Maximizing Throughput in Flow Shop Real-Time Scheduling

Lior Ben Yamin
Computer Science Department, Technion, Haifa, Israel
lior.b@cs.technion.ac.il

Jing Li
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
jingli@njit.edu

Kanthi Sarpatwar
IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
sarpatwa@us.ibm.com

Baruch Schieber
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
sbar@njit.edu

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel
hadas@cs.technion.ac.il

Abstract

We consider scheduling real-time jobs in the classic flow shop model. The input is a set of n jobs, each consisting of m segments to be processed on m machines in the specified order, such that segment I_i of a job can start processing on machine M_i only after segment I_{i-1} of the same job completed processing on machine M_{i-1}, for $2 \leq i \leq m$. Each job also has a release time, a due date, and a weight. The objective is to maximize the throughput (or, profit) of the n jobs, i.e., to find a subset of the jobs that have the maximum total weight and can complete processing on the m machines within their time windows. This problem has numerous real-life applications ranging from manufacturing to cloud and embedded computing platforms, already in the special case where $m = 2$. Previous work in the flow shop model has focused on makespan, flow time, or tardiness objectives. However, little is known for the flow shop model in the real-time setting. In this work, we give the first nontrivial results for this problem and present a pseudo-polynomial time $(2m + 1)$-approximation algorithm for the problem on $m \geq 2$ machines, where m is a constant. This ratio is essentially tight due to a hardness result of $\Omega(\frac{m}{\log m})$ for the approximation ratio. We further give a polynomial-time algorithm for the two-machine case, with an approximation ratio of $(9 + \varepsilon)$ where $\varepsilon = O(1/n)$. We obtain better bounds for some restricted subclasses of inputs with two machines. To the best of our knowledge, this fundamental problem of throughput maximization in the flow shop scheduling model is studied here for the first time.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Scheduling algorithms

Keywords and phrases Flow shop, real-time scheduling, throughput maximization, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.48

Category APPROX

Funding *Jing Li*: Supported by NSF CNS-1948457.

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020).
Editors: Jarosław Byrka and Raghu Meka; Article No. 48; pp. 48:1–48:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Flow shop is a fundamental scheduling model where a set of jobs needs to be processed in multiple stages in a specified order that is the same for all jobs. There are many real-world applications of flow shop scheduling, ranging from production planning and computing platforms to satellite systems and service centers. For instance, an autonomous car runs applications, such as obstacle detection and route planning, applying deep neural networks on an embedded computing platform, which are composed of a CPU host and a GPU accelerator connected via a non-preemptive bidirectional bus. Each execution instance (i.e., job) of the applications is first initiated in the CPU to preprocess the input data, then transfers the data from CPU to GPU via the bus, executes the computation on the GPU, and finally transfers the results back via the bidirectional bus. There are multiple such jobs running in real-time with different release times and deadlines, e.g., multiple images to be processed by the object detection application in a time window. Hence, the computing platform needs to schedule the execution on the CPU and GPU, as well as the data transfers on the bus, to meet preset deadlines, e.g., to maximize the number of images processed in a time window.

The flow shop model has been widely studied for minimizing the latest completion time of any job (or, *makespan*) since the 1950s, starting with the seminal work of Johnson [16], which showed that makespan minimization in two-machine flow shop can be solved in polynomial time. However, most extensions of the problem are strongly NP-hard [7]. For example, makespan minimization for flow shop with three machines is already NP-complete, even if the input length is measured by the sum of the job lengths [12]. Hence, later works studied approximation algorithms for the problem (see, e.g., [13, 18, 20, 22, 24]).

In this paper, we are interested in flow shop scheduling for jobs with different release times, due dates, and weights, and the scheduling objective is to maximize the throughput – the total weight of the jobs that are completed by their due dates. Surprisingly, in contrast to the extensive results on minimizing the makespan, flow time, and tardiness in the flow shop model, there is little work on maximizing the throughput of jobs with due dates. On the other hand, the problem of maximizing the throughput of jobs with release times, due dates, and weights, also known in the literature as (aperiodic) *real-time scheduling*, has been widely studied. In this classic model, each job can be processed to completion on a *single* machine or any of the *parallel* machines (see, e.g., [1–4, 9, 15, 17, 19, 25]).

We now formalize our problem. In the m-machine flow shop model, there is a set of n jobs, $\mathcal{J} = \{J_1, \dots, J_n\}$, and m machines, $M_1, \dots, M_m$. Each job J_j, $1 \le j \le n$, has a release time, a due date, and a weight, given by $r_j \ge 0$, $d_j \ge 0$, and $w_j > 0$, respectively. A job can only start executing on machine M_i, $2 \le i \le m$, after it has finished its execution on the previous machine M_{i-1}. In addition, at any time $t \ge 0$, each of the machines can process at most one job. For a job J_j, we denote the processing time of its i-th segment to be executed on machine M_i by $p_{j,i}$. We assume that $p_{j,i}, r_j$ and d_j are rational numbers. We further assume that all job segments are non-preemptive. In other words, once a job J_j has started its execution on a machine, this machine cannot stop or switch to another job until J_j has finished its execution on this machine. We seek a subset of jobs $\mathcal{J}' \subseteq \mathcal{J}$ that can be *feasibly* scheduled (i.e., each job J_j can complete processing on all machines in a flow shop manner in its time window $(r_j, d_j]$) and has a maximum total weight. We denote this *maximum throughput* objective by MaxT. We obtain results for MaxT in the m-machine flow shop model, with a focus on the special case of two-machine flow shop.

1.1 Applications

In the following, we motivate MaxT in the flow shop model with some real-life applications.

Scheduling in Cloud Data Centers. A *cloud-data-center (CDC)* consists of a set of server clusters connected with clients through a network. Since all the resources are stored on the servers, clients generate resource requests from the CDC. A data request consists of two steps: task *execution*, in which data is obtained from a disk or distributed storage systems and stored in memory, and then *transmission* from memory to the client over the network (see, e.g., [26]). When data requests have release times and due dates, a natural goal is to maximize the total number of requests that can be processed by the CDC in a given time interval. This yields an instance of MaxT in the flow shop model.

Earth Observation Satellites. An earth observation satellite (EOS) is equipped with high-resolution cameras for observing target objects across the surface of Earth. There are available time windows for multiple EOSs to observe a given object and to download the acquired image/video data to ground receiver stations. The problem of observation scheduling (at stage 1) and data downlink scheduling (at stage 2) with the objective of maximizing the number of satellites that can complete processing in their time windows yields an instance of MaxT in the two-machine flow shop model (see, e.g., [26]).

Autonomous Vehicle Navigation. An object detection application running in autonomous cars takes images from a front-facing camera as input and produces car steering angles as output (see, e.g., [6]). Since the algorithm uses deep neural networks (DNN), each image is handled in stages (preprocessing data in CPU, data transfers between CPU and GPU, and DNN computation in GPU), the process of handling the images in real-time so as to maximize the number of images processed in a given time window can be viewed as a MaxT instance in the flow shop model.

1.2 Contributions and Techniques

We say that $\mathcal{A}$ is a ρ-approximation algorithm for a maximization problem Π, for $\rho \geq 1$, if for any instance I of Π, $\mathcal{A}(I) \geq \frac{OPT(I)}{\rho}$, where $OPT(I)$ is the value of an optimal solution for I.

In this paper, we study the fundamental problem of throughput maximization in the flow shop scheduling model. Our main result is a polynomial-time $(9 + \varepsilon)$-approximation algorithm for MaxT in the two-machine flow shop, where $\varepsilon = O(1/n)$ for an input of size n (i.e., $n = |\mathcal{J}|$). We derive the algorithm by first obtaining a pseudo-polynomial time $(2m + 1)$-approximation algorithm for MaxT on m machines, where $m \geq 2$ is a constant. We note that the ratio of $(2m + 1)$ is essentially tight for any $m \geq 3$, due to a known hardness of approximation result for a ratio $\Omega(\frac{m}{\log m})$ [14].

We show that MaxT admits better approximations on some restricted instances of the two-machine model. In particular, we present a 4-approximation algorithm for instances where all jobs have the same release time, i.e., $r_j = 0$ for all $J_j \in \mathcal{J}$, and uniform weights. For the special case where all jobs have the same time window and arbitrary weights, we give a $(3 + \varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$.

