

# THE SINGLE MACHINE JUST-IN-TIME SCHEDULING PROBLEM WITH PREEMPTIONS

Yann Hendel

*LIP6*

*8 rue du capitaine Scott*

*75015 Paris, France*

Yann.Hendel@lip6.fr

Francis Sourd

*LIP6*

*8 rue du capitaine Scott*

*75015 Paris, France*

Francis.Sourd@lip6.fr

**Abstract** This paper deals with a variation of the single machine earliness/tardiness scheduling problem: preemption is allowed and, in order to penalize job interruptions, earliness costs depend on the starting time of the job while tardiness costs depend on the end times as in the usual models. We propose dominance properties and a polynomial time algorithm is given to solve the particular case where the sequence of the start and completion of the jobs is fixed.

**Keywords:** Scheduling, Single machine, just-in-time, preemption

## Introduction

Just-in-time scheduling has interested both practitioners and researchers for over a decade. A very common idea is to recognize that a job that completes either tardily or early in a schedule induces extra costs. Most of the time, just-in-time costs are function of the job completion time. However, in a context where preemption is allowed, such functions may not render the will of the scheduler: indeed, when starting

a job, the goal is generally to complete it as soon as possible so that it can be removed from the production line. If the cost only depends on the completion time of the job, idle time within the execution of the job will not be penalized. In order to model these costs, it seems natural to attach the earliness costs to the start time of the job, while tardiness costs are still tied to the completion of the jobs. Such an approach has been used in [Hendel and Sourd, 2003], for the job-shop problem with two jobs. In this problem, preemption is not allowed but the idea is to penalize waiting times in between operations.

In this paper, we adapt the classic one-machine earliness/tardiness problem when preemption is allowed and to render the earliness/tardiness costs, we will consider that each job has two due dates instead of one: one tied to the start time of the job and the other tied to the completion time of the job.

Formally, we consider a set of jobs  $\mathcal{J} = \{J_1, \dots, J_n\}$  which has to be scheduled on a single machine. Each job  $J_j \in \mathcal{J}$  has a processing time  $p_j$ . We define tardiness costs as  $T_j = \max(0, C_j - d_j^c)$  where  $C_j$  is the completion time of the job and  $d_j^c$  is the *ideal completion time* of job  $J_j$ , which corresponds to the usual due date. Earliness costs are defined as  $E_j = \max(0, d_j^s - S_j)$  where  $S_j$  is the starting time of job  $J_j$  and  $d_j^s = d_j^c - p_j$  is the *ideal starting time* of job  $J_j$ . Therefore,  $J_j$  would ideally be processed between  $d_j^s$  and  $d_j^c$ , which would cause no penalty costs. We want to minimize the total cost  $\sum_{j=1}^n (\alpha_j \cdot E_j + \beta_j \cdot T_j)$ .

We first notice that if preemption is not allowed, the problem is equivalent to the classic earliness/tardiness problem and if preemption is allowed but earliness costs are tied to the completion of the jobs, then the problem is equivalent to the single machine problem with only weighted tardiness costs.

The  $1||\sum_i f_i(C_i)$  problem has been extensively studied. One way to approach this NP-complete problem [Garey et al., 1988], is to solve a particular case where the job order is given i.e. when the job sequence is fixed; then eventually use it in a branch and bound or in a metaheuristic. [Garey et al., 1988] propose for this subproblem a direct algorithm based on the blocks of adjacent jobs in  $O(n \log n)$ , valid for the just-in-time problem with symmetric earliness and tardiness penalties. [Chrétienne and Sourd, 2003] present a generalization for convex functions. Finally, [Sourd, 2002] proposes a dynamic programming algorithm for general non-convex piecewise linear cost functions (which also considers eventual costs for the idle periods) whose complexity depends on the number of segments of the cost functions given in input.

