International Journal of Operations Research

International Journal of Operations Research Vol. 5, No. 1, 61–67 (2008)

Job-Shop with Two Jobs and Irregular Criteria

Yann Hendel*, and Francis Sourd

Laboratoire d'Informatique de Paris 6 - UMR7606, 4, place Jussieu - 75005 Paris

Received August 2006; *Revised February 2007*; *Accepted March 2007*

Abstract—We use the Akers-Friedman geometric approach to solve the two jobs job-shop problem when there is an earliness cost on the first operation and a tardiness cost on the last operation of each job. We then generalize the problem by imposing earliness and tardiness costs on each operation and finally, we solve it using a dynamic programming algorithm.

Keywords—Earliness-tardiness scheduling, Polynomial algorithm, Two-job job-shop

1. INTRODUCTION

We consider the two jobs job-shop problem where the goal is to minimize earliness and tardiness costs and more generally the costs incurred by irregular criteria. A unique characteristic of the two jobs job-shop problem is that it can be represented by a grid (proposed by Akers and Friedman (1955)), that highlights the valid schedules. Based on this geometric representation, Brucker (1988) proposed a polynomial algorithm when the optimization criterion is the minimizing of the completion time of the last job. Sotskov (1991) generalized this algorithm in the case where the optimization criterion is the minimization of regular cost functions assigned to each job. In this case, a regular function is a non decreasing function whose parameter is the completion time of the last operation. In a just-in-time environment, Agnetis et al. (2001) proposed to put a quasi-convex cost function whose parameter is the completion time of the last operation of each job (a function f is said to be quasi-convex, if $f(\lambda x+(1-\lambda)y) \leq Max(f(x), f(y)), \quad \forall \lambda \in [0,1]$). This function is a generalization of the usual earliness-tardiness cost function. Agnetis et al. (2001) solve this problem using a polynomial time algorithm.

In the first part of this paper, we propose a different cost function than that used by Agnetis et al. (2001): in our model, the earliness costs are functions of each job's first operation starting time, whereas the tardiness costs are still tied to the completion of the last operation of each job. In fact, we observe that in a production environment, once a job is started, the goal is to complete it as soon as possible so that it can be taken off the production chain: if we place earliness and tardiness costs only on the last operation, we can obtain results in which most of the operations of the two jobs are carried out as early as possible, whereas the only scheduled operations that are optimized are the last ones (see Figure 1). This issue is especially relevant for shop problems since the jobs are divided into operations which may be very different from each other and therefore must all be executed just-in-time. Thus, the model we propose penalizes the idle times between job operations.

In order to solve this first problem (referred to as JS2JET) we consider earliness and tardiness costs independently. As we will demonstrate in Section 3, when the starting times of the two jobs are set, we can adapt Sotskov's algorithm to minimize the tardiness costs of the two jobs. We will see in Section 4 that we can extract a dominant set of starting times of the two jobs. Finally, we propose a polynomial algorithm to solve JS2JET.

In the second part of this paper (Section 5) , we address a more general case in which there is a general cost function whose parameter is the completion time of each operation (the time scale is discretized and the cost is given for each step). We solve this second problem, JS2JG, using dynamic programming. We obtain a pseudopolynomial complexity directly related to the horizon of the schedule.

2. DEFINITION AND NOTATIONS OF JS2JET

We consider two jobs, A and B, whose sets of operations are: $\{A_1, A_2, ..., A_{n_A}\}$ and $\{B_1, B_2, ..., B_{n_B}\}$. The operations of each job have to be executed according to their index on a set of machines. Let us call the processing times of operation A_i and B_i, p_i^A and p_i^B, their completion times C_i^A and C_i^B, their starting times S_i^A and S_i^B. Total processing times for jobs A and B are $P^A = \sum_{i=1}^{n_A} p_i^A$ and $P^B = \sum_{i=1}^{n_B} p_i^B$.

Let d^A and d^B be the due dates of jobs A and B, respectively. The weighted tardiness cost T^A of job A is $\beta_A * (C_{n_A}^A - d^A)$ if $C_{n_A}^A \geq d^A$, 0 otherwise. The weighted tardiness cost T^B of job B is $\beta_B * (C_{n_B}^B - d^B)$ if $C_{n_B}^B \geq d^B$, 0 otherwise. We now introduce the ideal starting time of A_1 and B_1: $d_0^A = d^A - P^A$. The earliness cost E^A of job A is $\alpha_A * (d_0^A - S_1^A)$ if $d_0^A \geq S_1^A$, 0 otherwise. The

* Corresponding author's email: yann.hendel@gmail.com

earliness cost E^B of job B is $\alpha_B * (d_0^B - S_1^B)$ if $d_0^B \geq S_1^B$, 0 otherwise. We want to minimize the weighted sum of earliness and tardiness costs of the two jobs, i.e., $Min\ E^A + E^B + T^A + T^B$. This can be noted $J|n = 2|E^A + E^B + T^A + T^B$.

