overall makespan into $K+1$ parts. The idea of this step is to decouple $[0, C_{max}]$ into one part where the tasks in the terminal chain are being processed and K other parts associated with each machine group. Dependent on the choices of group assignment rule, we can further bound these $K+1$ parts.

(iii) Bound the idle time in between tasks. The greedy nature of GETF makes it possible to bound the length of the idle time intervals between tasks by communication delays of task pairs.

(iv) Combine (ii) and (iii) to bound the overall makespan in terms of the communication time of the terminal chain.

(i) *Define a terminal chain* $\mathbb{C}$. To find a terminal chain of length N, we start with one of the tasks that ends last, denoted as c_N. According to the definition of $h(\cdot)$ and $t(\cdot)$, task c_N is assigned to machine $h(c_N)$ in group $f(c_N)$ with a starting time $t(c_N)$. Among all the immediate predecessors of task c_N, we pick one of the tasks that finishes last and define it as c_{N-1}. In such a fashion, we construct a chain $\mathbb{C}$ of tasks $c_1 \prec c_2 \prec \ldots \prec c_N$ of length N such that c_1 does not have any predecessor.

(ii) *Partition* $[0, C_{max}]$ *into* $K+1$ *parts,* $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_K$. Recall that $K = O(\log m / \log\log m)$ is the number of groups for machines by the group assignment rule as we describe in the previous section. Let $\mathcal{T}_0$ denote the union of the time intervals during which tasks of chain $\mathbb{C}$ are being processed. Consider the time interval between the end of task c_{j-1} and the start of task c_j for $j = 2, 3, \ldots, N$, and assign it to $\mathcal{T}_k$ where $M_k = f(c_j)$. As a set of time intervals, $\mathcal{T}_k$ can be possibly empty or have more than one time interval. Essentially, $\mathcal{T}_k$ is a set of time intervals that tasks in the terminal chain $\mathbb{C}$ assigned to machines in group M_k have to wait before being processed. In such a fashion, we define $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K$ since $f(\cdot)$ maps each task to one of the K machine groups. The length of the union of $\mathcal{T}_i$ for $i = 0, 1, \ldots, K$ is the makespan.

(iii) *Bound the idle time in between tasks.* Consider a task c_j assigned to machine $h(c_j)$. For each machine $i \in f(c_j)$, let $E(c_{j-1}, c_j, i)$ denote a union of disjoint empty time intervals on machine i between the end time of task c_{j-1} and the start time of task c_j. Between the end time of task c_{j-1} and the start time of task c_j, there can be multiple tasks being processed on machine i in serial, possibly resulting in more than one idle time interval on machine i during that time interval $E(c_{j-1}, c_j, i)$. Precedence constraints between task pairs can also possibly make a successor wait before it gets started. Regardless of the reason for idle time between tasks, each task can not possibly start earlier on any machine in the assigned group due to the greedy feature of GETF. Thus the length of $E(c_{j-1}, c_j, i)$ is bounded above by the communication time between task c_{j-1} and task c_j, i.e.,

$$|E(c_{j-1}, c_j, i)| \leq \frac{w_{c_{j-1}, c_j}}{s_{h(c_{j-1}), i}} \quad \forall i \in f(c_j).$$

This is true because if it were not the case then task c_j could have started earlier on machine i. Note that the end time of task c_j could possibly be earlier if it were allowed to be scheduled on a faster machine with a slightly bigger communication delay, since the processing speeds of machines in the same group vary.

Let e_i be idle time on machine i in group M_k during the time interval $\mathcal{T}_k$, and let $\bar{e}_k$ be maximum idle time on any machine in group M_k during the time intervals $\mathcal{T}_k$, i.e., $e_i \leq \bar{e}_k$ for all $i \in M_k$. Thus,

$$\begin{aligned}\sum_{k=1}^{K} \bar{e}_k &\leq \sum_{j=2}^{N} \frac{w_{c_{j-1}, c_j}}{\min_{i' \in f(c_j)} s_{h(c_{j-1}), i'}} \\ &\leq \sum_{j=2}^{N} \frac{w_{c_{j-1}, c_j}}{\bar{s}(c_{j-1}, c_j)}. \end{aligned} \tag{4}$$