In this paper, we will use the same approach. However, in contrast to the non-preemptive problem, the notion of job sequencing is not clear in

the preemptive case. Indeed, in an optimal solutions, starting time order usually differs from end time order. Let us first assume that the order of the starting times is given (but the order of the completion times is still to be determined). It can be easily shown that this problem is NP-complete by reducing the weighted tardiness problem, which is NP-complete [Lenstra et al., 1977], to it.

Therefore, we will study the particular case where the order of the starting and completion times of the jobs is given, that is we assume that all the events  $S_1, C_1, S_2, C_2, \dots, S_n, C_n$  are completely ordered. Basically, this constraint is formulated as a list of  $2n - 1$  inequalities (for instance  $S_1 \leq S_2 \leq C_2 \leq C_1 \leq S_3 \leq C_3$ ). Clearly, this ordering must satisfy  $S_i \leq C_i$  for each  $i$ .

In section 1, we propose dominance properties for this problem, and extract a dominant set of sequences. In section 2, we provide a strongly polynomial algorithm for solving the problem with dominant sequences.

## 1. Dominance properties

For any given sequence (that satisfies  $S_i \leq C_i$  for each  $i$ ), we can obtain an optimal schedule by solving the following linear program:

$$\begin{aligned}
 \text{(LP) : } & \min \sum_{i=1}^n (\alpha_i \cdot E_i + \beta_i \cdot T_i) \\
 \text{s.t.:} & \quad E_i \geq d_i^s - S_i, & i = 1 \dots n \\
 & \quad T_i \geq C_i - d_i^c, & i = 1 \dots n \\
 & \quad C_j - S_i \geq p_{ij}, & ((i, j) | C_j \geq S_i) \\
 & \quad S_i, C_i \geq 0 & i = 1 \dots n \\
 & \quad E_i, T_i \geq 0 & i = 1 \dots n
 \end{aligned}$$

The constants  $p_{ij}$  describe the mandatory processing time that occurs between  $S_i$  and  $C_j$ , which is the processing of all the jobs that have started after  $S_i$  and finished before  $C_j$ . This LP have  $O(n)$  variables but  $O(n^2)$  constraints. In the following, we will show that we can obtain by a direct strongly polynomial algorithm an optimal schedule for a set of restricted sequences with a better theoretical complexity than the one of the LP.

**Property 1.** *A sequence that contains two jobs such that  $S_i \leq S_j \leq C_i \leq C_j$  is dominated.*

*Proof:* We consider the time intervals when  $J_i$  and  $J_j$  are executed, the sum of the length of these intervals being  $p_i + p_j$ , the execution intervals of  $J_i$  and  $J_j$  can be rearranged such that  $J_i$  is totally executed

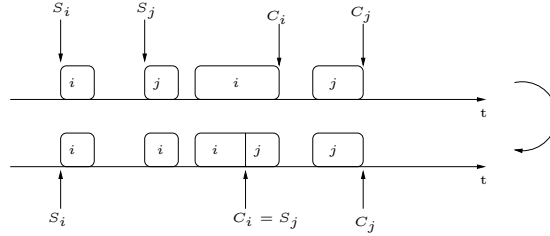


Figure 1. Proof of Property 1

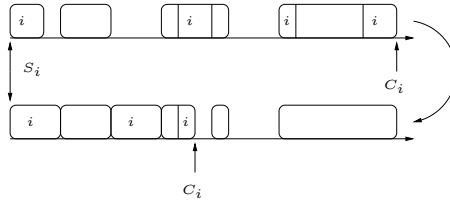


Figure 2. Proof of Property 2

before the start of  $J_j$ . Consequently the completion time of  $J_i$  is decreased and the starting time of  $J_j$  is increased. Thus the costs induced by the two jobs can only decrease (see Figure 1).

**Property 2.** *A sequence such that there is some idle time between the starting time and the completion time of a job is dominated.*

*Proof:* Consider a schedule where there is some idle time between the start of job  $J_i$  and its completion. The execution intervals of time of job  $J_i$  can be rearranged in order to fill the gaps between the start and the completion of  $J_i$ . In that manner,  $C_i$  may only decrease (see Figure 2).