3. MINIMIZING THE TRADINESS WHEN THE STARTING TIMES OF BOTH JOBS ARE SET

3.1 Geometric approach and Brucker's algorithm

The geometric approach, shown in Figure 2 is based on a grid of length P^A and width P^B. The x-axis represents the operations of job A and the y-axis represents the operations of job B. Obstacles are put on the grid: each obstacle is composed of 2 operations, 1 from A and 1 from B, that need to be executed on the same machine. If A_i and B_j need to be executed on the same machine, we call Δ_{ij} the obstacle whose southwest coordinate is $(\sum_{k=1}^{i-1} p_k^A, \sum_{k=1}^{j-1} p_k^B)$ and whose northeast coordinate is $(\sum_{k=1}^{i} p_k^A, \sum_{k=1}^{j} p_k^B)$. The northeast, northwest, southwest and southeast corners of Δ_{ij} are denoted by Δ_{ij}^{NE}, Δ_{ij}^{NW}, Δ_{ij}^{SW} *and* Δ_{ij}^{SE}. We denote by r the number of obstacles in the grid.

A valid schedule Σ is a path composed of vertical, horizontal or diagonal (with angle $\pi/4$) segments, starts from the southwest corner O and ends at the northeast corner F. At time $t = 0$, no operation has been executed, so the path starts at point O in the grid. If a path reaches a point with coordinates (x, y) at time t, it means that job A has been executed during x units of time and job B during y since $t = 0$. A vertical segment $[(x, y), (x + k, y)]$ (resp. horizontal segment $[(x, y), (x, y + k)]$) means that only job A (resp. B) is executed between times t and $t + k$. A diagonal segment $[(x, y), (x + k, y + k)]$ means that both jobs A and B are executed in parallel between times t and $t + k$. Finally, the path has to avoid the interior of any obstacle.

Brucker (1988) has shown that solving the problem $J|n = 2|Cmax$ corresponds to finding the shortest path in a network $N = (V, A)$ where V is the set of northwest and southeast corners of obstacles augmented by O and F. In order to build an arc from a vertex k, we go diagonally through the grid until we hit an obstacle. If the obstacle is the grid's edge, then F is the only successor of i, otherwise, we meet an obstacle Δ_{ij} and then k has two successors: Δ_{ij}^{NE} and Δ_{ij}^{SE}.

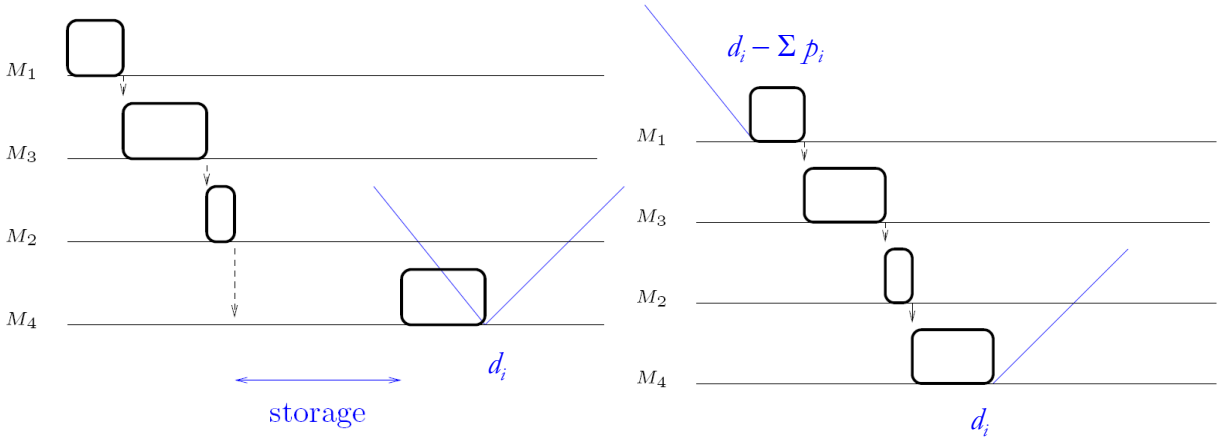

Figure 1. Two different just-in-time functions.

