Maximizing Throughput in Flow Shop Real-time Scheduling

Lior Ben-Yamin
Maximizing Throughput in Flow Shop Real-time Scheduling

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Lior Ben-Yamin

Submitted to the Senate of the Technion — Israel Institute of Technology
Adar 5782 Haifa February 2022
This research was carried out under the supervision of Prof. Hadas Shachnai, in the Department of Computer Science.

Some results in this thesis have been published as an article by the author and research collaborators in a conference during the course of the author’s masters research period, the most up-to-date version of which being:


The generous financial help of the Technion is gratefully acknowledged.
Contents

List of Figures

Abstract

1 Introduction
  1.1 Problem Definition
  1.2 Applications
  1.3 Contributions and Techniques

2 Related Work

3 Preliminaries

4 Approximation Algorithm for Fixed Number of Machines
  4.1 The Algorithm

5 Approximating MaxT on Two Machines
  5.1 Eliminating the Almost-Large Jobs
  5.2 The Selected Occurrences of Small and Large Jobs
  5.3 The Modified Linear Program
  5.4 The Induced Solution of the Modified Linear Program
  5.5 Rounding the Solution of the Modified Linear Program

6 Better Approximations for Special Cases on Two Machines
  6.1 A 4-approximation Algorithm for Unit Weight Jobs with the Same Release Time
  6.2 A (3 + ε)-approximation Algorithm for Jobs with the Same Release Time and Due Date

7 No-Wait Flowshop

8 Conclusion and Future Work
  8.1 Future Directions

Hebrew Abstract

Technion - Computer Science Department - M.Sc. Thesis MSC-2022-12 - 2022
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>An input for two machine flow shop: $a_j, b_j$ are the processing times of $J_j$ on $M_1$ and $M_2$, respectively; also, $J_j$ has release time $r_j$, due date $d_j$ and weight $w_j$, $1 \leq j \leq 3$.</td>
<td>9</td>
</tr>
<tr>
<td>3.2</td>
<td>An optimal permutation schedule of total weight 16</td>
<td>9</td>
</tr>
<tr>
<td>3.3</td>
<td>An optimal schedule of total weight 24</td>
<td>10</td>
</tr>
<tr>
<td>4.1</td>
<td>The job clique $(J_1^3, J_2^3, J_1^4, J_1^4)$ is defined by $z$, the right endpoint of the $i$th segment of $J_i^3$. The clique contains two occurrences of $J_1$: $J_1^3$ and $J_1^4$. The next clique, $(J_1^2, J_1^4)$, is defined by the segment having its right endpoint at $z'$.</td>
<td>12</td>
</tr>
<tr>
<td>6.1</td>
<td>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 $t_k^1$.</td>
<td>26</td>
</tr>
<tr>
<td>7.1</td>
<td>The transformation in the proof of Lemma 7.2 to a schedule where each job is completely processed on one of the machines.</td>
<td>32</td>
</tr>
</tbody>
</table>
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. 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.
Chapter 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. Specifically, 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 \( 2 \leq i \leq m \).

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 [Joh54], 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 [CPW98]. 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 [GJS76]. Hence, later works studied approximation algorithms for the problem (see, e.g., [SSS95, Hal98, Svi04, NS09, MS11]).

A closely related scheduling model is permutation flow shop, in which it is also re-
quired that each machine processes the set of all jobs in the same order. For makespan minimization, Johnson [Joh54] showed the existence of a permutation schedule in two machine flow shop. The existence of an optimal permutation schedule for three machines follows from a results of Conway et al. [CMM03]. However, it is easy to construct an instance with $m = 4$ machines where no permutation schedule is optimal for the flow shop problem. In fact, in [PSW91] Potts showed a family of instances for which the makespan of the best permutation schedule is worse than the optimum by a factor of more than $\frac{1}{2}\sqrt{m}$, where $m$ is the number of machines. Sviridenko [Svi04] presented an $O(\sqrt{m\log m})$-approximation algorithm for the permutation flow shop problem with the makespan objective. Nagarajan and Sviridenko [NS09] improved the ratio to $O(\min\{m^{\frac{1}{2}},n^{\frac{1}{2}}\})$.

A special case of the permutation flow shop is no-wait flow shop. In this model, once a job completes its execution on a machine it must immediately start running on the next machine. This model is widely applied in various industries, such as food-processing and industrial pipelines (see more at [HS96]). Most of the research work dealt with the objective of minimizing the makespan. For the case of two machines, Gilmore and Gomory [GG64] presented an optimal $O(n\log(n))$ algorithm for the problem.

In this work, 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., [Moo68, BGNS01, BBF01, COR06, ALLM18, BCK07, ILMT14, Woe94, KP00]).

### 1.1 Problem Definition

We now formalize our problem. In the $m$-machine flow shop model, there is a set of $n$ jobs, $J = \{J_1, \ldots, J_n\}$, and $m$ machines, $M_1, \ldots, M_m$. Each job $J_j$, $1 \leq j \leq n$, has a release time, a due date, and a weight, given by $r_j \geq 0$, $d_j \geq 0$, and $w_j > 0$, respectively. A job can only start executing on machine $M_i$, $2 \leq i \leq m$, after it has finished its execution on the previous machine $M_{i-1}$. In addition, at any time $t \geq 0$, 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 $J' \subseteq J$ 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 $\text{MaxT}$. We obtain results for $\text{MaxT}$ in the $m$-machine flow shop model, with a focus on the special case of two-machine flow shop.

1.2 Applications

In the following, we motivate $\text{MaxT}$ 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., [XZY+19]). 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 $\text{MaxT}$ 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 $\text{MaxT}$ in the two-machine flow shop model (see, e.g., [XZY+19]).

**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., [BMKY18]). 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 $\text{MaxT}$ instance in the flow shop model.

1.3 Contributions and Techniques

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

In this work, we study the fundamental problem of throughput maximization in the
flow shop scheduling model. Our main result (in Chapter 5) 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., \(n = |J|\)). 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 \(\Omega(\frac{m}{\log m})\) [HSS03].

We show (in Chapter 6) 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 \(J_j \in J\), 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 \(\varepsilon > 0\).

Finally, in Chapter 7 we give a simple polynomial-time \((2m + \varepsilon)\)-approximation algorithm for general MaxT instances in the no-wait flow shop model, for any \(\varepsilon > 0\) and arbitrary \(m > 1\). In the special case where all jobs have the same release time, we obtain a better ratio of \(m + \varepsilon\).

Techniques: In Chapter 4, 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 \(J\) can be modified to an instance \(J_{new}\) (by replacing some jobs with new jobs) in which for every job \(J_j \in J_{new}\) 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 \(J_{new}\) to be of size polynomial in \(|J_{new}|\) (and consequently also in \(|J|\), 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 Chapter 5). 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 6.1). This allows the use of approximation algorithms for the single machine case for solving our problem.
Chapter 2

Related 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 [GJ79]. Some special cases of the problem are known to be solvable in polynomial time. Moore [Moo68] 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 [Sah76] a fully polynomial time approximation scheme (FPTAS), whose running time is $O(n^3/e)$, for the more general case where jobs have the same release time, arbitrary due dates, and arbitrary weights.

