An O(n log n) algorithm for the two-machine flow shop problem with controllable machine speeds

An $O(n \log n)$ algorithm for the two-machine flow shop problem with controllable machine speeds

C.P.M. van Hoesel / Limburg and Eindhoven University;
Department of Economics, P.O. Box 616, 6200 MD Maastricht;
Email: S.vanHOESEL@KE.RULIMBURG.NL

A.P.M. Wagelmans / Erasmus University Rotterdam;
Department of Economics, P.O. Box 1738, 3000 DR Rotterdam;
Email: WAGELMAN@OPRES.FEW.EUR.NL

M. van Vliet / Erasmus University Rotterdam;
Department of Economics, P.O. Box 1738, 3000 DR Rotterdam

August 31, 1995

Abstract

In this paper we consider the two-machine flow shop problem with varying machine speeds. We present an algorithm which determines the optimal permutations for all machine speeds in $O(n \log n)$ time, where n is the number of jobs. To achieve this bound on the running time, the algorithm employs an elementary dominance relation.

Key words: Scheduling, Flow Shop, Algorithm Analysis, Controllable Machine Speeds.

The flow shop problem with controllable machine speeds has been introduced by Ishii et al. [3]. For the case of two machines and n jobs they proposed an algorithm which determines the optimal permutations for all machine speeds in $O(n^2 \log n)$ time. The corresponding problem with fixed machine speeds is solved in $O(n \log n)$ time by the well-known algorithm of Johnson [4]. In this paper we show that the problem with controllable machine speeds can also be solved in $O(n \log n)$ time. To achieve this bound we introduce the notion of *dominance* and the related notion of *potentially critical job*, a generalization of the well-known *critical job*. Priority queues like 2-3 trees are used in the implementation of the algorithm to support operations like searching, deleting and adding elements in $O(\log n)$ time.

Besides the two-machine flow shop problem, other scheduling environments with controllable machine speeds have been considered in the literature. Van Vliet [7] presents a linear time algorithm for the two-machine open shop problem, and Strusevich [6] gives an $O(n^3)$ algorithm for the two-machine flow shop problem with no-wait in process. All the above mentioned papers deal with two-machine environments. If more machines are considered the simple flow shop problem with fixed machine speeds is already NP-hard. Van Vliet [8] derives some worst-case bounds for a class of approximation algorithms for the m-machine flow shop problem with variable machine speeds.

In Section 1 we describe the two-machine flow shop problem with fixed machine speeds, and Johnson's $O(n \log n)$ algorithm for this problem. Furthermore, a dominance relation is introduced and some of its properties are derived. In Section 2 we describe an algorithm to determine the optimal makespan as a function of the speed of one machine. This algorithm makes use of a special set of jobs, the so-called *potentially critical jobs*. Some properties of this set of jobs are derived at the beginning of section 2. Finally, we discuss how the algorithm can be implemented to run in $O(n \log n)$ time. Section 3 contains some concluding remarks.

1 The two-machine flow shop problem

In the standard two-machine flow shop problem, two machines, M_1 and M_2, have to process n jobs. Each job $i \in \{1, \ldots, n\}$ has a non–negative processing time a_i on M_1 and a non–negative processing time b_i on M_2. The processing of job i on M_2 can only start when its processing on M_1 is finished; see figure 1. Finally, the jobs should be processed without preemption on both machines.

The objective is to minimize the makespan C_{max}, i.e., the completion time of the last job on M_2. An optimal strategy to solve the two-machine flow shop problem is due to Johnson [4].

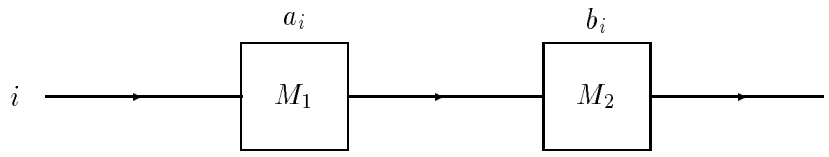

Figure 1: The two-machine flow-shop.

Johnson's algorithm

Partition the job set into two sets L_1 and L_2, where $L_1 := \{i | a_i < b_i\}$ and $L_2 := \{i | a_i \geq b_i\}$. Order the jobs in L_1 according to non–decreasing processing times on M_1. Order the jobs in L_2 according to non–increasing processing times on M_2. The optimal job sequence consists of the jobs in L_1 as ordered above first, and the jobs in L_2 as ordered above second. This order is maintained on both machines. A schedule like this, where the order of the jobs on all machines is the same, is called a *permutation schedule.*

A simple exchange argument proves the optimality of Johnson's rule. The complexity of the algorithm is $O(n \log n)$: partitioning the jobs in L_1 and L_2 requires $O(n)$ time; sorting the jobs in L_1 and in L_2 requires $O(n \log n)$ time.

