# Throughput Maximization in a Scheduling Environment with Machine-Dependent Due-Dates

**Shaul Rosner**
Tel-Aviv University, Israel

**Tami Tamir**
Reichman University, Herzliya, Israel

## Abstract

We consider a scheduling environment in which jobs are associated with *machine-dependent* due-dates. This natural setting arises in systems where clients' tolerance depends on the service provider.

The objective is to maximize throughput, defined as the number of non-tardy jobs. The problem exhibits significant differences from previously studied scheduling models. We analyze its computational complexity both in general and for the special case of unit-length jobs.

In the unit-length setting, we provide an optimal algorithm that also extends to cases with machine-dependent release times and machine-dependent weights (i.e., rewards depending on the machine that completes the job).

For jobs with different lengths, we show that even the unweighted problem without release times, with only two different lengths, specifically, for all $j$, $p_j \in \{1, 2\}$, is APX-hard. To isolate the role of machine-dependent due-dates in this hardness result, we present an optimal algorithm for the case where all $p_j \in 1, 2$ and due-dates are not machine-dependent. This algorithm further extends to instances with a constant number of integer processing times.

**2012 ACM Subject Classification** Theory of computation → Scheduling algorithms

**Keywords and phrases** Scheduling, Throughput maximization, Machine-dependent due-dates, Computational Complexity

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2025.5

## 1 Introduction

In many scheduling environments, jobs are associated with due-dates, and each job is expected to be completed before its due-date. This classical setting has been extensively studied since the 1950s [18], and the landscape is well understood for various objectives, such as minimizing the number of tardy jobs, maximum tardiness, or total tardiness. In this paper, we consider a more general setting in which due-dates are *machine-dependent* – that is, the allowed completion time for a job may vary depending on the machine to which it is assigned.

In our model, for every job $j$ and machine $i$, we are given the *tolerance* of job $j$ if processed on machine $i$, interpreted as its due-date on that machine. Such a setting arises in a range of real-world systems. For example, consider an online retailer during a busy season. The company operates multiple fulfillment centers, each with distinct capacities, locations, and shipping capabilities. While customer satisfaction is generally influenced by delivery speed, other factors – such as packaging quality, proximity to pickup points, and customer service – can also play a role. Thus, a customer's tolerance for delivery time may vary depending on the fulfillment center handling the order.

As another example, consider an electronics manufacturer with multiple production lines. Each line may differ in specialization and workload. A task could be completed quickly on a lightly loaded line, but a more specialized line may produce a higher-quality result, justifying a longer wait. In other words, the allowed due-date for the product may depend on the production line to which it is assigned.

Our model also arises in transportation systems, where different routes or depots lead to different tolerances for travel time. For example, in a city's public transit network, buses or trains may be dispatched from different depots or yards, each serving routes with different congestion patterns and travel conditions. Passengers' tolerance for delays may be shorter when the route is already prone to heavy traffic, but longer for routes that are typically more reliable or comfortable. Likewise, in a tourist ferry system serving the same two ports via different paths, some vessels take the fastest direct route while others meander along scenic coastlines or through picturesque islands. Travelers are willing to accept a significantly longer trip when the journey itself offers a richer experience, effectively extending the due-date for arrival on that specific route. These examples demonstrate that machine-dependent due-dates naturally arise in real-life applications, motivating our study beyond its theoretical interest.

In this paper, we provide initial results for this setting, focusing on the fundamental problem of maximizing throughput, defined as the number of non-tardy jobs. Surprisingly, to the best of our knowledge, this variant has not been previously studied. We show that for unit-length jobs, known scheduling techniques can be adapted to compute optimal solutions efficiently. In contrast, when jobs have variable lengths, the problem exhibits substantial differences from previously studied models.

## 2 Problem Statement and Preliminaries

An instance of a scheduling problem with machine-dependent due-dates (SMDD, for short) is given by $I = \langle J, M, \{p_j\}_{j\in J}, \{d_{i,j}\}_{i\in M, j\in J}\rangle$, where $J$ is a set of $n$ jobs, and $M$ is a set of $m$ parallel *machines*. For every job $j \in J, p_j$ is the length of job $j$. For every machine $i \in M$ and job $j \in J$, $d_{i,j}$ is the due-date of job $j$ if processed on machine $i$. Some of our results refer to instances with unit-length jobs, in which $p_j = 1$ for all $j \in J$.