Bar-Noy et al. [BGNS01, BBF01] 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 [BBF01] a $(2+\varepsilon)$-approximation algorithms, using the local ratio technique. A quasi-polynomial time dynamic programming framework was proposed in [ILMT14], 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 $\frac{e}{1+\frac{1}{m}} < 1.582$ approximation algorithm [COR06], 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 $i$th segments intersect. For this problem, the paper [BHN06] 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 $\Omega(\frac{m}{\log m})$ \cite{HSS03}. 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 \cite{BHN+06}. However, further steps are required to obtain polynomial running time, which is the main contribution of this work.

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 \cite{CY07} 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 \cite{ELT13} (see also \cite{SB12}).

We are not aware of earlier studies of throughput maximization in the flow shop model.
Chapter 3

Preliminaries

Our study includes also no-wait flow shop instances, in which the segments of each job are processed by the \( m \) machines one after the other with no delay. It is easy to verify that in any no-wait schedule, the jobs are processed in the same order on all machines; namely, the schedule defines a permutation of the \( n \) jobs. Thus, any optimal no-wait flow shop schedule for MaxT is a permutation schedule. A natural question is whether general MaxT instances also admit optimal permutation schedules. We show below that this is not true, already in the two machine case. Consider the instance outlined in Figure 3.1. Figure 3.2 illustrates an optimal permutation schedule for this instance, of total weight 16. In Figure 3.3 we give an optimal schedule of total weight 24.

\[
\begin{array}{ccccc}
 a_j & b_j & r_j & d_j & w_j \\
 J_1 & 5 & 5 & 0 & 20 & 6 \\
 J_2 & 3 & 3 & 5 & 15 & 8 \\
 J_3 & 2 & 2 & 8 & 12 & 10 \\
\end{array}
\]

Figure 3.1: An input for two machine flow shop: \( a_j, b_j \) are the processing times of \( J_j \) on \( M_1 \) and \( M_2 \), respectively; also, \( J_j \) has release time \( r_j \), due date \( d_j \) and weight \( w_j \), \( 1 \leq j \leq 3 \).

\[
\begin{array}{ccccccc}
 M_1 & J_1 & J_3 \\
 0 & 5 & 8 & 10 & 12 & 15 & 20 \\
 M_2 & J_1 & J_3 \\
 0 & 5 & 8 & 10 & 12 & 15 & 20 \\
 J_1 & J_2 & J_3 \\
 0 & 5 & 8 & 10 & 12 & 15 & 20 \\
\end{array}
\]

Figure 3.2: An optimal permutation schedule of total weight 16.
Figure 3.3: An optimal schedule of total weight 24
Chapter 4

Approximation Algorithm for Fixed Number of Machines

In this chapter, 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 \( J \), 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. Given these integral values, 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 \( (d_j - r_j)^m \). Clearly, at most one occurrence can be scheduled for each job.

We give some notation towards solving MaxT on \( m \) machines using a linear program. Let \( L_j \) denote the set of occurrences of job \( J_j \), so the number of occurrences of \( J_j \) is \( |L_j| \). Let \( L = \bigcup_{j=1}^{n} L_j \). Clearly, \( |L| = \sum_{j=1}^{n} |L_j| \). Let \( x^{\ell}(j) \in \{0,1\} \) be an indicator variable for the selection of the \( \ell \)-th occurrence \( J^\ell_j \) of \( J_j \) to the solution, where \( 1 \leq \ell \leq |L_j| \).

We note that the number of variables and the number of 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 \( w \in \mathbb{R}^n \), \( x \in \mathbb{R}^{|L|} \) be a weight vector and a relaxed indicator vector, respectively\(^1\). Then, \( w \cdot x = \sum_{j=1}^{n} \sum_{\ell=1}^{|L_j|} w_j x^{\ell}(j) \).

We say that a subset of job occurrences forms a job clique \( C \) on machine \( i \), for some

\(^1\)In the relaxed indicator vector \( x \in \mathbb{R}^{|L|} \) each component takes a value in \([0,1]\).
Figure 4.1: The job clique \((J_1^3, J_2^3, J_1^2, J_1^1)\) is defined by \(z\), the right endpoint of the \(i\)th segment of \(J_1^3\). The clique contains two occurrences of \(J_1^1\): \(J_1^3\) and \(J_2^1\). The next clique, \((J_2^2, J_2^1)\), is defined by the segment having its right endpoint at \(z'\).

1 \(\leq i \leq m\), if each pair of job occurrences \(J_{j}^{k}, J_{k}^{j}\) in this subset, the \(i\)th segments of \(J_{j}^{i}\) and \(J_{k}^{i}\) intersect. We note that the set of job cliques of a given instance is defined by the set of right endpoints for the \(i\)th segments of all job occurrences. Indeed, from each clique \(C\) on machine \(i\), at most one job occurrence can be scheduled. This is the heart of the linear program relaxation that we use to solve our problem. As it suffices to consider only maximal job cliques (i.e., job cliques containing a maximal subset of mutually intersecting job occurrences on machine \(i\)), in fact we do not need to consider all the right endpoints of \(i\)th segments.

Thus, we define the 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 \(C\) consisting of all job occurrences whose \(i\)th segment intersects the time point \(z\). The next clique is defined by the earliest right endpoint \(z'\) of an \(i\)th segment of a job, for which the following holds: there exists a job occurrence \(J_{k}^{j}\), such that the \(i\)th segment of \(J_{k}^{i}\) intersects \(z'\), but \(J_{k}^{j} \notin C\), as shown in Figure 4.1. 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.

\[
\begin{align*}
\text{(P)} \quad & \text{maximize} \quad w \cdot x \\
\text{subject to :} \\
\sum_{j \in C} x_j^i & \leq 1 \quad \text{for each clique } C \\
\sum_{\ell=1}^{\left|\mathcal{L}_i\right|} x_{\ell}^j & \leq 1 \quad \forall 1 \leq j \leq n \\
x_{\ell}^j & \geq 0 \quad \forall 1 \leq j \leq n, \ 1 \leq \ell \leq \left|\mathcal{L}_i\right|
\end{align*}
\]

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, \( J^*_j \), defines a configuration, \( \forall J_j \in \mathcal{J} \).

Considering the neighbors of a job occurrence \( J^*_j \), we define two subsets of job occurrences.

1. Let \( \tilde{N}_1(J^*_j) \) be the set of all job occurrences \( J^*_k \) where \( k \neq j \), such that a segment of \( J^*_k \) intersects a segment of \( J^*_j \) (we note that two segments can intersect only if both need to be processed on the same machine).

2. Let \( \tilde{N}_2(J^*_j) \) be the set of all job occurrences \( J^*_r \) where \( r \neq \ell \), i.e., other occurrences of \( J_j \).

The neighborhood of a job occurrence \( J^*_j \) is defined as \( \tilde{N}(J^*_j) = \tilde{N}_1(J^*_j) \cup \tilde{N}_2(J^*_j) \).