(iv) *Bound the makespan.* For $1 \leq k \leq K$, the total speed of machines in group M_k is

$$s(M_k) = \sum_{i \in M_k} s_i.$$

Denote the total length of the intervals in $\mathcal{T}_k$ by t_k. There must be at least a sum of $(t_k - e_i)\, s_i$ units of processing done on each machine i in group M_k during the time intervals $\mathcal{T}_k$. Thus for $1 \leq k \leq K$,

$$\sum_{i \in M_k} (t_k - e_i)\, s_i \leq \sum_{j : f(j) = M_k} w_j.$$

Therefore,

$$t_k \leq \frac{\sum_{j:f(j)=M_k} w_j}{s(M_k)} + \frac{\sum_{i\in M_k} e_i s_i}{s(M_k)}. \tag{5}$$

We now bound $\mathcal{C}_{max}$:

$$\begin{aligned}
\mathcal{C}_{max} = & \sum_{k=1}^{K} t_k + t_0 \\
\leq & \sum_{k=1}^{K} \left(\frac{\sum_{j:f(j)=k} w_j}{s(M_k)} + \frac{\sum_{i\in M_k} e_i s_i}{s(M_k)} \right) + \\
& \sum_{c_j \in \mathbb{C}} \frac{w_{c_j}}{s_{h(c_j)}} \qquad (6a) \\
\leq & \; P + \sum_{k=1}^{K} D_k + \sum_{k=1}^{K} \bar{e}_k \frac{\sum_{i\in M_k} s_i}{s(M_k)} \\
= & \; P + \sum_{k=1}^{K} D_k + \sum_{k=1}^{K} \bar{e}_k \\
\leq & \; P + \sum_{k=1}^{K} D_k + C, \qquad (6b)
\end{aligned}$$

where (6a) is due to (5) and (6b) is due to (4). □

5.2 Proof of Theorem 4.1

In order to apply the Separation Principle to prove Theorem 4.1, we need to prove bounds on P and $\sum_{k=1}^{K} D_k$ in the case of the group assignment rule defined in Section 3. For this, we consider the scheduling problem with zero communication time. Note that the design of group assignment function $f_{\mathsf{mksp}}(\cdot)$ is based on the optimal solution x^* of the relaxed LP for a scheduling problem with zero communication time, hence the upper bounds for both P and $\sum_{k=1}^{K} D_k$ are associated with the optimal objective of the relaxed LP in the setting with zero communication time as well.

The bounds of P and $\sum_{k=1}^{K} D_k$ are given in the following two lemmas, which are adapted from results in [7]. Theorem 4.1 follows directly from these two lemmas, the Separation Principle, and the fact that $T^* \leq \text{OPT}^{(i)}$, where T^* is the optimal solution to the LP.

Lemma 5.2. $P \leq 2\gamma T^*$.

Proof. Recall that $x^*_{M',j} = \sum_{i\in M'} x^*_{i,j}$ and ℓ_j as the largest group index such that at least more than half of tasks are assigned to machines in groups $M_\ell, \dots, M_K$. For every task j and any machine $i \in f(j)$, by definition of the largest index ℓ_j,

$$\sum_{k=1}^{\ell_j} x^*_{M_k,j} > \frac{1}{2}. \tag{7}$$

Thus,

$$\begin{aligned}
\sum_{i'\in M} \frac{x^*_{i',j}}{s_{i'}} &= \sum_{k=1}^{K} \sum_{i'\in M_k} \frac{x^*_{i',j}}{s_{i'}} \qquad (8a) \\
&\geq \sum_{k=1}^{\ell_j} \sum_{i'\in M_k} \frac{x^*_{i',j}}{s_{i'}} \\
&\geq \frac{1}{2}\gamma^{-\ell_j} \qquad (8b) \\
&\geq \frac{1}{2\gamma s_i}, \qquad (8c)
\end{aligned}$$