A *schedule* for an instance $I$ is defined by a tuple $s = \langle s_1, s_2, \ldots, s_n\rangle$, where $s_j = (m_j, o_j) \in (M \cup \{\perp\}) \times \mathbb{N}$. $m_j \in M \cup \{\perp\}$ indicates the machine to which job $j$ is assigned, or $\perp$ if the job is *rejected* (i.e., not scheduled on any machine). $o_j \in \mathbb{N}$ indicates the order on which it is assigned to the machine. We omit $o_j$ when it is clear from the context. For each machine $i \in M$, let $J_i = \{j \in J \mid s_j = i\}$ denote the set of jobs assigned to machine $i$. The *load* on machine $i$ in schedule $s$, denoted $L_i(s)$, is the total processing time of jobs in $J_i$: $L_i(s) = \sum_{j\in J_i} p_j$. In the unit-length case, this simplifies to $L_i(s) = |J_i|$.

The *completion time* $C_j(s)$ of a job $j \in J_i$ is defined as the sum of the processing times of all jobs scheduled on machine $i$ before job $j$, plus $p_j$. The *lateness* of job $j$, assigned to machine $i$, is $C_j(s) - d_{i,j}$, and its *tardiness* is $T_j(s) = \max(0, C_j(s) - d_{i,j})$. Job $j$ is called *tardy* if $T_j(s) > 0$. Let $U_j(s) \in \{0, 1\}$ denote the binary indicator of whether job $j$ is either rejected or tardy: $U_j(s) = 1$ iff $s_j = \perp$ or $C_j(s) > d_{s_j,j}$. We omit $s$ from the notation when it is clear from context.

Our objective is to *maximize the throughput*, which is the number of jobs which are not non-rejected and non-tardy. We denote this number, of *satisfied jobs*, by $\text{sat}(s) = \left|\{j \in J \mid s_j \neq \perp \text{ and } C_j(s) \leq d_{s_j,j}\}\right|$. Since we care only about throughput and not about minimizing tardiness, we may assume w.l.o.g., that all non-satisfied jobs are rejected (i.e., $s_j = \perp$ if $C_j(s_j) > d_{s_j,j}$). This assumption simplifies the model without affecting the objective. Using standard three-field scheduling notation [5], this problem can be described as: $P \mid d_{i,j} \mid \sum_j (1 - U_j)$.

## 2.1 Our Results

We define the SMDD problem and present initial results. We first consider instances with unit-length jobs. In Section 3 we present an optimal algorithm for the problem $P|p_j = 1, d_{i,j}|\sum_j(1-U_j)$. Our algorithm is based on reducing the problem to a weighted maximum matching problem [13], and it can be extended for instances in which jobs have machine-dependent release times and machine-dependent weights (reward for completion), that is, $P|p_j = 1, r_{i,j}, d_{i,j}|\sum_j w_{i,j}(1-U_j)$.

In Section 4 we turn to consider instances with arbitrary job lengths. We show that even if we allow only two different lengths, specifically, if for all $j$, $p_j \in \{1,2\}$ then the problem becomes APX-hard. To isolate the role of the machine-dependent due-dates in this hardness result, we present, in Section 5, an optimal algorithm for the case that for all $j$, $p_j \in \{1,2\}$, and the due-dates are not machine-dependent. That is, every job $j$ is associated with a due-date $d_j$ such that for all $i$, $d_{i,j} = d_j$. Our optimal algorithm for this case can be extended to solve $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ for any set $\mathcal{P}$ of integer job lengths, in time $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$. We conclude in Section 6 with a discussion and directions for future work regarding additional objectives relevant for environments with machine-dependent due-dates.

## 2.2 Related Work

Scheduling theory is a very well studied field, dating back to the early 1950's [5]. When jobs are associated with due-dates, and processed sequentially by the machines, maximizing the throughput is a classical well-studied objective. For a single machine, Moore-Hodgson's algorithm ([14]) solves the problem optimally. For parallel machines, the problem is NP-hard even with preemptions allowed [10]. A more general setting in which jobs are also associated with release times is also well-studied. Approximation algorithms, as well as optimal algorithms for several restricted classes are presented in [4, 19, 1, 8]. The paper [2] analyzes the impact of different scheduling policies on the throughput. When jobs have equal-lengths, an optimal solution can be produced using max-flow techniques [3, 7]. A variant of scheduling with machine-dependent due-dates is discussed in [6, 15]. In their setting, each machine has a sorted list of due-dates, and the $i$'th job assigned to a machine has the $i$'th due-date.