**Lemma 4.1.** Let \( \mathbf{x} \) be a feasible solution to (P). For some \( 1 \leq j \leq n \) there exists a job occurrence \( J^*_j \) satisfying:

\[
x^\ell(j) + \sum_{J^*_k \in \tilde{N}(J^*_j)} x^r(k) \leq 2m + 1.
\]

**Proof** We first show that there exists a job occurrence \( J^*_j \) for which the following holds:

\[
x^\ell(j) + \sum_{J^*_k \in \tilde{N}_1(J^*_j)} x^r(k) \leq 2m. \tag{4.1}
\]

For two “neighboring” job occurrences \( J^*_j \) and \( J^*_k \) (i.e., \( J^*_k \in \tilde{N}_1(J^*_j) \) and \( J^*_j \in \tilde{N}_1(J^*_k) \)), define \( z(J^*_j, J^*_k) = x^\ell(j) \cdot x^r(k) \). We also define \( z(J^*_j, J^*_k) = (x^\ell(j))^2 \). We prove (4.1) using a weighted averaging argument, where the weights are the values \( z(J^*_j, J^*_k) \) for all pairs of job occurrences which have intersecting segments.

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

\[
\sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{E}_j|} \left( z(J^*_j, J^*_j) + \sum_{J^*_k \in \tilde{N}_1(J^*_j)} z(J^*_j, J^*_k) \right) \leq 2 \sum_{j=1}^{n} \sum_{\ell=1}^{|\mathcal{E}_j|} \sum_{I \in \mathcal{I}(J^*_j)} \sum_{J^*_k \in R(J^*_j, I)} z(J^*_j, J^*_k). \tag{4.2}
\]
By the first constraint in (P), the definition of job cliques and the definition of \( z(J^*_j, J^*_k) \), we have
\[
\sum_{J^*_l \in R(J^*_j, I)} z(J^*_j, J^*_l) \leq x^F(j), \quad \sum_{J^*_l \in R(J^*_j, I)} x^r(k) \leq x^F(j). \tag{4.3}
\]

Using (4.2), (4.3), and the fact that \(|\mathcal{I}(J^*_j)| \leq m\), we get that
\[
\sum_{j=1}^{n} \sum_{\ell=1}^{L_j} \left( z(J^*_j, J^*_j) + \sum_{J^*_l \in N_1(J^*_j)} z(J^*_j, J^*_l) \right) \leq 2 \sum_{j=1}^{n} \sum_{\ell=1}^{L_j} \sum_{i \in \mathcal{I}(J^*_j)} x^F(j) \leq 2m \sum_{j=1}^{n} \sum_{\ell=1}^{L_j} x^F(j).
\]

Hence, there exists a job occurrence \( J^*_j \) satisfying
\[
z(J^*_j, J^*_j) + \sum_{J^*_l \in N_1(J^*_j)} z(J^*_j, J^*_l) = (x^F(j))^2 + \sum_{J^*_l \in N_1(J^*_j)} x^r(k)x^F(j) \leq 2m \cdot x^F(j).
\]

By factoring out \( x^F(j) \) from both sides we get inequality (4.1).

To complete the proof of the lemma, we note that for a job \( J^*_j \) satisfying (4.1) it also holds that
\[
x^F(j) + \sum_{J^*_l \in N_1(J^*_j)} x^r(k) = x^F(j) + \sum_{J^*_l \in N_1(J^*_j)} x^r(k) + \sum_{J^*_l \in N_2(J^*_j)} x^r(k) \leq 2m + \sum_{r=1}^{L_j} x^r(j) \leq 2m + 1.
\]

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

### 4.1 The Algorithm

We now show how to use Lemma 4.1 to get a \((2m + 1)\)-approximation for \( \text{MaxT} \) 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 \( J^*_j \) the following properties are satisfied:

1. All the subintervals in \( \psi(J^*_j) \) are disjoint.
2. The total size of the subintervals in \( \psi(J^*_j) \) is \( x^F(j) \).
3. None of the subintervals in \( \psi(J^*_j) \) intersects any of the subintervals in \( \bigcup_{J^*_l \in N(J^*_j)} \psi(J^*_l) \).

The mapping is constructed for one job occurrence at a time according to a hierarchical order induced by Lemma 4.1. We first define this hierarchical order. The last job occurrence in the order is the occurrence \( J^*_j \) that satisfies the inequality of Lemma 4.1. 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 4.1 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 \( \psi(J_j^f) \) for one job occurrence at a time from the first to the last in the hierarchical order defined above. When \( \psi(J_j^f) \) is computed, we remove from the interval \((0, 2m + 1]\) all the subintervals in \( \bigcup_{j \in N} \psi(J_j^f) \), where \( N \subseteq \tilde{N}(J_j^f) \) is the set of all job occurrences in \( \tilde{N}(J_j^f) \) that precede \( J_j^f \) in the hierarchical order. By Lemma 4.1, the total size of these subintervals is no more than \( 2m + 1 - x^f(j) \). Thus, the remainder contains a set of disjoint subintervals of a total size at least \( x^f(j) \). If we assign \( \psi(J_j^f) \) greedily, that is, we assign the leftmost collection of such disjoint subintervals, then it can be shown that \(| \psi(J_j^f) |\) is bounded by \(|L|\). 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 \( \phi(y) \subseteq L \) be the subset of \( L \) consisting of all job occurrences \( J_j^f \) for which one of the subintervals in \( \psi(J_j^f) \) contains the point \( y \). From the definition of the mapping \( \psi \), 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 \( w(y) = \sum_{j \in \phi(y)} w_j \). Let \( y^* = \arg \max_{y \in (0, 2m + 1]} \{ w(y) \} \). Note that if the mapping is computed greedily there are at most \(|L|^2\) 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 given in Algorithm 4.1.

---

**Algorithm 4.1 Flowshop_Time_Windows**

1: Find an optimal solution \( x \) for the linear program \((P)\).
2: Order the job occurrences according to the hierarchical order.
3: **for** each job occurrence \( J_j^f \) in order **do**
4: \hspace{1cm} Remove from the interval \((0, 2m + 1]\) all subintervals assigned to “neighbors” of \( J_j^f \) that precede it in the hierarchical order.
5: \hspace{1cm} Assign to \( J_j^f \) the leftmost collection of subintervals of total size \( x^f(j) \).
6: **end for**
7: Let \( \text{max}\,wy = 0 \).
8: **for** a point \( y \) that is a right endpoint of a subinterval **do**
9: \hspace{1cm} Let \( \phi(y) \subseteq L \) be the subset of all job occurrences \( J_j^f \) for which one of the subintervals in \( \psi(J_j^f) \) contains the point \( y \).
10: \hspace{1cm} Let \( w(y) = \sum_{j \in \phi(y)} w_j \).
11: \hspace{1cm} **if** \( w(y) > \text{max}\,wy \) **then**
12: \hspace{1cm} \hspace{1cm} Let \( \text{max}\,wy = w(y) \) and \( y^* = y \).
13: \hspace{1cm} **end if**
14: **end for**
15: Return \( \phi(y^*) \).