For a pair of jobs  $(J_i, J_j)$ , we then have two possibilities: either  $S_i \leq S_j \leq C_j \leq C_i$  or  $S_i \leq C_i \leq S_j \leq C_j$ . In the first case, we say that  $J_j$  is *nested* in  $J_i$ . In the second case, we say that  $J_i$  and  $J_j$  are *separated*. The first property ensure a structure analogous to the nested parenthesis structure [Aho and Ullman, 1994]: for a given job  $J_j$ , each job that has started after  $S_j$ , has finished before  $C_j$ , and is therefore nested in  $J_j$ . According to the second property, there is no idle time between  $S_j$  and  $C_j$ . We call  $B_j$  the set of jobs that must be processed between  $S_j$  and  $C_j$  ( $J_j$  included). Since there is no idle time, we will call this set the block

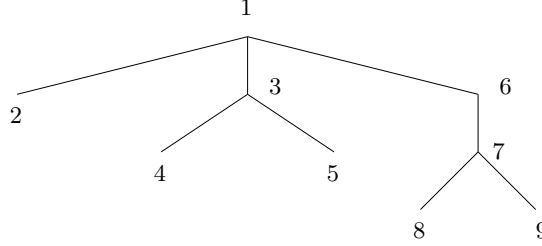
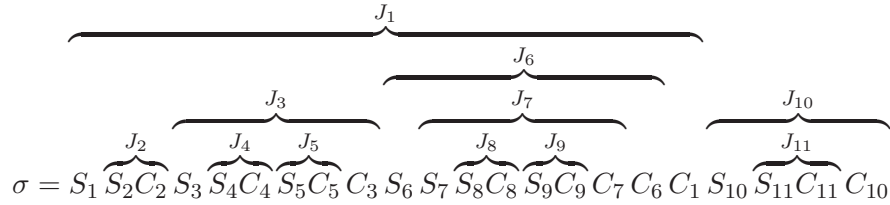


Figure 3. The date tree associated to the first *base-block* of  $\sigma$

associated with the job  $J_j$ . We finally denote by  $P_j$  the total processing time of the block  $B_j$ , which is equal to  $P_j = p_j + \sum_{J_k \in B_j} p_k$ .

A sequence, which has this nested parenthesis structure, is said *valid*. A *main block* is a block not nested by any other job. Thus a valid sequence can be divided into  $m$  main blocks which represent the jobs that are not *nested* by any other job. We call  $B_1 \dots B_m$  (in the order of the sequence) these main blocks. In the following section, we will show how to compute the cost for scheduling each of the main blocks.

Below is an example of a valid sequence of 11 jobs:



Sequence  $\sigma$  has two main blocks, one tied to  $J_1$  and one tied to  $J_{10}$ .

## 2. Solving the subproblem

### 2.1 Data Structure

According to the well-known connection between nested parenthesis expression and forest of trees, we use a tree to encode a main block: a leaf represents a job  $J_j$  such that the immediate successor in the sequence of  $S_j$  is  $C_j$  (therefore,  $C_j - S_j = p_j$ ). A node  $r$  with  $k$  sons represents a job  $J_r$  embracing the  $k$  sub-blocks represented by the  $k$  sons in their left-right order. Therefore the number of nodes in the tree corresponds to the number of jobs in the main block. The tree represented in figure 3 corresponds to the first main block of  $\sigma$ .

In each node  $r$ , we store:

- $P_r$  the total processing time between  $S_r$  and  $C_r$

- $f_r(t)$  the cost function for scheduling the block  $B_r$  such that  $C_r = t$  (and  $S_r = t - P_r$ ).

