# Truthful Prompt Scheduling for Minimizing Sum of Completion Times

**Alon Eden**
Tel Aviv University, Israel
alonarden@gmail.com

**Michal Feldman**
Tel Aviv University and Microsoft Research, Israel
michal.feldman@cs.tau.ac.il

**Amos Fiat**
Tel Aviv University, Israel
fiat@tau.ac.il

**Tzahi Taub**
Tel Aviv University, Israel
tzahita@gmail.com

**Abstract**

We give a prompt online mechanism for minimizing the sum of [weighted] completion times. This is the first prompt online algorithm for the problem. When such jobs are strategic agents, delaying scheduling decisions makes little sense. Moreover, the mechanism has a particularly simple form of an anonymous menu of options.

**2012 ACM Subject Classification** Theory of computation → Scheduling algorithms, Theory of computation → Algorithmic mechanism design

**Keywords and phrases** Scheduling, Mechanism design, Online algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2018.27

**Related Version** A full version of the paper is available at [6], `https://arxiv.org/abs/1804.03244`.

**Funding** This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 337122, by the ISF (grant number 317/17), and by the ISF (grant number 1841/14).

## 1 Introduction

The setting herein includes [multiple] service queues and selfish agents that arrive online over time and can be processed on one of $m$ machines. Agents may have some (private) *processing time* $p$ and/or some private *weight* $w$.

The goal is to improve service as much as possible. Minimizing the sum of [weighted] completion times is one measure of how good (or bad) service really is.

This problem has long been studied, as a pure optimization problem, without strategic considerations [11]. Given a collection of jobs, processing times, and weights, the shortest weighted processing time order [26], also known as Smith's rule, produces a minimal sum of weighted completion times with a non-preemptive schedule on a single machine.


Schedules can be preemptive (where jobs may be stopped and restarted over time) or non-preemptive (where a job, once execution starts, cannot be stopped until the job is done).

To the best of our knowledge, all online algorithms for this problem have the following property: when a job arrives, there are no guarantees as to when it will finish. If preemption is allowed, even if the job starts, there is no guarantee that it will not be preempted, or for how long. If preemption is disallowed, the online algorithm keeps the job "hanging about" for some unknown length of time, until the algorithm finally decides that it is time to start it.

Essentially, this means that when one requests service, the answer is "OK – just hang around and you will get service at some unknown future date". It is in fact impossible to achieve any bounded ratio for the sum of [weighted] completion times if one has to start processing the job as soon as possible[1]. Some delay is inevitable. However, the issue we address is "does the job know when it will be served?". All of these issues are fundamental when considering that every such "job" is a strategic agent. It is not only that one avoids uncertainty, knowing the future schedule allows one to make appropriate plans for the interim. Consider a setting where you call up to arrange a 2 hour dental appointment and you are told: "Show up ASAP but there are no guarantees as to when the dentist will see you." This is a non-prompt schedule. It has some disadvantages when compared with the prompt alternative: "Show up at 17:00 and the dentist will see you immediately then."

In this paper we present *prompt* online algorithms that immediately determine as to when an incoming job will be processed (without preemption). The competitive ratio is the best possible, amongst all prompt online algorithms, even if randomization is allowed (the algorithm is in fact deterministic). The competitive ratio compares the sum of completion times of the online algorithm with the [harder to achieve] sum of completion times of an optimal preemptive schedule. Moreover, viewed in the context of jobs being strategic agents, our algorithms are also dominant strategy incentive compatible, and have a particularly simple form. We describe the algorithms in the strategic setting, but – even ignoring strategic issues – no non-trivial online algorithm for the problem of prompt scheduling of jobs was known prior to this study.

Upon arrival, agents are presented with a menu of possible options, where a menu entry is of the form $([b, e], q, \pi)$. This means that the period from $b$ to $e$ is available on machine $q$ and will cost the agent $\pi$. These menus are anonymous and do not depend on the agent that arrives. The agent then chooses one of the options. Rational agents will never choose an interval that is shorter than the processing time. (If so the agent cost is $\infty$).

The cost to the agent is the sum of two components: (a) The time spent waiting, weighted by the agents' [private] weight. *I.e.*, highly impatient agents will have high weight, less impatient agents will have lower weight. (b) The price, $\pi$, associated with an option on the menu. Agents seek to minimize their cost.

Consider the case of a single queue, a selfish agent will simply join the queue immediately upon arrival, there is no reason to delay. Thus, jobs will be processed in first-in-first-out (FIFO) order. However, this may be quite bad in terms of the sum of completion times. Imagine a job with processing time $L$, arriving at time zero, followed by $\sqrt{L}$ jobs of processing time 1, all of which arrive immediately after the first.

As the first job will only be done at time $L$, the sum of completion times for these $1+\sqrt{L}$ jobs is about $L^{3/2}$. Contrariwise, if the $\sqrt{L}$ size one jobs were processed before the size $L$ job, the sum of completion times would be about $2L$. Obviously it seems a good idea to delay longer jobs and expedite shorter jobs.

[1] This is illustrated in the introduction of the full version [6].

Similarly, consider a first batch of $L$ jobs, each of size 1 and weight 1, immediately followed by a single job of size 1 and weight $W$. For FIFO processing, the weighted sum of completion times is $L^2/2$ (for the weight 1 jobs) plus $(L+1) \cdot W$ (for the job of weight $W$). Optimally, the weight $W$ job should be processed first, followed by the size 1 jobs. The weighted sum of completion times is then about $W + L^2/2$. For any constant $L$ and sufficiently large $W$, the ratio between the two sums approaches $L+1$.

The main question addressed in this paper is how to produce such dynamic menus so as to incentivize selfish agents towards behavior that achieves some desirable social goal, specifically, minimizing the sum of completion times. The dynamic menu is produced based on the past decisions of the previous agents and the current time[2].