---

**Theorem 4.2.** 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,

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

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

**Corollary 4.3.** There is a pseudo-polynomial time $(2m + 1)$-approximation algorithm for MaxT on $m$ machines, where $m \geq 2$ is some constant.
Chapter 5

Approximating MaxT on Two Machines

We now show that, with a slight degradation of the approximation ratio, we can use the algorithm presented in Chapter 4 to obtain a polynomial-time algorithm for \( m = 2 \).

We start with some notation. Consider two machines, \( M_1 \) and \( M_2 \), where each job consists of two continuous 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 \( J_j \in J \) 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 \( J' \subseteq J \) in the flow shop model, \( J_j \in J' \) 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 completed no later than its due date \( d_j \).

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

(i) **Small jobs** \( J_S \): 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** \( J_L \): 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** \( J_{AL} \): Job \( J_j \) is an 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 Chapter 4 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 (\( P_{\text{new}} \)) 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 for the modified problem.
of \((P_{\text{new}})\) with a slight degradation in the value of the objective function. Finally, we show how a feasible solution of \((P_{\text{new}})\) can be “rounded” to a schedule whose weight is \(\frac{1}{9}\) of the objective value of this feasible solution of \((P_{\text{new}})\). This schedule is a \((9 + \epsilon)\)-approximation of the optimal solution.

## 5.1 Eliminating the Almost-Large Jobs

We first partition the set \(J_{AL}\) of \textit{Almost-Large} jobs into two subsets \(J_{AL}^1\) and \(J_{AL}^2\).

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

For any job \(J_j \in J_{AL}^1\), consider the time point \(t_j = d_j - a_j - (2n^2 - 1)b_j\), i.e., \(b_j = \frac{d_j - t_j - a_j - b_j}{2n^2 - 2}\). Partition the occurrences of \(J_j\) into two subsets. The first subset, \(L_j^1\), consists of all occurrences \(J_j^1\) in which the first segment of \(J_j^1\) starts (on \(M_1\)) at or after \(t_j\), and the complement subset, \(L_j^2 = L_j \setminus L_j^1\), consists of all occurrences \(J_j^2\) in which the first segment of \(J_j^2\) starts processing before time \(t_j\). We replace job \(J_j\) by three new jobs as follows.

Note that the job occurrences in \(L_j^1\) 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 \textit{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 \(L_j^2\), we ignore (for now) all the occurrences where the second segment starts before \(t_j + a_j\). Let \(L_j^2\) be the rest of the job occurrences in this set. Then the job occurrences in \(L_j^2\) 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 \(a_j \geq \frac{d_j - t_j - a_j - b_j}{2n^2 - 2}\), we have \(a_j \geq \frac{t_j + a_j - r_j - a_j}{2n^2 - 2}\). Thus, the job \(J_{2n+j}\) is \textit{Large}. The job \(J_{3n+j}\) is \textit{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 \(J_j \in J_{AL}^2\) 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, \(L_j^1\), consists of all occurrences where the second segment ends at or before \(t_j\), and the complement subset, \(L_j^2\), consists of all occurrences where the second segment ends processing (on \(M_2\)) after time \(t_j\). The job occurrences in \(L_j^1\) are the same as the job occurrences of a new 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 \textit{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 \(L_j^2\), we again ignore (for now) all the occurrences where the first segment finishes after \(t_j - b_j\). Let \(L_j^2\) be the rest of the job occurrences in this set.
Then the job occurrences in $\mathcal{L}_j^2$ 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.

5.2 The Selected Occurrences of Small and Large Jobs

After eliminating Almost-Large jobs, the set of SMALL jobs in the modified LP becomes

$$\mathcal{J}_{S}^{new} = \mathcal{J}_{S} \cup \{J_{3n+j} | J_j \in \mathcal{J}_{AL}\} \text{ and } |\mathcal{J}_{S}^{new}| = |\mathcal{J}_{S}| + |\mathcal{J}_{AL}|.$$

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

$$\mathcal{J}_{L}^{new} = \mathcal{J}_{L} \cup \{J_{n+j}, J_{2n+j} | J_j \in \mathcal{J}_{AL}\} \text{ and } |\mathcal{J}_{L}^{new}| = |\mathcal{J}_{L}| + 2|\mathcal{J}_{AL}|.$$

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., 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 $J_j^\ell$, for $1 \leq \ell \leq (2n^2 + 1)^2$, be all the selected job occurrences.

5.3 The Modified Linear Program

The set of jobs in the modified LP is $\mathcal{J}_{new} = \mathcal{J}_{S}^{new} \cup \mathcal{J}_{L}^{new}$. Thus, $|\mathcal{J}_{new}| = |\mathcal{J}_{S}| + |\mathcal{J}_{L}| + 3|\mathcal{J}_{AL}| \leq 3n$. 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.

\[
\begin{align*}
(P_{\text{new}}) \quad \text{maximize} \quad & w \cdot x \\
\text{subject to :} \quad & \sum_{j^f \in C} x^f(j) \leq 2 \quad \text{for each clique } C \\
& \sum_{\ell=1}^{\mid L_1^j \mid} x^\ell(j) \leq 1 \quad \forall J_j \in J_{\text{new}} \\
& \sum_{\ell=1}^{\mid L_2^j \mid} x^\ell(n + j) + \sum_{\ell=1}^{\mid L_2^j \mid} x^\ell(2n + j) \leq 1 \quad \forall J_j \in J_{AL} \\
& \sum_{\ell=1}^{\mid L_2^j \mid} x^\ell(2n + j) - \sum_{\ell=1}^{\mid L_2^j \mid} x^\ell(3n + j) = 0 \quad \forall J_j \in J_{AL} \\
& x^\ell(j) \geq 0 \quad \forall J_j \in J_{\text{new}}, \ 1 \leq \ell \leq (2n^2 + 1)^2
\end{align*}
\]

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, \( \forall J_j \in J_{\text{new}} \). The third and fourth constraints deal with the jobs that replace the ALMOST-LARGE jobs. Recall that in Section 5.1 the occurrences of any \( J_j \in J_{AL} \) were partitioned into two subsets: \( L_1^j \) and \( L_2^j \). The third constraint ensures that the total value of the variables associated with the two LARGE jobs that replace a single ALMOST-LARGE job \( J_j \) (one LARGE job for each subset of job occurrences of \( J_j \)) is at most one. The fourth constraint ensures that for each pair of LARGE job and SMALL job that replace the subset of occurrences \( L_2^j \) 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.

5.4 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 \( y^r(j) \), for \( 1 \leq j \leq n \), \( 1 \leq r \leq \mid L_j \mid \). We show how it induces a solution to the modified linear program \( P_{\text{new}} \) as follows. If \( J_j \in J_S \), then for \( \ell = 1, \ldots, n^2 \),

\[
x^\ell(j) = \frac{1}{n} \sum_{r=1}^{\mid L_1^j \mid} y^r(j).
\]

If \( J_j \in J_L \), then for each selected job occurrence \( J_\ell^j \), the variable \( x^\ell(j) \) is the sum of all variables \( y^r(j) \), over all job occurrences \( J^r_j \) such that the right endpoint of the first segment of \( J^r_j \) is in the same time slot as the right endpoint of the first segment of
$J'_j$, and the right endpoint of the second segment of $J'_j$ is in the same time slot as the right endpoint of the second segment of $J'_j$.