We are going to prove by induction that the information stored in each node can be derived from the information stored in the son nodes. Moreover, we also prove that the cost functions  $f_i$  are piecewise linear and convex. First, since a leaf represents a stand-alone job —say  $J_i$ — that is executed without preemption, its cost function is therefore equal to  $\max(\beta_i(t - d_i^c), \alpha_i(d_i^c - t))$ . Clearly, this function is piecewise linear and convex.

We then consider a subtree whose root encodes information about the job  $J_r$ . We rename  $\mathbb{T}_1 \dots \mathbb{T}_k$  the descendant trees of this root in the left-right order. The tree  $\mathbb{T}_i$  holds the cost function of  $B_i$ , which is piecewise linear and convex by the induction hypothesis, and the length  $P_i$  of block  $B_i$ . We first have  $P_r = p_r + \sum_{j=1}^k P_j$ . If we fix  $C_r$ ,  $S_r$  is also fixed ( $S_r = C_r - P_r$ ). The blocks  $B_1, \dots, B_k$ , in this order, have to be optimally scheduled within the time interval  $[S_r, C_r]$ , the sum of the idle periods in this interval being  $p_r$ . However, we need to know the cost of the block  $B_r$  when  $C_r$  varies. In the following, we propose a method to compute  $f_r(C_r)$  efficiently. The idea is first to get the ideal completion time of every sub-block when they are not constrained by  $S_r$  and  $C_r$  (In that case, there is no preemption, so we can solve the problem using the extension of [Garey et al., 1988] algorithm proposed by [Chrétienne and Sourd, 2003]): it provides a new block decomposition (see figure 4 - case a). Then, we show that for any  $C_r$ , the scheduling of  $B_1, \dots, B_k$  in  $[S_r, C_r]$  can be derived from the non-constrained schedule so that we finally have an algorithm to compute  $f_r$  from  $f_1, \dots, f_k$ .

## 2.2 Computation of a node

**2.2.1 Determining the sub-blocks of  $B_r$ .** We apply the algorithm of Chrétienne and Sourd to the sequence made of the blocks  $B_1 \rightarrow B_k$ : it provides the schedule  $B_1, \dots, B_k$  with the minimal cost. A new block structure is extracted: let  $B'_j$  be the concatenation of consecutive blocks  $B_u, \dots, B_v$  in this schedule such that there is no idle time between  $B_u$  and  $B_v$  but idle time before  $B_u$  and after  $B_v$ . The cost function of  $B'_j$  is  $f'_j(t) = \sum_{i=u}^v f_i(t - \sum_{w>i} p_w)$  (see figure 4 - case a).  $f'_j$  is piecewise linear and convex as a sum of piecewise linear and convex functions.

**2.2.2 Computing the contribution of a block.** We next have to add the two constraints which state that the jobs have to be executed after  $S_r$  and before  $C_r$ : setting  $S_r$  force the far left blocks to

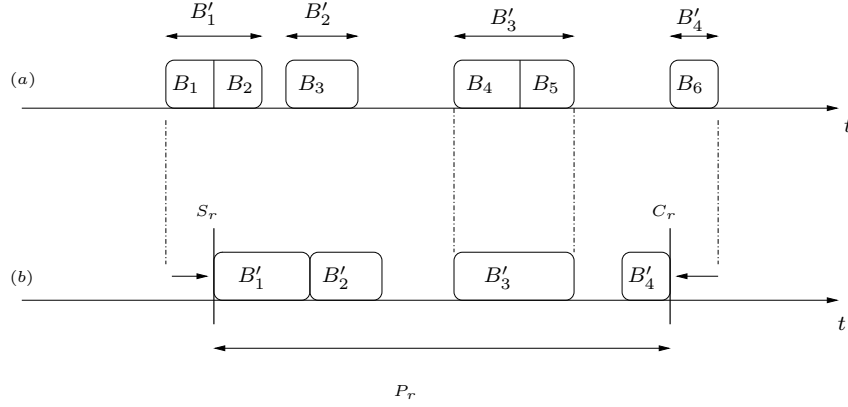


Figure 4. Optimal schedule without a completion time constraint, next constrained by  $C_r$ .