Techniques. In Section 2, we give an approximation algorithm for instances of MaxT on m machines, where m is some constant. As our algorithm requires solving a Configuration Linear Program (LP), this implies a pseudo-polynomial running time. Showing that this algorithm can be implemented in polynomial time, with only a slight degradation in the approximation ratio, is a major challenge even in the two machine case. We use the following key observation. Any instance $\mathcal{J}$ can be modified to an instance $\mathcal{J}_{new}$ (by replacing some jobs with *new* jobs) in which for every job $J_j \in \mathcal{J}_{new}$ either *both* $p_{j,1}$ and $p_{j,2}$ are *large* relative to $d_j - r_j$, or *both* $p_{j,1}$ and $p_{j,2}$ are *small* relative to $d_j - r_j$. Then, by an intricate analysis, we show how to reduce the number of variables associated with the jobs in $\mathcal{J}_{new}$ to be of size polynomial in $|\mathcal{J}_{new}|$ (and consequently also in $|\mathcal{J}|$, since we add only a polynomial number of new jobs) with only a minor degradation in the quality of the solution. The resulting polynomial-size linear program can then be solved and rounded in polynomial time to obtain an approximate solution (details are in Section 3). In one of the special cases, we establish a precise relation between the approximability of classic real-time scheduling on a single machine and MaxT in the two-machine flow shop (details are in Section 4.1). This allows the use of approximation algorithms for the single machine case for solving our problem.

1.3 Prior Work

The problem of real-time scheduling with the objective of throughput maximization is discussed widely in the literature. A general instance of the problem consists of a set of n jobs and k machines, for some $k \geq 1$, where each job j has a weight $w_j > 0$, a release time r_j, a due date d_j and a processing time p_{ji} on machine i, for $1 \leq i \leq k$ and $1 \leq j \leq n$. The goal is to find a non-preemptive schedule that maximizes the weight of jobs that meet their respective due dates. Note that all the related works in this domain do not consider the flow shop model. Instead, the k machines form a single stage, where each job needs to be processed on *any one* of the machines.

The problem is known to be NP-complete already in the single machine case (i.e., $k = 1$), where all jobs have the same (unit) weight [11]. Some special cases of the problem are known to be solvable in polynomial time. Moore [19] showed for the single machine case and uniform job weights that, if $r_j = 0 \ \forall j$ the problem can be solved in time $O(n^2)$. Sahni presented in [21] a *fully polynomial time approximation scheme (FPTAS)*, whose running time is $O(\frac{n^3}{\varepsilon})$, for the more general case where jobs have the same release time and arbitrary weights.

Bar-Noy et al. [3, 4] considered the real-time scheduling problem for general instances with k machines, for some $k \geq 1$, where jobs may have arbitrary weights and arbitrary release times and due dates. They presented in [3] a $(2 + \varepsilon)$-approximation algorithm, using the local ratio technique. A quasi-polynomial time dynamic programming framework was proposed in [15], which gives a $(1+\varepsilon)$-speed $(1+\varepsilon)$-approximation algorithm for the weighted throughput problem on k machines. The best known result without speed augmentation is an $\frac{e}{e-1} < 1.582$ approximation algorithm [9], for a single machine and uniform job weights.

We note that a variant of MaxT, where for every job $J_j \in \mathcal{J}$, the start-times of all segments of J_j are given explicitly, yields an instance of *maximum weight independent set* in m-union graphs. Recall that an m-union graph can be modeled as the intersection graph of m-segments, i.e., each vertex in the graph can be represented by at most m segments on the real line. Two vertices are adjacent if their ith segments intersect. For this problem, the paper [5] presented a $2m$-approximation algorithm that was shown to be close to the best possible, due to a hardness of approximation result for a ratio of $\Omega(\frac{m}{\log m})$ [14]. The hardness result carries over to MaxT in the flow shop model and m machines, for any $m \geq 3$. Our approximation algorithm for a constant number of machines builds on an algorithm presented in [5]. However, further steps are required to obtain polynomial running time, which is the main contribution of this paper.

Another line of work that relates to MaxT deals with maximizing the total weight of *just-in-time (JIT)* jobs, i.e., the weighted number of jobs that are completed *exactly* on their due dates. All previous studies assume that $r_j = 0\ \forall j$. Choi and Yoon [8] show that JIT two-machine flow shop with arbitrary job weights is NP-complete. The special case of uniform job weights is solvable in polynomial time on two machines and is strongly NP-hard for instances with three machines. The best known result is an FPTAS in [10] (see also [23]).

We are not aware of earlier studies of throughput maximization in the flow shop model.

2 Approximation Algorithm for Fixed Number of Machines

In this section, we present a pseudo-polynomial time algorithm for MaxT on flow shop instances with $m \geq 2$ machines, where m is some constant. Given the set of jobs $\mathcal{J}$, each job J_j, $1 \leq j \leq n$, is associated with m segments and a weight $w_j \geq 0$. Also, J_j has a release time and a due date, $r_j \geq 0$ and $d_j \geq 0$, respectively. We seek a subset of the jobs that can be feasibly scheduled on the machines in a flow shop manner, such that the total weight of scheduled jobs is maximized.

As the processing time $p_{j,i}$ on machine M_i, release time r_j, and deadline d_j of job J_j are all rational numbers, we can obtain integer values for these parameters by appropriate scaling. Since all these values are integral, it is easy to see that any feasible solution can be "tweaked" so that the start times of all segments of all jobs begin at an integral time point. Thus, from now on we assume that this is the case. This allows us to discretize the input and consider all the possible occurrences of a job J_j in its time window $(r_j, d_j]$. An occurrence of J_j specifies the start times of all segments of J_j on the m machines in $(r_j, d_j]$. Note that the number of such possible occurrences of job J_j is upper bounded by $(d_j - r_j)^m$.

We give some notations towards solving MaxT on m machines using a linear program. Let $\mathcal{L}_j$ denote the set of occurrences of job J_j, so the number of the occurrences of J_j is $|\mathcal{L}_j|$. Let $\mathcal{L} = \bigcup_{j=1}^n \mathcal{L}_j$. Clearly, $|\mathcal{L}| = \sum_{j=1}^n |\mathcal{L}_j|$. Let $x^\ell(j) \in \{0,1\}$ be an indicator variable for the selection of the ℓ-th occurrence J_j^ℓ of J_j in the solution, where $1 \leq \ell \leq |\mathcal{L}_j|$. We note that the number of variables and the number constraints in the linear program is $O(\sum_{j=1}^n (d_j - r_j)^m)$, and is thus pseudo-polynomial in the input size. Let $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^{|\mathcal{L}|}$ be a weight vector and a relaxed indicator vector, respectively. Then, $\mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^n \sum_{\ell=1}^{|\mathcal{L}_j|} w_j x^\ell(j)$.

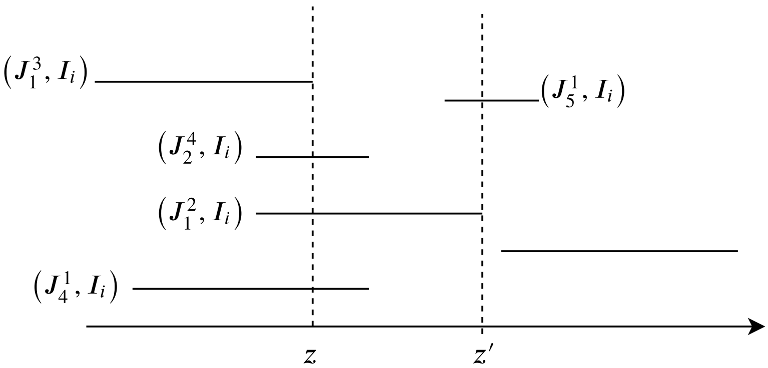

Figure 1 The job clique $(J_1^3, J_2^4, J_1^2, J_4^1)$ is defined by z, the right endpoint of the ith segment of J_1^3. The clique contains two occurrences of J_1: J_1^3 and J_1^2. The next clique, (J_1^2, J_5^1), is defined by the segment having its right endpoint at z'.