where (8b) is due to (7) and the fact that processing speed of machine i' in group M_k for task j is at most γ^{ℓ_j} for $k \leq \ell_j$, and (8c) is due to the fact that processing speed of machine i in group $f(j)$, whose group index is not smaller than ℓ_j, is at least $\gamma^{\ell_j - 1}$. Using this, we can bound P as follows:

$$\begin{aligned} P &= \sum_{c_j \in \mathbb{C}} \frac{w_{c_j}}{s_{h(c_j)}} \\ &\leq 2\gamma \sum_{c_j \in \mathbb{C}} w_{c_j} \sum_{i' \in M} \frac{x^*_{i',c_j}}{s_{i'}} && \text{(9a)} \\ &\leq 2\gamma \sum_{c_j \in \mathbb{C}} C^*_{c_j} && \text{(9b)} \\ &\leq 2\gamma T^*, && \text{(9c)} \end{aligned}$$

where (9a) is due to (8), (9b) is due to constraint (1d) of the LP and (9c) is due to constraint (1c) of the LP. □

Lemma 5.3. $\sum_{k=1}^{K} D_k \leq 2KT^*$.

Proof. For any task j, by definition of ℓ_j, $\sum_{k=\ell_j}^{K} x^*_{M_k,j} \geq \frac{1}{2}$. Thus,

$$\begin{aligned} \frac{1}{2s(f(j))} &\leq \sum_{k=\ell_j}^{K} \frac{x^*_{M_k,j}}{s(f(j))} \\ &\leq \sum_{k=\ell_j}^{K} \frac{x^*_{M_k,j}}{s(M_k)} && \text{(10)} \\ &\leq \sum_{k=1}^{K} \frac{x^*_{M_k,j}}{s(M_k)}. \end{aligned}$$

Inequality (10) is due to the fact that the assigned group $f(j)$ maximizes the total speeds of machines in that group among the candidates $M_{\ell_j}, \ldots, M_K$. Thus,

$$\begin{aligned} \sum_{k=1}^{K} D_k = \sum_{k=1}^{K} \frac{\sum_{j: f(j)=M_k} w_j}{s(M_k)} &= \sum_{j \in V} \frac{w_j}{s(f(j))} \\ &\leq 2 \sum_{j \in V} w_j \sum_{k=1}^{K} \frac{x^*_{M_k,j}}{s(M_k)} \\ &= 2 \sum_{k=1}^{K} \frac{1}{s(M_k)} \sum_{j \in V} w_j x^*_{M_k,j} \\ &\leq 2 \sum_{k=1}^{K} T^* && \text{(11)} \\ &= 2KT^*. \end{aligned}$$

The total load assigned to machines in group M_k is $\sum_{j \in V} w_j x^*_{M_k,j}$ while its total speed is $S(M_k)$. Summing over machines in group M_k on both sides for constraint (1d) leads to (11). □

5.3 Proof of Proposition 4.2

We now show how the Separation Principle can be used to provide a new, simpler proof of the state-of-the-art approximation ratio of ETF in the case of identical machines. Recall that the group assignment function is not required for GETF in this case.

To prove Proposition 4.2, we use the same approach as we used for proving the Separation Principle. However, we can tighten the analysis in the final step of the argument. Specifically, the proof can be broken into three steps, instead of four:

(i) Define a terminal chain $\mathbb{C}$. This step is identical to the definition of a terminal chain in the proof of the Separation Principle.

(ii) Bound the idle time in between tasks. As the machines are identical in terms of processing speed, communication speed between different machine pairs are still heterogeneous due to the possible geolocations of machines.

(iii) Combine (i) and (ii) to bound the overall makespan in terms of the communication time of the terminal chain.

Compared with the proof of the Separation Principle, Step (i) defines a terminal chain in the exactly same way. In Step (ii), bounding the idle time in the case of identical machines is also similar. Step (iii) requires more work. Here, we further tighten the bound by eliminating the processing time of the terminal chain to improve the constant factor.

(i) *Define a terminal chain* $\mathbb{C}$. This step is identical to the definition of a terminal chain in the proof of the Separation Principle.