be right-shifted and setting  $C_r$  force the the far right blocks to be left-shifted (see figure 4 - case b). We then say a block is *critical* when it is constrained by the job  $J_r$ : a block  $B'_j$  is said to be *right-critical* (resp. *left-critical*), when there is no idle time between  $B'_j$  and  $C_r$  (resp. when there is no idle time between  $S_r$  and  $B'_j$ ). At any moment, a block  $B'_j$  is either left-critical, either *on time*, or right-critical; indeed, if a block is not critical, it is set at its due date in order to minimize its cost.

Formally, we have  $l$  blocks,  $B'_1 \dots B'_l$ , and we know the cost function  $f'_j(t)$  of each block. We denote by  $t_j$  the earliest time at which  $f'_j$  is minimum. We say that block  $B'_j$  become *on time* at time  $\tau_j = t_j + p_r + \sum_{i>j} P_i$  and  $B'_j$  become *right-critical*, at time  $\tau'_j = t_j + \sum_{i>j} P_i$ . While  $t > \tau_j$ ,  $B'_j$  is left critical, its cost is given by  $f'_j(t - p_r - \sum_{i>j} P_i)$ . Between  $\tau'_j < t \leq \tau_j$ ,  $B'_j$  is on time, its cost is given by  $f'_j(t_j)$ . Finally, when  $t \leq \tau'_j$ ,  $B'_j$  is right critical, its cost is given by  $f'_j(t - \sum_{i>j} P_i)$ .

In figure 4 - case b,  $B'_1$  and  $B'_2$  are left-critical,  $B'_3$  is on time and  $B'_4$  is right critical.

Eventually, we obtain the cost of the block  $B_r$  by adding all the block's contribution, which we have supposed piecewise linear and convex, and the earliness and tardiness cost of  $B_r$ . This sum is piecewise linear and convex too. We have thus proved the induction hypothesis stated in section 2.1.

**2.2.3 Complexity.** The cost functions  $f_1, \dots, f_k$  are piecewise linear and convex and we denote by  $\|f_1\|, \dots, \|f_k\|$ , their number of segments. Indeed, each cost function is fully described by the sorted list

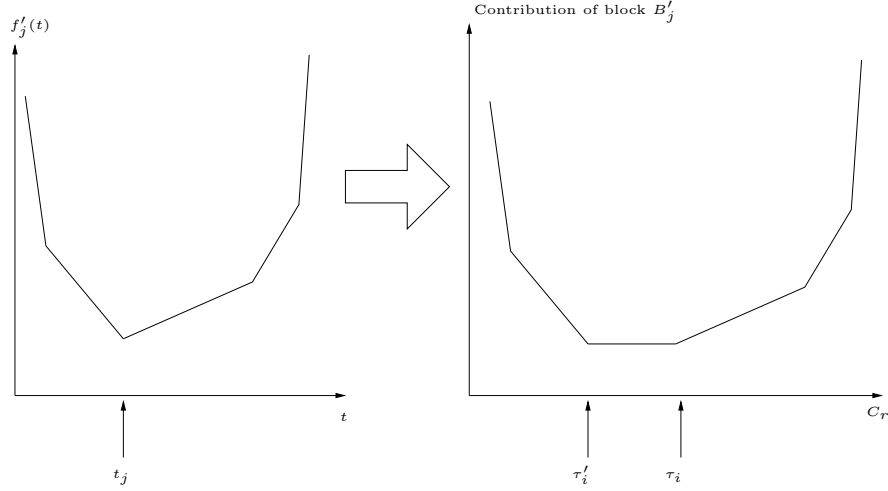


Figure 5. Cost function of a block  $B'_j$  and its contribution in the cost function  $B_r$ .

of its segments (each segment is represented by the coordinates of its endpoint). In [Chrétienne and Sourd, 2003], it is proved that the non constrained schedule can be computed in  $O(\sum_{i=1}^k \|f_i\| \cdot \log(\sum_{i=1}^k \|f_i\|))$ . The computing of the  $(B'_j)_{1 \leq j \leq l}$ , is then done in  $O(k)$  time and the sum of the number of segments of the  $B'_j$  is at most  $\sum_{i=1}^k \|f_k\|$ . To compute the contribution of a block  $B'_j$  in the calculus of  $B_r$ , one segment is added (see figure 5): finally, adding the contributions of each  $B'_j$  is done in  $O(\sum_{i=1}^k \|f_k\|)$ .

