Preemptive scheduling on uniformly related machines: Minimizing the sum of the largest pair of job completion times

Leah Epstein* Ido Yatsiv†

Abstract

We revisit the classic problem of preemptive scheduling on m uniformly related machines. In this problem, jobs can be arbitrarily split into parts, under the constraint that every job is processed completely, and that the parts of a job are not assigned to run in parallel on different machines. We study a new objective which is motivated by fairness, where the goal is to minimize the sum of the two maximal job completion times. We design a polynomial time algorithm for computing an optimal solution. The algorithm can act on any set of machine speeds and any set of input jobs. The algorithm has several cases, many of which are very different from algorithms for makespan minimization (algorithms that minimize the maximum completion time of any job), and from algorithms that minimize the total completion time of all jobs.

1 Introduction

In preemptive scheduling problems on uniformly related machines, we are given a set $J = \{1, \ldots, n\}$ of independent jobs with possibly different sizes, where job j has a size $p_j > 0$ associated with it. Each job must be scheduled to run to completion on a given set $M = \{1, \ldots, m\}$ of related machines, where machine i has speed s_i. The processing time of a part of a job of size x on machine i is $\frac{x}{s_i}$.

We deal with preemptive scheduling; each job may be split arbitrarily into parts, and these parts can be scheduled on one or more machines, under the following constraints: Each machine is assigned to run at most one job at each time, and at any given time, a job is scheduled to be executed on at most one machine (where machines are available starting time zero)[1]. The total size of the parts of job j that are assigned to the machines has to be equal to p_j. Our objective is to minimize the sum of the two largest job completion times. That is, letting $C_j(\sigma)$ denote the completion time of j in schedule σ (where the completion time of job j is the last completion time of any of its parts), and letting $c(\sigma) = (c_1(\sigma), c_2(\sigma), \ldots, c_n(\sigma))$ be a sorted permutation of $C(\sigma) = (C_1(\sigma), C_2(\sigma), \ldots, C_n(\sigma))$ such that $c_1(\sigma) \geq c_2(\sigma) \geq \cdots \geq c_n(\sigma)$, we let $\textsc{cost}(\sigma) = c_1(\sigma) + c_2(\sigma)$.

This objective is motivated by fairness issues. The standard makespan measure, which is equivalent to minimizing $c_1(\sigma)$, deals with minimizing the last completion time, while another standard

*Department of Mathematics, University of Haifa, Haifa, Israel. lea@math.haifa.ac.il.

†Department of Mathematics, University of Haifa, Haifa, Israel. idoyatsiv@gmail.com.

[1]The number of parts for each job must be finite, and the parts must have positive sizes.

measure is the total completion time, where the goal function is $\sum_{j=1}^{n} c_j(\sigma) = \sum_{j=1}^{n} C_j(\sigma)$. This last measure is equivalent to minimizing the average completion time. In this work, we study a measure that combines the two well-studied measures. We call this problem MTLCT (Minimizing the Two Largest Completion Times). Both two classic measures are known to have bias issues; the total completion time measure favors smaller jobs, finishing them as soon as possible, while the makespan has only one goal in mind: all jobs have to work towards finishing the last job as fast as possible, even at the price of completing all jobs simultaneously at the end. The motivation for the problem studied here is to suggest a model that bridges the gap between these two extremes.

We design a polynomial time algorithm that finds an optimal solution for MTLCT. The algorithm has several cases, and while we partially exploit ideas used in previous work, in many of the cases the output is very different from optimal solutions for the two standard goals. Let $\tilde{m} = \min\{m, n\}$. Using sorting methods, median selection methods, and partitioning, our algorithms for MTLCT will assume that $s_1 \geq s_2 \geq \cdots \geq s_{\tilde{m}}$, and that machines $\tilde{m}+1, \ldots, m$ are slower (that is, if $\tilde{m} < m$, then $s_i \leq s_{\tilde{m}}$ for any $i > \tilde{m}$), and additionally, we assume that $p_1 \geq p_2 \geq \cdots \geq p_{\tilde{m}}$, and that jobs $\tilde{m}+1, \ldots, n$ are smaller (that is, if $n > \tilde{m}$, then $p_j \leq p_{\tilde{m}}$ for any $j > \tilde{m}$). This modification of the input can be done in time $O(m+n+\tilde{m}\log\tilde{m})$, as machines $\tilde{m}+1, \ldots, m$ and jobs $\tilde{m}+1, \ldots, n$ are not sorted (at least one of these sets is empty). For $1 \leq i \leq \tilde{m}-1$, let $S_i = \sum_{\ell=1}^{i} s_\ell$, and $P_i = \sum_{\ell=1}^{i} p_\ell$. Let $S = S_{\tilde{m}} = \sum_{i=1}^{\tilde{m}} s_i$ and $P = P_n = \sum_{j=1}^{n} p_j$. Let $L_i(\sigma)$ denote the completion time of machine i in schedule σ, also called its load, which is the last time that the machine completes any part of a job assigned to it (thus, $\max_{1\leq i\leq m} L_i(\sigma) = c_1(\sigma)$). The number of preemptions of a given schedule is defined as the total number of parts that jobs have been split into minus the original number of parts (which is the number of jobs, n). In what follows, we assume $n > 1$ as the case $n = 1$ can be trivially solved by scheduling the (unique) job continuously to run on machine 1 starting time 0 and until time $\frac{p_1}{s_1}$ (if $n = 1$, then the second completion time is defined to be zero).

1.1 Previous work

Recall that in preemptive scheduling problems, such as the one discussed in this paper (unlike non-preemptive scheduling problems) a job does not need to be assigned to run continuously on one machine. The most basic preemptive scheduling problems are minimizing the total completion times of jobs (also known as minimizing the sum of completion times) and minimizing the last completion time of any job (often referred to as makespan minimization).

A polynomial time algorithm that finds an optimal solution with respect to minimizing the total completion time was designed as early as in the 1970's [8, 2]. The principle that leads to minimizing this objective is to complete the execution of the smallest jobs as soon as possible. Thus, such an algorithm gives priority to smaller jobs, both by starting them earlier, and by assigning them to faster machines. More accurately, at each point in time, the algorithm processes the smallest jobs that have not been completed yet, where smaller jobs are allocated to faster machines (so that the fastest machine runs the smallest job, the second fastest machine runs the second smallest job, etc.). This algorithm does not necessarily minimize the makespan, as can be seen for the next

example showing that the two goals are very different. Consider an input where $m = n = 2$, $s_1 = 3$, $s_2 = 1$, $p_1 = 32$ and $p_2 = 24$. An algorithm for makespan minimization can complete both jobs at time 14 (by running job 1 on machine 1 and job 2 on machine 2 till time 9, and then running job 1 on machine 2 and job 2 on machine 1 till time 14), while the algorithm for minimizing the total completion time processes job 2 on machine 1 until time 8, while job 1 runs on machine 2 at that time, and the remaining part of job 1 will be processed to completion (until time 16) on machine 1 afterwards (and the total completion time is 24, which is smaller than 28, the total completion time in the first solution).

The preemptive scheduling problem with the goal of makespan minimization can be solved in polynomial time for identical machines, where all speeds are equal, and for uniformly related machines [11, 7, 5, 12]. The solution for identical machines is fairly simple, using the greedy algorithm of McNaughton [11], while the case of uniformly related machines requires a more complicated treatment, and several approaches are known for this case. Liu and Yang [10] presented m lower bounds on the makespan of a schedule minimizing the makespan. For simplicity, we present the bounds for the case $m = \tilde{m}$ (in the case $\tilde{m} < m$, the slowest $m - \tilde{m}$ machines are neglected). These bounds are as follows. The first bound is the completion time that would be achieved if there is no idle time, where an idle time on a given machine is a time when it does not run any job, while it runs some job later than this time, and the m machines have equal completion times (that is, the ratio of the total size of all jobs and the total speed of all machines: $\frac{P}{S}$). For $1 \leq k \leq m-1$, a similar bound is computed, taking into account a sub-input with the largest k jobs and the fastest k machines (that is, the lower bound is $\frac{P_k}{S_k}$). Obviously, it is not always possible to obtain a flat schedule on all machines or on a subset of the machines, where a flat schedule is one where the machines have equal completion times. Horvath et al. [7] devised an algorithm that uses a large number of preemptions, where the achieved makespan is the maximum of these m bounds. This last result showed, in particular, that the minimum makespan is in fact equal to the maximum of the m bounds stated above for any input. Another approach was employed by Gonzalez and Sahni [5]. They designed a polynomial time algorithm that creates an optimal schedule for which the number of preemptions is at most $2(m-1)$. This is the minimum possible number of preemptions, as there exist inputs for which any optimal solution must use at least that number of preemptions [5]. Shachnai et al. [12] generalized the above algorithm for jobs of limited splitting constraints, that is, a job cannot be split into an arbitrary number of parts, but only to a limited number of parts, depending on the job.