An instance of SMDD is given by an $m \times n$ matrix representing, for every job $j$ and machine $i$, the due-date of job $j$ on machine $i$. There are several additional well-explored scheduling problems whose input is given by an $m \times n$ matrix, stating a value for each job-machine pair: $(i)$ In scheduling on *unrelated machines*, for every job $j$ and machine $i$, we are given the processing time of job $j$ if processed on machine $i$. The most classical problem, of minimizing the makespan of a schedule on unrelated machines $(R||C_{max})$ is studied in [12], where a 2-approximation algorithms is given, as well as various hardness results, $(ii)$ In a *shop scheduling* environment, each job consists of $m$ tasks. For each job $j$ and machine $i$, we are given the length of $j$'s task on machine $i$. The order according to which the tasks should be processed may be flexible (openshop), uniform (flowshop) or job-specific (jobshop). Unfortunately, except for some positive results for two machines, solving shop-scheduling problems is computationally hard [11, 18]. $(iii)$ In environments where each machine has a different scheduling policy [20], for every machine we are given its priority list - an order of the $n$ jobs according to which it processes the jobs assigned to it. The analysis of the above settings shows that having machine-dependent parameters makes the problem computationally harder relative to the related-machines settings.

## 3 Optimal Algorithm for Unit-length Jobs

This section refers to instances in which all jobs have unit processing lengths, that is, for all $j \in J, p_j = 1$. We present an optimal algorithm for the problem. In fact, our algorithm fits a more general setting, in which, in addition to the due-dates, for every job $j$ and machine $i$, we have machine-dependent release times and rewards. Formally, let $r_{i,j}$ be the release time of job $j$ if processed on machine $i$. Let $\omega_{i,j}$ denote the reward from completing $j$ on time on machine $i$. For completeness, define $\omega_{\perp,j} = 0$. Note that the throughput maximization problem $P|p_j = 1, d_{i,j}|\sum_j(1-U_j)$ is the special case in which $r_{i,j} = 0$ and $\omega_{i,j} = 1$ for every job $j$ and machine $i$. The proof of the following theorem is based on a reduction to a weighted maximum matching problem [13].

▶ **Theorem 1.** *The problem $P|p_j = 1, r_{i,j}, d_{i,j}|\sum_j \omega_{s_j,j}(1-U_j)$ is polynomially solvable.*

**Proof.** We reduce the problem to a weighted maximum matching problem in a bipartite graph. Given an instance $\langle J, M, \{r_j\}_{j\in J}, \{d_j\}_{j\in J}, \omega\rangle$, of our scheduling problem, construct a weighted bipartite graph $(V \cup U, E)$. The set $V$ includes $n$ job-vertices, $\{v_j\}$ for every job $1 \le j \le n$. The set $U$ includes $mn$ slot-vertices, $\{u_i^\ell\}$, for each $1 \le \ell \le n$ and machine $i \in M$. The set of edges is $E = \{(v_j, u_i^\ell) \mid r_j(i) < \ell \le d_j(i)\}$. That is, every job-vertex is connected to all the slot-vertices corresponding to a feasible assignment of $j$ with regards to release time and due-date. The weight of an edge $(v_j, u_i^\ell)$ is $\omega_{j,i}$.

It is easy to see that every schedule with total reward $W$ corresponds to a matching of total weight $W$. Also, every feasible matching of weight $W$ induces a valid schedule. All jobs in this schedule are scheduled after their release time and before their due-date, and their total reward is $W$. As a maximum weight matching can be found efficiently, and the reduction is polynomial, the whole algorithm is polynomial.

Note that machines may have idle slots in the produced schedule. The number of idle blocks can be minimized by shifting earlier jobs with early release times. Some idles may be unavoidable due to the release times. ◀

## 4 APX-hardness Proof for $p_j \in \{1, 2\}$

Unfortunately, the positive result for unit-length jobs cannot be extended even to highly restricted classes of variable-length jobs. The following hardness result is based on a reduction technique used in [12] to show NP-hardness of the minimum makespan problem on unrelated machines $(R||C_{max})$. We note, however, that its adaptation to SMDD with $p_j \in \{1, 2\}$ is different, as the classical minimum makespan problem is efficiently solvable for such instances.

▶ **Theorem 2.** *$P|p_j \in \{1, 2\}, d_{i,j}|\sum_j(1-U_j)$ is APX-hard.*

**Proof.** In order to prove APX-hardness, we use an $L$-reduction [16], defined as follows:

▶ **Definition 3.** *Let $\Pi_1, \Pi_2$ be two optimization problems. We say $\Pi_1$ $L$-reduces to $\Pi_2$ if there exist polynomial time computable functions $f, g$ and constants $\alpha, \beta > 0$ such that, for every instance $I \in \Pi_1$ the following holds:*

1. *$f(I) \in \Pi_2$ such that , $OPT(f(I)) \le \alpha \cdot OPT(I)$.*
2. *Given any solution $\varphi$ to $f(I)$, $g(\varphi)$ is a feasible solution to $I$ such that $|OPT(I) - value(g(\varphi))| \le \beta \cdot |OPT(f(I)) - value(\varphi)|$.*