Suppose $J_j \in \mathcal{J}_{AL}$. Recall that in Section 5.1 the occurrences of $J_j$ were partitioned into two subsets, denoted as $\mathcal{L}_1^j$ and $\mathcal{L}_2^j$. (The subset $\mathcal{L}_2^j$ includes the job occurrences we ignored in Section 5.1.) For each selected job occurrence $J_1^{n+j}$, the variable $x^\ell(n+j)$ is the sum of all variables $y^r(j)$, over all job occurrences $J_1^r \in \mathcal{L}_1^j$ such that the right endpoint of the first (second) segment of $J_1^r$ is in the same time slot as the right endpoint of the first (second) segment of $J_1^j$. Similarly, for each selected job occurrence $J_2^{2n+j}$, the variable $x^\ell(2n+j)$ is the sum of all variables $y^r(j)$, over all $J_2^r \in \mathcal{L}_2^j$ such that the right endpoint of the first segment of $J_2^r$ is in the same time slot as the right endpoint of the first segment of $J_2^j$.

Symmetrically, suppose $J_j \in \mathcal{J}_{AL}^2$. For each selected job occurrence $J_1^{n+j}$, the variable $x^\ell(n+j)$ is defined as above. For each selected job occurrence $J_2^{2n+j}$, the variable $x^\ell(2n+j)$ is the sum of all variables $y^r(j)$, over all $J_2^r \in \mathcal{L}_2^j$ such that the right endpoint of the second segment of $J_2^r$ is in the same time slot as the right endpoint of the second segment of $J_2^j$.

Finally, suppose $J_j \in \mathcal{J}_{AL}$. Then for $\ell = 1, \ldots, n^2$, we have $x^\ell(3n+j) = \frac{1}{n^2} \sum_{J_1^r \in \mathcal{L}_1^j} y^r(j)$.

It is straightforward to verify that the induced solution of the modified linear program (P\textsubscript{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 $J_j^k$ has the right endpoint of its first segment in 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, $\sum_{J_k^r \in \mathcal{C}} x^r(k) \leq \sum_{J_k^r \in S} y^r(k)$. 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 $\sum_{J_k^r \in \mathcal{C}} x^r(k) \leq 2$. 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 $J_j$, $x^\ell(j) \leq \frac{1}{n^2}$, for $1 \leq \ell \leq n^2$. Still, adding these variables may render the solution infeasible. Since for each SMALL job $J_j$, the job occurrences $J_j^1$, $1 \leq \ell \leq n^2$ are nonoverlapping, any job clique $\mathcal{C}$ contains at most one segment out of all segments of the job occurrences $J_j^1$, $1 \leq \ell \leq n^2$. 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 $(1 - \frac{1}{n})$. 

21
5.5 Rounding the Solution of the Modified Linear Program

Since the clique constraint is relaxed, we need to reformulate Lemma 4.1.

**Lemma 5.1.** Let $\mathbf{x}$ be a feasible solution to $(P_{\text{new}})$. Then, there exists a job occurrence $J^f_j$ satisfying $x^f(j) + \sum_{J'_k \in \tilde{N}(J^f_j)} x^r(k) \leq 9$.

*Proof* The proof is similar to the proof of Lemma 4.1. We first show that there exists a selected job occurrence $J^f_j$ for which

\[ x^f(j) + \sum_{J'_k \in \tilde{N}(J^f_j)} x^r(k) \leq 8. \tag{5.1} \]

As before, we define $z(J^f_j, J^f_k) = x^f(j) \cdot x^r(k)$, and $z(J^f_j, J^f_k) = (x^f(j))^2$. By the first constraint in $(P_{\text{new}})$, the definition of job cliques and the definition of $z(J^f_j, J^f_k)$, we have

\[ \sum_{J'_k \in R(J^f_j)} z(J^f_j, J^f_k) \leq x^f(j) \cdot \sum_{J'_k \in R(J^f_j)} x^r(k) \leq 2x^f(j). \tag{5.2} \]

Using (4.2), (5.2), and the fact that $|I(J^f_j)| \leq 2$, we have that

\[ \sum_{j=1}^{n} \sum_{\ell=1}^{n} \left( z(J^f_j, J^f_k) + \sum_{J'_k \in \tilde{N}(J^f_j)} z(J^f_j, J^f_k) \right) \leq 4 \sum_{j=1}^{n} \sum_{\ell=1}^{n} \sum_{J \in \mathcal{I}(J^f_j)} x^f(j) \leq 8 \sum_{j=1}^{n} \sum_{\ell=1}^{n} x^f(j). \]

Hence, there exists a job occurrence $J^f_j$ satisfying

\[ z(J^f_j, J^f_k) + \sum_{J'_k \in \tilde{N}(J^f_j)} z(J^f_j, J^f_k) = (x^f(j))^2 + \sum_{J'_k \in \tilde{N}(J^f_j)} x^r(k)x^f(j) \leq 8 \cdot x^f(j). \]

By factoring out $x^f(j)$ from both sides we get inequality (5.1).

To complete the proof of the lemma, we note that for a job $J^f_j$ satisfying (5.1) it also holds that

\[ x^f(j) + \sum_{J'_k \in \tilde{N}(J^f_j)} x^r(k) = x^f(j) + \sum_{J'_k \in \tilde{N}(J^f_j)} x^r(k) + \sum_{r=1}^{ \left| \mathcal{E}_j \right| } x^r(k) \leq 8 + \sum_{r=1}^{ \left| \mathcal{E}_j \right| } x^r(k) \leq 9. \]

The last inequality follows from the second constraint in $(P_{\text{new}})$.

We apply Lemma 5.1 to obtain a mapping (as defined in Section 4.1). 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 $J^1_{\text{new}}$ as defined in Section 4.1 with total weight $\frac{1}{8}$ of the objective value of the feasible solution of $(P_{\text{new}})$. 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 5.2.** There is a polynomial time \((9 + \epsilon)\)-approximation algorithm for MaxT on two machines, where \(\epsilon = O\left(\frac{1}{n}\right)\).
Chapter 6

Better Approximations for Special Cases on Two Machines

6.1 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., $r_j = 0 \forall j \in J$, 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. Let MaxT5 denote the problem of real-time scheduling on a single machine so as to maximize throughput. Recall that MaxT5 is solvable in polynomial time on instances with the same release times and unit job weights, using Moore’s algorithm [Moo68].

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 [Moo68]. We call this problem below MaxT5.

Moore’s algorithm can thus be used as a subroutine in our algorithm for MaxT, Split_the_Schedule. We give the pseudocode in Algorithm 6.1.