We now define *job cliques* on each of the m machines as follows. For machine $1 \leq i \leq m$, we examine the time axis from left to right and find among the segments that need to be processed on machine i a segment whose right endpoint is earliest. Let z be the time point where this segment ends. We now define a clique $\mathcal{C}$ consisting of all job occurrences whose ith segment intersects the time point z. The next clique is defined by the earliest endpoint z' of an ith segment of a job, for which the following holds: there exists a job occurrence J_k^r, such that the ith segment of J_k^r intersects z', but $J_k^r \notin \mathcal{C}$, as shown in Figure 1. Intuitively, the endpoints z and z' in the definition of job cliques capture the maximum intersecting job occurrences in the time window $[z, z']$. Hence, a feasible schedule can only select one job occurrence in each clique.

We formulate the linear programming relaxation for MaxT as follows.

$$\begin{array}{llll} \text{(P)} & \text{maximize} & \mathbf{w}\cdot\mathbf{x} & \textit{subject to:} \\ \displaystyle\sum_{J_j^\ell \in \mathcal{C}} x^\ell(j) & \leq & 1 & \text{for each clique } \mathcal{C} \\ \displaystyle\sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(j) & \leq & 1 & \forall 1 \leq j \leq n \\ x^\ell(j) & \geq & 0 & \forall 1 \leq j \leq n,\ 1 \leq \ell \leq |\mathcal{L}_j| \end{array}$$

The first constraint ensures that at most one job occurrence is selected from each clique. The second constraint guarantees that at most one occurrence of a job J_j is selected for the solution, $\forall J_j \in \mathcal{J}$. We note that (P) can be viewed as a Configuration LP, where each occurrence, J_j^ℓ, defines a *configuration*, $\forall J_j \in \mathcal{J}$.

Considering the neighbors of a job occurrence J_j^ℓ, we define two subsets of job occurrences.

1. Let $\tilde{N}_1(J_j^\ell)$ be the set of all job occurrences J_k^r where $k \neq j$, such that a segment of J_k^r intersects a segment of J_j^ℓ (recall that two segments can intersect only if both need to be processed on the *same* machine).
2. Let $\tilde{N}_2(J_j^\ell)$ be the set of all job occurrences J_j^r where $r \neq \ell$, i.e., other occurrences of J_j.

The *neighborhood* of a job occurrence J_j^ℓ is defined as $\tilde{N}(J_j^\ell) = \tilde{N}_1(J_j^\ell) \bigcup \tilde{N}_2(J_j^\ell)$.

▶ **Lemma 1.** *Let* $\mathbf{x}$ *be a feasible solution to* (P)*. There exists a job occurrence* J_j^ℓ *satisfying:*

$$x^\ell(j) + \sum_{J_k^r \in \tilde{N}(J_j^\ell)} x^r(k) \leq 2m + 1.$$

Proof. We first show that there exists a job occurrence J_j^ℓ for which the following holds:

$$x^\ell(j) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} x^r(k) \leq 2m. \tag{1}$$

For two "neighboring" job occurrences J_j^ℓ and J_k^r (i.e., $J_k^r \in \tilde{N}_1(J_j^\ell)$ and $J_j^\ell \in \tilde{N}_1(J_k^r)$), define $z(J_j^\ell, J_k^r) = x^\ell(j) \cdot x^r(k)$. We also define $z(J_j^\ell, J_j^\ell) = (x^\ell(j))^2$. We prove (1) using a weighted averaging argument, where the weights are the values $z(J_j^\ell, J_k^r)$ for all pairs of job occurrences which have intersecting segments. The full proof is given in Appendix A. ◀

2.1 The Algorithm

We now show how to use Lamma 1 to get a $(2m+1)$-approximation for MaxT on m machines for some constant $m \geq 2$. Let $\mathcal{I}$ be the set of all half open subintervals of the interval $(0, 2m+1]$. Given an optimal solution $\mathbf{x}$ for the linear program (P), we construct a mapping $\psi : \mathcal{L} \to 2^{\mathcal{I}}$ such that for each job occurrence J_j^ℓ the following properties are satisfied:

1. All the subintervals in $\psi(J_j^\ell)$ are disjoint.
2. The total size of the subintervals in $\psi(J_j^\ell)$ is $x^\ell(j)$.
3. None of the subintervals in $\psi(J_j^\ell)$ intersects any of the subintervals in $\bigcup_{J_k^r \in \tilde{N}[J_j^\ell]} \psi(J_k^r)$.

The mapping is constructed for one job occurrence at a time according to a *hierarchical* order induced by Lemma 1. We first define this hierarchical order. The *last* job occurrence in the order is the occurrence J_j^ℓ that satisfies the inequality of Lemma 1. We then remove this job occurrence from the feasible solution to (P). We still remain with a feasible solution to (P) and we can apply Lemma 1 again and find yet another job occurrence that satisfies the inequality of the lemma. We append this job occurrence to the order. We continue in the same manner until we order all the job occurrences.

We compute $\psi(J_j^\ell)$ for one job occurrence at a time from the first to the last in the hierarchical order defined above. When $\psi(J_j^\ell)$ is computed, we remove from the interval $(0, 2m+1]$ all the subintervals in $\bigcup_{J_k^r \in \bar{N}} \psi(J_k^r)$, where $\bar{N} \subset \tilde{N}[J_j^\ell]$ is the set of all job occurrences in $\tilde{N}[J_j^\ell]$ that precede J_j^ℓ in the hierarchical order. By Lemma 1 the total size of these subintervals is no more than $2m+1-x^\ell(j)$. Thus, the remainder contains a set of disjoint subintervals of a total size at least $x^\ell(j)$. If we assign $\psi(J_j^\ell)$ greedily, that is, we assign the leftmost collection of such disjoint subintervals, then it can be shown that $|\psi(J_j^\ell)|$ is bounded by $|\mathcal{L}|$. This is because each job occurrence may increase the number of subintervals by at most one.

For a point $y \in (0, 2m+1]$, let $\phi(y) \subseteq \mathcal{L}$ be the subset of $\mathcal{L}$ consisting of all job occurrences J_j^ℓ for which one of the subintervals in $\psi(J_j^\ell)$ contains the point y. From the definition of the mapping ψ, it is evident that the subset $\phi(y)$ does not contain two job occurrences that intersect and also does not contain two job occurrences of the same job. Thus, the job occurrences in $\phi(y)$ can be scheduled feasibly to yield a weight of $w(y) = \sum_{J_j^\ell \in \phi(y)} w_j$. Let $y^* = \arg\max_{y \in (0,2m+1]} \{w(y)\}$. Note that if the mapping is computed greedily there are at most $|\mathcal{L}|^2$ possible values of $w(y^*)$. These values are determined by the right endpoints of all subintervals. The pseudocode of the algorithm, Flowshop_Time_Windows, is in Algorithm 1.

Algorithm 1 Flowshop_Time_Windows.

Find an optimal solution $\mathbf{x}$ for the linear program (P).
Order the job occurrences according to the hierarchical order.
for each job occurrence J_j^ℓ in order **do**
    Remove from the interval $(0, 2m+1]$ all subintervals assigned to "neighbors" of J_j^ℓ that precede it in the hierarchical order.
    Assign to J_j^ℓ the leftmost collection of subintervals of total size $x^\ell(j)$.
end for
Let $maxwy = 0$.
for a point y that is a right endpoint of a subinterval **do**
    Let $\phi(y) \subseteq \mathcal{L}$ be the subset of all job occurrences J_j^ℓ for which one of the subintervals in $\psi(J_j^\ell)$ contains the point y.
    Let $w(y) = \sum_{J_j^\ell \in \phi(y)} w_j$.
    if $w(y) > maxwy$ **then**
        Let $maxwy = w(y)$ and $y^* = y$.
    end if
end for
Return $\phi(y^*)$.