Epstein and Tassa [4] obtained similar results for a wide class of goal functions, including minimization of the ℓ_p norm of the vector of machine completion times. This vector is of length m, with one component for each machine representing its completion time, such that component i represents the completion time of machine i. They showed that for makespan minimization, the best schedule one can get is a schedule where all the machines have exactly the same completion time (though this is not always possible), but this does not necessarily apply to other objective functions such as the ℓ_2 norm of the vector of machine completion times. They proved that a schedule without idle time always exists (for a wide class of objectives), and can be found by their

algorithm. Ebenlendr and Sgall [3] designed an algorithm that we will call InTime. The algorithm InTime receives a parameter $T > 0$ (in addition to a set of machines and a set of jobs), and creates a schedule of the input jobs on the input machines such that all jobs are completed no later than time T, if this is possible, that is, if the optimal makespan for the given input does not exceed T. This algorithm is similar to those of [5, 12, 4], but it can act on an unsorted list of jobs. Its running time (for m machines and n jobs) is $O(n + m \log m)$ if $m \geq n$, and therefore, using our notation, its running time is $O(m + n + \tilde{m} \log \tilde{m})$. Note that the solutions provided by InTime may have idle time.

All the preemptive problems discussed above can be solved in polynomial time for uniformly related machines. However, the non-preemptive makespan minimization problem is strongly NP-hard already for identical machines. In the more general model of unrelated machines (where the processing time of each job can be different on every machine, and the processing time of job j on machine i can take any non-negative value), the preemptive makespan minimization problem is still polynomially solvable. More specifically, Lawler and and Labetoulle [9] showed that this last problem can be solved by using a linear program formulation that converts the problem into another kind of scheduling problem, which is solvable in polynomial time. Interestingly, the preemptive problem of minimizing the total completion time on unrelated machines is NP-hard [13], while the non-preemptive variant is polynomially solvable (for unrelated machines) via the assignment problem [6, 1].

2 Preliminaries

In what follows, when we discuss an optimal schedule, we mean an optimal schedule for MTLCT, i.e., a schedule that minimizes the sum of the largest two job completion times. For two vectors $\bar{a} = (a_1, a_2, \ldots, a_k)$ and $\bar{b} = (b_1, b_2, \ldots, b_k)$, we say that $\bar{a} = \bar{b}$ holds if $a_\ell = b_\ell$ for any $1 \leq \ell \leq k$. We say that $\bar{a}$ is lexicographically larger than $\bar{b}$, if there exists $1 \leq \ell \leq k$ such that $a_d = b_d$ for any $1 \leq d \leq \ell - 1$ and $a_\ell > b_\ell$, and we write $\bar{a} > \bar{b}$. We say that $\bar{a}$ is lexicographically larger than or equal to $\bar{b}$ and write $\bar{a} \geq \bar{b}$ if either $\bar{a} = \bar{b}$ or $\bar{a} > \bar{b}$ holds. For a vector $\bar{a} = (a_1, a_2, \ldots, a_k)$, the sorted vector $\bar{a}' = (a'_1, a'_2, \ldots, a'_k)$ of $\bar{a}$ is a permutation of the components of $\bar{a}$ that satisfies $a'_1 \geq a'_2 \geq \cdots \geq a'_k$. Recall that for a schedule σ for a given input (with n jobs), letting $c(\sigma) = (c_1, c_2, \ldots, c_n)$ be the sorted vector of $C(\sigma) = (C_1(\sigma), C_2(\sigma), \ldots, C_n(\sigma))$, we have $\text{COST}(\sigma) = c_1 + c_2$.

In this section, we will define several types of schedules having specific properties. We will discuss the relation of such schedules to optimal schedules, and show that we can restrict our attention to particular schedule types.

Lemma 1 *Consider two schedules σ and σ' for the job sets J and $I \subseteq J$ (with a common set of machines), where $|I| = n'$. If $C_j(\sigma) \geq C_j(\sigma')$ for any $j \in I$, then $\text{COST}(\sigma') \leq \text{COST}(\sigma)$. If $C_j(\sigma) \geq C_j(\sigma')$ for any $j \in I$, and $\max_{j \in I} C_j(\sigma') < \max_{j \in J} C_j(\sigma)$, then $\text{COST}(\sigma') < \text{COST}(\sigma)$.*

Proof. We start with the first claim. Consider $c(\sigma)$, and let $c(\sigma') = (c'_1, c'_2, \ldots)$ (originally of length n') be augmented with $n - n'$ zeroes (by letting $C_j(\sigma') = 0$ for any $j \in J \setminus I$), every component of

$c(\sigma')$ now corresponds to a job of J, where the last $n-n'$ components correspond to jobs of $J \setminus I$. We have $c'_1 = C_j(\sigma')$ for some j, and $C_j(\sigma') \leq C_j(\sigma) \leq c_1$, as $C_j(\sigma)$ is a component of the (sorted) vector $c(\sigma)$. Thus, $c'_1 \leq c_1$. Assume by contradiction that $c'_2 > c_2$. Let $j' \neq j$ be such that $c'_2 = C_{j'}(\sigma')$. Since $c_2 < c'_2 = C_{j'}(\sigma') \leq C_{j'}(\sigma)$, which holds by our assumptions, as $C_{j'}(\sigma)$ is a component of $c(\sigma)$, it must be the case that $C_{j'}(\sigma) = c_1$ and the component of value $C_j(\sigma)$ in $c(\sigma)$ is no larger than c_2, as it is a different component of $c(\sigma)$. We find $c'_2 \leq c'_1 = C_j(\sigma') \leq C_j(\sigma) \leq c_2$, a contradiction. Thus, $\textsc{cost}(\sigma) = c_1 + c_2 \geq c'_1 + c'_2 = \textsc{cost}(\sigma')$. If additionally $\max_{1 \leq j \leq n} C_j(\sigma') < \max_{1 \leq j \leq n} C_j(\sigma)$ holds, then we have $c_1 > c'_1$, while $c_2 \geq c'_2$ holds as before, and the claim follows. ■

Corollary 2 *Consider an input with jobs $J = \{1, 2, \ldots, n\}$, and another input with the same set of machines and a subset of the jobs $J' \subseteq J$. Let OPT and OPT' be optimal schedules for the input with jobs sets J and J', respectively. Then $\textsc{cost}(OPT') \leq \textsc{cost}(OPT)$.*

Proof. Remove the jobs of $J \setminus J'$ from OPT to obtain a schedule σ' for J'. We find $\textsc{cost}(OPT') \leq \textsc{cost}(\sigma')$ as σ' is an arbitrary solution for the second input. Using $\sigma = OPT$ and Lemma 1, $\textsc{cost}(\sigma') \leq \textsc{cost}(OPT)$. ■

If job j (or a part of it) starts running on machine i at time t and stops running at time t', we say that it runs during the time interval $[t, t')$. As mentioned above, an idle time interval $[t_1, t_2)$ on machine i is such that no job or a part of it runs during this time interval (a job or a part of a job can be completed at time t_1 or can start running at time t_2), but the machine starts at least one job or a part of a job at time t_2 or later. If machine i does not run any part of job during $[t_1, t_2)$, that is, it is idle, or it completes all parts of jobs assigned to it at time t_1 or earlier, then we say that i is inactive (or not active) during this time, and otherwise, if it is running a part of a job, it is called active.

A **fixed time slot** of machine i in a schedule σ is a maximum length time interval $[t, t') \subseteq [0, \infty)$, where i is either inactive or it is assigned to run one job or one part of a job continuously during $[t, t')$ (note that the last fixed time slot of any machine i is the infinite time slot $[L_i(\sigma), \infty)$). That is, a fixed time slot of a machine is a maximum length time interval where the machine is engaged in one activity, on it is not active. For a given machine, the time is split into fixed time slots of this machine in a unique way by the following times: time zero, and times at which a job or a part of a job start or stops running on this machine.

A **fixed time slot of a schedule** is a maximum length time interval $[t, t') \subseteq [0, \max_{1 \leq i \leq m} L_i(\sigma))$ where every machine satisfies the property that it either inactive during $[t, t')$, or it is assigned to run one job or one part of a job continuously. That is, time zero, times that jobs or parts of jobs start running on some machine, and times when jobs or parts of jobs stop running on some machine, split the schedule into fixed time slots of the schedule in a unique way.

A **block** in a schedule σ is a maximum set of machines of consecutive indices with a common completion time.

Definition 1 *A schedule σ is called* **nice**, *if in every fixed time slot of σ the following holds: If machine i is not active and machine i' is active, then $i' < i$. That is, a schedule is nice if at every time, the set of active machines is a prefix of the machine set.*

Lemma 3 *For every input, there exists an optimal schedule that is nice.*

Proof. We define a process that converts an optimal schedule σ into a nice schedule without increasing job completion times. Thus, the resulting schedule is optimal as well, and it is an optimal schedule that is nice. The process is applied separately for every fixed time slot of σ. Note that the completion time of a job must be the ending time of a fixed time slot. Thus, as we will only rearrange the schedules inside fixed time slots, the completion times of jobs cannot increase. It can, however, happen that the fixed time slots of the resulting schedule will be different.