Throughout this paper we will assume that all processing times are positive, because the general problem can be reduced in $O(n)$ time to this special case, as follows. If an instance contains jobs that have operations with zero processing times, we solve the smaller problem, where these jobs are deleted. Afterwards, the jobs with zero processing time on machine M_1 are added at the beginning of the optimal ordering, and the jobs with zero processing time on machine M_2 are added at the end of the optimal ordering. This ensures an optimal solution for the original problem as can be concluded from the algorithm of Johnson.

The critical job

Let π be a permutation that defines a sequence of the jobs, i.e., $\pi(i)$ is the position of job i in the schedule defined by π. For a job i we define

$$A^{\pi}(0,i) = \sum_{k:\pi(k)\leq\pi(i)} a_k \qquad \text{and} \qquad B^{\pi}(i,n+1) = \sum_{k:\pi(k)\geq\pi(i)} b_k \ .$$

For the computation of C^{π}_{max}, the makespan of the schedule defined by π, it is convenient to introduce the notion of a *critical job.* A job i is called critical with respect to π if $A^{\pi}(0,i) + B^{\pi}(i,n+1)$ is maximum among all jobs. If i is a critical job, then the makespan of the schedule defined by π is (cf. Monma and Rinnooy Kan [5])

$$C^{\pi}_{max} = A^{\pi}(0,i) + B^{\pi}(i,n+1) = \max_j \{A^{\pi}(0,j) + B^{\pi}(j,n+1)\} \ .$$

The minimum makespan C_{max} equals the minimum of C^{π}_{max} over all permutations π, i.e.,

$$C_{max} = \min_{\pi} \max_j \{A^{\pi}(0,j) + B^{\pi}(j,n+1)\} \ .$$

A permutation for which the minimum makespan is attained is called an optimal permutation.

Example

i	1	2	3	4	5	6
a_i	2	4	6	4	4	5
b_i	3	5	7	4	3	2

$L_1 = \{1,2,3\}$, $L_2 = \{4,5,6\}$. An optimal schedule is shown in Figure 2.

In the example job 3 is the critical job and $C_{max} = a_1 + a_2 + a_3 + b_3 + b_4 + b_5 + b_6 = 28$.

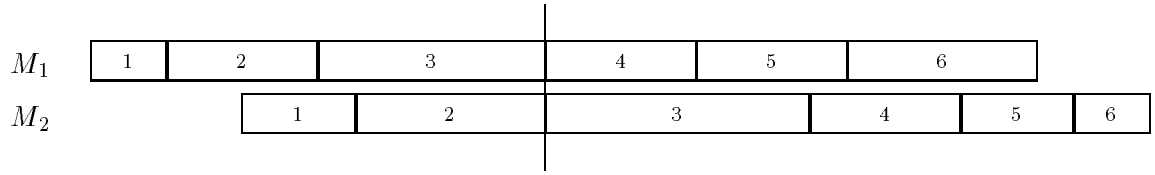

Figure 2: A schedule.

Dominance
The concept of a critical job can be generalized using the more elementary concept of *dominance.* For a pair of jobs i and j, where i precedes j in the ordering defined by π, i.e., $\pi(i) < \pi(j)$, we define

$$A^{\pi}(i,j) = \sum_{k:\pi(i)<\pi(k)\leq\pi(j)} a_k \qquad \text{and} \qquad B^{\pi}(i,j) = \sum_{k:\pi(i)\leq\pi(k)<\pi(j)} b_k \ .$$

Consider two jobs i and j with $\pi(i) < \pi(j)$. Job i is said to dominate job j with respect to π if $A^{\pi}(i,j) < B^{\pi}(i,j)$; if $A^{\pi}(i,j) \geq B^{\pi}(i,j)$, then j is said to dominate i. If i dominates each job in a set S, then i is said to dominate S.

The following propositions give two important properties of the dominance relation. The first proposition describes a structural property, and the second proposition relates the dominance concept to critical jobs.

Proposition 1 *(transitivity)*
Let $i,j,k \in \{1,\ldots,n\}$. If i dominates j and j dominates k, then i dominates k.

Proof. The relative order of $\pi(i)$, $\pi(j)$ and $\pi(k)$ defines six cases. We only prove the cases $\pi(i) < \pi(j) < \pi(k)$ and $\pi(j) < \pi(k) < \pi(i)$. The other cases can be proved analogously.