▶ **Theorem 2.** Flowshop_Time_Windows *yields a* $(2m+1)$*-approximation for* MaxT *on* m *machines.*

Proof. Consider $\int_0^{2m+1} w(y)dy$. By our definitions,

$$\int_0^{2m+1} w(y)dy = \sum_{j=1}^{n}\sum_{\ell=1}^{|\mathcal{L}_j|}\int_{\psi(J_j^\ell)} w_j dz_j = \sum_{j=1}^{n}\sum_{\ell=1}^{|\mathcal{L}_j|} w_j x^\ell(j) = \mathbf{w}\cdot\mathbf{x}$$

The first equality is derived by a variable substitution and the second equality follows from the second property of the mapping. Since $\int_0^{2m+1} w(y)dy = \mathbf{w}\cdot\mathbf{x}$ it follows that $(2m+1)w(y^*) \geq \mathbf{w}\cdot\mathbf{x}$. ◀

▶ **Corollary 3.** *There is a pseudo-polynomial time* $(2m+1)$*-approximation algorithm for* MaxT *on* m *machines, where* $m \geq 2$ *is some constant.*

3 Approximating MaxT on Two Machines

We now show that, with a slight degradation of the approximation ratio, we can use the algorithm presented in Section 2 to obtain a polynomial-time algorithm for $m = 2$.

We start with some notations. Consider two machines, M_1 and M_2, and each job consists of two non-preemptive segments. For notation simplicity, in the following sections, we denote the processing times of J_j on M_1 and M_2 as a_j and b_j, respectively. Recall that a job $J_j \in \mathcal{J}$ has a release time $r_j \geq 0$, a due date $d_j \geq 0$, and a weight $w_j \geq 0$. Thus, in any feasible schedule of $\mathcal{J}' \subseteq \mathcal{J}$ in the flow shop model, $J_j \in \mathcal{J}'$ is processed first for a_j time units on M_1 after its release time r_j, then processed for b_j time units on M_2 and finished no later than its due date d_j.

We distinguish between three types of jobs based on their *slackness*:

(i) Small jobs $\mathcal{J}_{\mathcal{S}}$: Job J_j is a Small job, if it has a large slack in its time window $[r_j, d_j]$, satisfying $a_j + b_j < \frac{d_j - r_j - a_j - b_j}{n^2-1}$. Note that this implies $a_j + b_j < \frac{d_j - r_j}{n^2}$.

(ii) Large jobs $\mathcal{J}_{\mathcal{L}}$: Job J_j is a Large job, if $a_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$.

(iii) Almost-Large jobs $\mathcal{J}_{\mathcal{AL}}$: Job J_j is a Almost-Large job, if it satisfies $a_j + b_j \geq \frac{d_j - r_j - a_j - b_j}{n^2-1}$ (and hence $a_j + b_j \geq \frac{d_j - r_j}{n^2}$), and also one of the following:

- **(a)** $a_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j < \frac{d_j - r_j - a_j - b_j}{2n^2-2}$.
- **(b)** $a_j < \frac{d_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$.

We modify the linear program (P) in Section 2 to solve it in polynomial time in the following steps. First, we eliminate Almost-Large jobs and replace each Almost-Large job by two Large jobs and one Small job. Then, we define the modified linear program (P$_{new}$) of polynomial size by identifying only a polynomial number of job occurrences for each job included in this formulation. We call these job occurrences the *selected* job occurrences. All the unselected job instances will not be scheduled (fractionally). We show that any feasible solution of (P) induces a feasible solution of (P$_{new}$) with a slight degradation in the value of the objective function. Finally, we show how a feasible solution of (P$_{new}$) can be "rounded" to a schedule whose weight is $\frac{1}{9}$ of the objective value of this feasible solution of (P$_{new}$). This schedule is a $(9+\epsilon)$-approximation of the optimal solution.

3.1 Eliminating the Almost-Large Jobs

We first partition the set $\mathcal{J}_{\mathcal{AL}}$ of Almost-Large jobs into two subsets $\mathcal{J}^1_{\mathcal{AL}}$ and $\mathcal{J}^2_{\mathcal{AL}}$.

(1) The subset $\mathcal{J}^2_{\mathcal{AL}}$ of jobs J_j satisfying $a_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j < \frac{d_j - r_j - a_j - b_j}{2n^2-2}$.

(2) The subset $\mathcal{J}^2_{\mathcal{AL}}$ of jobs J_j satisfying $a_j < \frac{d_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$.

Consider a job $J_j \in \mathcal{J}_{\mathcal{AL}}^1$. Let time point $t_j = d_j - a_j - (2n^2-1)b_j$, so $b_j = \frac{d_j - t_j - a_j - b_j}{2n^2-2}$. Partition the occurrences of J_j into two subsets. The first subset consists of all occurrences J_j^ℓ in which the first segment of J_j^ℓ starts (on M_1) at or after t_j, and the complement subset consists of all occurrences J_j^ℓ in which the first segment of J_j^ℓ starts processing before time t_j. We replace job J_j with three new jobs as follows.

Note that the job occurrences in the first subset are essentially the job occurrences of a new job consisting of two segments of lengths a_j, b_j with new release time t_j and due date d_j. We add such a job J_{n+j} to the input. This job is LARGE, since $a_j \geq \frac{d_j - t_j - a_j - b_j}{2n^2-2}$ and $b_j = \frac{d_j - t_j - a_j - b_j}{2n^2-2}$.

For the second subset of job occurrences, we ignore (for now) all the occurrences where the second segment starts before $t_j + a_j$. Note that the rest of the job occurrences in the second subset are essentially the job occurrences of two new jobs: one job consisting of a single segment of length a_j (to be processed on M_1) with release time r_j and new due date $t_j + a_j$, and a second job consisting of a single segment of length b_j (to be processed on M_2) with new release time $t_j + a_j$ and due date d_j. We add these two jobs J_{2n+j} and J_{3n+j} to the input. Since $d_j - b_j = t_j + a_j + (2n^2-2)b_j \geq t_j + a_j$ and $a_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$, we have $a_j \geq \frac{t_j + a_j - r_j - a_j}{2n^2-2}$. Thus, the job J_{2n+j} is LARGE. The job J_{3n+j} is SMALL, since $b_j = \frac{d_j - t_j - a_j - b_j}{2n^2-2} < \frac{d_j - t_j - a_j - b_j}{n^2-1}$. We make sure that J_{2n+j} and J_{3n+j} are scheduled together by modifying the linear program.

Jobs $J_j \in \mathcal{J}_{\mathcal{AL}}^2$ are handled symmetrically. Let t_j be the time point satisfying $a_j = \frac{t_j - r_j - a_j - b_j}{2n^2-2}$. Partition the occurrences of J_j into two subsets. The first subset consists of all occurrences where the second segment ends at or before t_j, and the complement subset consists of all occurrences where the second segment ends processing (on M_2) after time t_j. The job occurrences in the first subset are the same as the new job occurrences of a job with two segments of lengths a_j, b_j, release time r_j and new due date t_j. We add such a job J_{n+j} to the input. This job is LARGE since $a_j = \frac{t_j - r_j - a_j - b_j}{2n^2-2}$ and $b_j \geq \frac{t_j - r_j - a_j - b_j}{2n^2-2}$. For the second subset, we again ignore (for now) all the occurrences where the first segment finishes after $t_j - b_j$. Then the rest job occurrences in the second subset are the same as the job occurrences of two new jobs: one job with a single segment of length b_j (to be processed on M_2) with new release time $t_j - b_j$ and due date d_j, and a second job consisting of a single segment of length a_j (to be processed on M_1) with release time r_j and new due date $t_j - b_j$. We add these jobs J_{2n+j} and J_{3n+j} to the input. Since $t_j - b_j = r_j + a_j + (2n^2-2)a_j \geq r_j + a_j$ and $b_j \geq \frac{d_j - r_j - a_j - b_j}{2n^2-2}$, we have $b_j \geq \frac{d_j - (t_j - b_j) - b_j}{2n^2-2}$. Thus, the job J_{2n+j} is LARGE. Since $a_j = \frac{t_j - r_j - a_j - b_j}{2n^2-2} < \frac{(t_j - b_j) - r_j - a_j}{n^2-1}$, the job J_{3n+j} is SMALL.

3.2 The Selected Occurrences of Small and Large Jobs

After eliminating ALMOST-LARGE jobs, the set of SMALL jobs in the modified LP becomes

$$\mathcal{J_S}^{new} = \mathcal{J_S} \bigcup \{J_{3n+j} |\ J_j \in \mathcal{J_{AL}}\} \text{ and } |\mathcal{J_S}^{new}| = |\mathcal{J_S}| + |\mathcal{J_{AL}}|.$$

For each SMALL job $J_j \in \mathcal{J_S}^{new}$, find n^2 non-overlapping occurrences of J_j: $J_j^1, \ldots, J_j^{n^2}$, such that in each such occurrence, the two segments of J_j are scheduled with no wait, i.e., the second segment is scheduled on M_2 immediately after completing the first segment on M_1. We can find n^2 such job occurrences since $a_j + b_j < \frac{d_j - r_j}{n^2}$. These non-overlapping occurrences are the selected occurrences of job J_j.