Consider a fixed time slot $[\theta, \theta + \Delta)$. If all machines are active during this last time slot, we are done. Otherwise, create "bins" of sizes $s_k \cdot \Delta$ for $1 \leq k \leq m$, and for each machine i that is active during this fixed time slot, create an "item" of size $s_i \cdot \Delta$. Pack the items into the bins using Fractional Next Fit Decreasing (FNFD). This approach is similar to McNaughton's algorithm [11]. The algorithm FNFD sorts the items by non-increasing size, and packs an item into a bin of minimum index that still has empty space, cutting the item if it cannot be packed completely and moving to the next bin in this case. All items can be packed as their total size does not exceed $\Delta \cdot \sum_{i=1}^{m} s_i$, which is the total size of bins. Moreover, if $1 \leq k' \leq m$ is the minimum index of an inactive machine (during the analyzed time slot), then any machine $1 \leq i < k'$ receives item i completely. Seeing this packing as a schedule (the parts of jobs corresponding to parts of items assigned to a given bin are scheduled on the machine exactly in the order that they have been assigned to the bin), jobs may have been shifted (completely or partially) to machines of smaller indices. Thus, as any given part of a job is processed on machines whose speeds are not smaller than the machine that runs it in σ (during $[\theta, \theta + \Delta)$), no overlaps are created; a part of a job of size x that was previously assigned on machine i' has $x = \Delta \cdot s_{i'}$, and its processing time on machines in $1, \ldots, i'$ (whose speeds are at least $s_{i'}$) is at most Δ. The completion time of a job that is scheduled to run during $[\theta, \theta + \Delta)$ was at least $\theta + \Delta$, and it may decrease as $[\theta, \theta + \Delta)$ is not necessarily a fixed time slot after the modification. If its completion time was strictly larger than $\theta + \Delta$, then it is unchanged, and otherwise is will be no larger than $\theta + \Delta$. In the resulting schedule, a prefix of the machines will be active during $[\theta, \theta + \Delta)$. Out of the remaining machines, the machine of minimum index is either inactive or busy during a time slot $[\theta, \theta + \Delta')$ for some $0 < \Delta' < \Delta$, and the other machines (if at least one such machine exists) are inactive. Even though $[\theta, \theta + \Delta)$ may be split into several fixed time slots, each of them will satisfy the property that at any time, a prefix of the machines is active, and therefore it is not necessary to repeat the process, and it is possible to move to the next fixed time slot. ■

Given the last lemma, we discuss several properties of nice schedules.

Lemma 4 *If σ is a nice schedule, then $L_1(\sigma) \geq L_2(\sigma) \geq \cdots \geq L_m(\sigma)$ holds, that is, the machines are sorted by non-increasing completion times.*

Proof. If $L_i(\sigma) < L_{i+1}(\sigma)$ for some $1 \leq i \leq m-1$, then consider the fixed time slot of the schedule that ends at time $L_{i+1}(\sigma)$. Since time $L_{i+1}(\sigma)$ is the completion time of a job or a part of a job, this time must be the end of a fixed time slot. Machine i cannot be active during this fixed time

slot, as it completes earlier than $L_{i+1}(\sigma)$, and thus the set of active machines during this fixed time slot is not a prefix of the machines, contradicting the definition of a nice schedule. ■

For an input where the number of jobs is smaller than the number of machines, i.e, $n < m$, machines $n+1, n+2, \ldots, m$ are not active in an optimal schedule that is nice. Given the last property, in what follows we will assume without loss of generality that $m = \tilde{m}$, possibly by removing $m - \tilde{m}$ machines of largest indices from the input, and will only consider nice schedules. Thus, for a schedule σ, the first component of $c(\sigma)$ will be equal to $L_1(\sigma)$.

Lemma 5 *For every input and any optimal schedule σ for it that is nice, there is one fixed time slot for machine* 1 *during* $[L_2(\sigma), L_1(\sigma))$.

Proof. Let $L_1 = L_1(\sigma)$ and $L_2 = L_2(\sigma)$. We assume by contradiction that there are at least two fixed time slots during $[L_2, L_1)$. If there is an idle fixed time slot $[t, t') \subset [L_2, L_1)$ (where $t' < L_1$), then all jobs running on machine 1 during $[t', L_1)$ are moved to $[t, L_1 - t' + t)$ to create σ'. As a result, no job has a larger completion time, and the maximum completion time of any job decreases. By Lemma 1, we have $\textsc{cost}(\sigma') < \textsc{cost}(\sigma)$, a contradiction. Otherwise, since there is at least one fixed time slot on machine 1 during $[L_2, L_1)$ except for the last one, there is a fixed time slot $[t, t') \subsetneq [L_2, L_1)$ (where $t' < L_1$), during which the job j that runs on machine 1 is not the same job that runs in the last fixed time slot. We assume that $[t, t')$ is the last such time slot. Denote the last fixed time slot by $[\tau, \tau')$, where $\tau = t'$ and $\tau' = L_1$, and let the job running during this time on machine 1 be j'. We have $L_1 = C_{j'}(\sigma)$, $c_1(\sigma) = L_1$, $c_2(\sigma) = C_j(\sigma) = \tau$, and $\textsc{cost}(\sigma) = L_1 + \tau$. Let $\Delta = \min\{(t' - t)s_2, (\tau' - \tau)s_1\} > 0$. Move a part of j' from $[L_1 - \frac{\Delta}{s_1}, L_1)$ to $[t, t + \frac{\Delta}{s_2})$ on machine 2. The size of the part of j' that was moved is $\Delta \leq (\tau' - \tau)s_1$, i.e., no larger than the part that was previously assigned on machine 1 during $[\tau, \tau')$. No machine was previously running a job during the time slot $[t, t')$ except for machine 1, and $t + \frac{\Delta}{s_2} \leq t'$, so machine 1 runs j at the time that j' is now scheduled on machine 2. The completion times of all jobs except for j' are unchanged, and the completion time of j' strictly decreased since the completion time of the schedule becomes $L_1 - \frac{\Delta}{s_1} < L_1$. By Lemma 1, the resulting schedule has a smaller cost. This contradicts the property that σ is optimal, a contradiction. ■

Definition 2 *A schedule that is nice and has no idle time is called* **good** *(note that a nice schedule may have idle time).*

Definition 3 *A schedule is called* **structured** *if it satisfies the following three conditions.*

- *There is no idle time in the schedule.*
- *The loads satisfy* $L_1(\sigma) \geq L_2(\sigma) \geq \cdots \geq L_m(\sigma)$.
- *If* $L_1(\sigma) > L_2(\sigma)$, *then machine* 1 *runs a single job during the time* $[L_2(\sigma), L_1(\sigma))$.

Note that a good schedule is not necessarily structured. For example, a schedule where all (at least two) jobs are assigned to machine 1 (without any idle time) is good, but it is not structured, as it does not satisfy the third condition.

Proposition 6 *Let π be a schedule that is either structured, or it is an optimal schedule that is nice. The cost of π satisfies* $\text{COST}(\pi) = L_1(\pi) + L_2(\pi)$.

Proof. For any schedule σ for which $L_1(\sigma) \geq L_2(\sigma) \geq \cdots \geq L_m(\sigma)$, we have $c_1(\sigma) = L_1(\sigma)$, as there is a job that is completed at time $L_1(\sigma)$ and no job is completed later. The schedule π satisfies the property of sorted machine completion times; if it is structured this holds by definition, and otherwise by Lemma 4.

If $L_1(\pi) = L_2(\pi)$, then there are at least two jobs that are completed at this time, and no job is completed later, so $c_2(\pi) = L_2(\pi)$. Otherwise, since machine 1 runs a single job j during $[L_2(\pi), L_1(\pi))$ (this holds for structured schedules by definition, and for optimal and nice schedules by Lemma 5), no job except for j is completed strictly after $L_2(\pi)$. Since just before time $L_2(\pi)$, both machines are active, and starting time $L_2(\pi)$ only one machine is active, there is a job that completes at time $L_2(\pi)$. Thus, $c_2(\pi) = L_2(\pi)$ holds in this case as well. ■

Proposition 7 *Any structured schedule is good, and an optimal schedule σ that is good is structured.*

Proof. First, we show that any optimal schedule that is good satisfies the three conditions of a structured schedule. The first condition holds by definition. The second condition holds since any good schedule is nice, and by Lemma 4. The third condition holds since any good schedule is nice, and by Lemma 5, as the schedule is optimal and nice.

Next, consider an arbitrary schedule satisfying the three conditions. We will show that the schedule is nice, and by the first condition we will conclude that it is good. To show that a schedule is nice, it is sufficient that the set of machines that are active at each time is a prefix of the machines list, which holds due to the first and second conditions. ■

Lemma 8 *For every input, there exists an optimal schedule that is good.*

Proof. Recall that for every input there exists an optimal schedule that is nice, by Lemma 3. Consider such a schedule σ. We will create a nice schedule of the same cost that has no idle time, which will show that it is optimal and good. The cost of a nice schedule π that is optimal is equal to twice the makespan if $L_1(\pi) = L_2(\pi)$ (that is, if in the last fixed time slot at least two machines are active), or it is equal to twice the makespan minus the length of the last fixed time slot otherwise. This holds by Proposition 6 for both cases (see also Lemma 5, which discusses optimal schedules that are nice, for the case $L_1(\pi) > L_2(\pi)$).

Note that since $n \geq 2$, if there is a single fixed time slot in the schedule, then at least two machines are active in it, and moreover, there cannot be any idle time. Thus, we assume that the schedule has at least two fixed time slots.

The schedule σ' is created from σ by reordering the fixed time slots such that they are sorted by a non-increasing number of active machines. The fixed time slots are assigned consecutively (with respect to time) without any gaps (thus the makespan is unchanged). The fixed time slots with equal numbers of machines are ordered according to their order in σ. Thus, if in the last fixed

time slot there is one active machine (this must be machine 1 as σ is nice), it will appear last in σ' as well. Since in each time slot, a prefix of the machines is active, as a result of the reordering, the schedule will not have any idle time. The property of a nice schedule is kept since the schedule of every fixed time slot is not altered. Thus, it remains to show that σ' is optimal.