Our $L$-reduction is from 3-bounded 3-dimensional matching (3DM3). The input to the problem is a set of triplets $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = k$. The number of occurrences of every element of $X \cup Y \cup Z$ in $T$ is at most 3. The number of triplets is

$|T| = m$. The goal is to find a maximal subset $T' \subseteq T$, such that every element in $X \cup Y \cup Z$ appears at most once in $T'$. This problem is known to be APX-hard [9]. Moreover, a stronger hardness result shows the problem has an hardness gap even on instances with a matching of size $k$ [17], namely instances where there is a solution with $k$ triplets.

Given an instance $T$ of 3DM3, we construct an instance $f(T)$ of SMDD. For each triplet containing element $x \in X$, we say it is a triplet of type $x$. Let $t_x$ be the number of triplets of type $x$. For each triplet, there is a corresponding machine $i$ for a total of $m$ machines.

There are $2k$ element jobs, one for each element $y \in Y$, and one for each element $z \in Z$. A job $j_y$ corresponding to an element $y \in Y$ has $d_{i,j_y} = 1$ if $y \in T_i$ where $T_i$ is the triplet corresponding to machine $i$, and $d_{i,j_y} = 0$, otherwise. A job $j_z$ corresponding to an element $z \in Z$ has $d_{i,j_z} = 2$ if $z \in T_i$ where $T_i$ is the triplet corresponding to machine $i$, and $d_{i,j_z} = 0$ otherwise. For every such element job $j$, $p_j = 1$.

For every $x \in X$, there are $t_x - 1$ dummy jobs $j_x$ such that $d_{i,j_x} = 2$ if $x \in T_i$ where $T_i$ is the triplet corresponding to machine $i$, and $d_{i,j_x} = 0$ otherwise. Note that there is one dummy job for all but one of the $t_x$ machines corresponding to triplets of type $x \in X$, for a total of $m - k$ dummy jobs. For every dummy job $j$, $p_j = 2$.

In order to complete the $L$-reduction, we prove the conditions of Definition 3 are met. As a preliminary, we show that if $T$ has a matching $T'$ of size $k$, then $OPT(f(T)) = m + k$. For a schedule $s$, let $sat(s) = \sum_j (1 - U_j)$ be its throughput.

Let $T'$ be a matching of size $k$. For each triplet $(x, y, z) \in T'$, we can schedule the jobs corresponding to $y, z$ on the machine corresponding to the triplet. Note that both jobs are satisfied. This leaves $t_x - 1$ idle machines to which dummy jobs corresponding to $x$ can be assigned with completion time 2. Thus, we have a schedule in which $m - k$ dummy jobs are satisfied, and $2k$ element jobs are satisfied for a total of $m + k$ satisfied jobs, proving $sat(OPT(f(T)) \geq m + k$. As there are no additional jobs, $sat(OPT(f(T)) = m + k$.

Given a schedule $s$, let $g(s)$ be the set of triplets corresponding to machines on which two jobs are assigned. The theorem follow from the following claims, showing that Definition 3 is satisfied for $\alpha = 4$ and $\beta = 1$.

▷ Claim 4. If $OPT(T) = k$ then $OPT(f(T)) \leq 4k$.

Proof. Since every element of $X \cup Y \cup Z$ appears at most 3 times in $T$, $m \leq 3k$. Thus, $OPT(f(T)) = m + k \leq 3k + k = 4k$. ◁

▷ Claim 5. $OPT(T) - g(s) \leq (OPT(f(T)) - sat(s))$.

Proof. Since we assume that $T$ has a perfect matching, we have that $OPT(T) = k$. Since we proved that $value(OPT(f(T)) = m+k$, it is sufficient to prove that $k-g(s) \leq m+k-value(s)$. Consider a schedule $s$ of $f(T)$. Recall that every machine has $0, 1$ or $2$ jobs assigned to it. Thus, $sat(s) \leq 2g(s) + (m - g(s)) = m + g(s)$. Therefore, $k - g(s) \leq m + k - sat(s) \leq m + k - (m + g(s)) = k - g(s)$ as needed. ◁

◀

## 5 Optimal Algorithm for Job-dependent Due-dates

To isolate the impact of machine-dependent due-dates on the computational complexity of the problem, We consider the case of machine-independent due-dates, that is, the problem $P||\sum_j (1 - U_j)$. A simple reduction from the *Partition* problem implies that this problem is NP-hard even on two machines, when all the jobs have the same machine-independent due-date. Let $\mathcal{P}$ denote the set of possible job lengths. We assume that $\mathcal{P} \subseteq \mathbb{N}$. We

present an optimal algorithm for the problem assuming $\mathcal{P} = \{1, 2\}$. We then generalize this algorithm and present an optimal algorithm for $P|p_j \in \mathcal{P}| \sum_j (1 - U_j)$ whose running time is $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$. We note that the this problem is already known to be polynomially solvable. However, our algorithm is based on dynamic programming, while previously known algorithms ([19, 4]) use different techniques.