Algorithm 6.1 Split_the_Schedule

1. For any job $J_j \in J$ let $p_j = a_j + b_j$.
2. Solve optimally MaxT5, 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 6.1. 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 6.1.
Lemma 6.2. For any instance of MaxT where \( r_j = 0 \ \forall J_j \in J \), 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 \( s_j^1, s_k^1 \) be 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 6.1). 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 6.1: A non-permutation schedule.](image)

Lemma 6.3. Let \( OPT \) be an optimal solution for a MaxT instance \( J \) for which there is a permutation schedule. Then there exists a subset of jobs \( OPT_{\text{single}} \subseteq OPT \) satisfying:

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

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

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 6.1:** Recall that for an instance in which $r_j = 0 \forall J_j \in J$ we can use Moore’s algorithm [Moo68] to solve $\text{MaxT}_S$ optimally. By Lemma 6.3, there exists a subset of $|OPT|/4$ jobs that can be scheduled feasibly on a single machine, where OPT is an optimal solution for $\text{MaxT}$ on two machines. Since Moore’s algorithm outputs an optimal solution on a single machine, we have the statement of the theorem.

### 6.2 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 $1 \leq j \leq n$, $r_j = 0$ and $d_j = T$, for some $T > 0$. We note that $\text{MaxT}$ on such instances is NP-hard, as Knapsack is the special case where $b_j = 0$ for all $1 \leq j \leq n$.

Algorithm 6.2 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 [Joh54]. 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 $J_j$, $1 \leq j \leq n$. Also, let $\Delta_0 = 0$. For a set of jobs $J$, define the weight of $J$ to be $w(J) = \sum_{J_j \in J} w_j$.

**Theorem 6.4.** For any fixed $0 < \varepsilon < 1$, Algorithm 6.2 runs in polynomial time and yields a $(3 + \varepsilon)$-approximation for $\text{MaxT}$ on instances where $r_j = 0$ and $d_j = T$, $\forall j$.

To prove Theorem 6.4 we need an observation and a few lemmas.
Algorithm 6.2 Pack_and_Schedule

1: Fix $0 < \varepsilon < 1$.
2: Let $\mathcal{L} = \{J_j \in \mathcal{J} \mid \Delta_j \geq \frac{\varepsilon T}{6}\}$ and $\mathcal{S} = \mathcal{J} \setminus \mathcal{L}$.
3: Let $M_L = 0$
4: for all $R \subseteq \mathcal{L}$ such that $|R| \leq \frac{12}{\varepsilon}$ jobs do
5:   if $R$ can be scheduled with makespan at most $T$ then
6:     if $w(R) > M_L$ then
7:       Let $\text{SOL}_L = R$
8:     Let $M_L = w(R)$
9:     end if
10: end if
11: end for
12: Order the jobs in $\mathcal{S}$ in non-ascending order of the ratio $\frac{w_i}{\Delta_j}$. Assume w.l.o.g. that
13: $\mathcal{S} = \{J_1, \ldots, J_{|\mathcal{S}|}\}$, and $\frac{w_1}{\Delta_1} \geq \frac{w_2}{\Delta_2} \geq \ldots \geq \frac{w_{|\mathcal{S}|}}{\Delta_{|\mathcal{S}|}}$
14: Find the maximum index $k$ such that $\sum_{j=1}^k \Delta_j \leq T(1 - \frac{\varepsilon}{6})$
15: Let $\text{SOL}_S = \{J_1, \ldots, J_k\}$ \hspace{1cm} $\triangleright$ SOL$_S$ can be scheduled feasibly as shown below
16: If $w(\text{SOL}_L) > w(\text{SOL}_S)$ then $\text{SOL} = \text{SOL}_L$; else $\text{SOL} = \text{SOL}_S$.
17: Return $\text{SOL}$

Observation 6.5. Any feasible solution $R$ of MaxT on instances where $r_j = 0$ and $d_j = T \forall j$ satisfies $\sum_{j \in R} \Delta_j \leq 2T$.

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

Lemma 6.6. 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 6.5 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. $\blacksquare$

Lemma 6.7. 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 $\Delta_j$. Let $\pi$ 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 $s^1_{\pi(j)} = \sum_{i=0}^{j-1} \Delta_{\pi(i)}$ on $M_1$ and at time $s^2_{\pi(j)} = s^1_{\pi(j)} + \Delta_{\pi(1)}$ 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 $\Delta_{\pi(1)} = \max_{j \in [1..k]} \Delta_j$. $\blacksquare$

Lemma 6.8. The set SOL$_S$ is a $(2 + \varepsilon)$-approximation of the optimal solution for MaxT on the input $\mathcal{S}$.

28
Proof First, consider a knapsack problem with set of items corresponding to the jobs in $\mathcal{S}$, where the size of item $j$ is $\Delta_j$ 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 $\text{MaxT}$ on the input $\mathcal{S}$. To see this, consider such an optimal solution $O$ for $\mathcal{S}$, 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 $w(O)$. Since $O$ is a feasible solution for $\text{MaxT}$, 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 of weight at least $\frac{w(O)}{2}$. 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 $J_j \in \mathcal{S}$, $\Delta_j < \frac{e^2}{12} T$, it follows that $\sum_{j=1}^{k} \Delta_j > T(1 - \frac{e}{2})$. 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 $(1 - \frac{e}{2})$ of the weight of the optimal solution to the knapsack problem, and thus it is at least $\frac{1}{2}(1 - \frac{e}{2})w(O)$. Since $(2 + \varepsilon) \cdot \frac{1}{2}(1 - \frac{e}{2}) = 1 + \frac{1}{6}(\varepsilon - \varepsilon^2) \geq 1$, for $0 < \varepsilon < 1$, it follows that $SOL_S$ is a $(2 + \varepsilon)$-approximation of the optimal solution for $\text{MaxT}$ on the input $\mathcal{S}$. \hfill \blacksquare

**Proof of Theorem 6.4:** Consider an optimal solution $OPT$ for input $\mathcal{J}$, and partition the jobs in this solution into two disjoint sets: $OPT_L = OPT \cap \mathcal{L}$ and $OPT_S = OPT \cap \mathcal{S}$. By Lemmas 6.6 and 6.8, we have that $w(OPT_L) \leq w(SOL)$ and $w(OPT_S) \leq (2 + \varepsilon) \cdot w(SOL)$. It follows that $w(OPT) \leq (3 + \varepsilon) \cdot w(SOL)$.

By the algorithm and Lemma 6.7, the jobs in $\text{SOL}$ can be scheduled feasibly. For the running time, we note that the most time consuming part in Algorithm 6.2 is the loop defined in Step 4, where we enumerate over all subsets of $L$ of size at most $\frac{12}{\varepsilon}$. Thus, the time complexity is $O(n^{\frac{12}{\varepsilon}})$ which is polynomial.