Case 1: $\pi(i) < \pi(j) < \pi(k)$.
Thus, $A^{\pi}(i,j) < B^{\pi}(i,j)$ and $A^{\pi}(j,k) < B^{\pi}(j,k)$. Now $A^{\pi}(i,k) = A^{\pi}(i,j) + A^{\pi}(j,k) < B^{\pi}(i,j) + B^{\pi}(j,k) = B^{\pi}(i,k)$. Therefore, i dominates k.

Case 2: $\pi(j) < \pi(k) < \pi(i)$.
Thus, $A^{\pi}(j,i) \geq B^{\pi}(j,i)$ and $A^{\pi}(j,k) < B^{\pi}(j,k)$. Now $A^{\pi}(k,i) = A^{\pi}(j,i) - A^{\pi}(j,k) \geq B^{\pi}(j,i) - B^{\pi}(j,k) = B^{\pi}(k,i)$. Therefore, i dominates k.

□

By transitivity, the dominance relation defines a linear ordering on the jobs. Thus, there exists a job that dominates all others. The following proposition shows the importance of this job.

Proposition 2 *If a job dominates all other jobs, then it is a critical job.*

Proof. Let i dominate all other jobs. We have to prove that $A^\pi(0,i) + B^\pi(i,n+1) \geq A^\pi(0,j) + B^\pi(j,n+1)$ for an arbitrary $j \neq i$.

Case 1: $\pi(i) < \pi(j)$: $A^\pi(i,j) < B^\pi(i,j)$.
$A^\pi(0,i) + B^\pi(i,n+1) > A^\pi(0,i) + A^\pi(i,j) - B^\pi(i,j) + B^\pi(i,n+1) = A^\pi(0,j) + B^\pi(j,n+1)$.

Case 2: $\pi(i) > \pi(j)$: $A^\pi(j,i) \geq B^\pi(j,i)$.
$A^\pi(0,i) + B^\pi(i,n+1) \geq A^\pi(0,i) - A^\pi(j,i) + B^\pi(j,i) + B^\pi(i,n+1) = A^\pi(0,j) + B^\pi(j,n+1)$.

□

2 Varying the speed of M_1

In this section we assume that only the speed of M_1 may change. Let a_i denote the processing time of job i on M_1 when this machine runs at its "normal speed". Varying the speed of M_1 causes all processing times on M_1 to change by the same factor, i.e., the processing time of job i becomes αa_i for some $\alpha > 0$. The optimal permutation depends, of course, on the value of α. Let $C_{max}(\alpha)$ denote the makespan of the optimal permutation with respect to α. Hence,

$$C_{max}(\alpha) = \min_{\pi} \max_{i} \{\alpha A^\pi(0,i) + B^\pi(i,n+1)\} \ .$$

One can easily verify (cf. Ishii et al. [3]) that $C_{max}(\alpha)$ is continuous, piecewise linear and monotonically non-decreasing in α. However, in general, $C_{max}(\alpha)$ is neither convex nor concave. Ishii et al. describe an $O(n^2 \log n)$ algorithm to determine all breakpoints of $C_{max}(\alpha)$. Their analysis implies that the number of breakpoints is $O(n^2)$, and they show that the breakpoints can be used to determine optimal machine speeds with respect to a general class of cost functions. In this paper we derive an algorithm that determines the breakpoints of $C_{max}(\alpha)$ for all positive values of α in $O(n \log n)$ time. It is not difficult to see that breakpoints only occur at values of α for which the permutation or the critical job changes. Our analysis implies that the number of breakpoints is $O(n)$. To obtain this time bound we introduce a set of jobs, the so-called potentially critical jobs, that contains the critical job as one of its elements. The improvement in the running time is achieved by using the fact that changes in the set of potentially critical jobs are limited and can be maintained easily during the algorithm.

In Subsection 2.1 we introduce the notion of potentially critical jobs, and we derive some results on the changes that may occur in the set of these jobs. Subsection 2.2 contains a description of the algorithm, which will be proven to be correct in Subsection 2.3. Finally, in Subsection 2.4, we discuss implementation issues.

2.1 Potentially critical jobs

In order to be able to determine the makespan of a schedule it is sufficient to find a critical job, or more precisely, the job that dominates all other jobs. This job may vary for different speed factors. For that reason, we introduce the set of *potentially critical jobs*. This set consists of those jobs that dominate all their successors with respect to a given permutation

π, and a given value of α. It is denoted by P_α^π. By convention, the last job in a sequence, i.e., the job i with $\pi(i) = n$, is potentially critical.