Our algorithm consists of two steps. In the first step, we guess the jobs that are going to be satisfied. In the second step, we use dynamic programming to assign these jobs, such that they are all satisfied in the resulting schedule.

Consider an optimal schedule $s^*$. For $p \in \{1, 2\}$, let $k_p$ be the number of satisfied jobs of length $p$ in $s^*$. By a simple exchange argument, we can assume that these $k_p$ jobs have the maximal due-date among the jobs of length $p$ in the instance. Thus, the choice of jobs is determined by the values of $k_1$ and $k_2$, and the number of guesses needed is $O(n^2)$.

Next, we show how to construct a schedule of the chosen jobs. Similarly to Moore-Hodgson's algorithm for a single machine [14], we consider the jobs in EDD order. With $m$ parallel machines, a considered job has $m$ possible assignments. Two natural heuristics are to assign each job $j$ to a least loaded machine, or to a most loaded machine on which it is non-tardy. We start by providing simple examples showing that these approaches as well as other greedy heuristics are sub-optimal.

**Sub-optimality of a greedy approach.** Consider 5 jobs $\{j_1, \ldots, j_5\}$, such that $p_1 = p_2 = 1$, $p_3 = p_4 = p_5 = 2$, $d_1 = d_2 = 2$, and $d_3 = d_4 = d_5 = 4$. There are 2 machines. In an optimal schedule all jobs are satisfied, for example by assigning $j_1, j_2, j_3$ to one machine, and $j_4, j_5$ to the other. However if the jobs are assigned to lightly loaded machines in EDD order, $j_1$ and $j_2$ are assigned to different machines, and only 4 jobs are assigned in the resulting schedule.

The next example demonstrates that a greedy approach in which the jobs are considered in EDD order and each job is assigned to a most loaded machine on which it is non-tardy, is sub-optimal. Consider 4 jobs $\{j_1, \ldots, j_4\}$, such that $p_1 = p_2 = 1$, $p_3 = p_4 = 2$, $d_1 = d_2 = 2$, and $d_3 = d_4 = 3$. There are 2 machines. In an optimal schedule all jobs are satisfied, for example by assigning $j_1, j_3$ to one machine, and $j_2, j_4$ to the other. However if the jobs are assigned to a most loaded feasible machine in EDD order, $j_1$ and $j_2$ are assigned to one machine, and only one of the longer jobs can be satisfied.

Note that the above examples are valid regardless of tie breaking between jobs having the same due-date.

Instead, in our algorithm, if there are multiple machines to which $j$ can be assigned without being late, we maintain all possibilities for the loads of the machines after its assignment. In order to maintain a polynomial runtime, we only consider assignments to machines such that the difference in loads between any two machines is at most 2. Formally, we assume that after a job is assigned, for every machine $i$, its load is in $\{L, L-1, L-2\}$ for some value $L$. We can then store all possibilities of machine loads using a dynamic programming approach, with an $n \times n \times (n^3)$ table $S$ for all possible load options after a job is considered.

Formally, the DP table includes boolean values such that $S[j; L; x_0, x_1, x_2] = True$ if and only if there exists a schedule of the first $j$ jobs in the EDD order such that, for $z \in \{0, 1, 2\}$, exactly $x_z$ machines have load $L - z$, where $L$ is the only integer for which $\sum_{\ell=1}^{j} p_\ell = x_0 L + x_1(L-1) + x_2(L-2)$, such that $x_0, x_1, x_2$ are integers, and $x_0 > 0$ (that is, the most loaded machine has load $L$). No machine has load higher than $L$ or lower than $L - 2$. Note that for every choice of $j, x_0 > 0, x_1, x_2$, there is a unique value of $L$ for which $S[j; L; x_0, x_1, x_2]$ may be true. Thus, the table can be constructed without the $L$-dimension, resulting in a $n \times (n^3)$ table. In the sequel, we include the value of $L$ in the table $S$ for clarity.

Initially, $S[0;0;m,0,0] = True$, and for every other value of $x_0, x_1, x_2$, $S[0;0;x_0,x_1,x_2] = False$. That is, before any jobs are considered, the only possible schedule is of $m$ machines with load 0.