We measure the quality of the solution achieved by the competitive ratio, the ratio between the sum of completion times of the selfish agents, when presented with the dynamic menus, and the minimal sum of completion times, when the future arrivals and their private values are known and there are no incentive considerations. In fact, the comparison is with the optimal preemptive schedule (which could definitely be better than the optimal non-preemptive schedule).

We consider several scenarios:

1. All agents have weight 1 and arbitrary processing times, nothing known apriori on the processing times. This models cases where all agents are equally impatient but have different processing requirements. The underlying idea here is to offer menu options that delay longer jobs so that they do not overly delay many shorter jobs that arrive later.
2. All agents have processing time 1 and arbitrary weight, nothing known apriori on the weights. The underlying idea here is to set prices so as to delay jobs of small weight and thus to allow later jobs of large weight to finish early.
3. Jobs with arbitrary processing times and weights bounded by a known bound $B_{\max}$. This means that we have to delay long jobs and simultaneously have to leave available time slots for jobs with large weights.

The competitive ratios for the different scenarios appear in Table 1. We remark that the lower bounds hold even if one assumes that the machines used are arbitrarily faster than the machines used by the optimal schedule that minimizes the sum of weighted completion times.

## 1.1 Related Work

For one machine, weighted jobs, available at time zero, ordering the jobs in order of weight/-processing time minimizes the sum of competition times [26]. For one machine, unweighted jobs with release times, a preemptive schedule that always processes the job with the minimal remaining processing time minimizes the sum of weighted completion times [24, 23]. As an offline problem, where jobs cannot be executed prior to some earliest time, finding an optimal non-preemptive schedule is computationally hard [12].

For parallel machines, where jobs arrive over time, a preemptive schedule that always processes the jobs with the highest priority – weight divided by remaining processing time – is a 2 approximation [20]. This algorithm is called *weighted shortest remaining processing time* (WSRPT). If all weights are one this preemptive algorithm is called *shortest remaining processing time* (SRPT). Other online and offline algorithms to minimize the sum of completion times appear in [1, 25, 12].

[2] For clarity we describe the menu as though it was infinite. In fact, one can think of the process as though the menu is presented entry by entry. The selfish job will provably choose an option early on.

**Table 1** Competitive Ratios of our Dynamic Menus, and associated lower bounds. $P_{\max}$ is the longest job processing time in the input sequence, it is not known apriori. $W_{\max}$ is the maximal job weight in the sequence, it is not known apriori. $B_{\max}$ is an apriori upper bound on $W_{\max}$.

| Processing Time | Job Weight | Menu entries | Upper Bound (Deterministic) | Lower Bound (Randomized) |
|---|---|---|---|---|
| $p_j \in \mathbb{Z}^+$ | $w_j = 1$ | intervals (various lengths) no prices | $O(\log P_{\max})$ | $\Omega(\log P_{\max})$ |
| $p_j = 1$ | $w_j \in \mathbb{Z}^+$ | unit length intervals with prices | $O\left(\begin{matrix}\log W_{\max}\cdot \\ (\log\log W_{\max} + \log n)\end{matrix}\right)$ | $\Omega(\log W_{\max})$ |
| $p_j \in \mathbb{Z}^+$ | $w_j \in \mathbb{Z}^+$ | intervals (various lengths) with prices | $O\left(\begin{matrix}\log B_{\max}\cdot \\ (\log P_{\max} + \log n)\end{matrix}\right)$ | $\Omega\left(\max\left(\begin{matrix}\log B_{\max}, \\ \log P_{\max}\end{matrix}\right)\right)$ |

[22] show how to convert a preemptive online algorithm into a non-preemptive online algorithm while increasing the completion time of the job by no more than a constant factor. This transformation strongly depends on not determining immediately when the job will be executed. This is in comparison to a prompt algorithm that determines when the job is executed immediately upon the job's arrival.

When selfish agents are involved, it is valuable to keep things simple [13]. Offering selfish agents an anonymous menu of options is an example of such a simple process. More complicated mechanisms require trust on the part of the agents.

Recently, [7] considered a similar question to ours, where a job with private processing time had to choose between multiple FIFO queues, where the servers had different speeds. Here, dynamic posted prices were associated with every queue, with the goal of [approximately] minimizing the makespan, the length of time until the last job would finish. Shortly thereafter, [15] used dynamic pricing to minimize the maximal flow time. Dynamic pricing schemes were considered for non-scheduling cost minimization problems in [3].

A constant approximation mechanism for minimizing sum of completion times for selfish jobs was considered in [9], where the setting was an offline setting, the processing time was known in advance and the weight was private information. In an online setting, [14] show a constant approximation preemptive mechanism that gives an $O(1/\epsilon^2)$ approximation to the sum of flow times when using machines that are faster by a factor of $1+\epsilon$.

Online mechanisms were considered in [17, 8], whereas prompt online mechanisms are defined in [4].

In this paper our goals are pricing schemes that affect agents as to behave in a manner that [approximately] minimizes the sum of weighted completion times.

There is a vast body of work on machine scheduling problems, in offline and online settings, with strategic agents involved and not, and in a host of models. It is impossible to do justice to this body of work but a very short list of additional relevant papers includes [10, 19, 11, 18, 21, 2, 16].

## 1.2 Organization of this paper

In Section 2 we describe our model. In Section 3 we give an optimal $O(\log P_{\max})$-competitive menu based mechanism for the case of arbitrary [unknown] lengths and identical weights, and in Section 4 we show a matching $\Omega(\log P_{\max})$ lower bound that holds for any [randomized, non-truthful] prompt online algorithm. In the full version [6] we handle the case of arbitrary weighted jobs with identical processing times and give a $O(\log W_{\max}(\log\log W_{\max} + \log n))$-competitive pricing menu. An $\Omega(W_{\max})$ lower bound for this case is given.