(ii) *Bound the idle time in between tasks.* Let $I(c_{j-1}, c_j)$ be the time interval between the end time of task c_{j-1} and the start time of c_i for $j = 2, 3, \ldots, N$. As we explained in the Separation Principle, there can possibly be multiple idle time intervals on a machine during the time interval $I(c_{j-1}, c_j)$. For each machine $i \in M$, define $E(c_{j-1}, c_j, i)$ as a union of disjoint empty time intervals on machine i during the time interval $I(c_{j-1}, c_j)$. For any machine i, the length of $E(c_{j-1}, c_j, i)$ is bounded above by the communication time between task c_{j-1} and task c_j, i.e.,

$$|E(c_{j-1}, c_j, i)| \leq \frac{w_{c_{j-1},c_j}}{s_{h(c_{j-1}),i}} \quad \forall i \in M, j = 2, 3, \ldots, N.$$

Otherwise task c_j could have started earlier on machine i.

(iii) *Bound the makespan.* During the time intervals $I(c_{j-1}, c_j)$ for $j = 2, 3, \ldots, N$, there must be at least $\sum_{j=2}^{N} \sum_{i=1}^{m} (|I(c_{j-1}, j_i)| - |E(c_{j-1}, c_j, i)|)$ processing units done, and it is bounded by a sum of the processing units for all the tasks except those in the terminal chain. This leads to the following bound:

$$\sum_{j=2}^{N} \sum_{i=1}^{m} (|I(c_{j-1}, c_j)| - |E(c_{j-1}, c_j, i)|) \leq \sum_{j=1}^{n} w_j - \sum_{j=1}^{N} w_{c_j}. \tag{12a}$$

Finally, applying (12a), we have

$$\begin{aligned} \mathcal{C}_{max} =& \sum_{j=2}^{N} |I(c_{j-1}, c_j)| + \sum_{j=1}^{N} w_{c_j} \\ \leq& \frac{1}{m} \sum_{j=1}^{n} w_j + \frac{m-1}{m} \sum_{j=1}^{N} w_{c_j} + \\ & \frac{1}{m} \sum_{j=2}^{N} \sum_{i=1}^{m} |E(c_{j-1}, c_j, i)| \\ \leq& \left(2 - \frac{1}{m}\right) \mathrm{opt}^{(i)} + C'. \end{aligned} \tag{13a}$$

The total processing time $\sum_{j=1}^{n} w_j$ divided by the number of machines m is the smallest possible makespan, i.e., $\frac{1}{m} \sum_{j=1}^{n} w_j \leq \mathrm{OPT}^{(i)}$. At the same time, the makespan of any schedule should at least cover the processing time of any chain $\mathbb{C}$ in the DAG. These two facts lead to the last inequality (13a).

5.4 Proof of Theorem 4.3

To establish the bound on the total weighted completion time for the group assignment rule $f_{\mathsf{twct}}(\cdot)$, we first apply the Separation Principle to separate the requirements on communication and processing times. Second, we break the tasks into subsets based on the task completion times and, for each subset, we form an LP for those tasks alone. For each such LP, we construct a feasible solution $\tilde{x}, \tilde{C}$ and $\tilde{T}$ to bound processing time of the tasks. The feasibility of $\tilde{x}, \tilde{C}$ and $\tilde{T}$ enables us to take advantage of Lemmas 5.2 and 5.3 with only a loss of an additional constant factor.

Given a schedule $\mathcal{S}$ for a DAG G, we use the same notation as in Section 4.2, $G(\mathcal{S}, j)$, to denote subsets of DAG. For each DAG $G(\mathcal{S}, j)$, there is a terminal chain $\mathbb{C}(\mathcal{S}, j)$ with task j as the ending task in the schedule $\mathcal{S}(j)$. Similarly, define $P(\mathcal{S}, j)$ as a sum of the processing time along the terminal chain $\mathbb{C}(\mathcal{S}, j)$,

$$P(\mathcal{S}, j) = \sum_{c_j \in \mathbb{C}(\mathcal{S}, j)} \frac{w_{c_j}}{s_{h(c_j)}}, \tag{14}$$

and let $D_k(\mathcal{S}, j)$ denote the total load assigned to machines in group M_k in DAG $G(\mathcal{S}, j)$,

$$D_k(\mathcal{S}, j) = \frac{\sum_{j : j \in G(\mathcal{S}, j), k \in f(j)} w_j}{s(M_k)}. \tag{15}$$