After eliminating ALMOST-LARGE jobs, the set of LARGE jobs in the modified LP is

$$\mathcal{J_L}^{new} = \mathcal{J_L} \bigcup \{J_{n+j}, J_{2n+j} |\ J_j \in \mathcal{J_{AL}}\} \text{ and } |\mathcal{J_L}^{new}| = |\mathcal{J_L}| + 2|\mathcal{J_{AL}}|.$$

For every $J_j \in \mathcal{J_L}^{new}$, define $2n^2+1$ dividers on the time axis for machine M_1, at the time points $r_j + h \cdot \frac{(d_j - b_j - r_j)}{2n^2}$, for $h = 0, \ldots, 2n^2$, and $2n^2+1$ dividers on the time axis for machine M_2, at the time points $r_j + a_j + h \cdot \frac{(d_j - r_j - a_j)}{2n^2}$, for $h = 0, \ldots, 2n^2$. The $|\mathcal{J_L}^{new}|(4n^2+2)$ dividers define half open time slots for M_1 and M_2, where each time slot is between adjacent dividers. We note that for any $J_j \in \mathcal{J_L}^{new}$, no segment of J_j is completely contained in a time slot, i.e., it lies between two adjacent dividers.

Consider a Large job $J_j \in \mathcal{J_L}^{new}$. For each time slot s for M_1 and time slot t for M_2, consider the set of all job occurrences of J_j where the right endpoint of its first segment is in time slot s and the right endpoint of its second segment is in t. Select one arbitrary job occurrence from this set. Let J_j^ℓ, for $1 \le \ell \le (2n^2+1)^2$, be all the selected job occurrences.

3.3 The Modified Linear Program

The set of jobs in the modified LP is $\mathcal{J}_{new} = \mathcal{J_S}^{new} \bigcup \mathcal{J_L}^{new}$. Thus, $|\mathcal{J}_{new}| = |\mathcal{J_S}| + |\mathcal{J_L}| + 3|\mathcal{J_{AL}}| \le 3n$. All the jobs in $\mathcal{J}_{new}$ are either Small or Large. We only consider variables that correspond to the selected job occurrences. We define job cliques as before but only with respect to the selected job occurrences. The modified linear program is as follows.

$$
\begin{array}{llll}
(\mathrm{P}_{new}) \quad \text{maximize} \quad \mathbf{w} \cdot \mathbf{x} \quad \textit{subject to}: & & & \\
\displaystyle\sum_{J_j^\ell \in \mathcal{C}} x^\ell(j) & \le & 2 & \text{for each clique } \mathcal{C} \\
\displaystyle\sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(j) & \le & 1 & \forall J_j \in \mathcal{J}_{new} \\
\displaystyle\sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(n+j) + \sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(2n+j) & \le & 1 & \forall J_j \in J_{\mathcal{AL}} \\
\displaystyle\sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(2n+j) - \sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(3n+j) & = & 0 & \forall J_j \in J_{\mathcal{AL}} \\
x^\ell(j) & \ge & 0 & \forall J_j \in \mathcal{J}_{new},\ 1 \le \ell \le (2n^2+1)^2
\end{array}
$$

The first constraint is a relaxation of the original clique constraint and ensures that the total value of the variables associated with the selected job occurrences in each clique is at most *two*. The second constraint guarantees that at most one occurrence of a job J_j is selected for the solution, $\forall J_j \in \mathcal{J}_{new}$. The third and fourth constraints deal with the jobs that replace the Almost-Large jobs. Recall that in Section 3.1 the occurrences of any $J_j \in \mathcal{J_{AL}}$ were partitioned into two subsets. The third constraint ensures that the total value of the variables associated with the two Large jobs that replace a single Almost-Large job (one Large job for each subset of job occurrences of the Almost-Large job) is at most one. The fourth constraint ensures that for each pair of Large job and Small job that replace the second subset of job occurrences of a single Almost-Large job, the total value of the variables associated with the replacement Large job is the same as the total value of the variables associated with the replacement Small job.

3.4 The Induced Solution of the Modified Linear Program

Consider a solution of the linear program (P) for an instance with two machines. Denote this solution $y^\ell(j)$, for $1 \le j \le n$, $1 \le \ell \le \mathcal{L}_j$. We show how it induces a solution to the modified linear program (P_{new}) as follows. If $J_j \in \mathcal{J}_\mathcal{S}$, then for $\ell = 1, \ldots, n^2$, $x^\ell(j) = \frac{1}{n^2} \sum_{r=1}^{\mathcal{L}_j} y^r(j)$.

If $J_j \in \mathcal{J}_\mathcal{L}$, then for each selected job occurrence J_j^ℓ, the variable $x^\ell(j)$ is the sum of all variables $y^\ell(r)$, over all job occurrences J_j^r such that the right endpoint of the first segment of J_j^r is in the same time slot as the right endpoint of the first segment of J_j^ℓ and the right endpoint of the second segment of J_j^r is in the same time slot as the right endpoint of the second segment of J_j^ℓ.

Suppose $J_j \in \mathcal{J}^1_{\mathcal{AL}}$. Recall that in Section 3.1 the occurrences of J_j were partitioned into two subsets, denoted as S_1 and S_2. (The subset S_2 includes the job occurrences we ignored in Section 3.1.) For each selected job occurrence J^ℓ_{n+j}, the variable $x^\ell(n+j)$ is the sum of all variables $y^\ell(r)$, over all job occurrences $J_j^r \in S_1$ such that the right endpoint of the first (second) segment of J_j^r is in the same time slot as the right endpoint of the first (second) segment of J_j^ℓ. Similarly, for each selected job occurrence J^ℓ_{n+2j}, the variable $x^\ell(n+2j)$ is the sum of all variables $y^\ell(r)$, over all $J_j^r \in S_2$ such that the right endpoint of the first segment of J_j^r is in the same time slot as the right endpoint of the first segment of J_j^ℓ.

Symmetrically, suppose $J_j \in \mathcal{J}^2_{\mathcal{AL}}$. For each selected job occurrence J^ℓ_{n+j}, the variable $x^\ell(n+j)$ is defined as above. For each selected job occurrence J^ℓ_{n+2j}, the variable $x^\ell(n+2j)$ is the sum of all variables $y^\ell(r)$, over all $J_j^r \in S_2$ such that the right endpoint of the second segment of J_j^r is in the same time slot as the right endpoint of the second segment of J_j^ℓ.

Finally, suppose $J_j \in \mathcal{J}_{\mathcal{AL}}$ for $\ell = 1, \ldots, n^2$, we have $x^\ell(n+3j) = \frac{1}{n^2} \sum_{J_j^r \in S_2} y^r(j)$.

It is straightforward to verify that the induced solution of the modified linear program (P_{new}) satisfies all constraints but the relaxed clique constraint. We show that the relaxed clique constraint is satisfied as well. First, we show that it is satisfied when we ignore the variables associated with Small jobs.

Consider a time slot s for M_1, and assume that at least one selected job occurrence J_j^ℓ has the right endpoint of its first segment is time slot s. Let $\mathcal{C}$ be the clique defined by this endpoint. Let S be the set of all job occurrences J_k^r whose first segment intersects time slot s. Clearly, $\sum_{J_k^r \in \mathcal{C}} x^r(k) \le \sum_{J_k^r \in S} y^r(r)$. Since all jobs are Large, the first segment of any $J_k^r \in S$ intersects at least one of the dividers that define time slot s. It follows $\sum_{J_k^r \in \mathcal{C}} x^r(k) \le 2$. The same argument holds for any time slot t for M_2.