## 2 The Model

We consider a job scheduling setting with $m$ machines and $n$ jobs that arrive in real time, where $p_j$, $w_j$, and $r_j$ are, respectively, the processing time, weight, and release time of the $j$th job to arrive. It may be that $r_j = r_{j+1}$, *i.e.*, more than one job arrive at the same time. However, job decisions are made sequentially in index order.

A valid input for this problem can be described as a sequence of jobs

$$\sigma = (r_1, w_1, p_1), (r_2, w_2, p_2), \ldots, (r_n, w_n, p_n),$$

where the *release time* $r_i \leq r_{i+1}$ for $i = 1, \ldots, n-1$, the *job weight* $w_i \geq 1$ for $i = 1, \ldots, n$, and the job *processing time* $p_i \geq 1$ for $i = 1, \ldots, n$. We refer to the $j$th job in this sequence as job $j$. We use the terms *size* and processing time interchangeably. Let $\sigma[1..\ell]$ be the length $\ell$ prefix of $\sigma$. The total volume of a set of jobs $D$, denoted $vol(D)$ is the sum of processing times of the jobs in $D$, i.e., $vol(D) = \sum_{j \in D} p_j$.

Let $s_j \geq r_j$ be the time at which job $j$ starts processing (on some machine $1 \leq q \leq m$). The completion time of job $j$ is $c_j = s_j + p_j$.

The objective considered in this paper is to minimize the sum of [weighted] completion times; *i.e.*, we wish to minimize $\sum_{j=1}^{n} w_j \cdot c_j$.

For jobs $j$, $j'$, with $j < j'$ and with $r_j = r_{j'}$, job $j$ is assigned (or chooses) machine $q_j$ at time $s_j$ before job $j'$ is assigned machine $q_{j'}$ at $s_{j'}$. We say that $(q_j, s_j)$ and $(q_{j'}, s_{j'})$ *overlap*, if $q_j = q_{j'}$ and $(s_j \leq s_{j'} < c_j = s_j + p_j$ or $s_{j'} \leq s_j < c_{j'} = s_{j'} + p_{j'})$.

A *valid* (non-preemptive) schedule for an input $\sigma$ is a sequence

$$(m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$$

where no overlaps occur. An *online* algorithm determines $(m_j, s_j)$ after seeing $\sigma[1 \ldots j]$ and before seeing job $j+1$.

We consider online mechanisms where jobs are selfish agents, processing times and weights are private information, and job $j$ is presented with a menu of options upon arrival. Every option on the menu is of the form $(I, q, \pi)$ where (i) $I$ is a time interval $[b(I), e(I)]$, with integer endpoints, and where $b(I) \geq r_j$, (ii) $1 \leq q \leq m$ is some machine, and (iii) $\pi$ is the price for choosing this entry. The menu of options presented to job $j$ is computed after jobs $1, \ldots, j-1$ have all made their choices and also depends on the release time of job $j$, $r_j$ (because one cannot process a job in the past). We assume *no feedback* from jobs after they choose their menu options, i.e., if a job of size $p$ chooses an interval $I$ of length $|I| > p$ on some machine, we do not know the interval is only partly used, and specifically, cannot offer the $|I| - p$ remaining to future jobs.

For job $j$ that chooses menu entry $([b(I), e(I)], q, \pi)$ we use the following notation (i) $I(j)$ for the interval chosen by job $j$, $[b(I), e(I)]$, (ii) $M(j)$ for the machine chosen by job $j$, $q$, and (iii) $\Pi(j)$ for the price of the entry chosen by $j$, $\pi$.

Although the menus we describe are infinite, one can present the menu items sequentially. With unit weight jobs, a job of processing time $p$ will make its choice within the first $\log p$ options presented. With unit length jobs, a job of weight $w$ will make its choice within the first $\log w$ options presented. With arbitrary lengths and arbitrary weights, a job of processing time $p$ and of weight $w$ will make its choice within the first $\log p \cdot \log w$ options presented.

The cost to job $j$ with weight $w_j$ and processing time $p_j$ for choosing the menu entry $([b, e], q, \pi)$ is $\infty$ if the time interval is too short: $e - b < p_j$. If $e - b \geq p_j$ then the cost to job $j$ is a cost of $w_j$ for every unit of time until job $j$ starts processing, plus the extra price

from the menu. *I.e.*, the cost to job $j$ with release time $r_j$, processing time $p_j$ and weight $w_j$, for choosing menu entry $([b,e],q,\pi)$, $e-b \geq p_j$, is

$$(b+p_j)\cdot w_j + \pi.$$

For the specialized cases of weight one jobs or unit length jobs the general model above is somewhat simpler:

## 2.1 Modeling weight one jobs with arbitrary Processing times

If jobs have weight one, we give (optimal) menus that do not require pricing menu entries. Any entry on the menu is available for free. Therefore, we can simplify the menu structure as follows: The job chooses a time interval and a machine from a menu with entries of the form $([b,e], 1 \leq q \leq m)$ where the first entry is a time interval, and the second entry is a machine[3].

Jobs choose from the menu one of the entries immediately upon arrival. As above, we say that job $j$ chooses menu entry $(I(j), M(j))$ where $I(j)$ is an interval, and $1 \leq M(j) \leq m$.

For job $j$ with arrival time $r_j$, and processing time $p_j$ the cost associated with choosing the menu item $([b,e], 1 \leq q \leq m)$ is $\infty$ if $p_j > e-b$ and $(b+p_j)$ otherwise. Jobs always seek to minimize their cost.

## 2.2 Modeling unit length jobs of arbitrary weight

Every job requires one unit of processing time on one of $m$ different processors. Every job $j$ is a selfish agent that has a private weight $w_j$, the cost to the job of one unit of delay.

The job chooses a machine and time slot from a menu with entries of the form $([i,i+1], 1 \leq q \leq m, \pi)$ where the first entry is a time slot, the second entry is a machine, and the third entry is the price of this time slot on the machine.