For $j > 0$, consider first the case $p_j = 1$. The table $S$ is updated as follows:

1. If $x_2 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0,x_1+1,x_2-1] = True$. This corresponds to adding $j$ to a machine with load $L-2$. Note that since the jobs are sorted in EDD order, it must be that $d_j \geq L > L-1$, so $j$ is satisfied.
2. If $x_1 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0+1,x_1-1,x_2] = True$. This corresponds to adding $j$ to a machine with load $L-1$. Note that since the jobs are sorted in EDD order, it must be that $d_j \geq L$, so $j$ is satisfied.
3. If $d_j \geq L+1$, $x_0 > 0$, and $S[j-1;L;x_0,x_1,0] = True$, then $S[j;L+1;1,x_0-1,x_1] = True$. This corresponds to adding $j$ to a machine with load $L$. Note that we only consider this if no machines have load $L-2$ when $j$ is considered, thus maintaining a maximum difference of 2 between the loads of different machine.

For $j > 0$, and $p_j = 2$, the table $S$ is updated as follows:

1. If $x_2 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0+1,x_1,x_2-1] = True$. This corresponds to adding $j$ to a machine with load $L-2$. Note that since the jobs are sorted in EDD order, $d_j \geq L$, so $j$ is satisfied.
2. If $d_j \geq L+1$, $x_1 > 0$, and $S[j-1;L;x_0,x_1,0] = True$, then $S[j;L+1;1,x_0,x_1-1] = True$. This corresponds to adding $j$ to a machine with load $L-1$. Note that we only consider this if no machines have load $L-2$ when $j$ is considered, thus maintaining a maximum difference of 2 between the loads of different machines.
3. If $d_j \geq L+2$, $x_0 > 0$, and $S[j-1;L;x_0,0,0] = True$, then $S[j;L+2;1,0,x_0-1] = True$. This corresponds to adding $j$ to a machine with load $L$. Note that we only consider this if no machines have load $L-2$ or $L-1$ when $j$ is considered, thus maintaining a maximum difference of 2 between the loads of different machines.

We are now ready to describe the algorithm for $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

**Algorithm 1** Algorithm for $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

Guess $0 \leq k_1 \leq |\{j|p_j = 1\}|$, and $0 \leq k_2 \leq |\{j|p_j = 2\}|$.
Let $S$ be an $n \times n \times n^3$ table initialized to $False$.
Let $S[0;0;m,0,0] = True$
Let $\mathcal{J}$ be the set of $k_1$ highest due-date 1-jobs, and $k_2$ highest due-date 2-jobs.
**for** all jobs $j$=1 to $|\mathcal{J}|$, considered in EDD order **do**
    Update $S[j;L;x_0,x_1,x_2]$ for every $L, x_0, x_1, x_2$.
**end for**
Let $x_0, x_1, x_2, L$ be values for which $S[|\mathcal{J}|;L;x_0,x_1,x_2] = True$. Return a corresponding schedule.

▶ **Theorem 6.** *Algorithm 1 computes an optimal schedule for* $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

**Proof.** We first justify the fact that the table $S$ considers only partial assignments of jobs in EDD order, in which the maximal gap in the loads of different machines is 2.

▷ Claim 7. There exists an optimal schedule $O$ such that $(i)$ the internal order of jobs on any machine is EDD, and $(ii)$ when the jobs are considered in EDD order, for every $j$, after the assignment of job $j$, the maximal gap in loads between machines is 2.

Proof. Consider an optimal schedule $O$. Simple exchange argument implies the first property in the claim, that is, the jobs assigned to each machine in $O$ are sorted in EDD order. Therefore, we can assume $O$ is constructed by iterating over jobs in EDD order, and assigning the jobs one after the other with no intended idle. Assume by contradiction that there is some job such that after it is assigned to machine $i$, there is a machine $i'$ such that $L_i - L_{i'} > 2$. Let $j_0$ be the first job for which this occurs.

Let $A$ be the set of jobs considered after $j_0$ that are assigned to $i$, and let $B$ be the set of jobs considered after $j_0$ that are assigned to $i'$. If $B \neq \emptyset$, let $j_1$ be the first job in $B$. We describe how to convert $O$ to another schedule, $O'$, in which all the jobs are satisfied and the gap between $i$ and $i'$ reduces to at most 2.

For jobs considered before $j_0$, they are assigned to the same machine as in $O$. $j_0$ is assigned to $i'$ instead of $i$, and since the jobs are sorted in EDD order, it remains satisfied. We prove there is a valid assignment of the remaining jobs. For a job assigned to a machine other than $i$ or $i'$ in $O$, it is assigned to the same machine in $O'$, and is clearly satisfied. For jobs in $A \cup B$, we divide into cases. A visual description of the constructed assignment is given in Figure 1.