For every DAG $G(\mathcal{S}, j)$ associated with schedule $\mathcal{S}_j$ for $1 \leq j \leq n$, we are able to apply Separation Principle and then combine these inequalities as follows:

$$\sum_j \omega_j C_j \leq \sum_j \omega_j \left(P(\mathcal{S}, j) + \sum_k D_k(\mathcal{S}, j) \right) + \sum_j \omega_j C(\mathcal{S}, j).$$

Both $P(\mathcal{S}, j)$ and $D_k(\mathcal{S}, j)$ are independent of the communication constraints, which enables us to take advantage of any group assignment rule.

Using the group assignment rule $f_{\mathsf{twct}}(\cdot)$ helps further tighten the bound. To show this, we first divide the n tasks into Q sets based on $q(j)$, which can be viewed as a rough estimate of the completion time of task j. For the qth interval, we define $\mathcal{J}_q$ as a set of tasks such that $q(j) = q$:

$$\mathcal{J}_q = \{j : q(j) = q\}.$$

In this way, we have divided the n tasks into Q sets: $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_Q$.

Next, for $1 \leq q \leq Q$, we construct a set of feasible solutions for LP (1), $\tilde{x}, \tilde{C}$ and $\tilde{T}$, for every set of tasks in $\mathcal{J}_q$, based on the optimal solution of LP (2), i.e., x^* and C^*. Note that $\tilde{x}$ here is the same as in equation (3). Since precedence constraints are preserved in constraints of the LPs, we can concatenate these schedules together to obtain a feasible schedule for all of the tasks.

Lemma 5.4. *Consider a set of tasks $\mathcal{J}_q$ for a fixed q. A feasible solution for LP* (1) *is defined by*

$$\tilde{x}_{i,j} = \sum_{t=1}^{q} \frac{x^*_{i,j,t}}{\alpha_j} \qquad \forall i, j \in \mathcal{J}_q \tag{17a}$$

$$\tilde{C}_j = 2C^*_j \qquad \forall j \in \mathcal{J}_q \tag{17b}$$

$$\tilde{T} = 2^{q+1}. \tag{17c}$$

Proof. To show feasibility of such a candidate solution, we verify that $q, \tilde{x}, \tilde{C}$ and $\tilde{T}$ satisfy all the constraints in LP (1). Substitute $\tilde{x}$ into the left side of constraint (1a) for any task $j \in \mathcal{J}_q$, and it is clear that $\sum_i \tilde{x}_{i,j} = 1$. To validate that constraint (1b) is satisfied, note that $\alpha_j \geq 1/2$ by definition and so a direct substitution on the left hand side yields the right hand side due to (2b). Similarly, constraint (2c) ensures that constraint (1c) is satisfied and constraint (2f) ensures that constraint (1d) is satisfied. Finally, we obtain $C^*_j \leq 2^q$ by definition of $q(j)$ and thus constraint (1e) holds. □

Due to the similarity between group assignment rule $f_{\mathsf{mksp}}(\cdot)$ and $f_{\mathsf{twct}}(\cdot)$, we can further tighten the bound using Lemmas 5.2 and 5.3 from Section 5.2 directly. Combining Lemmas 5.2 and 5.4, we conclude that the total load along any chain $\mathbb{C}$ in the DAG formed by $\mathcal{J}_q$ is upper bounded by

$$\begin{aligned} \sum_{j \in \mathbb{C}} \frac{w_j}{s_{h(j)}} &\leq 2\gamma \tilde{T} \\ &= 2\gamma \cdot 2^{q+1}. \end{aligned}$$