**Corollary 6.9.** 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 $\varepsilon \in (0, 1)$, Algorithm 6.2 is a $(2 + \varepsilon)$-approximation algorithm for $\text{MaxT}$ 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 holds, then the weight of an optimal solution to the knapsack problem defined in Lemma 6.8 is at least the weight of the optimal solution for $\text{MaxT}_S$ on the input $\mathcal{S}$, and thus the set $SOL_S$ is a $(1 + \varepsilon)$-approximation for $\text{MaxT}_S$ on the input $\mathcal{S}$. \hfill \blacksquare
Chapter 7

No-Wait Flowshop

Consider a general weighted instance of no-wait flow shop with \( m \geq 2 \) machines, in which jobs have arbitrary release times and due dates. We show below that for such instances a simple algorithm yields an approximation ratio of \( 2m + \varepsilon \) for \( \text{MaxT} \) (see the pseudocode in Algorithm 7.1). We use as a subroutine an algorithm of Bar-Noy et al. [BBF+01] to solve \( \text{MaxT}_S \). Recall that \( \text{MaxT}_S \) represents the problem of real-time scheduling on a single machine so as to maximize throughput. The Algorithm of [BBF+01] uses the local ratio technique to obtain a \((2 + \varepsilon)\)-approximation for \( \text{MaxT}_S \).

Algorithm 7.1 Split_the_Schedule_No_Wait

1. For any job \( J_j \in \mathcal{J} \) let \( p_j = \sum_{i=1}^{m} p_{j,i} \).
2. Given \( \varepsilon > 0 \), run the algorithm of [BBF+01] with parameter \( \frac{\varepsilon}{m} \) to solve \( \text{MaxT}_S \), where each job \( J_j \) has a processing time \( p_j \), a weight \( w_j \), release time \( r_j \) and due date \( d_j \). Let \( \text{SOL} \) be the set of jobs in the solution.
3. Define the following flow shop schedule of \( \text{SOL} \) on the \( m \) machines: for any \( J_j \in \text{SOL} \) that is processed on the single machine in \((s_j, t_j)\), process \( J_j \) on \( M_i \) in \((s_j + \sum_{k=1}^{i-1} p_{j,k}, s_j + \sum_{k=1}^{i} p_{j,k})\) for each \( 1 \leq i \leq m \).
4. Return the schedule of \( \text{SOL} \).

Theorem 7.1. For any \( m \geq 2 \) and \( \varepsilon > 0 \), Algorithm 7.1 yields in polynomial time a \((2m + \varepsilon)\)-approximation for \( \text{MaxT} \) on general no-wait instances with \( m \) machines.

We prove the theorem using the next result.

Lemma 7.2. Let \( \text{OPT} \) be an optimal solution for a \( \text{MaxT} \) instance \( \mathcal{J} \) of no-wait flow shop with \( m \geq 2 \) machines. Then there is a subset of jobs \( \text{OPT}_{\text{single}} \subseteq \text{OPT} \) satisfying:

(i) The jobs in \( \text{OPT}_{\text{single}} \) can be feasibly scheduled on a single machine, taking the processing time of each \( J_j \in \text{OPT}_{\text{single}} \) to be \( p_j = \sum_{i=1}^{m} p_{j,i} \), with the same release time and due date.

(ii) \( w(\text{OPT}_{\text{single}}) \geq \frac{w(\text{OPT})}{m} \).

31
Consider an optimal subset of jobs, \( OPT \), which has a permutation schedule (since we discuss no-wait flow shop). To simplify the presentation, we renumber the jobs by \( J_0, ..., J_{n-1} \), using their order in this permutation, and the machines by \( M_0, ..., M_{m-1} \). Let \( (s_{j,i}, t_{j,i}] \) be the processing interval of \( J_j \) on \( M_i \), \( 0 \leq i \leq m-1 \). The next observation is used in the proof of Lemma 7.2.

**Observation 7.3.** Any two jobs \( J_j, J_{j+m} \) where \( 0 \leq j \leq j + m \leq n - 1 \), satisfy that \( s_{j+m,0} \geq t_{j,m-1} \). In other words, the first segment of job \( J_{j+m} \) starts processing after the last segment of \( J_j \) completes processing.

**Proof** By the ordering of the jobs, we have that the first segment of \( J_{j+m} \) starts processing after the first segment of \( J_{j+m-1} \) completes; that is,

\[
s_{j+m,0} \geq t_{j+m-1,0} \tag{7.1}
\]

Also, by the no-wait condition, we have that for any \( 1 \leq r \leq m - 1 \) it holds that \( t_{j+r,m-r} \geq t_{j+r-1,m-r-1} \), since segment \( m - r \) of \( J_{j+r} \) must be processed immediately after segment \( m - r - 1 \) of \( J_{j+r} \) completes processing. Hence, we have that \( t_{j+m-1,0} \geq t_{j+m-2,1} \geq \cdots \geq t_{j,m-1} \) and the claim follows from (7.1).

Figure 7.1 illustrates the statement of Observation 7.3 for an instance of 5 jobs and 3 machines. For example, taking \( j = 0 \) we have in the no-wait flow shop schedule that \( s_{3,0} \geq t_{0,2} \).

**Proof of Lemma 7.2:** We show how to convert an \( m \)-machine schedule to a schedule
of a subset of jobs in OPT, such that each job is completely processed on one of the
machines. We schedule all segments of $J_j \in OPT$ on the machine $M_{j \mod m}$. Figure 7.1 illustrates this transformation for the instance consisting of $J_0, \ldots, J_4$ and $M_0, \ldots, M_2$.

By Observation 7.3, the schedule obtained on each machine $M_i$ is feasible. In addition, each job $J_j \in OPT$ is scheduled on one of the $m$ machines. Therefore, taking the machine with the subset of jobs of largest weight as OPT single we have that $w(OPT_{\text{single}}) \geq \frac{w(OPT)}{m}$.

**Proof of Theorem 7.1:** It is easy to verify that Algorithm 7.1 returns a feasible no-wait flow shop schedule. Now, we prove that the approximation ratio is $(2m + \varepsilon)$. Let $OPT_{\text{MaxT}_5}$ be an optimal solution for MaxT$_5$. As the algorithm of [BBF+01] yields a $(2 + \varepsilon)$-approximation for MaxT$_5$, we have that

$$w(SOL) \geq \frac{w(OPT_{\text{MaxT}_5})}{2 + \frac{\varepsilon}{m}}.$$  \hfill (7.2)

By Lemma 7.2, OPT single is a feasible solution for MaxT$_5$. Therefore,

$$\frac{w(OPT_{\text{MaxT}_5})}{2 + \frac{\varepsilon}{m}} \geq \frac{w(OPT_{\text{single}})}{2 + \frac{\varepsilon}{m}} \geq \frac{w(OPT)}{2m + \varepsilon} \hfill (7.3)$$

Hence,

$$w(SOL) \geq \frac{w(OPT)}{2m + \varepsilon}. \hfill (7.4)$$

We note that Theorem 7.1 improves our approximation ratio for general flow shop instances with two machines (in Theorem 5.2) to $4 + \varepsilon$. In the special case where $r_j = 0$ for all $j \in J$ we can use in Step 2 of Algorithm 7.1 the PTAS of Sahni [Sah76] for MaxT$_5$. Moreover, if all jobs also have uniform weights we can use in Step 2 Moore’s algorithm [Moo68]. Hence, we have

**Corollary 7.4.** *If $r_j = 0$ for all $j \in J$, there is a polynomial-time $(m+\varepsilon)$-approximation algorithm for no-wait flow shop, for any $m \geq 2$. If, in addition, all jobs have the same (unit) weight, MaxT can be approximated within factor of $m$.*
Chapter 8

Conclusion and Future Work

In this work, we studied the flow shop real-time scheduling problem. While previous studies of the flow shop model have focused on the makespan, flow time or tardiness objectives, we focused on throughput maximization in the real-time setting \((\text{MaxT})\).