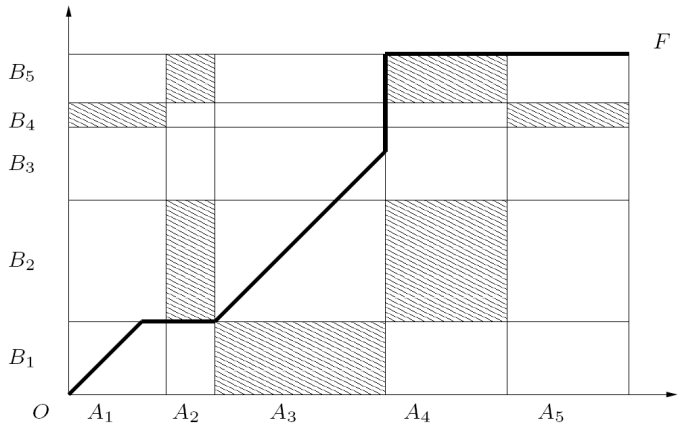

Figure 2. A valid path in the grid with its corresponding Gantt diagram.

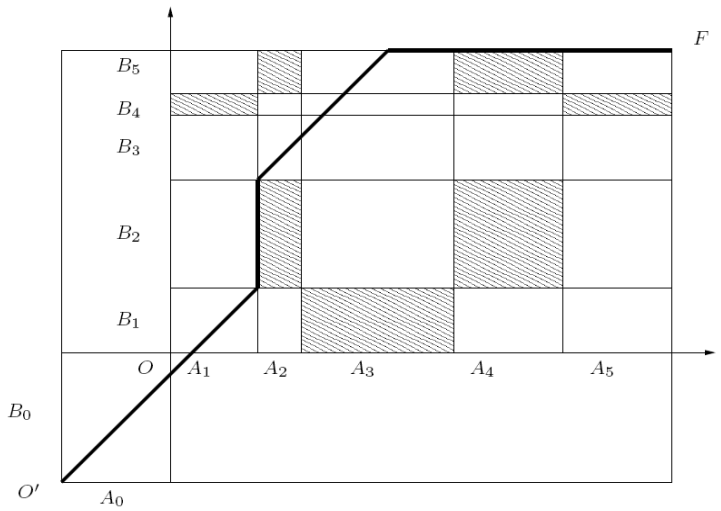

Figure 3. A valid path in a grid with starting dates P_0^A and P_0^B.

The crux of the algorithm is to execute the two jobs in parallel until an obstacle is found, at which point the algorithm "chooses" to go through the northwest corner or the southeast corner. Sotskov (1991) adapted this algorithm in order to minimize a regular function. Here the regular function is the minimizing of the sum of the two jobs tardiness. From now on, we call this algorithm the Brucker-Sotskov algorithm and use it as a "black box" when designing of our algorithm.

3.2 Adapting the Brucker-Sotskov algorithm when the starting times of both jobs are set

In this section, we suppose that the starting times of the jobs are set (as are the earliness costs). We now show that we can minimize the weighted tardiness by adapting the Brucker-Sotskov algorithm. In the following section, we show that we can extract a dominant set ε of pairs of starting times of jobs A and B, that allows for an optimal solution of JS2JET.

Finally, the algorithm we propose consists of the minimized weighted tardiness sum of jobs A and B when their starting times are pairs of E. In the remainder of this section, the starting times of jobs A and B are fixed and respectively equal to S_1^A and S_1^B and we want to minimize the weighted tardiness sum, that is $J|n = 2,\ S_1^A,\ S_1^B\ |T_A + T_B$. In order to achieve this, we add two extra dummy operations, A_0 and B_0, that are to be executed before A_1 and B_1, respectively. They are to be executed on two dummy machines. When using the Brucker-Sotskov geometric algorithm, those two operations represent periods of inactivity before the start of jobs A and B. Since those operations are executed on dummy machines, they do not have to compete with other operations of A and B: B_0 is executed in parallel with operations of B and A_0 with operations of A. In Akers and Friedman's representation, it comes down to introducing a point O' with coordinates $(-P_0^A\ ,\ -P_0^B)$. According to the Brucker-Sotskov algorithm, we have a diagonal segment from O' which length is $Min(P_0^A\ ,\ P_0^B) * \sqrt{2}$ (see Figure 3). $\left|P_0^A - P_0^B\right|$ represents the time lag between the starting times of the two jobs. If we use the Brucker-Sotskov algorithm from O', the operations A_1 and B_1 start respectively at P_0^A and P_0^B, and we obtain a minimum cost for the sum of the tardiness cost.