Next, since the terminal chain $\mathbb{C}(\mathcal{S}, j)$ can be represented as a concatenation of chains in the DAGs formed by tasks in $\mathcal{J}_q$ for $1 \leq q \leq q(j)$, we have

$$\begin{aligned} P(\mathcal{S}, j) &\leq \sum_{t=1}^{q(j)} 2\gamma \cdot 2^{t+1} \\ &\leq 8\gamma \cdot 2^{q(j)}. \end{aligned}$$

Using Lemmas 5.3 and 5.4 together gives the following inequality:

$$\begin{aligned}\sum_{j\in\mathcal{J}_q}\frac{w_j}{s(f_{\mathsf{twct}}(j))} &\le 2K\tilde{T}\\ &= 2K\cdot 2^{q+1}.\end{aligned}$$

The left side can be viewed as $\sum_k D_k$ for a DAG formed by tasks in $\mathcal{J}_q$. Since the tasks in DAG $G(\mathcal{S},j)$ form a subset of $\cup_{t=1}^{q(j)}\mathcal{J}_q$, the following inequality holds:

$$\begin{aligned}\sum_k D_k(\mathcal{S},j) &\le \sum_{t=1}^{q(j)}\sum_{j'\in\mathcal{J}_q}\frac{w_{j'}}{s(f_{\mathsf{twct}}(j'))}\\ &\le \sum_{t=1}^{q(j)} 2K\cdot 2^{t+1}\\ &\le 8K\cdot 2^{q(j)},\end{aligned}$$

which immediately yields

$$P(\mathcal{S},j)+\sum_k D_k(\mathcal{S},j)\le 8(\gamma+K)\cdot 2^{q(j)}.$$

Finally, the remaining piece of the proof is to upper bound $2^{q(j)}$ with a multiplicative factor of its optimal completion time C_j^* in the LP (2). By definition of $q(j)$, for task j either

$$\sum_{t=1}^{q(j)-1}\sum_i x_{i,j,t}^* < \frac{1}{2} \tag{20}$$

or

$$C_j^* > 2^{q(j)-1}. \tag{21}$$

If inequality (20) holds, then

$$\begin{aligned}2^{q(j)-1} &= \tau_{q(j)-1}\\ &\le 2\tau_{q(j)-1}\left(\sum_{t=q(j)}^{Q}\sum_i x_{i,j,t}^*\right) && \text{(22a)}\\ &\le 2\left(\sum_{t=q(j)}^{Q}\tau_{t-1}\sum_i x_{i,j,t}^*\right)\\ &\le 2\left(\sum_t \tau_{t-1}\sum_i x_{i,j,t}^*\right)\\ &\le 2C_j^*. && \text{(22b)}\end{aligned}$$

Inequality (22a) is due to (20) and the definition of $q(j)$, and constraint (2e) in the LP (2) leads to (22b). If inequality (21) is true, then

$$2^{q(j)-1} < C_j^* \le 2C_j^*.$$

In both cases, $2^{q(j)-1}$ is upper bounded by $2C_j^*$. Thus, we achieve

$$P(\mathcal{S},j)+\sum_k D_k(\mathcal{S},j)\le 32(\gamma+K)\cdot C_j^*.$$

Since $\sum_j \omega_j C_j^*$ is lower bounded by wOPT$^{(i)}$, we conclude that

$$\begin{aligned}\sum_j \omega_j C_j &\le \sum_j \omega_j\left(P(\mathcal{S},j)+\sum_k D_k(\mathcal{S},j)\right)+\sum_j \omega_j C(\mathcal{S},j) && \text{(23a)}\\ &\le 32(\gamma+K)\sum_j \omega_j C_j^* + \sum_j \omega_j C(\mathcal{S},j) && \text{(23b)}\\ &\le O(\log m/\log\log m)\cdot \text{wOPT}^{(i)} + \sum_j \omega_j C(\mathcal{S},j), && \text{(23c)}\end{aligned}$$

which completes the proof.

6 Concluding Remarks

This paper studies the problem of scheduling tasks with precedence constraints on related machines with machine-dependent communication times, and addresses two long-standing open problems in the area. We introduce a new scheduler, GETF, and prove worst-case approximation ratios for it in the case of (i) scheduling to minimize the makespan and (ii) scheduling to minimize the total weighted completion time. These results represent the first progress on this problem in the 30 years since [8] provided a bound on the makespan under ETF in the case of identical servers and communication time. No previous bounds exist for the case of total weighted completion time when communication time is considered.