We first presented 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. Using this result, we derived a polynomial-time \((9 + \varepsilon)\)-approximation algorithm for \(\text{MaxT}\) on two machines, where \(\varepsilon = O(1/n)\) for an input of size \(n\).

We obtained better bounds for some restricted subclasses of inputs with two machines. In particular, we gave a 4-approximation algorithm for flow shop instances where all jobs have the same release time and uniform weights. For the special case where all jobs have the same time window and arbitrary weights, we presented a \((3 + \varepsilon)\)-approximation algorithm, for any fixed \(\varepsilon > 0\).

Finally, we showed that a simple polynomial-time algorithms yields \((2m + \varepsilon)\)-approximation for general \(\text{MaxT}\) instances in the no-wait flow shop model, for any \(\varepsilon > 0\) and arbitrary \(m > 1\). In the special case where all jobs have the same release time, we obtained a better ratio of \(m + \varepsilon\).

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.

8.1 Future Directions

Our results leave several open questions for future work.

Polynomial-time \(O(m)\)-approximation. We presented a pseudo-polynomial time \((2m + 1)\)-approximation algorithm for \(\text{MaxT}\) on \(m \geq 2\) machines, where \(m\) is a constant. An immediate question is: Can we obtain a polynomial-time \(O(m)\)-approximation for this problem?
The running time of our algorithm was dominated by the size of the linear programming relaxation for our problem, which depends on the number of job occurrences in each time window for the given input. Thus, obtaining a polynomial-time algorithm either requires to apply different techniques, or to use an LP relaxation of smaller size, as we did in Chapter 5 for the two machine case. We note that obtaining a polynomial-time approximation for MaxT is open already in the three machine case.

Another important direction is to obtain a non-trivial approximation ratio for MaxT on general instances where \( m \geq 2 \) is arbitrary.

**Better bounds for the no-wait flow shop model.** We showed that in the no-wait flow shop model MaxT admits polynomial-time \((2m + \varepsilon)\)-approximation algorithm for any \( m \geq 2 \) and \( \varepsilon > 0 \). We obtained better approximations for some restricted cases.

We note that the hardness of approximation result for general MaxT instances does not carry over to the no-wait flow shop model; thus, it may be the case that our bound for this model can be significantly improved. Obtaining such better bounds may require the use of different techniques. For makespan minimization on two machines, the algorithm of Johnson [Joh54] yields an optimal solution for general instances. In contrast, for no-wait flow shop, an optimal schedule is obtained by solving a restricted instance of the traveling salesman problem, as shown by Gilmore and Gomory [GG64].

**Existence of optimal permutation schedule.** In this study, we encountered several special cases of MaxT for which there exist an optimal permutation schedule. This includes instances where all jobs have the same release time \( r_j = 0 \), or instances of no-wait flow shop. Designing algorithms for MaxT that yield permutation schedules may be somewhat easier, since we can solve first the throughput maximization problem on a single machine and then generate a permutation schedule on multiple machines, as we did in Section 6.1. Thus, it would be interesting to identify other subclasses of instances for MaxT that admit optimal permutation schedules. Furthermore, assuming we need to find a permutation schedule, can we use techniques similar to those applied in Section 6.1 to obtain efficient approximations?
Bibliography


Technion - Computer Science Department - M.Sc. Thesis MSC-2022-12 - 2022
In general, and in contrast to previous results in the field, there is a flow shop
(tardiness) problem with due date constraints, with the objective to minimize the
makespan of the shop, i.e., the time when all jobs are completed.

The problem is formally stated as follows. Given a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$,
and for each job $J_i$, a release time $r_i$, a due date $d_i$, and a weight $w_i$, find a
schedule that minimizes the makespan.

In particular, for the flow shop problem, we consider the MaxT problem, where
the goal is to minimize the maximum tardiness among all jobs.

The problem is NP-hard, and there are several approximation algorithms proposed
for solving it. One of the most well-known is the 1.5-approximation algorithm
by Asahiro et al. [1996], which guarantees a solution no more than 1.5 times
the optimal makespan.

For the flow shop problem, a 2-approximation algorithm is known. This
algorithm, due to Asahiro et al. [1996], runs in polynomial time and guarantees
a solution within a factor of 2 of the optimal makespan.

In this thesis, we consider the no-wait flow shop problem, where
all jobs must be completed without any waiting time between
operations. This problem is known to be NP-hard, and
there are several approximation algorithms proposed for it.

One of the most well-known algorithms is the
no-wait flow shop algorithm of Asahiro et al. [1996],
which guarantees a solution within a factor of 2 of the optimal makespan.

In the following sections, we discuss the theoretical results and algorithms
for the MaxT problem, and present experimental results that
support the theoretical findings.
In this study, we focus on the problem of maximizing the throughput of the flow shop model with a given number of machines. At time t, a set of parts is available, and each part requires a specific sequence of operations on the machines. The goal is to find a schedule that maximizes the throughput, i.e., the number of parts processed in a given time frame.

In reality, there is a wide range of scenarios involving time constraints. For example, Johnson's problem is a classic case involving autonomous vehicles. Autonomous vehicles are equipped with different applications whose purpose is to monitor and plan the route.

To process a photo, it is taken by a camera, and the image is captured. It is then processed on a GPU using a deep learning model, and the result is passed to a CPU for further processing. This process is repeated for each image in the set.

In the context of flow shop models, we can see that the throughput is maximized by finding an optimal schedule that minimizes the makespan, i.e., the time it takes for all parts to be processed.

In the Johnson problem, where we have a single machine, the optimal schedule is known. However, when we have multiple machines, the problem becomes much harder. For example, when we have three machines, the optimal solution is known, but for four or more machines, the problem is NP-hard.

In the permutation flow shop model, we consider the case where the jobs are permutations of the machines. In this case, the optimal schedule is known, and the problem becomes significantly easier.

In conclusion, the study focuses on the flow shop model and its variants, exploring various optimization techniques to improve throughput and efficiency.
The research was conducted under the supervision of Professor Hadas Shachnai, in the Faculty of Mathematics.

The results in the thesis were published in an article that the author of the research wrote in a seminar in the course of his master's thesis, which has been updated and corrected:


A draft of the comprehensive report for the master's thesis has been submitted.
מקסום התפקה של תחומי בורימה קויה
במעורבות שנה-אמה

 العربية

 לייאור בן-מיי

thinkableבשם תחומי בורימה קויה
במעורבות שנה-אמה

לייאור בן-מיי

הוגשלסנטה טכניון — מכון תכנון לישראלי
אדר החשפים"ב חיפה מבראואר 2022
מקסום התפוקה של תזמון בזירה קיימת
במערכות זמ"ן-אמית

ליאור בר-ימי