4. DETERMINING THE JOBS STARTING TIMES AND SOLVING JS2JET

4.1 Dominance properties

Let $\mathcal{P}$ be a set of pairs of starting times for jobs A and B. In this section, we establish properties of optimal schedules. These properties are used to reduce $\mathcal{P}$ to set ε:

Property 1.

- Either, there exists an optimal schedule and two integers $i \le n_A$ and $j \le n_B$ such that the i first operations of A and the j first operations of B are executed without idle times and such there exists an obstacle Δ_{ij} and we then have:
  - either $C_i^A = S_j^B$
  - or $S_i^A = C_j^B$
- or for each job, there is no idle time at all between the executions of all its operations.

Proof. We provide a constructive demonstration: we consider a feasible schedule σ. We consider the first block of operations of job A, i.e. operations A_1, ..., A_k such that there is no idle time between these operations. We right shift this block on the time scale. We denote by σ^* the modified current schedule obtained from σ. Three kinds of event may happen:

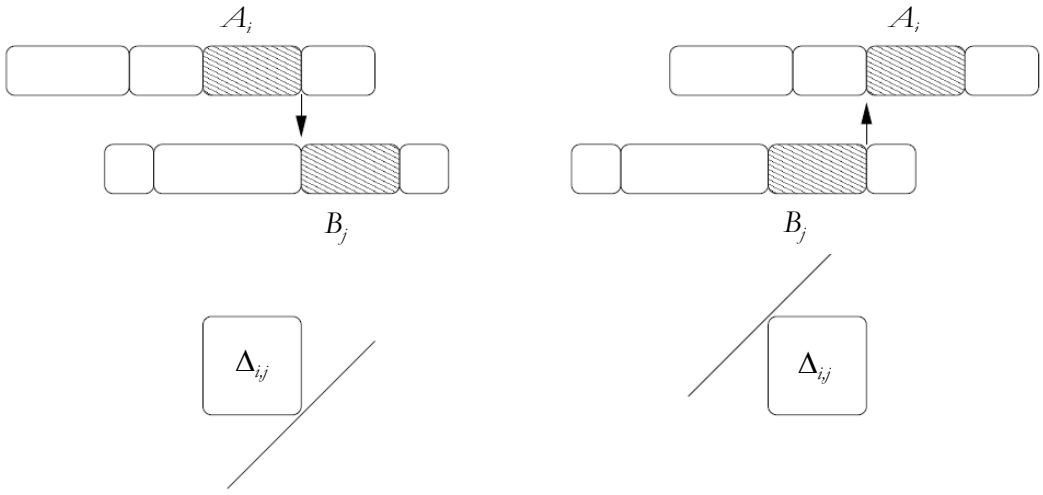

Figure 4. The first two cases of property 1.

1. Either the current block merge with another block of operation of job *A*. We proceed with the right shifting of this new block.
2. Or, for one A_i of the block, there is an operation B_j such that $C_i^A = S_j^B$ and Δ_{ij} exists. The shifting is then stopped (this is the left case in Figure 4. The path in the grid corresponding to σ^* goes through Δ_{ij}^{SE}).
3. Or A_{n_A} is in the current block and the shifting is stopped.

We do the same with the leftmost block of operations of job *B*. The current block is rigth-shifted. the second event is replaced by $S_i^A = C_j^B$ (it is the right case in Figure 4, the path in the grid corresponding to σ^* goes through Δ_{ij}^{NE}). Among the events $C_i^A = S_j^B$ and $S_i^A = C_j^B$, we choose the one that happened earlier in the time scale. If these two events do not happen, it means that *A* and *B* are executed in unique blocks.

For the newly obtained schedule σ^*, the earliness costs of the jobs *A* and *B* may only decrease. Therefore, the cost of σ^* is lower than the one of σ.

From now on, let Σ_1 be the set of schedules verifying property 1, we establish a symmetric property for the rightmost operations of *A* and *B*.

Property 2.

- Either, there exists an optimal schedule which belong to Σ_1 and two integers $i \le n_A$ *and* $i \le n_B$ such that the *i* last operations of *A* and the *j* last operations of *B* are executed without idle times and such there exists an obstacle Δ_{ij} and we then have:
  - either $C_i^A = S_j^B$
  - or $S_i^A = C_j^B$
- or for each job, there is no idle time at all between the executions of all its operations.