If there are no fixed time slots with a single active machine in σ, then there are no such fixed time slots in σ', and for both schedules the cost is twice the makespan (which is equal for both schedules). Otherwise, consider the last fixed time slot of σ, $[t, L_1(\sigma))$. Letting $\Delta = L_1(\sigma) - t$, we find $\textsc{cost}(\sigma) = 2L_1(\sigma) - \Delta$. By the construction of σ', its last fixed time slot is also a fixed time slot with a single active machine, and its length is at least Δ (as by the construction of σ', during $[t, L_1(\sigma))$, both schedules have one active machine, and they process the same job). As the two schedules have the same makespan, we get $\textsc{cost}(\sigma') \leq 2L_1(\sigma) - \Delta$, showing that σ' is an optimal schedule. ■

We summarize this section with the next corollary, which follows from Proposition 7 and Lemma 8, and will be used in the next section.

Corollary 9 *There exists an optimal schedule that is good and structured.*

3 The algorithm and its optimality

In this section, we design a polynomial time algorithm that creates an optimal schedule that is structured. While the set of good schedules and the set of structured schedules are not the same set, as we saw, the two sets of optimal schedules that are good and optimal schedules that are structured are the same set. Thus, we use the terms good and structured interchangeably. Since we construct this kind of schedule, it will not contain idle time, and our objective is to minimize the sum of loads of the first two machines.

We start with discussing algorithms for makespan minimization, one of which will acts as a procedure in our algorithm. Afterwards, we define another auxiliary procedure, and the algorithm for MTLCT, which is called SMTLCT (schedule MTLCT). Afterwards, we prove the optimality of SMTLCT.

The algorithm ComputeMakespanLoads is based on algorithms for computing schedules that are optimal with respect to makespan minimization. It receives a set of $\hat{m} > 0$ machines (which can be the entire set of machines or a subset of it), and a set of $\hat{n} > 0$ jobs. Its output is a list of loads for the machines, where the load of machine i is L_i, and if $i_1 < i_2$, then $L_{i_1} \geq L_{i_2}$. The set of loads is such that any corresponding schedule is strongly optimal with respect to makespan, and does not contain any idle time. Strongly optimal means the following: for each schedule, define a load vector of length $\hat{m}$ associated with it, where the vector consists of the list of machine loads sorted by non-increasing order. A schedule is strongly optimal if the sorted vector of machine loads is minimal lexicographically, and not only the maximum load (which is the value of the first component) is minimized. The algorithm of [7] creates such a schedule, but its running time is large and we use a different algorithm here. By the properties of outputs of the algorithm of [7], if

$\hat{m} > \hat{n}$, the $\hat{m} - \hat{n}$ machines of largest indices do not receive any jobs. Thus, if $\hat{m} > \hat{n}$, the $\hat{m} - \hat{n}$ machines of largest indices are removed.

Assuming $\hat{m} \leq \hat{n}$, the action of the algorithm that we use (which is not the algorithm of [7]) is as follows. The first step is to compute the $\hat{m}$ bounds on the makespan. For $1 \leq k \leq \hat{m}$, the kth bound is computed using the ratio between the total size of largest k jobs and the sum of speeds of the k machines of smallest indices, while the $\hat{m}$th bound is the ratio between the total size of all jobs to be assigned and the sum of speeds of all machines. The second step is to find the maximum bound out of the $\hat{m}$ bounds, denoted by T_1. The third step is to find the maximum index k, for which the kth bound is equal to T_1. The fourth step is as follows. If $k = \hat{m}$, define loads of T_1 for all machines and halt. Otherwise, define loads of T_1 for the k machines of smallest indices, remove these machines and the largest k jobs from the input, and apply the algorithm recursively on the remaining jobs and machines, starting from the first step.

Consider the case where the algorithm is applied recursively. Since in this case $k < \hat{m} \leq \hat{n}$, the remaining input consists of at least one machine and at least one job. For an input of n jobs and $m \leq n$ machines, the running time is $O(n + m^2)$; this includes sorting of the m largest jobs and the machines, and computing at most m bounds at most m times (where the set of bounds of one iteration can be computed in time $O(m)$ using prefix sums). Note that the function is linear in n, as we do not need to sort all the jobs except for the largest n jobs (which can be found by methods of median selection and partitioning), and for the $n - m + 1$ smallest jobs, only their total size is required, and it is computed only once for all iterations together. The sorting of m machines and $O(m)$ jobs is done in time $O(m \log m)$.

The algorithm FindMakespan acts similarly to ComputeMakespanLoads, but it applies just one iteration, that is, it only finds the maximum load. The running time for an input of n jobs and $m \leq n$ machines is $O(n + m \log m)$.

The algorithm InTime which we use applies the algorithm of [3] to a subset of jobs and machines. It receives a threshold value T, a set of machines, and a set of jobs, and it creates a preemptive schedule such that the completion time of each machine does not exceed T, if this is possible. We use the algorithm as a black box[2]. Even though the algorithm InTime could introduce idle time, we will only use it in a way that it does not do this. More specifically, we only use it for cases where the jobs require all the processing time during $(0, T]$ on all machines, and thus, in order not to exceed the completion time T, no idle time will be created. Given a list of loads calculated by ComputeMakespanLoads, a schedule can be created by applying InTime to each block separately. That is, it is applied to a set of machines of consecutive indices and the set of jobs that are removed from the input together with these machines before the recursive call or before termination. This gives us an algorithm for creating a strongly optimal schedule with respect to makespan, and we call

[2] A short summary of the action of the algorithm of [3] is as follows. It keeps an infinite number of machines that can be used during the time $[0, T)$. The machines have time dependent speeds. The actual machines are initialized with their actual speeds, while dummy machines are initialized with zero speeds. Each job is assigned to run during $[0, T)$, possibly on several machines, with the convention that running on a machine of speed zero during some time means not running at all at that time. By assigning a job, two machines are merged into what will be seen as one machine (this merged machine usually will not have a constant speed).

this algorithm ScheduleMakespan. The running time of InTime for an input of n jobs and $m \leq n$ machines is $O(n + m \log m)$. Even if machines are split into blocks and InTime is applied on each block separately, the total running time of all applications of InTime remains $O(n + m \log m)$ as it is the sum of $O(n_i + m_i \log m_i)$ over the iterations, where m_i and n_i are the numbers of machines and jobs, respectively, in the ith iteration, and the sums over all iterations of m_i and n_i are m and n, respectively.

Proposition 10 *Any schedule π that is created by ScheduleMakespan is structured.*

Proof. The first two properties hold by the action of the algorithm, since it produces a schedule of minimum makespan. If $L_1(\pi) = L_2(\pi)$, then we are done. Otherwise, machine 1 is the only machine of its block, and it receives one job that is scheduled non-preemptively (the largest job), thus the third condition holds as well. ■

In what follows, we will define our algorithm for all cases. We now define a function FindMaxPart, that is used by our algorithm[3]. The function FindMaxPart receives an index μ, where $2 \leq \mu \leq m$ and an index ν, where $\mu \leq \nu \leq n$. It also gets a threshold value T such that an optimal solution that minimizes the makespan for the jobs of indices $2, 3, \ldots, \nu$ and the machines of indices $1, 2, \ldots, \mu$ is T. If T is unknown, it can be computed using FindMakespan. The function finds a maximum value $0 \leq x \leq p_1$, such that it is possible to create a schedule of the jobs $2, \ldots, \nu$ ($\nu - 1$ jobs) and a part of the first job of size x, on the same set of machines, during $[0, T)$. For $j = 1, \ldots, \nu$, let $P'_j = P_j - p_1$.

The action of FindMaxPart is as follows.
An array X of length μ is defined; for $1 \leq a \leq \mu - 1$, let $X[a] = T \cdot S_a - P'_a$, and $X[\mu] = T \cdot S_\mu - P'_\nu$. The value x is defined as $\min_{1 \leq a \leq \mu} X[a]$, and the index b (where $1 \leq b \leq \mu$) is selected as the maximum index for which $X[b] = x$. The output consists of x and b.

The value $X[\mu]$ is the maximum part of the first job that can be added to jobs $2, \ldots, \nu$ such that the total size of jobs does not exceed the total size that can be completed by time T. For $1 \leq a \leq \mu - 1$, $X[a]$ is the maximum part x of the first job that can be added if we only consider the sub-input of the first a jobs and the first a machines, such that all jobs except for the first one are assigned completely, and a part of size x of the first job is scheduled. Since the array X is computed in time $O(\mu + \nu)$, the running time of FindMaxPart is $O(\mu + \nu)$.

Lemma 11 *The function FindMaxPart with the parameters T, μ, ν finds a maximum value x such that $0 \leq x \leq p_1$, and it is possible to schedule jobs $2, \ldots, \nu$ and a part of the first job of size x to machines $1, \ldots, \mu$ during the time $[0, T)$. If $\mu = \nu$, then $x > 0$. The value b given as an output satisfies $b \geq 2$.*

Proof. We start with showing that $0 \leq x \leq p_1$. Recall that T is the makespan for jobs $2, \ldots, \nu$ and machines $1, \ldots, \mu$. Thus, $T \geq \frac{P'_\nu}{S_\mu}$, and $T \geq \frac{P'_{a+1}}{S_a}$ for $1 \leq a \leq \mu - 1$. We get $X[\mu] \geq 0$, and for

[3]The action of the function can be seen as solving a linear program, but since the program is simple, we solve it directly. We do not use this property (that a linear program is being solved) in the proof, and thus, we do not describe the function as such.