Jobs choose from the menu one of the entries immediately upon arrival. Job $j$ is said to choose menu item $(I(j), M(j), \Pi(j))$ where $I(j)$ is a length one interval, $1 \leq M(j) \leq m$, and $\Pi(j)$ is the price to be paid for choosing this option.

For job $j$ with arrival time $r_j$, and weight $w_j$ the cost associated with choosing the menu item $([i,i+1], 1 \leq q \leq m, \pi)$ is $w_j(i+1)+\pi$. Jobs always seek to minimize their cost.

# 3 Dynamic Menu for Jobs with Heterogeneous Processing Times

In this section we introduce a dynamic menu based mechanism, for jobs of weight one and heterogeneous processing times, with competitive ratio $O(\log P_{\max})$, where $P_{\max}$ is the maximal job processing time among all jobs. Due to lack of space, the analysis of the mechanism is deferred to the full version [6].

In Section 3.1 we provide integer sequences and corresponding interval sequences that serve as a building block for our dynamic menu mechanism, which is presented in Section 3.2.

## 3.1 The $S_k$ Integer and Interval Sequences

We define sequences of integers $S_k$, $k = 0, 1, \ldots$, as follows: Let $S_0 = \langle 1 \rangle$ and for $k > 0$ let $S_k = S_{k-1} \| S_{k-1} \| \langle 2^k \rangle$ where $\|$ denotes concatenation. Ergo,

$$S_0 = \langle 1 \rangle; \quad S_1 = S_0 \| S_0 \| \langle 2^1 \rangle = \langle 1,1,2 \rangle; \quad S_2 = S_1 \| S_1 \| \langle 2^2 \rangle = \langle 1,1,2,1,1,2,4 \rangle; \quad \cdots$$

[3] Although the general setting allows pricing menu items, it turns out that for weight 1 jobs the optimal menu does not need to differentiate entries by price.

Let $n_k = 2^{k+1} - 1$ denote the length of $S_k$ (follows inductively from $n_0 = 1$ and $n_k = 2n_{k-1} + 1$). Let $S_k[i]$, $i = 1, \ldots, n_k$ be the $i$th element of $S_k$. Let $S_\infty$ be an infinite sequence whose length $n_k$ prefix is $S_k$ (for all $k$):

$$S_\infty = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 16, 1, \ldots \rangle.$$

Let $S_\infty[i]$, $i = 1, 2, \ldots$ be the $i$th element of $S_\infty$. Note that $S_k[i] = S_{k'}[i]$ for all $k \leq k'$ and all $i = 1, \ldots, n_k$, ergo, $S_k$ is a prefix of $S_{k'}$ for $k \leq k'$.

▶ **Lemma 1.** *For all $d \geq 0$, for all $0 \leq k \leq d$, the sum of all the $2^k$ value items in $S_d$ is equal $2^d$. That is,* $\sum_{1 \leq i \leq n_d : S_d[i] = 2^k} 2^k = 2^d$.

We use the $S_k$ sequences to define interval sequences. Let $\gamma_i$ be the sum of the first $i$ entries in $S_\infty$, $\gamma_i = \sum_{j=1}^{i} S_\infty[j]$ (*i.e.*, $\gamma_1 = 1$, $\gamma_2 = 2$, $\gamma_3 = 4$, etc.).

We define $S_k(t)$, $t \geq 0$, to be a sequence of $n_k$ consecutive intervals, the first of which starts at time $t$, and where the length of the $j$th interval equals $S_k[j]$. *I.e.*,

$$S_k(t) = \langle [t, t + \gamma_1], [t + \gamma_1, t + \gamma_2], \ldots, [t + \gamma_{n_k - 1}, t + \gamma_{n_k}] \rangle .$$

For example

$$S_2(2) = \langle [2, 3], [3, 4], [4, 6], [6, 7], [7, 8], [8, 10], [10, 14] \rangle. \tag{1}$$

For any interval sequence $S$ let $b(S)$ be the start of the first interval in $S$ and let $e(S)$ be the end of the last interval in $S$. For example, $b(S_2(2)) = 2$ and $e(S_2(2)) = 14$.

We say that $S_k$ *appears* in $S_d(t)$ if there exists some $t'$ such that the interval sequence $S_k(t')$ is a contiguous subsequence of $S_d(t)$. In this case we also say that $S_k(t')$ appears in $S_d(t)$. Note that while $S_k$ is a sequence of integers, both $S_k(t')$ and $S_d(t)$ are interval sequences.

By construction, for any $k$ and any $t \neq t'$ if $S_k(t)$ and $S_k(t')$ appear in some $S_d(\tilde{t})$, then $[b(S_k(t)), e(S_k(t))]$ and $[b(S_k(t')), e(S_k(t'))]$ are disjoint except for, possibly, their endpoints. Let $I$ be an interval of length $2^k$ that appears in $S_\infty(t)$. Then there is a unique $t'$ such that $S_k(t')$ appears in $S_\infty(t)$ and $I$ is the last interval of $S_k(t')$. It follows from Lemma 1 that

▶ **Corollary 2.** *For all $k \leq d$, for all $t$,*

1. *$S_k$ appears in $S_d(t)$ $2^{d-k}$ times.*
2. *The sum of the lengths of the intervals in $S_d(t)$ is $(d+1)2^d$.*

The interval sequences defined above suggests a new possible static algorithm. Divide the timeline of each machine into intervals as in $S_\infty(0)$, and let any job that arrives occupy the first unoccupied interval it fits in. Unfortunately, when the competitive ratio is evaluated as a function of $P_{\max}$ alone, this algorithm is $\Omega\left(\sqrt{P_{\max}}\right)$ competitive[4] (When the competitive ratio may be a function of $P_{\max}$ and $n$, this algorithm is $O\left(\log P_{\max} + \log n\right)$ competitive as analyzed in Theorem 5 of the full version).