A variety of open questions are raised by the work in this paper. Most importantly, while we have provided theoretical bounds on the performance of GETF, it is also important to investigate how GETF performs in real settings via an implementation study. GETF could be particularly powerful in the context of large-scale machine learning platforms, where workflows are typically specified as DAGs. As part of such a study, it would be interesting to understand how to best choose a tie-breaking rule, how to adjust the group assignment rules for the best performance, and how various choices for these rules compare with heuristics that have been suggested in the literature. Further, it will be important to see if it is possible to obtain some theoretical results characterizing how the optimal choices for these rules depend on properties of real-world workloads. Moreover, it will also be interesting to extend the results of this work to stochastic settings, e.g., when task sizes are unknown.

On the analytic side, it will be interesting to discover other applications of the Separation Principle. It may be possible to revisit other scheduling problems for precedence-constrained tasks and obtain more general results because of the separation this result provides. Further, it is possible to consider other performance measures, such as energy usage and resource augmentation, using the Separation Principle.

References

[1] Edward Grady Coffman and John L Bruno. *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[3] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[4] David Chappell. Introducing azure machine learning. *A guide for technical professionals, sponsored by Microsoft Corporation*, 2015.

[5] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[6] Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[7] S. Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–294, Oct 2017.

[8] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D Anger, and Chung-Yee Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[9] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. GRASS: Trimming stragglers in approximation analytics. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 289–302, Seattle, WA, 2014. USENIX Association.

[10] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 379–392, New York, NY, USA, 2015. ACM.

[11] M-Y Wu and Daniel D Gajski. Hypertool: A programming aid for message-passing systems. *IEEE transactions on parallel and distributed systems*, 1(3):330–343, 1990.

[12] Yuming Xu, Kenli Li, Ligang He, Longxin Zhang, and Keqin Li. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. *IEEE Transactions on parallel and distributed systems*, 26(12):3208–3222, 2015.

[13] Tao Yang and Apostolos Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[14] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[15] Fuhui Wu, Qingbo Wu, and Yusong Tan. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing*, 71(9):3373–3418, 2015.

[16] Ruben Mayer, Christian Mayer, and Larissa Laich. The tensorflow partitioning and scheduling problem: it's the critical path! In *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, pages 1–6. ACM, 2017.

[17] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 185–198, 2013.

[18] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, Vancouver, BC, 2010. USENIX Association.

[19] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[20] Minghong Lin, Li Zhang, Adam Wierman, and Jian Tan. Joint optimization of overlapping phases in mapreduce. *Performance Evaluation*, 70(10):720–735, 2013.

[21] Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. Purlieus: locality-aware resource allocation for mapreduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 58. ACM, 2011.

[22] Jian Tan, Xiaoqiao Meng, and Li Zhang. Delay tails in mapreduce scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):5–16, 2012.

[23] Abhishek Verma, Ludmila Cherkasova, and Roy H Campbell. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 11–18. IEEE, 2012.

[24] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Transactions on Networking (TON)*, 24(1):190–203, 2016.

[25] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[26] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H. Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. *CoRR*, abs/1803.03288, 2018.

[27] Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[28] Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 745–754. ACM, 2010.

[29] Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 453–462. IEEE, 2009.

[30] Leslie A Hall, David B Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *SODA*, volume 96, pages 142–151, 1996.

[31] Alix Munier, Maurice Queyranne, and Andreas S Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 367–382. Springer, 1998.

[32] Maurice Queyranne and Andreas S Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computing*, 35(5):1241–1253, 2006.

[33] Abbas Bazzi and Ashkan Norouzi-Fard. Towards tight lower bounds for scheduling problems. In *Algorithms-Esa 2015*, pages 118–129. Springer, 2015.

[34] Maciej Drozdowski. *Scheduling for parallel processing*. Springer, 2009.

[35] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[36] Michael R Garey and David S Johnson. Computers and intractability: a guide to np-completeness, 1979.