In conclusion,  $B_r$  (standing for the node  $r$ ) is computed in  $O(\sum_{i=1}^k \|f_i\| \cdot \log(\sum_{i=1}^k \|f_i\|))$  and has at most  $\sum_{i=1}^k \|f_k\| + k + 2$  (2 stands for the two segments added by the  $J_r$  earliness/tardiness function) segments.

### 2.3 Computing the minimum cost associated with a tree and solving the sub-problem

Consider a main block described by a tree with  $n'$  nodes, we prove by induction that the cost function stored at the root has less than  $3 \cdot n' - 1$  segments: each cost function stored at a leaf has two segments ( $3 \cdot 1 - 1$ ). Suppose that the root has  $k$  subtrees which represent  $n'_1, \dots, n'_k$  nodes such that  $\sum_{i=1}^k n'_i = n' - 1$  and each subtree has less than  $3 \cdot n'_i - 1$  segments. From the preceding section, we have clearly the maximal number of segments of  $B'_r$ :  $\sum_{i=1}^k 3 \cdot (n'_i - 1) + 2 + k$  which is less than  $3 \cdot n' - 1$ .

Every node of the tree has to be computed, starting at its leaves. Each node has at most  $3 \cdot n' - 1$  segments. In the preceding section, we have proved that its computation can be done in  $O(n' \cdot \log(n'))$ . Since there are  $n'$  nodes, the cost function of the main block can therefore be computed in  $O(n'^2 \cdot \log(n'))$ .

Suppose that we have  $m$  main blocks. We first have to compute their respective cost functions  $f_1, \dots, f_m$  (in left-to-right order): it can be done in  $O(n^2 \cdot \log(n))$ . To obtain the final schedule, we can again apply [Chrétienne and Sourd, 2003] algorithm. It's done in  $O(\sum_{i=1}^m \|f_i\| \cdot \log \sum_{i=1}^m \|f_i\|)$  which is equal to  $O(n \log(n))$ .

The global complexity of the algorithm is therefore  $O(n^2 \cdot \log(n))$ .

### 3. Conclusion and perspectives

We have introduced a variation of the one-machine earliness/tardiness scheduling problem and proposed a direct algorithm to solve the subproblem where the sequence of the starting and completion of the jobs is given. Future works could address the general resolution of this problem. The search of lower bounds could be of interest in order to use the above subproblem in a branch and bound algorithm. Neighborhood search could be tried too. Actually, the nested parenthesis structure provide a convenient scheme in these two axes.

### References

- [Aho and Ullman, 1994] Aho, A. V. and Ullman, J. D. (1994). *Foundations of Computer Science*. W. H. Freeman.
- [Chrétienne and Sourd, 2003] Chrétienne, P. and Sourd, F. (2003). Pert scheduling with convex cost functions. *Theoretical Computer Science*, 292:145–164.
- [Garey et al., 1988] Garey, M., Tarjan, R., and Wilfong, G. (1988). One processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13:330 – 348.
- [Hendel and Sourd, 2003] Hendel, Y. and Sourd, F. (2003). Job-shop à deux tâches avec critères irréguliers. *Ecole d'Automne de Recherche Opérationnelle*.
- [Lenstra et al., 1977] Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.
- [Sourd, 2002] Sourd, F. (2002). Scheduling a sequence of tasks with general completion costs. *European Journal of Operational Research*, to appear.