The following lemma characterizes P_α^π. For notational convenience, we define $i_0 = 0$ and $\pi(0) = 0$.

Lemma 3 *Consider a set of jobs $i_1, i_2, \ldots, i_R$ such that $\pi(i_r) < \pi(i_{r+1})$ for $r = 1, \ldots, R-1$, and $\pi(i_R) = n$.*

$P_\alpha^\pi = \{i_1, i_2, \ldots, i_R\}$ if and only if the following dominance relations hold with respect to π and α.

(P1) i_r dominates i_{r+1} for $r = 1, \ldots, R-1$;

(P2) i_r dominates $\{i | \pi(i_{r-1}) < \pi(i) < \pi(i_r)\}$ for $r = 1, \ldots, R$.

Proof. The fact that condition (P1) is necessary follows from the definition of P_α^π as the set of jobs that dominate all their successors. To prove necessity of (P2), suppose it does not hold for some r. Let j with $\pi(i_{r-1}) < \pi(j) < \pi(i_r)$ be the latest job in the permutation π that is not dominated by i_r. From the transitivity of the dominance relation it follows that j dominates all its successors. This is a contradiction to the fact that j does not belong to P_α^π.

Sufficiency of the conditions (P1) and (P2) follows by induction from the transitivity property. Let $\{i_1, i_2, \ldots, i_R\}$ be a set of jobs that satisfies (P1) and (P2), such that $\pi(i_R) = n$. By definition, i_R is potentially critical. Moreover, because of (P2), the jobs between i_{R-1} and i_R are not potentially critical. Next suppose that $\{i_{r+1}, \ldots, i_R\}$ are the potentially critical jobs after i_r in the sequence defined by π. Then, by induction and (P2), job i_{r+1} dominates all the jobs after i_r. Thus, by (P1) and transitivity, i_r dominates all its successors. Finally, the jobs between i_{r-1} and i_r are dominated by i_r since (P2) holds. Thus, these jobs are not potentially critical.

□

Note that if $P_\alpha^\pi = \{i_1, i_2, \ldots, i_R\}$, then i_R is the last job in the sequence determined by π, i.e., $\pi(i_R) = n$. Moreover, using transitivity one can easily show that i_1 dominates all other jobs. Hence, i_1 is a critical job.

Of course, when α varies, the set P_α^π may change. However, as will be shown in the lemmas below, these changes are limited. For a fixed permutation π and each pair of jobs i and j with $\pi(i) < \pi(j)$, we define $\alpha^\pi(i,j) = B^\pi(i,j)/A^\pi(i,j)$. This is the switching point for the dominance relation between i and j, as the following lemma shows.

Lemma 4 *Consider a permutation π. If i and j are two jobs such that $\pi(i) < \pi(j)$, then*

$$i \text{ dominates } j \text{ for } \alpha < \alpha^\pi(i,j) \ ,$$

and

$$j \text{ dominates } i \text{ for } \alpha \geq \alpha^\pi(i,j) \ .$$

Proof. This follows immediately from the definition of dominance: j dominates i if and only if $\alpha A^\pi(i,j) \geq B^\pi(i,j)$.

□

This lemma shows that, for a fixed permutation π, and two values of α, say α_1 and α_2 ($\alpha_1 < \alpha_2$), the set of potentially critical jobs for $\alpha = \alpha_1$ contains the set of potentially critical jobs for $\alpha = \alpha_2$. The following lemma describes at which values of α the set of potentially critical jobs changes and how it changes.

Lemma 5 *Let π be a fixed permutation. Let the set of potentially critical jobs with respect to α_1 be $P^\pi_{\alpha_1} = \{i_1, i_2, \ldots, i_R\}$. Let $\alpha_2 := \min\{\alpha^\pi(i_r, i_{r+1}) \mid r \in \{1, \ldots, R-1\}$ and let $Q = \{i_r | \alpha^\pi(i_r, i_{r+1}) = \alpha_2\}$. Then $P^\pi_\alpha = P^\pi_{\alpha_1}$ for all $\alpha \in [\alpha_1, \alpha_2)$, and $P^\pi_{\alpha_2} = P^\pi_{\alpha_1} \backslash Q$.*

Proof. The first part is easily verified using Lemma 4: (P1) and (P2) continue to hold for $\alpha \in [\alpha_1, \alpha_2)$, since the dominance relations of the jobs mentioned in (P1) and (P2) do not change.