$1 \leq a \leq \mu-1$, $X[a] = T \cdot S_a - P'_a \geq P'_{a+1} - P'_a = p_{a+1} > 0$. On the other hand, T is equal to one of the bounds $\frac{P'_\nu}{S_\mu}$, $\frac{P'_{a+1}}{S_a}$ for $1 \leq a \leq \mu-1$, since it is the maximum of these bounds. If $T = \frac{P'_\nu}{S_\mu}$, then $X[\mu] = 0$, and $x = 0$. Otherwise, if $T = \frac{P'_{a+1}}{S_a}$ for some $1 \leq a \leq \mu-1$, then $X[a] = P'_{a+1} - P'_a = p_{a+1}$, so $0 < x = X[a] \leq p_1$, as p_1 is the maximum job size.

Assume $\nu = \mu$. To show that $x > 0$ holds in this case, we note that $x = 0$ would imply $X[\mu] = 0$, as we showed that $X[a] > 0$ holds for any a such that $1 \leq a \leq \mu-1$. Since T is the makespan of jobs $2, \ldots, \mu$, using the bound for jobs $2, \ldots, \mu$ and machines $1, \ldots, \mu-1$, $T \geq \frac{P'_\mu}{S_{\mu-1}}$. Thus, $X[\mu] = T \cdot S_\mu - P'_\mu \geq T \cdot S_\mu - T \cdot S_{\mu-1} = T \cdot s_\mu > 0$, and therefore $x > 0$.

Next, we show that for any $x' > x$, assigning a part of size x' instead of a part of size x results in a higher makespan. Let a be such that $x = X[a]$. If $a = \mu$, then $x = 0$, and for $x' > 0$, $\frac{P'_\nu + x'}{S_\mu} = T + \frac{x'}{S_\mu} > T$. Otherwise, consider the a largest jobs among the following $a+1$ jobs: jobs $2, \ldots, a+1$ and the part of job of size x'. We have $x' > x = X[a] \geq p_{a+1}$. Thus, the a largest jobs are jobs $2, \ldots, a$ and the job of size x'. We have $\frac{P'_a + x'}{S_a} = \frac{T \cdot S_a - x + x'}{S_a} > T$.

Now, we show that the makespan for the jobs $2, \ldots, \nu$ and the job of size x (on machines $1, \ldots, \mu$) is at most T. We consider the μ bounds on the makespan (that determine the makespan). The last bound is $\frac{P'_\nu + x}{S_\mu} \leq \frac{P'_\nu + X[\mu]}{S_\mu} = T$. For $1 \leq a \leq \mu-1$, if $p_{a+1} \geq x$, then the ath bound is the same as the bound that was computed when dealing with jobs $2, \ldots, \nu-1$ (which is based on jobs $2, \ldots, a+1$). Otherwise, we have $\frac{P'_a + x}{S_a} \leq \frac{P'_a + X[a]}{S_a} = T$.

Finally, assume by contradiction $b = 1$. Thus, we have $X[1] = x$ and $X[i] > x$ for $2 \leq i \leq \mu$, and we have $x = T \cdot s_1 > 0$ (as the input is non-empty), $T \cdot S_i - P'_i > x$ for $2 \leq i \leq \mu-1$ and $T \cdot S_\mu - P'_\nu > x$. We will show that the bounds on the makespan for machines $1, \ldots, \mu$ and jobs $2, \ldots, \nu$ are all strictly smaller than T, which will contradict the choice of T as the minimum makespan for jobs $2, \ldots, \nu$ on machines $1, \ldots, \mu$. These bounds are $\frac{P'_{i+1}}{S_i}$ for $1 \leq i \leq \mu-1$ and $\frac{P'_\nu}{S_\mu}$. The last bound satisfies $\frac{P'_\nu}{S_\mu} < \frac{T \cdot S_\mu - x}{S_\mu} < T$. Using $x = s_1 \cdot T$, we get $T \cdot (S_i - s_1) - P'_i > 0$ for $2 \leq i \leq \mu-1$, and $T \cdot (S_\mu - s_1) - P'_\nu > 0$. Since $s_1 \geq s_i$ for $2 \leq i \leq \mu$, $S_i - s_1 \leq S_{i-1}$ for $2 \leq i \leq \mu$, and we find $T \cdot S_{i-1} - P'_i > 0$ for $2 \leq i \leq \mu-1$ and $T \cdot S_{\mu-1} - P'_\nu > 0$. Using $P'_\nu \geq P'_\mu$ and $T \cdot S_{\mu-1} - P'_\nu > 0$, we get $T \cdot S_{\mu-1} - P'_\mu > 0$. Thus, $T \cdot S_{i-1} - P'_i > 0$ holds for $2 \leq i \leq \mu$, that is, $T \cdot S_i - P'_{i+1} > 0$ holds for $1 \leq i \leq \mu-1$, proving $\frac{P'_{i+1}}{S_i} < T$ for $1 \leq i \leq \mu-1$, as required. ■

Next, we define algorithm SMTLCT, which is our main contribution. Without loss of generality it is assumed that $n \geq m$. Otherwise, the input is modified first by finding the n fastest machines and removing the remaining machines from the input. The algorithm has running time $O(n+m+\tilde{m}^2)$. In the case $n \geq m$, for the smallest $n-m+1$ jobs, only their total size is required for the calculations of ComputeMakespanLoads, and it is computed before executing the algorithm, thus the resulting running time is linear in n. For the case $n \geq m$, the algorithm applies algorithms of running time $O(n+m^2)$ a constant number of times, and one algorithm whose running time is $O(n)$ (FindMaxPart).

Algorithm SMTLCT

1. Run ComputeMakespanLoads with J and M. Let L_i be the resulting completion time of machine i (these are not necessarily the loads of the schedule that will be given as the output).

Let the blocks (sequences of consecutive indices of machines with equal completion times) be denoted by $B_1, B_2, \ldots, B_q$. For convenience, define an additional (empty) block B_{q+1}, and let $L_{m+1} = 0$ (that is, the load associated with block B_{q+1} is zero).

2. Let m' be a maximum index such that $L_{m'} = L_2$, that is, the last machine of the block of machine 2.

3. If $s_1 \leq s_2 + s_3 + \cdots + s_{m'}$, then run ScheduleMakespan with J and M. Give its output as an output and halt.

4. Define an integer ℓ, such that $m' + 1 \leq \ell \leq m + 1$, as follows.

   - If $s_1 > s_2 + s_3 + \cdots + s_m$, then let $\ell = m + 1$.
   - Otherwise, let $\ell \leq m$ be such that $s_1 > s_2 + s_3 + \cdots + s_{\ell-1}$ and $s_1 \leq s_2 + s_3 + \cdots + s_\ell$.

5. Let B_r, where $2 \leq r \leq q$, be the block that contains machine ℓ (if $\ell = m+1$, then $r = q+1$).

6. Let k, $2 \leq k \leq m$, be the maximum index machine of block B_{r-1} (so $\ell \geq k + 1$).

7. Let $y' = n$ if $k = m$, and otherwise let $y' = k$. Run ComputeMakespanLoads with jobs $2, 3, \ldots, y'$ and machines $1, 2, \ldots, k$. Let $z > 0$ be resulting makespan, and let $L' = L_\ell$ (if $\ell = m + 1$, let $L' = 0$).

8. If $z \leq L'$, $x = L' \cdot (s_1 + s_2 + \cdots + s_k) - (p_2 + p_3 + \cdots + p_{y'})$. Let $b = k$.

9. If $z > L'$, then run FindMaxPart with jobs $2, 3, \ldots, y'$ and machines $1, 2, \ldots, k$ and the threshold value z. Let x and b (where $2 \leq b \leq k$) be the results of FindMaxPart.

10. If $b = m$, then let $y = n$, and otherwise $y = b$. Define a modified job $\tilde{\jmath}$ of size x. Run ScheduleMakespan with jobs $\tilde{\jmath}, 2, 3, \ldots, y$ on machines $1, 2, \ldots, b$. Let L be the load of machine 1.

11. If $y < n$, then run ScheduleMakespan with jobs $b + 1, \ldots, n$ and machines $b + 1, \ldots, m$. Let this schedule, obtained in the previous step and the current step be denoted by σ.

12. Replace σ with a schedule σ' as follows. Jobs $2, \ldots, n$ are scheduled as in σ. A part of size x of job 1 is assigned instead of $\tilde{\jmath}$, and the remaining part of size $p_1 - x$ of job 1 is assigned during $[L, L + \frac{p_1 - x}{s_1})$. Give σ' as an output.

The general action of SMTLCT is as follows. The algorithm receives the input, consisting of J and M (n jobs and m machines). First, it applies ComputeMakespanLoads to obtain the loads of a strongly optimal schedule with respect to makespan, which is a structured schedule, by Proposition 10. Then, there are three cases, based on properties of the block of the second machine in the corresponding schedule. In the first case, the corresponding schedule is computed and given as output. In the other two cases, the algorithm tries to increase the completion time of the first

machine, still keeping the property that the schedule is structured. It analyzes a potential schedule for the block of machine 2 together with machine 1 (i.e., the schedule of one or two blocks) excluding job 1 (an optimal schedule with respect to makespan for these machines and jobs), keeping the schedule of slower machines as computed before. If those machines (of later blocks) have sufficiently small loads, then it combines a part of job 1 into the schedule, such that the maximum completion time does not increase, and the remainder of this job is assigned to machine 1. Otherwise, the loads of machine 1 and the machines of the block of machine 2 are defined such that they are equal to the loads for the next block (as it was computed by ComputeMakespanLoads).