Proof. The proof is similar to the one of property 1 except that instead of right-shifting the leftmost operations of *A* and *B*, it is the rightmost operations of *A* and *B* that are left-shifted and we choose the event that happened later on the time scale.

From now on, let Σ_2 be the set of schedules verifying property 2. The following is an example illustrating how to go from any schedule to one that verifies Σ_2. We consider the 2 jobs of Tables 1 and 2. On Figure 5, the steps involved in the transformation are represented:

1. A random valid schedule.
2. A_1 and A_2 are right-shifted until A_2 encounters B_3.
3. B_1 is right-shifted until it encounters B_2. At this point, the schedule verifies property 1 and the schedule goes through $\Delta_{2,3}^{SE}$.
4. A_6 and A_7 are left-shifted until A_6 encounters B_5.
5. B_7 is left-shifted until it encounters A_6. At this point, the schedule verifies property 2 and the schedule goes through $\Delta_{6,7}^{SE}$.

The properties 1 and 2 ensure that the time lags between the starting times of the two jobs (and respectively the completion times of the two jobs) of an optimal schedule can be known. Indeed, if we consider that Δ_{ij} is the first obstacle and that $C_i^A = S_j^B$ (so the path goes through Δ_{ij}^{SE}), then we have $S_1^A - S_1^B = \sum_{k=1}^{i} p_k^A - \sum_{l=1}^{j-1} p_l^B$ assuming that both jobs are executed at the same time starting from O' (which is the case when the Brucker-Sotskov algorithm is applied). Similarly, if $S_i^A = C_j^B$ (the path goes through Δ_{ij}^{NE}), we then have $S_1^A - S_1^B = \sum_{k=1}^{i-1} p_k^A - \sum_{l=1}^{j} p_l^B$. There are *r* obstacles, then let $\omega = (\omega_1, ..., \omega_{2r})$ be the list of these constants. However, we can notice that some obstacles may never be the first obstacle, therefore, for some *i*, ω_i are not useful to obtain optimal schedules (in the algorithm presented in the next section, they lead to valid schedules, so they may remain in the list).

Table 1. Job A

-	Processing time	Machine	Incompatibility
A_1	5	M_1	B_2
A_2	4	M_2	B_3
A_3	1	M_3	B_1
A_4	4	M_4	B_6
A_5	2	M_5	B_4
A_6	2	M_6	B_5,B_7
A_7	1	M_7	-

Table 2. Job B

-	Processing time	Machine	Incompatibility
B_1	2	M_3	A_3
B_2	3	M_1	A_1
B_3	2	M_2	A_2
B_4	5	M_5	A_5
B_5	1	M_6	A_6
B_6	2	M_4	A_4
B_7	3	M_6	A_6

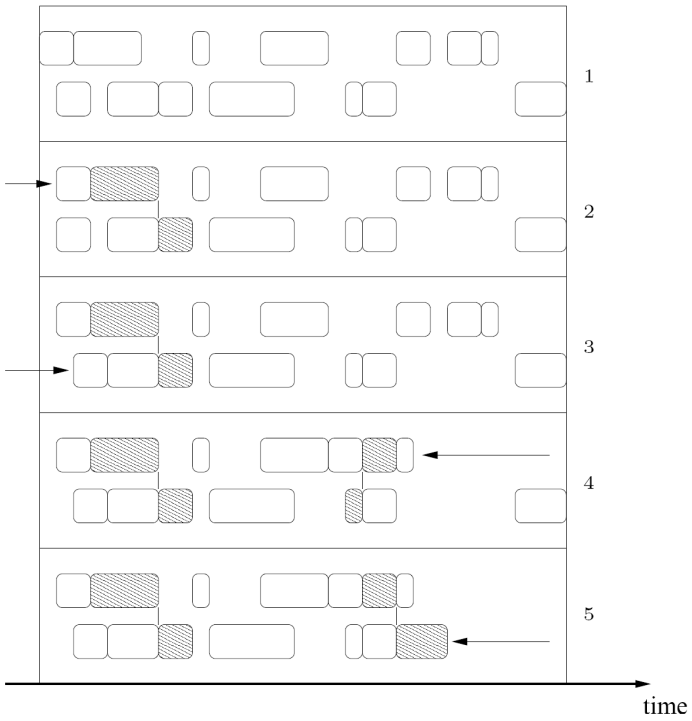

Figure 5. The different steps to transform any schedule. In each rectangle, job A is represented on top.