▶ **Definition 3.** A *state* is a vector of consecutive interval sequences of the form

$$\begin{aligned} A &= \langle A_1, A_2, \cdots, A_\ell \rangle \text{ where} \\ A_i &= S_{k_i}(t_i) \text{ for every } 1 \leq i \leq \ell, \end{aligned}$$

for some $\ell$ (which we refer to as the *length* of $A$) and integers $k_i$ for $1 \leq i \leq \ell$, and where $e(A_i) = e(S_{k_i}(t_i)) \leq t_{i+1} = b(A_{i+1})$ for $1 \leq i \leq \ell - 1$. This means that the interval sequences are disjoint and ordered by their starting times. Note that there might be gaps between two consecutive state entries, i.e., $e(A_i) < b(A_{i+1})$ for some $1 \leq i \leq \ell - 1$.

[4] This is shown in Appendix B in the full version [6].

## 3.2 $O(\log P_{\max})$ Competitive Dynamic Menu

When job $j+1$ arrives the algorithm is in some *configuration* $\psi^j = \left(A^j, X^j\right)$, where $A^j$ is some state of length $\ell_j$, and $X^j$ is the set of intervals occupied by the previous $j$ jobs. State $A^j$ represents every machines' division of $\left[0, \max_{i\in[j]} c_i\right]$ into time intervals (same division for all machines). This division will be kept at any future time. For every $i < \ell_j$, $A^j_i$ is fixed and will be a part of every future state, while $A^j_{\ell_j}$ might be subject to change. We refer to $A^j_{\ell_j}$ as the *tentative sequence* of state $A^j$. $X^j$ keeps track of all previously allocated intervals (in all machines): $([b,e],q) \in X^j$ means that some job $j' \le j$ chose the interval $[b,e]$ on machine $1 \le q \le m$. Note that the size of job $j'$, $p_{j'}$, might be strictly smaller than the length of the interval $(e-b)$, yet it is still considered occupied.

We note that our mechanism has the property that, roughly, every time interval in state $A_j$ has a $\frac{1}{\log P_{\max}}$ fraction of its volume allocated to "small" jobs.

### Generating the Dynamic Menu

Given a state $A = (A_1, A_2, \ldots, A_\ell)$ and a time $t$, we define an interval sequence $\tau$ as follows:

$$\tau(A,t) = \begin{cases} A_1 \| A_2 \| \ldots \| A_\ell \| S_\infty(t) & t \ge e\left(A_\ell\right) \\ A_1 \| A_2 \| \ldots \| A_{\ell-1} \| S_\infty(b\left(A_\ell\right)) & t < e\left(A_\ell\right) \end{cases}$$

$\tau$ is used to create the menu presented to a job $j$. We present an algorithm for the creation of the menu, based on the previous configuration $\psi^{j-1}$, and the current time $t$.

- Let $\tau^j = \tau\left(A^{j-1}, r_j\right)$.
- Set $d_1$ to be the length of the first time interval in $\tau^j$ beginning at time $b_1 \ge t$.
- Add $([b_1, b_1+d_1], q)$ to the menu for all machines $1 \le q \le m$ in which $[b_1, b_1+d_1]$ is unoccupied (i.e, $([b_1, b_1+d_1], q) \notin X^{j-1}$).
- Set $i = 1$
- Repeat until job $j$ chooses an interval:
  - Let $d_{i+1}$ be the length of the first interval longer than $d_i$ in $\tau^j$ that starts at time $b_{i+1} \ge t$ (it follows that $b_{i+1} > b_i$).
  - Add $([b_{i+1}, b_{i+1}+d_{i+1}], q)$ to the menu for all machines $1 \le q \le m$ in which $[b_{i+1}, b_{i+1}+d_{i+1}]$ is unoccupied (i.e., $([b_{i+1}, b_{i+1}+d_{i+1}], q) \notin X^{j-1}$).
  - Set $i = i+1$.

By construction, no job will ever choose a time interval that starts before the job arrival time, nor will it ever choose a slot that has already been chosen.

A selfish job of length $p_j$ always chooses a menu entry of the form $([b,e],q)$ where $b$ is the earliest menu entry with $p_j \le e-b$.

### Updating States

After job $j$ makes its choice of menu entry, $(I(j), M(j))$, we update the configuration from $\psi^{j-1} = \left(A^{j-1}, X^{j-1}\right)$ to $\psi^j = \left(A^j, X^j\right)$. Clearly, $X^j = X^{j-1} \cup \{(I(j), M(j))\}$. In the rest of this section we describe how to compute $A^j$.

Recall that a state is a vector of consecutive and disjoint interval sequences. Initially, $A^0 = \langle\rangle$ with length $\ell_0 = 0$ and $A^0_{\ell_0}$ is an empty sequence with $b\left(A^0_{\ell_0}\right) = e\left(A^0_{\ell_0}\right) = 0$. $A^j$ always contains all of $A^{j-1}$'s interval sequences except possibly the tentative sequence $A^{j-1}_{\ell_{j-1}}$. When job $j$ of size $2^k$ chooses an interval, the new tentative sequence $A^j_{\ell_j}$ can be one of the following:

**Table 2** Update rules: After job $j$ makes its choice (and $c_j$ is determined), the new state $A^j$ is a function of (i) $A^{j-1}$, (ii) release time $r_j$, (iii) processing time $p_j = 2^k$, and (iv) completion time $c_j$.