Figure 1 illustrates the choice of m', ℓ, and k.

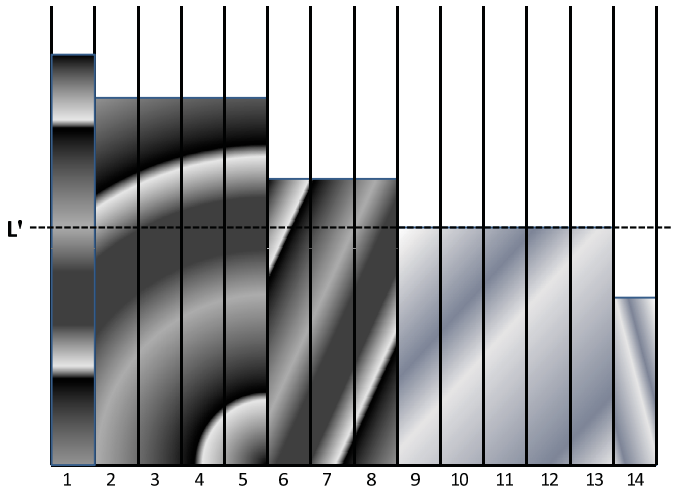

Figure 1: An example for the values m', ℓ, and k. The figure illustrates an optimal schedule with respect to makespan computed in step 1. The speeds of machine 1 is 25, and speeds of other machines are equal to 3. In this example $m' = 5$, $\ell = 10$, and $k = 8$.

We briefly present two examples for the action of algorithm SMTLCT.

For the first example, let $m = 3$, $n = 4$, $s_1 = 3$, $s_2 = s_3 = 2$, $p_1 = 9$, $p_2 = 7$, $p_3 = 3$, and $p_4 = 2$. The three bounds for the makespan are 3, 16/5, and 3. Thus, ComputeMakespanLoads assigns loads of 16/5 to machines 1 and 2, and calls the same function recursively with one machine of speed 2 and two jobs of sizes 3 and 2, resulting in a load of 5/2 for machine 3. There are two blocks, the first one consists of machines 1 and 2, and the second block consists of machine 3. We have $m' = 2$, and the condition of step 3 is not fulfilled. Machine ℓ is machine 3, $r = 2$, and $k = 2$. In step 7, $y' = 2$, $L' = 5/2$, and z is the makespan of running job 2 on machines 1 and 2, thus $z = 7/3$. Since $z < L'$, FindMaxPart is run with the threshold 5/2, and its value is 11/2, and $b = 2$. The job $\tilde{j}$ has size 11/2, and the jobs $\tilde{j}, 2, 3$ are scheduled by ScheduleMakespan. Then, the remaining part of job 1 is added to run on machine 1 during $[5/2, 11/3)$. See the left hand size of Figure 2 for the resulting schedule. The cost of the schedule is $11/3 + 5/2 = 37/6$.

For the second example, let $m = 6$, $n = 8$, $s_1 = 6$, $s_2 = 5$, $s_3 = s_4 = 4$, $s_5 = s_6 = 3$, $p_1 = p_2 = 15$, $p_3 = p_4 = 3$, $p_5 = p_6 = 2$, and $p_7 = p_8 = 1$. The result of ComputeMakespanLoads is

that machines 1 and 2 have loads of 30/11, and the other machines have loads of 6/7. The values of m', ℓ, r, k, and y' are as in the previous example. Here, $L' = 6/7$, $z = 5/2$. In this case $z > L'$, and z is the threshold for FindMaxPArt. The values of x is 25/2, and $b = 2$ again. The resulting schedule (without the details of the assignment of jobs $3, \ldots, 6$) is illustrated in the right hand side of Figure 2.

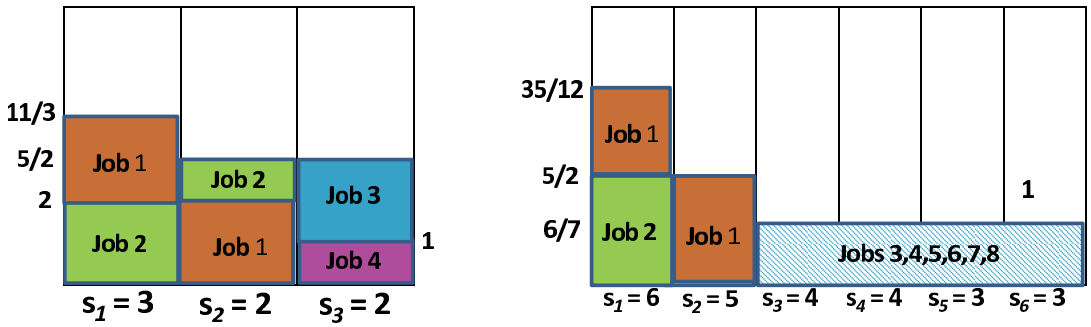

Figure 2: The outputs of the algorithm for the two examples given in the text (the first example is on the left hand side and the second example is on the right hand side).

We start with proving two simple properties.

Lemma 12 *The indices ℓ and r are well-defined. If $\ell = m + 1$, then $z < L'$.*

Proof. According to the condition of step 3, step 4 is reached in the case $s_1 > s_2 + s_3 + \ldots s_{m'}$. Thus, for any machine $2 \leq i \leq m'$, we have $s_1 > s_2 + s_3 + \ldots s_i$, and for any ℓ' such that $s_1 \leq s_2 + s_3 + \ldots s_{\ell'}$, $\ell' > m'$. If at least one such value ℓ' exists, ℓ is defined to be a minimum such value. Otherwise, $\ell = m + 1$. Since $\ell > m'$, and m' is the maximum index of a machine in a block, the block of ℓ is not B_1, and therefore $r \geq 2$.

In the case $\ell = m + 1$, it holds that $L' = 0$, while $z > 0$. ■

In what follows, we prove the optimality of the output solution for all cases.

Lemma 13 *If the algorithm halts in step 3, then the output is an optimal schedule.*

Proof. Let σ be the created schedule, and let σ' be the schedule of jobs that are assigned to machines $1, \ldots, m' \geq 2$. Let J' be the set of these jobs (scheduled in σ'), and let Q be the total size of jobs of J'. Both σ and σ' are schedules that can be created by ScheduleMakespan, and thus they are structured schedules by Proposition 10. We will show that σ' is optimal, and since the costs of both schedules are equal (the costs are $L_1(\sigma) + L_2(\sigma) = L_1(\sigma') + L_2(\sigma')$), this will imply that σ is optimal (by Corollary 2). Since machines $2, \ldots, m'$ belong to one block of ScheduleMakespan (and m' is the last machine of the block), we have

$$L_1(\sigma') + L_2(\sigma') = L_1(\sigma') + \frac{Q - s_1 L_1(\sigma')}{s_2 + \cdots + s_{m'}} = \frac{Q + ((s_2 + \cdots + s_{m'}) - s_1) L_1(\sigma')}{s_2 + \cdots + s_{m'}} .$$

By Corollary 9, for any input there exists an optimal schedule that is structured, and let π be such a schedule for J' and M. Its cost is $L_1(\pi) + L_2(\pi)$ by Proposition 6. As σ' is optimal with respect to makespan, $L_1(\pi) \geq L_1(\sigma')$. If $m' < m$, then J' consists of jobs $1, \ldots, m'$, and otherwise $J' = J$. Thus, if $m' < m$, a good schedule for J' does not use machines $m'+1, \ldots, m$, since $|J'| = m'$ in this case. In both cases ($m' < m$ and $m' = m$), π uses machines $1, \ldots, m'$. By averaging and since the loads are non-increasing in structures schedules, $L_2(\pi) \geq \frac{Q - s_1 \cdot L_1(\pi)}{s_2 + \cdots + s_{m'}}$. We find

$$L_1(\pi) + L_2(\pi) \geq L_1(\pi) + \frac{Q - s_1 \cdot L_1(\pi)}{s_2 + \cdots + s_{m'}} = \frac{Q + ((s_2 + \cdots + s_{m'}) - s_1) \cdot L_1(\pi)}{s_2 + \cdots + s_{m'}} \ .$$

As in this case, $s_1 \leq s_2 + s_3 + \cdots + s_{m'}$, and since $L_1(\pi) \geq L_1(\sigma')$, we find that the cost of π is at least the cost of σ', proving the optimality of σ' for J' and M, and thus the optimality of σ for J and M. ■

We are left with the cases that lead to step 12. The optimality is proved in a sequence of lemmas. The next lemma deals with the situation $z > L'$, which is the only case where $b < k$ is possible. We compare the machine loads of machines $1, 2, \ldots, b$ and machines $b+1, \ldots, m$.

Lemma 14 *If $z > L'$, then in the schedule found in step 11 all loads (of machines $b+1, \ldots, m$) are strictly below z. If $b < k$, the minimum makespan for jobs $2, \ldots, b$ and machines $1, \ldots, b$ is z.*

Proof. If $b = m$, then the claims hold trivially, as in this case the set $b+1, \ldots, m$ is empty and since $k \leq m$, $k = b$. Thus, we assume $b < m$. We define a possible valid schedule for jobs $b+1, \ldots, n$ on machines $b+1, \ldots, m$, and show that its makespan is no larger than z. The claim will follow, since an optimal schedule with respect to makespan for these sets (of jobs and machines) is found in step 11.