We proceed in the same manner for the last obstacles: if we consider that Δ_{ij} is the last obstacle and that $C_i^A = S_j^B$ (so the path go through Δ_{ij}^{SE}), then, we have $C_{n_A}^A - C_{n_B}^B = (P^A - \sum_{k=i+1}^{n_A} p_k^A) - (P^B - \sum_{l=j}^{n_B} p_l^B)$ assuming that both jobs are executed at the same time starting from Δ_{ij}^{SE} (which is the case when the Brucker-Sotskov algorithm is applied). Similarly, if $S_i^A = C_j^B$ (the path goes through Δ_{ij}^{NE}), we then have $C_{n_A}^A - C_{n_B}^B = (P^A - \sum_{k=i}^{n_A} p_k^A) - (P^B - \sum_{l=j+1}^{n_B} p_l^B)$. Let $\omega' = (\omega'_1, ..., \omega'_{2r})$ be the other constant list. Again, some obstacle may never be the last ones.

We now know the possible time lags between the starting times of the two jobs (and respectively between the completion times of the two jobs). We need another property to fix the starting time or completion time of either one of the two jobs:

Property 3. There is an optimal schedule which belongs to Σ_2 where at least one of the six following conditions is met :

1. $S_1^A = d_1^A$ (or $S_1^A = 0$)
2. $S_1^B = d_1^B$ (or $S_1^B = 0$)
3. $C_{n_A}^A = d^A$
4. $C_{n_B}^B = d^B$

Proof. Let σ be a schedule which belongs to Σ_2, we can either push backward or postpone the execution of all the two jobs' operations so as to diminish the scheduling cost. Cost variation is linear except when one of the operations draws to its completion time or when one of the operations A_1 or B_1 finds itself scheduled at $t = 0$.

4.2 Algorithm

We first consider the cases 1 and 2 of property 3. In the grid we apply the Brucker-Sotskov sub-routine with tardiness factor β_A and β_B and due dates d^A and d^B for the following pairs of starting times:

- if $S_1^A = d_1^A$, then $(d_1^A, Max(d_1^A - \omega_i, 0))\ \forall i$;
- if $S_1^A = 0$, then $(0, Max(\omega_i, 0))\ \forall i$;

- if $S_1^B = d_1^B$, then $(Max(d_1^B + \omega_i, 0), d_1^B)$ $\forall i$;
- if $S_1^B = 0$, then $(Max(\omega_i, 0), 0)$ $\forall i$.

For the last two cases of property 3, we need to reverse the time scale and consider the symmetric problem : we consider the grid where the *x*-axis represents the operations of job *B* and the *y*-axis represents the operations of job *A*. The operations of job *B* are given in the order $\{B_{n_B}, ..., B_2, B_1\}$ and operations of job *A*, $\{A_{n_A}, ..., A_2, A_1\}$. In this new grid, we apply the Brucker-Sotskov sub-routine with tardiness factor α_B and α_A and due dates d^B and d^A for the following pairs of starting times:

- if $C_{n_A}^A = d^A$, then $(Max(d_1^A - \omega_i', 0), d_1^A)$ $\forall i$;
- if $C_{n_B}^B = d^B$, then $(d_1^B, Max(d_1^B + \omega_i', 0)$ $\forall i$;

We refer to Brucker (2004) for the multiple constructions of the grid and the associated networks. It can be done in time $O(r\log r)$. For every couple of starting times provided below, a Brucker-Sotskov sub-routine has to be executed. It is done in time $O(r)$. There are $6r$ pairs of starting times. Therefore, the overall complexity of the algorithm is $O(r^2)$.

5. GENERAL END-TIME-DEPENDENT COSTS

In this section, the cost of each operation X is $c(X, t)$ when X completes at t. Each job must be completed before a time horizon T — the value T is assumed to be greater that the earliest completion time (or makespan) of the schedule. These costs are given in input as an array of T values $c(X, 1)$, $c(X, 2)$, ..., $c(X, T)$ for each operation X so that the size of the input is in $O(n_A n_B T)$. The problem is to compute a schedule whose total cost

$$\sum_{i=1}^{n_A} c(A_i, C_i^A) + \sum_{i=1}^{n_B} c(B_i, C_i^B)$$

is minimal.