Next, we show how the relaxed clique constraint is satisfied when we add the variables associated with Small jobs. Note that for each variable associated with a Small job J_j, $x^\ell(j) \le \frac{1}{n^2}$, for $1 \le \ell \le n^2$. Still, adding these variables may render the solution infeasible. Since for each Small job J_j, the job occurrences J_j^ℓ, $1 \le \ell \le n^2$ are nonoverlapping, any job clique $\mathcal{C}$ contains at most one segment out of all segments of the job occurrences J_j^ℓ, $1 \le \ell \le n^2$. Thus, the total sum of fractions assigned to Small jobs in any job clique $\mathcal{C}$ is at most $\frac{1}{n^2} \cdot n = \frac{1}{n}$. It follows that scaling the fractions assigned to Large jobs by a factor of $(1 - \frac{1}{n})$ will make the solution feasible. This scaling degrades the value of the objective function of the fractional solution by a factor of $(1 - \frac{1}{n})$.

3.5 Rounding the Solution of the Modified Linear Program

Since the clique constraint is relaxed, we need to reformulate Lemma 1.

▶ **Lemma 4.** *Let* $\mathbf{x}$ *be a feasible solution to* (P_{new}). *Then, there exists a job occurrence* J_j^ℓ *satisfying* $x^\ell(j) + \sum_{J_k^r \in \tilde{N}(J_j^\ell)} x^r(k) \le 9$.

Proof. The proof is similar to the proof of Lemma 1. We first show that there exists a selected job occurrence J_j^ℓ for which

$$x^\ell(j) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} x^r(k) \leq 8. \tag{2}$$

As before, we define $z(J_j^\ell, J_k^r) = x^\ell(j) \cdot x^r(k)$. The analysis is slightly different from the one in the proof of Lemma 1 since the first constraint in (P_{new}) is now relaxed. We omit the details. ◀

We apply Lemma 4 to obtain a mapping (as defined in Section 2.1). This can be done in polynomial time since we are guaranteed to have a polynomial number of nonzero variables that correspond to job occurrences. The mapping yields a schedule of a subset of jobs in $\mathcal{J}_{new}$ as defined in Section 2.1 with total weight $\frac{1}{9}$ of the objective value of the feasible solution of (P_{new}). Recall that this value is the objective value of the feasible solution of (P) scaled down by a factor of $(1 - \frac{1}{n})$. We summarize in the next theorem.

▶ **Theorem 5.** *There is a polynomial time $(9 + \epsilon)$-approximation algorithm for* MaxT *on two machines.*

4 Better Approximations for Special Cases on Two Machines

4.1 A 4-approximation Algorithm for Unit Weight Jobs with the Same Release Time

Consider instances of flow shop with two machines, in which all jobs have the same release time, i.e., $r_j = 0 \ \forall J_j \in \mathcal{J}$, arbitrary due dates, and unit weight. Below, we show that for such instances a simple algorithm yields an improved approximation ratio of 4 for MaxT. We note that the problem of maximizing throughput on a *single* machine with the same release times and unit job weights is solvable in polynomial time using Moore's algorithm [19]. We call this problem below $\mathsf{MaxT_S}$. Moore's algorithm can thus be used as a subroutine in our algorithm for MaxT, Split_the_Schedule. We give the pseudocode in Algorithm 2.

Algorithm 2 Split_the_Schedule.

1: For any job $J_j \in \mathcal{J}$ let $p_j = a_j + b_j$.
2: Solve optimally $\mathsf{MaxT_S}$, where each job J_j has a processing time p_j, a release time $r_j = 0$, and a due date d_j. Let SOL be the set of jobs in the solution.
3: Define the following flow shop schedule of SOL on M_1 and M_2: for any $J_j \in SOL$ that is processed on the single machine in $(s_j, t_j]$, process J_j on M_1 in $(s_j, s_j + a_j]$ and on M_2 in $(s_j + a_j, t_j]$.
4: Return the schedule of SOL on M_1 and M_2.

▶ **Theorem 6.** *Let OPT be the set of jobs in an optimal solution for* MaxT*. Then $|SOL| \geq \frac{|OPT|}{4}$.*

We use the following two lemmas to prove our main result in Theorem 6.

▶ **Lemma 7.** *For any instance of* MaxT *where $r_j = 0 \ \forall J_j \in \mathcal{J}$, there exists an optimal* permutation schedule*, i.e., a schedule where jobs are scheduled in the same order on both machines.*

Proof. Consider an optimal schedule that is not a permutation schedule, then we show that by swapping jobs we can obtain a feasible permutation schedule. Formally, given a schedule of the jobs on the two machines, we scan the schedule on M_2, starting from time $t = 0$. For any two consecutive jobs on M_2, J_k, J_j, if J_k precedes J_j on M_2, but J_j precedes J_k on M_1, we modify the schedule on M_1 as follows. Let s_j^1, s_k^1 the start-times of J_j, J_k on M_1, and t_k^1 the completion time of J_k on M_1 (see Figure 2). Then we schedule J_k on M_1 at time $t_k^1 - a_k - a_j$ and J_j at time $t_k^1 - a_j$. For M_1 we have:

- J_k completes processing earlier.
- J_j has a later completion time on M_1, but it still completes processing on M_2 by its due date. Indeed, as before, J_j starts processing on M_2 at time $s_j^2 \geq t_k^1$ and completes by d_j.
- For any other job J_r, $r \neq j, k$, the above swap can only result in an earlier completion time of J_r on M_1.

In addition, since we made no change on M_2, the schedule is still feasible. ◀

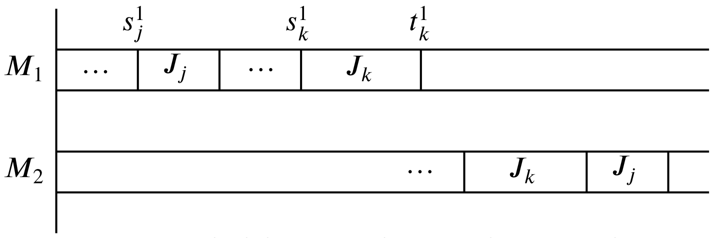

Figure 2 A non permutation schedule. J_j and J_k can be swapped on M_1 and scheduled consecutively, so that J_j completes processing on M_1 at time t_k^1.

▶ **Lemma 8.** *Let OPT be an optimal solution for a* MaxT *instance $\mathcal{J}$ for which there is a permutation schedule. Then there exists a subset of jobs $OPT_{single} \subseteq OPT$ satisfying:*

(i) *The jobs in OPT_{single} can be feasibly scheduled on a single machine, taking the processing time of each $J_j \in OPT_{single}$ to be $p_j = a_j + b_j$.*

(ii) $|OPT_{single}| \geq \frac{|OPT|}{4}$.

Proof. Consider an optimal subset of jobs, OPT, which has a permutation schedule. Assume w.l.o.g. that this permutation is the identity permutation. We now show how to move from a two machine schedule to a schedule of a subset of jobs in OPT, such that each job is completely processed either on M_1 or on M_2. We note that if $|OPT|$ is odd then we can always process the two segments of the last job on M_1. Hence, we assume from now on that $|OPT| = 2k$ for some integer $k \geq 1$. We now partition OPT to k pairs of jobs: $(J_1, J_2), \ldots, (J_{2i-1}, J_{2i}), \ldots$. Consider the jobs J_{2i-1}, J_{2i} with the processing times (a_{2i-1}, b_{2i-1}) and (a_{2i}, b_{2i}), respectively. We distinguish between two cases:

(i) If $a_{2i} > b_{2i-1}$ then we schedule J_{2i-1} on M_1 with processing time $p_{2i-1} = a_{2i-1} + b_{2i-1}$.

(ii) If $a_{2i} \leq b_{2i-1}$ then we schedule J_{2i} on M_2 with processing time $p_{2i} = a_{2i} + b_{2i}$.

We note that the schedules obtained on M_1 and M_2 are feasible. In addition, from each pair of jobs in OPT, one job is scheduled (either on M_1 or on M_2). Therefore, $|OPT|/2$ jobs are scheduled. Now, we choose the machine with a maximum number of jobs. This yields a solution consisting of at least $|OPT|/4$ jobs. ◀

Proof of Theorem 6. Recall that for an instance in which $r_j = 0 \ \forall J_j \in \mathcal{J}$ we can use Moore's algorithm to solve MaxT_S optimally. By Lemma 8, there exists a subset of $|OPT|/4$ jobs that can be scheduled feasibly on a single machine, where OPT is an optimal solution for MaxT on two machines. Since Moore's algorithm outputs an optimal solution on a single machine, we have the statement of the theorem. ◀

4.2 A $(3 + \varepsilon)$-approximation Algorithm for Jobs with the Same Release Time and Due Date

Consider instances of flow shop with two machines, in which all jobs have the same release time and the same due date. We assume below that for all $1 \le j \le n$, $r_j = 0$ and $d_j = T$, for some $T > 0$. We note that MaxT on such instances is NP-hard, as Knapsack is the special case where $b_j = 0$ for all $1 \le j \le n$.