Consider the following schedule. If $k < m$, then machines $k+1, \ldots, m$ receive jobs $k+1, \ldots, n$, where the schedule is created by ScheduleMakespan. Since machine $k+1$ is the first machine of a block in the schedule corresponding to the loads found by ComputeMakespanLoads in step 1, ScheduleMakespan will create exactly the loads $L_{k+1}, \ldots, L_m$ that were found by ComputeMakespanLoads in step 1. These loads do not exceed $L' = L_\ell$, which is the load of the machines of the block of machine $k+1$ in the list of step 1 (and the loads of blocks of higher indices, if they exist, are smaller). If $b = k$, then we are done, since $L' < z$. Otherwise, we consider the schedule of ScheduleMakespan of the unscheduled jobs out of jobs $b+1, \ldots, y'$ on machines $b+1, \ldots, k$ (these jobs $b+1, \ldots, n$ if $k = m$, and otherwise these are jobs $b+1, \ldots, k$). We have $b < k \leq m$, and recall that b was selected as the maximum index i' for which $X[i'] = x$ (and x is the minimum of the array X). Therefore, for any $b+1 \leq i \leq k$, $X[i] > x$. We have for $b+1 \leq i \leq k-1$, $z(s_1 + \cdots + s_i) - (p_2 + \cdots + p_i) > x = z(s_1 + \cdots + s_b) - (p_2 + \cdots + p_b)$, i.e., $z(s_{b+1} + \cdots + s_i) > p_{b+1} + \cdots + p_i$ holds, and similarly, $z(s_1 + \cdots + s_k) - (p_2 + \cdots + p_{y'}) > x = z(s_1 + \cdots + s_b) - (p_2 + \cdots + p_b)$, where $y' = k$ if $k < m$ and $y' = n$ otherwise, and we get, $z(s_{b+1} + \cdots + s_k) > p_{b+1} + \cdots + p_{y'}$. Thus, all $k - b$ bounds on the makespan for machines $b+1, \ldots, k$ and jobs $b+1, \ldots, y'$ are below z, and it is possible to schedule jobs $b+1, \ldots, y'$ on machines $b+1, \ldots, k$ with makespan strictly below z.

To prove the second claim, assume by contradiction that $b < k$, and the minimum makespan for jobs $2, \ldots, b$ and machines $1, \ldots, b$ is not z. The $b-1$ bounds on the minimum makespan are

exactly the first $b-1$ bounds for the minimum makespan of jobs $2,\ldots,y'$ and machines $1,\ldots,k$, thus the makespan must be strictly below z (since z is the maximum of the bounds in the latter case). We find that it is possible to schedule jobs $2,\ldots,b$ on machines $1,\ldots,b$ and jobs $b+1,\ldots,y'$ on machines $b+1,\ldots,k$ with makespan strictly below z (the property for jobs $2,\ldots,b$ follows from the discussion here regarding the $b-1$ bounds on the makespan, and for jobs $b+1,\ldots,k$, from the previous paragraph), contradicting the definition of z. ■

We have proved that if $b < k$, then the makespan for jobs $2,\ldots,b$ and machines $1,\ldots,b$ is exactly z, which is the makespan calculated for jobs $2,\ldots,y'$ and machines $1,\ldots,k$. We do not claim, however, that all machines $1,\ldots,b-1$ have completion times z in an optimal schedule of jobs $2,\ldots,b$ on machines $1,\ldots,b$ with respect to makespan, and this is not necessarily true. If $b = k$, then obviously the makespan for jobs $2,\ldots,y'$ and machines $1,\ldots,b$ is z by definition. In the next lemma we will show that after adding x, all the loads of machines $1,\ldots,b$ will be equal to z in a schedule that minimizes makespan.

Lemma 15 *If $z > L'$, the loads of machines $1,\ldots,b$ (where $b \geq 2$) are equal to z in the schedule found in step 10.*

Proof. The value x is selected such that the makespan of an optimal schedule with respect to makespan of jobs $\tilde{j}, 2, 3, \ldots, y'$ on machines $1, 2 \ldots, k$ does not exceed z, by Lemma 11. Since $x = X[b]$, the total size of jobs $\tilde{j}, 2, 3, \ldots, y$ is exactly $(s_1 + \cdots + s_b)z$. None of these machines can have load strictly below z as by averaging this would result in another machine having a load strictly above z, contradicting the choice of x (see Lemma 11). ■

Lemma 16 *If $z > L'$, then the output is a valid schedule, this schedule is structured, and it is optimal.*

Proof. Since $b \geq 2$ (by Lemma 11), machines $1, 2$ have loads of z before step 12 (by Lemma 15), and the additional part of job 1 that is scheduled in step 12 is not scheduled to run in parallel to any part of job. Thus, the schedule is valid. The schedule created in steps 10 and 11 does not contain idle time, as the schedule of each set of machines is created using ScheduleMakespan. The part of job 1 scheduled in step 12 is added without introducing any idle time.

By the previous two lemmas, we find that before applying step 12, machines $1,\ldots,b$ have loads of z, and the remaining machines have loads no larger than z. Moreover, the loads are sorted for machines $b+1,\ldots,m$. Since the loads of machines $1, 2$ are equal before step 12 is applied, the third condition of a structured schedule holds as well. Let J' be the set of jobs assigned to machines $1,\ldots,b$, that is, jobs $1,\ldots,y$. Let Q be the total size of these jobs. Let σ be the output schedule, and let σ' be the schedule of the jobs of J' only. The costs of the two schedules are equal, and therefore we will show that σ' is optimal for J' and M. This cost is $L_1(\sigma') + L_2(\sigma')$, where $L_2(\sigma') = z$, and $L_1(\sigma') = \frac{Q-(s_2+\ldots+s_b)z}{s_1}$. Thus, the cost is

$$L_1(\sigma') + L_2(\sigma') = \frac{Q-(s_2+\ldots+s_b)z}{s_1} + z = \frac{Q+(s_1-(s_2+\ldots+s_b))z}{s_1}\ .$$

Recall that $b \leq k \leq \ell - 1$, and thus $s_1 > s_2 + \cdots + s_b$. Consider an optimal schedule π for J' and M that is structured (and thus its cost is $L_1(\pi) + L_2(\pi)$ by Proposition 6). Remove the (unique) job that is run during $[L_2(\pi), L_1(\pi))$ from π. Since machines $1, 2$ both have loads of at least $L_2(\pi)$, the resulting makespan is exactly $L_2(\pi)$ (it cannot be larger as no machine remains active after time $L_2(\pi)$, but at least one machine remains active just before time $L_2(\pi)$ so it cannot be smaller). This results in a schedule of a subset of $y-1$ of the jobs $1, 2, \ldots, y$ on machines $1, \ldots, b$, and according to the definition of z, its makespan cannot be below z, since this is the minimum makespan of the $y-1$ smallest jobs out of these jobs. This follows from Lemma 14 if $b < k$, and it follows from the definition of z if $b = k$, as in this case $y' = y$, and z is the minimum makespan for scheduling jobs $1, 2, \ldots, y'$ on machines $1, 2, \ldots, k$. Thus, we find $L_2(\pi) \geq z$. Since $L_i(\pi) \leq L_2(\pi)$ for $3 \leq i \leq b$ (as π is structured), we have $L_1(\pi) \geq \frac{Q-(s_2+\ldots+s_b)L_2(\pi)}{s_1}$, and

$$L_1(\pi) + L_2(\pi) \geq \frac{Q - (s_2 + \ldots + s_b)L_2(\pi)}{s_1} + L_2(\pi) = \frac{Q + (s_1 - (s_2 + \ldots + s_b))L_2(\pi)}{s_1} \ .$$

As $s_1 - (s_2 + \ldots + s_b) > 0$ and $L_2(\pi) \geq z$, we find that the cost of σ' does not exceed the cost of π. ■

We are left with the case $z \leq L'$. In this case $L' > 0$ (recall that $z > 0$ as z is the makespan for scheduling a non-empty set of jobs), so $\ell \leq m$ is an actual machine index. Thus, $k < m$ must hold since $k < \ell \leq m$. In this case $y' = k$.

Lemma 17 *If $z \leq L'$, then in the output schedule of step 10, the loads of machines $1, \ldots, k$ are equal to L'.*

Proof. The total size of jobs $\tilde{\jmath}, 2, \ldots, k$ is $x + p_2 + \ldots + p_k = L' \cdot S_k$. Thus, the kth bound on the makespan is equal to L'.