| | $\ell_j$ | $A^j_{\ell_j}$ | $c_j \le e\left(A^{j-1}_{\ell_{j-1}}\right)$ | $r_j \ge e\left(A^{j-1}_{\ell_{j-1}}\right)$ | $A^{j-1}_{\ell_{j-1}} = S_d(t)$ $k \le d$ |
|---|---|---|---|---|---|
| 1 | $\ell_{j-1}$ | $A^{j-1}_{\ell_{j-1}}$ | True | - | - |
| 2 | $\ell_{j-1}+1$ | $S_k(r_j)$ | False | True | - |
| 3 | $\ell_{j-1}+1$ | $S_k\left(e\left(A^{j-1}_{\ell_{j-1}}\right)\right)$ | False | False | True |
| 4 | $\ell_{j-1}$ | $S_k\left(b\left(A^{j-1}_{\ell_{j-1}}\right)\right)$ | False | False | False |

1. *Unchanged from former:* The new tentative sequence in $A^j$ is the same as the former tentative sequence in $A^{j-1}$, *i.e.*, $A^j_{\ell_j} = A^{j-1}_{\ell_{j-1}}$. This happens when $I(j) \in A^{j-1}$, see entry 1 in Table 2.
2. *Disjoint from former:* The former tentative sequence, $A^{j-1}_{\ell_{j-1}}$ becomes *fixed*, and the new tentative sequence $A^j_{\ell_j}$ is disjoint from the former. The tentative sequence in $A^{j-1}$, $A^{j-1}_{\ell_{j-1}}$, is the $\ell_{j-1}$th element in all future states $A^i$, for $i \ge j$. See entries 2 and 3 in Table 2.
3. *Extension of former:* The new tentative sequence is an *extension* of the former tentative sequence. *I.e.*, if $A^{j-1}_{\ell_{j-1}} = S_d(t)$ then $\ell_j = \ell_{j-1}$ and $A^j_{\ell_j} = S_k(t), k > d$. See entry 4 in Table 2.

Let $A^j_i, A^j_{i+1}$ be two consecutive interval sequences in a state $A^j$. If $b\left(A^j_{i+1}\right) > e\left(A^j_i\right)$, we say the interval $\left[e\left(A^j_i\right), b\left(A^j_{i+1}\right)\right]$ is a *gap*.

Figure 1 is an example with 5 jobs that arrive over time, and the matching configuration changes. The jobs in Figure 1 illustrate cases 1–4 from Table 2 in the following order: case 2 for job 1, case 1 for job 2, case 3 for job 3, case 4 for job 4 and case 2 for job 5.

### 3.2.1 High level overview of the analysis

Due to lack of space, we only give a high level overview of the analysis. The full analysis appears in Section 3.4 of the full version [6]. We first show that w.l.o.g. one may assume that all job lengths are powers of 2 and that the adversary's schedule never includes gaps. In our analysis, we compare the completion time of each job under our mechanism with the completion time of the same job under SRPT. Let $j$ be a job in the input sequence. We define $D(j) = \{j' \le j | p_{j'} \le p_j\}$ to be the set of all jobs that arrived no later than job $j$ and that are no bigger (note $j \in D(j)$). These jobs are all completed no later than job $j$ both under our mechanism and under SRPT, implying $c^*_j \ge \frac{1}{m} vol(D(j))$ (where $c^*_j$ is the completion time of job $j$ under SRPT). Our analysis is based upon this observation. We show that the mechanism depicted above ensures that $c_j = O(\log P_{\max}) \cdot c^*_j$ for every job $j$. Since SRPT is an $O(1)$-competitive algorithm, this immediately implies the following.

▶ **Theorem 4.** *Our mechanism is $O(\log P_{\max})$-competitive.*

## 3.3 Arbitrary processing times, weight $\le B_{\max}$

The static algorithm suggested at the end of Section 3.1 used for weight one jobs of arbitrary sizes can be easily adapted to weights in some predetermined range from 1 to $B_{\max}$. Replicate every interval in the sequence $S_\infty(0)$ $\log B_{\max} + 1$ times. For $\ell = 0, \ldots, \log B_{\max}$, the $\ell$th copy is designed to hold only jobs of weight $\ge 2^\ell$. To achieve this, one associates prices with