For any value of $p \in [0, p^B]$, we will say that job *B* is *p*-processed if the sum of the lengths of the time intervals during which *B* is processed is equal to *p*. For some $i \in \{1, ..., n_B\}$, we have $\sum_{1\le j<i} p_j^B < p \le \sum_{1\le j\le i} p_j^B$, then operation B_i is said to be in process or complete at *p*. This operation is denoted by $B(p)$ and the index i is denoted by $i(p)$. If operation $B(p)$ is in process at t, $H(p) = p - \sum_{1\le j<i(p)} p_j^B$ denotes the length of the part of $B(p)$ which has already been executed and, respectively, $T(p) = \sum_{1\le j<i(p)} p_j^B - p$ is the amount of processing before the end of $B(p)$. We use the variable δ_p : $\delta_p = 1$ if $p = \sum_{1\le j<i} p_j^B$ (in this case, operation B_i is complete and $H(p) = p_i^B$ and $T(p) = 0$; otherwise $\delta_p = 0$.

In order to introduce the dynamic programming scheme to solve the problem, we define, for any $(t, k, p) \in [0, T] \times [0, n_A] \times [0, p^B]$, the subproblem $P(t, k, p)$ in which

- job *A* is restricted to its first *k* operations which must be processed before *t*,
- job *B* has to be *p*-processed at time *t*. The costs of operations that complete after *t* are ignored. To put it precisely, if operation B_i is in process at *t*, then B_i must start at $t - H(p)$.

We now analyze the properties of an optimal solution for $P(t, k, p)$. Let us first assume that $C_k^A < t$ so that $C_k^A < t-1$. If an operation of *B* is in process in $(t - 1, t)$ then job *B* must be $p - 1$-processed at time $t - 1$. We have $P(t, k, p) = P(t - 1, k, p - 1) + \delta_p C(B(p), t)$ (abusing notation, $P(k, t, p)$ denotes both the problem and its optimal value). If an operation of *B* is not in process in $(t - 1, t)$ we clearly have $P(t, k, p) = P(t - 1, k, p)$. So, if $C_k^A < t$, the cost for $P(t, k, p)$ is

$$Min(P(t-1, k, p-1) + \delta_p C(B(p), t), P(t-1, k, p)) \qquad (1)$$

Let us now assume that $C_k^A = t$. If *B* is in process at *t*, the machine on which $B(p, t)$ is processed must be different from the machine of A_k; otherwise the schedule is not feasible. If $H(p) \ge p_k^A$, the cost for $P(t, k, p)$ is

$$P(t - p_k^A, k-1, p - p_k^A) + C(A_k, t) + \delta_p C(B(p), t) \qquad (2)$$

We now consider that $H(p) < p_k^A$ (see Figure 6). Let $\overline{p} < p$ be such that *B* is $\overline{p}$-processed at the start time of A_k. For any $\pi \in [\overline{p}, p]$, operation $B(\pi)$ cannot be processed on the same machine as A_k. We have of course $i(\overline{p}) \le i(p)$ and $T(\overline{p}) + H(p) \le p_k^A$. We have two possibilities: either $\delta_p = 1$, and so, in an optimal schedule, the activity $B_{i(\overline{p})+1}, ..., B_{i(p)}$ must be optimally scheduled in the time interval $[t - p_k^A + T(\overline{p}), t]$; or $\delta_p = 0$ and $B_{i(\overline{p})+1}, ..., B_{i(p)-1}$ must be optimally scheduled in the time interval $[t - p_k^A + T(\overline{p}), t - H(p)]$. Since none of these operations have resource conflict with A_k, this subproblem is independent from the rest of the problem and it can be solved by dynamic programming. We define $Q(i, j, t, t')$ as the minimum cost (necessary) to schedule the sequence of operations $(B_i, ..., B_j)$ in the time interval $[t, t']$. If $\delta_p = 1$, the cost for $P(t, k, p)$ is:

$$\underset{\overline{p}\in V(p, A_k)}{Min} \begin{pmatrix} C(A_k, t) + (1 - \delta_{\overline{p}}) C(B(\overline{p}), t - p_k^A + T(\overline{p})) \\ + P(t - p_k^A, k-1, \overline{p}) \\ + Q(i(\overline{p}) + 1, i(p), t - p_k^A + T(\overline{p}), t) \end{pmatrix} \qquad (3)$$

If $\delta_p = 0$, the cost for $P(t, k, p)$ is:

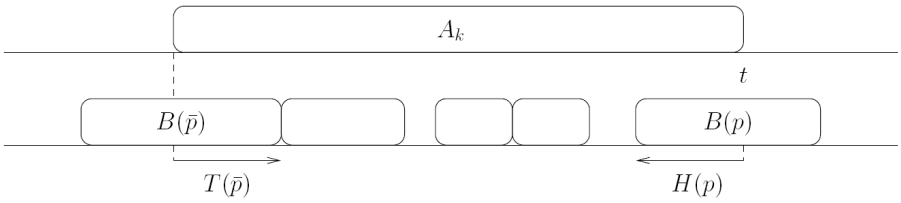

Figure 6. Decomposition of the problem with $H(p) < p_k^A$.