We compute the other $k-1$ bounds. For the ith bound (for $1 \leq i \leq k-1$), we select the i largest jobs out of jobs $\tilde{\jmath}, 2, \ldots, i, i+1$. If $x \leq p_{i+1}$, then these are jobs $2, \ldots, i+1$, and the ith bound is the same as for scheduling the jobs $2, \ldots, k$ on machines $1, \ldots, k$, so it does not exceed $z \leq L'$. Otherwise, assume by contradiction that $\frac{x+p_2+\ldots+p_i}{S_i} > L'$. Let i' be the last machine of the block of i in the list $L_1, \ldots, L_m$ (computed in step 1 of the algorithm), and let i'' be the first machine of this block. In the corresponding schedule (to the list of loads computed in step 1), machines $i'+1, \ldots, k$ would receive exactly jobs $i'+1, \ldots, k$ (as $k < m \leq n$), and their loads would be strictly above L' (since k is the last machine of a block, and since the loads in all blocks up to machine k are strictly larger than L', which is the load in the following block). Thus, $p_{i'+1} + \cdots + p_k \geq L'(s_{i'+1} + \cdots + s_k)$ (which holds even if $i' = k$). Let $L'' = L_i > L'$. In the schedule corresponding to the loads computed in step 1, the block of machine i receives jobs $i'', \ldots, i'$ (since $i' \leq k < m \leq n$). Since $p_{i''} + \cdots + p_i \leq L''(s_{i''} + \cdots + s_i)$ (the bound for jobs $i'', \ldots, i$ and the block of i) and $p_{i''} + \cdots + p_{i'} = L''(s_{i''} + \cdots + s_{i'})$ (the bound for jobs $i'', \ldots, i'$ and machines $i'', \ldots, i'$, that determines the load of all machines in the block of i in the list of step 1), we get $p_{i+1} + \cdots + p_{i'} \geq L''(s_{i+1} + \cdots + s_{i'})$ (which holds even if $i = i'$). Using $L'' > L'$, and taking the sum, we get $x + p_2 + \cdots + p_k > L' \cdot S_k$, a contradiction. ■

The next lemma has been used in previous work, and the bounds for the minimum makespan are based on similar arguments (see e.g. [4]).

Lemma 18 *In any good schedule ψ, for any $1 \leq k < m$, the total size of jobs assigned to machines $1, \ldots, k$ is at least P_k.*

Proof. Given ψ, remove jobs $k+1, \ldots, n$ from the schedule to create ψ'. Since ψ is good, the machines loads are sorted and there is no idle time. During the time $[L_i(\psi), L_{i-1}(\psi))$ (for $2 \leq i \leq k$), there are $i-1$ at most active machines at each time in ψ'. During $[0, L_k(\psi))$, there are at most k active machines at each time in ψ', since there are k jobs, and two parts of a job cannot be processed in parallel. Thus, the total size of these k jobs cannot exceed $\sum_{i=2}^{k}(L_{i-1}(\psi) - L_i(\psi))S_{i-1} + L_k(\psi) \cdot S_k = \sum_{i=1}^{k} s_i L_i(\psi)$, where the last expression is equal to the total size of jobs assigned to the first k machines in ψ (since there is no idle time), and we find, $P_k \leq \sum_{i=1}^{k} s_i L_i(\psi)$. ■

Lemma 19 *If $z \leq L'$, then the output is a valid schedule, this schedule is structured, and it is optimal.*

Proof. As machines $1, 2$ have loads of L' before step 12 (by Lemma 17 and since $k \geq 2$), the part of job 1 that is scheduled in step 12 is not scheduled to run in parallel to any part of job. Thus, the schedule is valid. In step 10, a schedule is created for machines $1, \ldots, k$ and in step 11, a schedule with the remaining jobs is created for machines $k+1, \ldots, m$. The loads of all machines of the first schedule are equal to L', while the other machines will have sorted loads that are at most L' (the machines of the first block, which contains machine ℓ will have this load, and the other machines have smaller loads that will be sorted). The schedule created in steps 10 and 11 does not contain idle time, as the schedule of each set of machines is created using ScheduleMakespan. The part of job 1 scheduled in step 12 is added without introducing any idle time.

Let c be the last machine of the block of ℓ (in the list of loads computed in step 1). Consider the set of jobs J_1 containing all jobs that are assigned to machines $1, \ldots, k$ (these are jobs $1, \ldots, k$), and let J_2 be the jobs of the block of ℓ. Let Q_1 and Q_2, respectively, be the total sizes of jobs of J_1 and J_2 (the jobs of J_2 are $k+1, \ldots, c$ if $c < m$ and otherwise they are the jobs $k+1, \ldots, n$). Let $J' = J_1 \cup J_2$ and $Q = Q_1 + Q_2$. Let σ be the output schedule, and let σ' be the schedule of the jobs of J' only. We have $Q_2 = (s_{k+1} + \ldots + s_c)L'$ and $Q_1 = (L_1(\sigma') - L_2(\sigma'))s_1 + (s_1 + \ldots + s_k)L'$. The cost of σ' is $L_1(\sigma') + L_2(\sigma')$, where $L_2(\sigma') = L'$. Thus, the cost is

$$L_1(\sigma') + L_2(\sigma') = \frac{Q - (s_2 + \ldots + s_c)L'}{s_1} + L' = \frac{Q + (s_1 - (s_2 + \ldots + s_c))L'}{s_1} \ ,$$

and since $Q_2 = (s_{k+1} + \ldots + s_c)L'$,

$$L_1(\sigma') + L_2(\sigma') = \frac{Q_1 - (s_2 + \ldots + s_k)L'}{s_1} + L' = \frac{Q_1 + (s_1 - (s_2 + \ldots + s_k))L'}{s_1} \ .$$

Recall that $s_1 > s_2 + \cdots + s_k$ and $s_1 \leq s_2 + \cdots + s_c$ (as $\ell \leq c$). Consider an optimal schedule π for J' and M that is structured. If $L_2(\pi) \geq L_2(\sigma')$, recall that since π is a good schedule,

$\sum_{i=1}^{k} s_i \cdot L_i(\pi) \geq P_k$ (by the previous lemma), and therefore $Q_1 = P_k \leq \sum_{i=1}^{k} s_i \cdot L_i(\pi) \leq s_1 L_1(\pi) + \sum_{i=2}^{k} s_i \cdot L_2(\pi) = s_1 L_1(\pi) + (s_2 + \cdots + s_k) L_2(\pi)$. We have

$$L_1(\pi) + L_2(\pi) \geq \frac{Q_1 - (s_2 + \ldots + s_k) L_2(\pi)}{s_1} + L_2(\pi) = \frac{Q_1 + (s_1 - (s_2 + \ldots + s_k)) L_2(\pi)}{s_1} .$$

As $s_1 - (s_2 + \ldots + s_k) > 0$ and $L_2(\pi) \geq L_2(\sigma')$, we find that the cost of σ' does not exceed the cost of π (using the second bound for σ').

If $L_2(\pi) < L_2(\sigma') = L'$, since in a structured schedule all the jobs are scheduled on machines $1, \ldots, c$, and since machine loads are sorted, similarly to the previous bound on $L_1(\pi)$, we have

$$L_1(\pi) + L_2(\pi) \geq \frac{Q - (s_2 + \ldots + s_c) L_2(\pi)}{s_1} + L_2(\pi) = \frac{Q + (s_1 - (s_2 + \ldots + s_c)) L_2(\pi)}{s_1} .$$

As $s_1 - (s_2 + \ldots + s_c) \leq 0$ and $L_2(\pi) < L_2(\sigma')$, we find that the cost of σ' does not exceed the cost of π. ■

To conclude, the algorithm finds an optimal solution in all cases.

Theorem 20 *The algorithm above finds an optimal schedule with respect to* MTLCT*, and its running time is* $O(m + n + \min\{m, n\}^2)$.

The running time in the case $m \leq n$ was shown to be $O(n + m^2)$. If $n < m$, a linear time pre-processing step is applied, where the n machines of largest speeds are found, and other machines are removed from the input. This is done using median selection and partitioning. The running time of the algorithm after pre-processing is $O(n^2)$, giving a running time of $O(m + n^2)$ in total in the case $m > n$.

4 Conclusion

In this paper, we designed a polynomial time algorithm for MTLCT. It is possible to define a generalized objective, where the goal is to minimize the sum of k largest completion times. The value k can be a constant integer, or a function of n such that $k \leq n$ (for example, valid cases are $k = 3$ and $k = n - 3$). The case $k = 2$ is MTLCT, the case $k = n$ is objective of minimizing the total completion time, and the case $k = 1$ is the makespan minimization objective. We believe that already for $k = 3$, the problem becomes more technical, and defining an algorithm for it as well as analyzing it would involve a long case-analysis.

References

[1] J. L. Bruno, E. G. Coffman Jr. and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7): 382–387, 1974.

[2] J. L. Bruno and T. Gonzalez. Scheduling independent tasks with release dates and due dates on parallel machines. Technical Report 213, The Pennsylvania State University, 1976.

[3] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. *Journal of Scheduling*, 12(5):517–527, 2009. Also in STACS 2004.

[4] L. Epstein and T. Tassa. Optimal preemptive scheduling for general target functions. *Journal of Computer and System Sciences*, 72(1):132–162, 2006.

[5] T. F. Gonzales and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25(1):92–101, 1978.

[6] W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973.

[7] E. C. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32–43, 1977.

[8] J. Labetoulle, E. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[9] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25(4):612–619, 1978.

[10] J. Liu and A. Yang. Optimal scheduling of independent tasks on heterogeneous computing systems. In *Proceedings of the ACM National Conference, vol. 1*, pages 38–45. ACM, 1974.

[11] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[12] H. Shachnai, T. Tamir, and G. J. Woeginger. Minimizing makespan and preemption costs on a system of uniform machines. *Algorithmica*, 42(3-4):309–334, 2005.

[13] R. Sitters. Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines. In *Proc. 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO2001)*, pages 396–405, 2001.