Algorithm 3 below is a $(3+\varepsilon)$-approximation algorithm for such instances. The algorithm partitions the jobs into two groups: *large* and *small* jobs. For the large jobs the algorithm finds an optimal solution by applying an algorithm for makespan minimization in two-machine flow shop due to Johnson [16]. For the small jobs it finds a $(2+\varepsilon)$-approximation by applying a greedy algorithm for the *knapsack* problem. The algorithm then outputs the better of the two solutions, to yield a $(3 + \varepsilon)$-approximation.

Let $\Delta_j = \max\{a_j, b_j\}$ for each job J_j, $1 \le j \le n$. Also, let $\Delta_0 = 0$. For a set of jobs $\mathcal{J}$, define the weight of $\mathcal{J}$ to be $w(\mathcal{J}) = \sum_{J_j \in \mathcal{J}} w_j$.

Algorithm 3 Pack_and_Schedule.

Fix $0 < \varepsilon < 1$.
Let $\mathcal{L} = \{J_j \in \mathcal{J} \mid \Delta_j \ge \frac{\varepsilon T}{6}\}$ and $\mathcal{S} = \mathcal{J} \setminus \mathcal{L}$.
Let $M_L = 0$
for all $R \subseteq \mathcal{L}$ such that $|R| \le \frac{12}{\varepsilon}$ jobs **do**
    if R can be scheduled with makespan at most T **then** ▷ use Johnson's Algorithm [16] to check this condition
        if $w(R) > M_L$ **then**
            Let $SOL_L = R$
            Let $M_L = w(R)$
        end if
    end if
end for
Order the jobs in $\mathcal{S}$ in non-ascending order of the ratio $\frac{w_j}{\Delta_j}$. Assume w.l.o.g. that $\mathcal{S} = \{J_1, \ldots, J_{|\mathcal{S}|}\}$, and $\frac{w_1}{\Delta_1} \ge \frac{w_2}{\Delta_2} \ge \ldots \ge \frac{w_{|\mathcal{S}|}}{\Delta_{|\mathcal{S}|}}$
Find the maximum index k such that $\sum_{j=1}^{k} \Delta_j \le T(1 - \frac{\varepsilon}{6})$
Let $SOL_S = \{J_1, \ldots, J_k\}$ ▷ SOL_S can be scheduled feasibly as shown below
If $w(SOL_L) > w(SOL_S)$ then SOL $= SOL_L$; else SOL $= SOL_S$.
Return SOL

▶ **Theorem 9.** *For any fixed $0 < \varepsilon < 1$, Algorithm 3 runs in polynomial time and yields a $(3 + \varepsilon)$-approximation for MaxT on instances where $r_j = 0$ and $d_j = T$, $\forall j$.*

To prove Theorem 9 we need an observation and a few lemmas.

▶ **Observation 10.** Any feasible solution R of MaxT on instances where $r_j = 0$ and $d_j = T$ $\forall j$ satisfies $\sum_{J_j \in R} \Delta_j \le 2T$.

Proof. We note that $\sum_{J_j \in R} \Delta_j = \sum_{J_j \in R} \max\{a_j, b_j\} \le \sum_{J_j \in R} (a_j + b_j) \le 2T$. The last inequality follows from the fact that R can be scheduled feasibly. ◀

▶ **Lemma 11.** *The set SOL_L is an optimal solution for input $\mathcal{L}$.*

Proof. Since for every job $J_j \in \mathcal{L}$ we have $\Delta_j \geq \frac{\varepsilon T}{6}$, it follows from Observation 10 that any feasible solution for $\mathcal{L}$ cannot include more than $\frac{12}{\varepsilon}$ jobs. Since we enumerate over all feasible schedules with up to this number we are guaranteed to find the optimum. ◀

▶ **Lemma 12.** *The jobs in $SOL_S = \{J_1, \ldots, J_k\}$ can be scheduled feasibly.*

Proof. Sort the jobs $J_1, \ldots, J_k$ in non-ascending order of Δ_j. Let π be the resulting permutation; that is, $\Delta_{\pi(1)} \geq \Delta_{\pi(2)} \geq \ldots \geq \Delta_{\pi(k)}$. Let $\pi(0) = 0$.

Schedule job $J_{\pi(j)}$ at time $t_1^j = \sum_{i=0}^{j-1} \Delta_{\pi(i)}$ on M_1 and at time $t_2^j = t_1^j + \Delta_{\pi(1)}$ on M_2. The schedule is feasible since (i) no two jobs overlap in any of the machines (recall that $\Delta_j = \max\{a_j, b_j\}$), (ii) the makespan of the schedule is $\Delta_{\pi(1)} + \sum_{j=1}^{k} \Delta_{\pi(j)} \leq T$ since $\Delta_{\pi(1)} < \frac{\varepsilon T}{6}$, and (iii) for any job J_j, its segment on M_2 is executed after the completion of its segment on M_1, since the schedule on M_2 is shifted by $\Delta_{\pi(1)} = \max_{j \in [1..k]} \Delta_j$. ◀

▶ **Lemma 13.** *The set SOL_S is a $(2+\varepsilon)$-approximation of the optimal solution for input $\mathcal{S}$.*

Proof. First, consider a knapsack problem with set of items corresponding to the jobs in $\mathcal{S}$, where the size of item j is Δ_j and its weight is w_j. Assume that the knapsack capacity is T. We claim that the weight of the optimal solution to this knapsack problem has weight that is at least $\frac{1}{2}$ of the weight of optimal solution for input $\mathcal{S}$. To see this consider an optimal solution for input $\mathcal{S}$ and partition the set of jobs in this solution into two disjoint sets: the first set O_1 consists of all jobs J_j in the solution for which $\Delta_j = a_j$, and the second set O_2 consists of all jobs J_j in the solution for which $\Delta_j > a_j$ (and $\Delta_j = b_j$). Let i be the index of the set whose total weight is larger; that is $w(O_i) \geq w(O_{3-i})$. Clearly $w(O_i)$ is at least half the optimum. Since we start from a feasible solution, the total size of the items corresponding to the jobs in O_i is bounded by T. Thus, there is a feasible solution to the knapsack problem with weight that is at least $\frac{1}{2}$ of the weight of optimal solution for input $\mathcal{S}$. Note that we may not be able to feasibly schedule the set of jobs corresponding to the items in an optimal solution of this knapsack problem.

From the way we chose k and since for all $J_j \in \mathcal{S}$, $\Delta_j < \frac{\varepsilon T}{6}$, it follows that $\sum_{j=1}^{k} \Delta_j > T(1 - \frac{\varepsilon}{3})$. Since the jobs are sorted in non-ascending order of the ratio $\frac{w_j}{\Delta_j}$, we are guaranteed that $w(SOL_S)$ is at least $(1 - \frac{\varepsilon}{3})$ of the weight of the optimal solution to the knapsack problem and thus it is at least $\frac{1}{2}(1 - \frac{\varepsilon}{3})$ of the weight of the optimal solution for input $\mathcal{S}$. Since $(2+\varepsilon) \cdot \frac{1}{2}(1 - \frac{\varepsilon}{3}) = 1 + \frac{1}{6}(\varepsilon - \varepsilon^2) \geq 1$, for $0 < \varepsilon < 1$, it follows that SOL_S is a $(2+\varepsilon)$-approximation of the optimal solution for input $\mathcal{S}$. ◀

▶ **Lemma 14.** *The time complexity of Algorithm 3 is $O(n^{\frac{12}{\varepsilon}})$.*

Proof of Theorem 9. Consider an optimal solution O for input $\mathcal{J}$, and partition the jobs in this solution into two disjoint sets $O_L = O \cap \mathcal{L}$ and $O_S = O \cap \mathcal{S}$. By Lemmas 11 and 13, we have that $w(O_L) \leq w(\mathsf{SOL})$ and $w(O_S) \leq (2+\varepsilon) \cdot w(\mathsf{SOL})$. It follows that $w(O) \leq (3+\varepsilon) \cdot w(\mathsf{SOL})$.