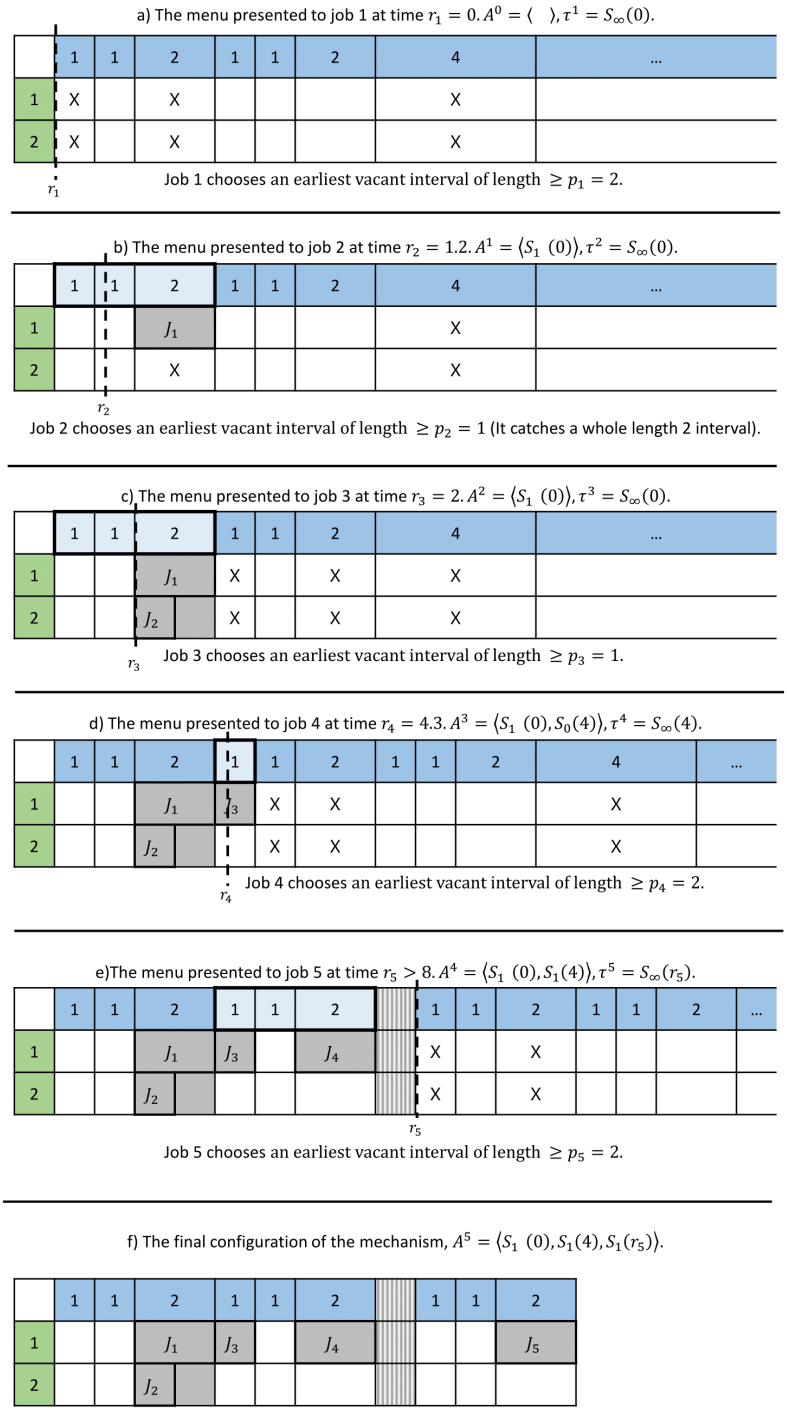


**Figure 1** Changing Menus of the Dynamic Menu Algorithm, as jobs arrive and make choices. The two bottom rows in the tables represents two machines. An $X$ in a machine cell represents an (interval,machine) entry in the currently presented menu. A dashed line marks the release time of the current job. Gray cells represent choices previously made by jobs. A gap is represented by a rectangle filled with vertical lines. A rectangle outline in the top row of a table represents the tentative sequence before job $j$ makes it choice, i.e., $A^{j-1}_{\ell_{j-1}}$. Note that this example does not make the simplicity assumptions made in the analysis.

such intervals, as done in Section 5 of the full version. The analysis preformed in Appendix A of the full version holds when multiplying every element with $\log B_{\max} + 1$, implying a competitive ratio of $O\left((\log P_{\max} + \log n) \cdot \log B_{\max}\right)$.

## 4 Lower Bound on the Competitive Ratio for any Prompt Online Algorithm, Arbitrary Lengths

We now show that any prompt online scheduling algorithm must have a competitive ratio of $\Omega\left(\log P_{\max}\right)$, even if randomization is allowed.

Let $c$ be the competitive ratio of some algorithm ALG as a function of $P_{\max}$. Consider the following sequence, for $\boldsymbol{P}$ to be determined later:

For $i = 0, \ldots, 16c$:
- $n_i = 2^i$ jobs of size $P_i = \frac{\boldsymbol{P}}{2^i}$ arrive one after the other (at time 0).
- If the expected number of $P_i$ sized jobs with completion time greater than $8c\boldsymbol{P}$ is at least $n_i/2$, stop the sequence. Let $j$ be the last iteration.

Note that for every $i = 0, \ldots, 16c$ it holds that $n_i \cdot P_i = \boldsymbol{P}$.

▶ **Lemma 5.** *There must be an iteration $j \in \{0, \ldots, 16c\}$ for which in expectation more than half of the jobs have completion time greater than $8c\boldsymbol{P}$.*

**Proof.** Let $X_i$ be a random variable representing the number of size $P_i$ jobs, with completion time greater than $8c\boldsymbol{P}$. If for all $i \in \{0, \ldots, 16c\}$, $\mathbb{E}\left[X_i\right] \leq n_i/2$, then the total expected volume of jobs completed before time $8c\boldsymbol{P}$ is at least

$$\sum_{i=0}^{16c} \mathbb{E}\left[(n_i - X_i) \cdot P_i\right] \geq \sum_{i=0}^{16c} \frac{n_i}{2} \cdot P_i = \sum_{i=0}^{16c} \frac{\boldsymbol{P}}{2} > 8c\boldsymbol{P},$$

a contradiction. ◀

▶ **Theorem 6.** *Any random prompt online algorithm must be $\Omega\left(\log P_{\max}\right)$ competitive for the above sequence.*

**Proof.** According to Lemma 5, there must be some $j \in \{0, \ldots, 16c\}$ for which in expectation at least half of the jobs are completed after time $8c\boldsymbol{P}$. Given this $j$, we give bounds on both OPT and ALG. Let $X_i$ be as in Lemma 5. In ALG, $\mathbb{E}\left[X_j\right] > n_j/2$, thus:

$$\mathbb{E}\left[\text{Cost(ALG)}\right] > \mathbb{E}\left[X_j \cdot 8c\boldsymbol{P}\right] > 8c\boldsymbol{P} \cdot \frac{n_j}{2} = 4c\boldsymbol{P} \cdot n_j. \tag{2}$$

In OPT, the jobs are scheduled from the smallest one (of size $P_j$) to the biggest one (of size $P_0 = \boldsymbol{P}$). The $k$th job of size $P_i$ to be scheduled, is completed after all jobs smaller than it (of sizes $P_{i+1}, \ldots, P_j$) and after $k-1$ jobs of size $P_i$, and therefore has a completion time of

$$\left(\sum_{\ell=i+1}^{j} n_\ell \cdot P_\ell\right) + P_i \cdot (k-1) + P_i = P_i \cdot k + \sum_{\ell=i+1}^{j} n_\ell \cdot P_\ell.$$