$$\underset{\bar{p}\in V(p,A_k)}{Min}\begin{pmatrix} C(A_k,t)+(1-\delta_{\bar{p}})C(B(\bar{p}),t-p_k^A+T(\bar{p})) \\ +P(t-p_k^A,k-1,\bar{p}) \\ +Q(i(\bar{p})+1,i(p)-1,t-p_k^A+T(\bar{p}),t-H(p)) \end{pmatrix} \quad (4)$$

where $V(p, A_k)$ is the set of possible values for $\bar{p}$: namely it is the largest interval $[p^*, p-H(p)]$ such that $p^* \geq p-p_k^A$ and, for all π in the interval, operation $B(\pi)$ is not to be executed by the machine that runs operation A_k.

The cost for $P(t, k, p)$ when $t = C_k^A$ is then given either by (2), (3) or (4) according to how $H(p)$ compares to p_k^A and if p corresponds to the completion of an operation.

Therefore, we can conclude that $P(t, k, p)$ is equal to the minimum between (1) and one of the three equations (2), (3) or (4).

We finally present the dynamic programming scheme to compute all the values $Q(i, j, t, t')$. For any fixed (i, t), the problem is to find the minimum cost of a sequence of tasks so that we can use the dynamic program proposed by Sourd (2005). For our problem, the recurrence equation becomes:

$$Q(i,j,t,t') = \begin{cases} \infty & \text{if } t+p_j^B > t' \\ Min(Q(i,i,t,t'-1), C(B_j,t')) & \text{if } i=j \\ \underset{t+p_j^B \leq \theta \leq t'}{Min}(Q(i,j-1,t,\theta-p_j^B)+c(B_j,\theta)) & \text{otherwise} \end{cases}$$

The costs $Q(i, j, t, t')$ must be calculated for any $i < j$ and $t \leq t' \leq t + p_{Max}^A$ where $p_{Max}^A = Max_{1\leq i\leq n_A} p_i^A$ is the maximal processing time of an operation of job A. Therefore, $O(n_B^2 p_{Max}^A T)$ values are calculated in $O((n_B p_{Max}^A)^2 T)$ time since the computation of a cost requires $O(p_{Max}^A)$ time.

Similarly, the time complexity to calculate the $O(n_A T p^B)$ values $P(t, k, p)$ is in $O(n_A p_{Max}^A p^B T)$. So the algorithm runs in $O((n_B p_{Max}^A)^2 T + n_A p_{Max}^A p^B T)$ time, which is in $O(n^2 p^2 T)$ with $n = Max(n_A, n_B)$ and $p = Max(p_{Max}^A, p_{Max}^B)$.

6. CONCLUSION

In this paper, we have proposed a new objective criterion to model earliness-tardiness for the 2-jobs Job-shop problem. We have solved the problem in polynomial time. Then, we have proposed a model where each operation incurs a cost which is given in a table. We have also solved this more general problem in polynomial time. However, the complexity of the latter algorithm depends of the size of the table, that is the horizon of the schedule. In particular, if the cost functions $c(X, t)$ are more compactly encoded, for example if $c(X, t)$ represents an earliness-tardiness function, the algorithm is no more polynomial since the size of the input is in $O(n)$. The existence of a polynomial-time algorithm for this problem is an open question.

ACKNOWLEDGMENTS

The authors are indebted to an anonymous referee for helpful suggestions and comments.

REFERENCES

1. Agnetis, A., Mirchandani, P., Pacciarelli, D., and Pacifici, A. (2001). Job-shop scheduling with two jobs and nonregular objectives functions. *INFOR: Information Systems and Operational Research*, 39: 227-244.
2. Akers, S.B. and Friedman, J. (1955). A non numerical approach to production scheduling problems. *Operations Research*, 3: 429-442.
3. Brucker, P. (1988). An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40: 353-359.
4. Brucker, P. (2004). *Scheduling Algorithms*, 4th ed., Springer-Verlag, Berlin.
5. Sourd, F. (2005). Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, 165: 82-96.
6. Sotskov, Y.N. (1991). The complexity of shop scheduling problems with two or tree jobs. *European Journal of Operational Research*, 53: 322-336.