By the algorithm and Lemma 12 the jobs in SOL can be scheduled feasibly, and by Lemma 14 the running time is polynomial in n. The theorem follows. ◀

▶ **Corollary 15.** *If $a_j \leq b_j$, for all $J_j \in \mathcal{J}$, or $a_j \geq b_j$, for all $J_j \in \mathcal{J}$, then, for any fixed $0 < \varepsilon < 1$, Algorithm 3 is a $(2+\varepsilon)$-approximation algorithm for MaxT on instances where $r_j = 0$ and $d_j = T$, $\forall j$.*

Proof. It is easy to see that if any of the conditions in the corollary hold then the weight of an optimal solution to the knapsack problem defined in Lemma 13 is at least the weight of the optimal solution for input $\mathcal{S}$, and thus the set SOL_S is a $(1+\varepsilon)$-approximation of the optimal solution for input $\mathcal{S}$. ◀

References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallelizable jobs online to maximize throughput. In *Latin American Symposium on Theoretical Informatics*, pages 755–776. Springer, 2018.

2 Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Baruch Schieber, and Cliff Stein. Non-preemptive min-sum scheduling with resource augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 614–624. IEEE, 2007.

3 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

4 Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001.

5 Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36(1):1–15, 2006.

6 Michael G Bechtel, Elise McEllhiney, Minje Kim, and Heechul Yun. Deeppicar: A low-cost deep neural network-based autonomous car. In *RTCSA*, pages 11–21, 2018.

7 Bo Chen, Chris N Potts, and Gerhard J Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization*, pages 1493–1641. Springer, 1998.

8 Byung-Cheon Choi and Suk-Hun Yoon. Maximizing the weighted number of just-in-time jobs in flow shop scheduling. *Journal of Scheduling*, 10(4-5):237–243, 2007.

9 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research*, 31(4):730–738, 2006.

10 Amir Elalouf, Eugene Levner, and Huajun Tang. An improved FPTAS for maximizing the weighted number of just-in-time jobs in a two-machine flow shop problem. *Journal of Scheduling*, 16(4):429–435, 2013.

11 Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

12 Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

13 Leslie A. Hall. Approximability of flow shop scheduling. *Math. Program.*, 82:175–190, 1998.

14 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-dimensional matching. In *Proceedings of APPROX*, pages 83–97. Springer, 2003.

15 Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1070–1086. SIAM, 2014.

16 Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

17 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM (JACM)*, 47(4):617–643, 2000.

18 Monaldo Mastrolilli and Ola Svensson. Hardness of approximating flow and job shop scheduling problems. *Journal of the ACM (JACM)*, 58(5):20, 2011.

19 J Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, 15(1):102–109, 1968.

20 Viswanath Nagarajan and Maxim Sviridenko. Tight bounds for permutation flow shop scheduling. *Mathematics of Operations Research*, 34(2):417–427, 2009.

21 Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

22 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

23 Dvir Shabtay and Yaron Bensoussan. Maximizing the weighted number of just-in-time jobs in several two-machine scheduling systems. *Journal of Scheduling*, 15(1):39–47, 2012.

24 Maxim Sviridenko. A note on permutation flow shop problem. *Annals of Operations Research*, 129(1-4):247–252, 2004.

25 Gerhard J Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

26 Yiyong Xiao, Siyue Zhang, Pei Yang, Meng You, and Jiaoying Huang. A two-stage flow-shop scheme for the multi-satellite observation and data-downlink scheduling problem considering weather uncertainties. *Reliability Engineering & System Safety*, 188:263–275, 2019.

A Some Proofs

Proof of Lemma 1. We first show that there exists a job occurrence J_j^ℓ for which the following holds:

$$x^\ell(j) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} x^r(k) \leq 2m. \tag{3}$$

For two "neighboring" job occurrences J_j^ℓ and J_k^r (i.e., $J_k^r \in \tilde{N}_1(J_j^\ell)$ and $J_j^\ell \in \tilde{N}_1(J_k^r)$), define $z(J_j^\ell, J_k^r) = x^\ell(j) \cdot x^r(k)$. We also define $z(J_j^\ell, J_j^\ell) = (x^\ell(j))^2$. We prove (3) using a weighted averaging argument, where the weights are the values $z(J_j^\ell, J_k^r)$ for all pairs of job occurrences which have intersecting segments.

Consider the sum $\sum_{j=1}^n \sum_{\ell=1}^{\mathcal{L}_j} \left(z(J_j^\ell, J_j^\ell) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} z(J_j^\ell, J_k^r) \right)$. We upper bound this sum as follows. Let $\mathcal{I}(J_j^\ell)$ denote the set of segments of a job occurrence J_j^ℓ. For each job occurrence J_j^ℓ, we consider all of its segments $I \in \mathcal{I}(J_j^\ell)$. For each such segment I, we sum up $z(J_j^\ell, J_k^r)$ for all job occurrences J_k^r having at least one segment that intersects with I (including J_j^ℓ itself). Let $R(J_j^\ell, I)$ be the set of job occurrences that have a segment intersecting the right endpoint of I (including J_j^ℓ itself). We note that it suffices to sum up $z(J_j^\ell, J_k^r)$ only for job occurrences $J_k^r \in R(J_j^\ell, I)$ and then multiply the total sum by 2. This is because, for the intersecting segment I of J_j^ℓ and segment I' of J_k^r, if the right endpoint of I precedes the right endpoint of I', then $J_k^r \in R(J_j^\ell, I)$; otherwise, $J_j^\ell \in R(J_k^r, I')$. Since $z(J_j^\ell, J_k^r) = z(J_k^r, J_j^\ell)$, each of them contributes the same value to the other. Therefore, it follows that

$$\sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{L}_j|} \left(z(J_j^\ell, J_j^\ell) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} z(J_j^\ell, J_k^r) \right) \leq 2 \sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{L}_j|} \sum_{I \in \mathcal{I}(J_j^\ell)} \sum_{J_k^r \in R(J_j^\ell, I)} z(J_j^\ell, J_k^r). \tag{4}$$

By the first constraint in (P), the definition of job cliques and the definition of $z(J_j^\ell, J_k^r)$, we have

$$\sum_{J_k^r \in R(J_j^\ell, I)} z(J_j^\ell, J_k^r) \leq \ x^\ell(j) \cdot \sum_{J_k^r \in R(J_j^\ell, I)} x^r(k) \leq x^\ell(j). \tag{5}$$

Using (4), (5), and the fact that $|\mathcal{I}(J_j^\ell)| \leq m$, we get that

$$\sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{L}_j|} \left(z(J_j^\ell, J_j^\ell) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} z(J_j^\ell, J_k^r) \right) \leq 2 \sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{L}_j|} \sum_{I \in \mathcal{I}(J_j^\ell)} x^\ell(j) \leq 2m \sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{L}_j|} x^\ell(j).$$

Hence, there exists a job occurrence J_j^ℓ satisfying

$$z(J_j^\ell, J_j^\ell) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} z(J_j^\ell, J_k^r) = (x^\ell(j))^2 + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} x^r(k) x^\ell(j) \leq 2m \cdot x^\ell(j).$$

By factoring out $x^\ell(j)$ from both sides we get inequality (3).

To complete the proof of the lemma, we note that for a job J_j^ℓ satisfying (3) it also holds that

$$x^\ell(j) + \sum_{J_k^r \in \tilde{N}(J_j^\ell)} x^r(k) = x^\ell(j) + \sum_{J_k^r \in \tilde{N}_1(J_j^\ell)} x^r(k) + \sum_{J_k^r \in \tilde{N}_2(J_j^\ell)} x^r(k) \leq 2m + \sum_{r=1}^{|\mathcal{L}_j|} x^r(j) \leq 2m+1.$$

The last inequality follows from the second constraint in (P). ◀

Proof of Lemma 14. It is easy to see that the most time consuming part is the loop defined in Step 4 where we enumerate over all subsets of $\mathcal{L}$ of size at most $\frac{12}{\varepsilon}$ and thus the time complexity. ◀