Summing over all jobs of all sizes, we have

$$\begin{aligned}\text{Cost(OPT)} &= \sum_{i=0}^{j}\left(\sum_{k=1}^{n_i}\left(P_i\cdot k+\sum_{\ell=i+1}^{j} n_\ell\cdot P_\ell\right)\right)\\ &= \underbrace{\sum_{k=1}^{n_j} P_j\cdot k}_{(i)}+\underbrace{\sum_{i=0}^{j-1}\sum_{k=1}^{n_i} P_i\cdot k}_{(ii)}+\underbrace{\sum_{i=0}^{j-1}\left(\sum_{\ell=i+1}^{j} n_\ell\cdot P_\ell\right)\cdot n_i}_{(iii)}. \end{aligned} \tag{3}$$

We now bound each term of Cost(OPT) separately.

$$(i): P_j\sum_{k=1}^{n_j} k < P_j\cdot n_j^2 = \boldsymbol{P}\cdot n_j. \tag{4}$$

$$(ii): \sum_{i=0}^{j-1} P_i\sum_{k=1}^{n_i} k < \sum_{i=0}^{j-1} P_i\cdot n_i^2 = \boldsymbol{P}\cdot\sum_{i=0}^{j-1} 2^i \le \boldsymbol{P}\cdot 2^j = \boldsymbol{P}\cdot n_j. \tag{5}$$

For $(iii)$ we have

$$\begin{aligned}(iii): \sum_{i=0}^{j-1}\left(\sum_{\ell=i+1}^{j} n_\ell\cdot P_\ell\right)\cdot n_i &= \sum_{i=0}^{j-1}\sum_{\ell=i+1}^{j}\boldsymbol{P}\cdot 2^i = \boldsymbol{P}\sum_{i=0}^{j-1}(j-i)2^i = \boldsymbol{P}\sum_{i=1}^{j} i\cdot 2^{j-i}\\ &= \boldsymbol{P}\cdot 2^j\sum_{i=1}^{j}\frac{i}{2^i} \le 2\boldsymbol{P}\cdot n_j. \end{aligned} \tag{6}$$

From Equations (4), (5) and (6), we get that $\text{Cost(OPT)} \le 4\boldsymbol{P}\cdot n_j$. Therefore,

$$\mathbb{E}\left[\text{Cost(ALG)}/\text{Cost(OPT)}\right] > c,$$

in contradiction to the assumption that ALG is $c$-competitive.

For the input sequence to be valid, it must be that $P_j \ge 1$. As $j \le 16c$, it is sufficient that $c(\boldsymbol{P}) \le \frac{1}{16}\log\boldsymbol{P}$, as in this case, $P_{16c} = \frac{\boldsymbol{P}}{2^{16c}} \ge 1$. So for every competitive ratio function $c$ such that $c(P_{\max}) = o(\log P_{\max})$ there exists a sufficiently large $\boldsymbol{P}$ for which $c(\boldsymbol{P}) \le \frac{1}{16}\log\boldsymbol{P}$, and our input is a valid counter example. ◀

## References


1. J. 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.
2. George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 345–357, 2004. doi:10.1007/978-3-540-27836-8_31.
3. Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 73–91. SIAM, 2015. doi:10.1137/1.9781611973730.7.
4. Richard Cole, Shahar Dobzinski, and Lisa Fleischer. Prompt mechanisms for online auctions. In *Proceedings of the 1st International Symposium on Algorithmic Game Theory*, SAGT '08, pages 170–181, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-79309-0_16.

**5** Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin, editors. *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*. ACM, 2017.

**6** Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub. Prompt scheduling for selfish agents. *CoRR*, abs/1804.03244, 2018. `arXiv:1804.03244`.

**7** Michal Feldman, Amos Fiat, and Alan Roytman. Makespan minimization via posted prices. In Daskalakis et al. [5], pages 405–422. `doi:10.1145/3033274.3085129`.

**8** Eric J. Friedman and David C. Parkes. Pricing wifi at starbucks: Issues in online mechanism design. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, EC '03, pages 240–241, New York, NY, USA, 2003. ACM. `doi:10.1145/779928.779978`.

**9** Vasilis Gkatzelis, Evangelos Markakis, and Tim Roughgarden. Deferred-acceptance auctions for multiple levels of service. In Daskalakis et al. [5], pages 21–38. `doi:10.1145/3033274.3085142`.

**10** R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

**11** Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

**12** Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997. `doi:10.1287/moor.22.3.513`.

**13** Jason D. Hartline and Tim Roughgarden. Simple versus optimal mechanisms. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 225–234, 2009. `doi:10.1145/1566374.1566407`.

**14** Sungjin Im and Janardhan Kulkarni. Fair online scheduling for selfish jobs on heterogeneous machines. In Christian Scheideler and Seth Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 185–194. ACM, 2016. `doi:10.1145/2935764.2935773`.

**15** Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. Minimizing maximum flow time on related machines via dynamic posted pricing. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 51:1–51:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.ESA.2017.51`.

**16** Nicole Immorlica, Li (Erran) Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theor. Comput. Sci.*, 410(17):1589–1598, 2009. `doi:10.1016/j.tcs.2008.12.032`.

**17** Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 233–241, New York, NY, USA, 2000. ACM. `doi:10.1145/352871.352897`.

**18** Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. `doi:10.1007/BF01585745`.

**19** J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977. `doi:10.1016/S0167-5060(08)70743-X`.

**20** Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Oper. Res. Lett.*, 32(5):485–490, 2004. `doi:10.1016/j.orl.2003.11.008`.

21 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

22 Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1):199–223, Jun 1998. doi:10.1007/BF01585872.

23 Linus Schrage. Letter to the editor-a proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

24 Linus E. Schrage and Louis W. Miller. The queue m / g /1 with the shortest remaining processing time discipline. *Operations Research*, 14(4):670–684, 1966.

25 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, 1995. doi:10.1137/S0097539793248317.

26 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi:10.1002/nav.3800030106.