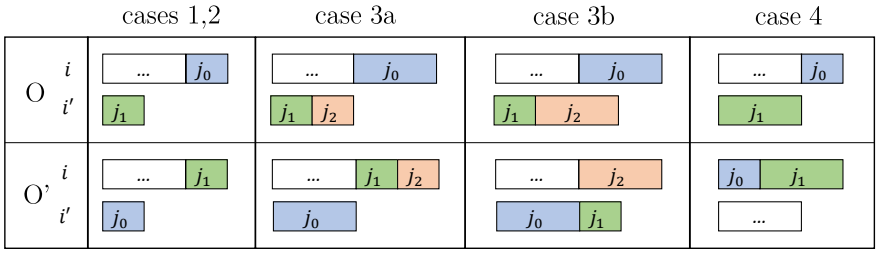


**Figure 1** Converting $O$ to $O'$.

1. If $B = \emptyset$, we assign all jobs in $A$ to $i$. They are clearly satisfied as the load on $i$ in $O'$ after a job is considered, is lower than the load on $i$ in $O$ after the same job is considered.
2. If $p_{j_0} = p_{j_1}$, assign $j_1$ to $i$, all jobs in $A$ to $i$, and all jobs in $B \setminus \{j_1\}$ to $i'$. $j_1$ is satisfied since the jobs are sorted in EDD order. All remaining jobs are satisfied as they are processed at the same time in $O$ and $O'$.
3. If $p_{j_0} = 2$ and $p_{j_1} = 1$. If $B = \{j_1\}$, assign $j_1$ to $i$ and all jobs in $A$ to $i$, and clearly all jobs are satisfied. Otherwise, let $j_2$ be the second job in $B$.
   If $p_{j_2} = 1$, assign $j_1$ and $j_2$ to $i$, and all jobs in $A$, $B \setminus \{j_1, j_2\}$ to $i$, $i'$ respectively. If $p_{j_2} = 2$, assign $j_1$ to $i'$, $j_2$ to $i$, and all jobs in $A$, $B \setminus \{j_1, j_2\}$ to $i$, $i'$ respectively. $j_1, j_2$ are satisfied since the jobs are sorted in EDD order, and the remaining jobs are satisfied as they are processed at the same time in $O$, $O'$.
4. If $p_{j_0} = 1$ and $p_{j_1} = 2$, since $j_0$ is the first job such that $L_i - L_{i'} > 2$ after its assignment, we have $L_i - L_{i'} = 3$. Thus, when $j_0$ is assigned to $i'$ instead of $i$, $L_i - L_{i'} = 1$. We assign $j_1$ to $i'$, all jobs in $A$ to $i'$, and all jobs in $B \setminus \{j_1\}$ to $i$.
   $j_1$ is satisfied as the jobs are sorted in EDD order. Since the due-dates are not machine-dependent, and all jobs in $A$, $B \setminus \{j_1\}$ are processed at the same time in $O$ and $O'$, they are also satisfied in $O'$.

Note that $O'$ is an optimal schedule with the same assigned jobs as $O$. Additionally, after $j_0$ is assigned the difference in loads between machines is at most 2. Therefore, by repeating this procedure we arrive at an optimal schedule such that the gap between machine loads is at most 2 after each job is assigned. ◁

Let $O$ be an optimal schedule fulfilling the conditions specified in Claim 7. Assume that for each $p \in \{1,2\}$, there are $k_p^\star$ satisfied $p$-jobs in $O$. Recall that we can assume these jobs are the $k_p^\star$ $p$-jobs with maximal due-date.

Consider Algorithm 1 for a guess of $k_1^\star, k_2^\star$ jobs with length $1,2$, respectively. The jobs in Algorithm 1 are assigned in EDD order, and by the definition of $S$, a valid assignment of $k_1^\star + k_2^\star$ jobs will be constructed. ◀

Algorithm 1 is for the specific case $p_j \in \mathcal{P} = \{1,2\}$ for every $j$. For other integer values of $\mathcal{P}$, it is possible to generalize Algorithm 1 to an algorithm for $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ that has a running time of $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$.

The generalization is based on the fact that regardless of $\mathcal{P}$, every instance has an optimal schedule in which the internal order of jobs on any machine is EDD, and the fact that if we can bound the maximal gap between the loads of different machines, then it is simple to extend the dynamic programming table $S$ to consider all partial schedules.

We generalize Claim 7 and show that for any $\mathcal{P}$, there exists an optimal assignment in which the maximal gap between any two machines is $lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$. The idea is to consider $j_0$, the first job such that after it is placed, $L_i - L_{i'} > lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$. Consider the set $A$ of jobs of total processing time at least $lcm(\mathcal{P}) \cdot |\mathcal{P}|$ most recently processed on $i$ when $j_0$ is assigned. Note that $j_0 \in A$. As there are only $|\mathcal{P}|$ different job sizes and they all divide $lcm(\mathcal{P})$, there must be a subset of jobs $A' \subseteq A$ with total length exactly $lcm(\mathcal{P})$ in $A$.

Let $B$ be the minimal set of jobs with total processing time at least $lcm(\mathcal{P}) \cdot |\mathcal{P}|$ assigned to $i'$ after $j_0$ is considered. As there are only $|\mathcal{P}|$ different job sizes and they all divide $lcm(\mathcal{P})$, there must be a subset of jobs $B' \subseteq B$ with total processing time exactly $lcm(\mathcal{P})$. We assign the jobs in $A'$ to $i'$, and the jobs in $B'$ to $i$.

As the jobs are considered in EDD order, they are all satisfied. All jobs assigned later are assigned to the same time slot, so they remain satisfied. By repeating this procedure, we are left with a schedule with a maximal gap of $lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$.

Thus, we can conclude the following.

▶ **Theorem 8.** *An optimal schedule for $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ can be computed in time $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$.*

## 6 Discussion

We introduced and studied a natural generalization of classical scheduling problems, in which due-dates are machine-dependent. This setting models practical environments such as manufacturing, transportation, and logistics systems where job tolerance depends on the specific machine handling the job.

We focused on the objective of maximizing throughput, defined as the number of non-rejected and non-tardy jobs. We presented a polynomial-time algorithm for unit-length jobs, which is suitable also in the presence of machine-dependent release times and weights, and showed that the problem becomes APX-hard even for instances with $p_j \in \{1,2\}$. To isolate the source of this hardness, we provided an optimal dynamic programming algorithm for instances with job-dependent due-dates and a constant number of job lengths.

Our results establish the computational landscape of this model and motivate further exploration. In particular, it would be interesting to study approximation algorithms for the general case, parameterized complexity with respect to the number of machines or job types, and extensions to additional objectives such as minimizing maximal or total tardiness ($T_{max}$ or $\sum_j T_j$) as well as instances with non-identical machines.

## References


1 A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001. doi:10.1137/S0097539799354138.

2 D. Briskorn and S. Waldherr. Anarchy in the $U_j$: Coordination mechanisms for minimizing the number of late jobs. *European Journal of Operational Research*, 301(3):815–827, 2022. doi:10.1016/j.ejor.2021.11.047.

3 J. L. Bruno, E. G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. doi:10.1145/361011.361064.

4 C. Durr and M. Hurand. Finding total unimodularity in optimization problems solved by linear programs. In *14th European Symp. on Algorithms (ESA)*, 2006. doi:10.1007/11841036_30.

5 R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math*, 5:287–326, 1979.

6 N. G. Hall. Scheduling problems with generalized due dates. *IIE Transactions*, 18(2):220–222, 1986.

7 W. A. Horn. Technical note—minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973. doi:10.1287/opre.21.3.846.

8 D. Hyatt-Denesik, M. Rahgoshay, and M. R. Salavatipour. Approximations for throughput maximization. *Algorithmica*, 86:1545–1577, 2024. doi:10.1007/s00453-023-01201-4.

9 V. Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37:27–35, 1991. doi:10.1016/0020-0190(91)90246-E.

10 E. L. Lawler. Recent results in the theory of machine scheduling. In A. Bachem, M. Groetschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 202–234. Springer, 1982. doi:10.1007/978-3-642-68874-4_9.

11 J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Ann. Discrete Math.*, 4:121–140, 1979.

12 J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. doi:10.1007/BF01585745.

13 L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Math Soc., 2009.

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

15 B. Mor, G. Mosheiov, and D. Shabtay. Scheduling problems on parallel machines with machine-dependent generalized due-dates. *Ann Oper Res.*, 2025. doi:10.1007/s10479-025-06468-0.

16 C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

17 E. Petrank. The hardness of approximation: gap location. *Computational Complexity*, 4(2):133–157, 1994. doi:10.1007/BF01202286.

18 M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2008.

19 J. Sgall. Open problems in throughput scheduling. In *20th European Symp. on Algorithms (ESA)*, 2012. doi:10.1007/978-3-642-33090-2_2.

20 V. R. Vijayalakshmi, M. Schröder, and T. Tamir. Minimizing total completion time with machine-dependent priority lists. *European J. of Operational Research*, 315(3):844–854, 2024. doi:10.1016/j.ejor.2023.12.030.