It remains to prove that $P^\pi_{\alpha_2} = P^\pi_{\alpha_1} \backslash Q$. From Lemma 4, it follows that $P^\pi_{\alpha_2} \subseteq P^\pi_{\alpha_1}$. Furthermore, by definition, the jobs in Q are not potentially critical for α_2, i.e., $Q \cap P^\pi_{\alpha_2} = \emptyset$. Hence, we only need to show that the jobs in $P^\pi_{\alpha_1} \backslash Q$ are in $P^\pi_{\alpha_2}$. Suppose this is not the case and let i_s be the largest indexed job that belongs to $P^\pi_{\alpha_1} \backslash Q$, but not to $P^\pi_{\alpha_2}$. Note that $i_s \neq i_R$, because $i_R = n$ trivially belongs to $P^\pi_{\alpha_2}$. We have $\alpha^\pi(i_s, i_{s+1}) > \alpha_2$, or equivalently, $\alpha_2 A^\pi(i_s, i_{s+1}) < B^\pi(i_s, i_{s+1})$.

Among the jobs in $P^\pi_{\alpha_2}$ with index higher than s, let i_t be the one with the smallest index. Note that i_t is well–defined, because $i_R \in P^\pi_{\alpha_2}$. Also note that, by its choice, i_t dominates each job i with $\pi(i_s) < \pi(i) < \pi(i_t)$.

We are going to prove that i_s dominates i_t. Using transitivity this implies that i_s dominates all its successors. Hence, i_s is potentially critical, which is a contradiction.

If $t = s + 1$, then it follows from Lemma 4 and $\alpha^\pi(i_s, i_{s+1}) > \alpha_2$ that the statement holds. So suppose $t > s + 1$, then all potentially critical jobs between s and t are in Q. Therefore, $\alpha^\pi(i_r, i_{r+1}) = \alpha_2$ for all $s < r < t$. This implies $\alpha_2 A^\pi(i_{s+1}, i_t) = B^\pi(i_{s+1}, i_t)$. We now have $\alpha_2 A^\pi(i_s, i_{s+1}) + \alpha_2 A^\pi(i_{s+1}, i_t) < B^\pi(i_s, i_{s+1}) + B^\pi(i_{s+1}, i_t)$. Hence, i_s dominates i_t.

Concluding, we have that $P^\pi_{\alpha_2} = P^\pi_{\alpha_1} \backslash Q$.

□

In the remainder of this subsection we present results concerning the effect of changes of the permutation π on the set of potentially critical jobs.

We view each change of position of a job as a deletion of that job from a certain position and the subsequent insertion of that job into another position. We will analyze these two parts (deletion and insertion) separately. However, the following lemma is relevant for both parts.

Lemma 6 *Consider a job k and let α be such that $b_k = \alpha a_k$. Let $\tilde{\pi}$ and π be permutations of $\{1, \ldots, k-1, k+1, \ldots, n\}$ and $\{1, 2, \ldots, n\}$, respectively, such that $\tilde{\pi}$ differs from π only in the omission of k.*

Let i and j be two jobs not equal to k. Then the dominance relations of i and j with respect to the two permutations are equal.

Proof. Consider an arbitrary pair $i, j \neq k$ with $\pi(i) < \pi(j)$. If job k is not between i and j in permutation π, then clearly $A^{\tilde{\pi}}(i,j) = A^{\pi}(i,j)$ and $B^{\tilde{\pi}}(i,j) = B^{\pi}(i,j)$, and the statement holds. So suppose $\pi(i) < \pi(k) < \pi(j)$. We show that the values $\alpha^{\pi}(i,j) - \alpha$ and $\alpha^{\tilde{\pi}}(i,j) - \alpha$ have the same sign, which is equivalent to the statement in the lemma. Using $\alpha a_k = b_k$, we obtain

$$\begin{aligned}
\alpha^{\pi}(i,j) - \alpha &= \tfrac{1}{A^{\pi}(i,j)} \left[B^{\pi}(i,j) - \alpha A^{\pi}(i,j)\right] \\
&= \tfrac{1}{A^{\pi}(i,j)} \left[B^{\tilde{\pi}}(i,j) + b_k - \alpha(A^{\tilde{\pi}}(i,j) + a_k)\right] \\
&= \tfrac{1}{A^{\pi}(i,j)} \left[B^{\tilde{\pi}}(i,j) - \alpha A^{\tilde{\pi}}(i,j)\right] \\
&= \tfrac{A^{\tilde{\pi}}(i,j)}{A^{\pi}(i,j)} \left[\alpha^{\tilde{\pi}}(i,j) - \alpha\right] \ .
\end{aligned}$$

Since both $A^{\pi}(i,j)$ and $A^{\tilde{\pi}}(i,j)$ are positive, the claim follows.

□

The lemma below refers to the deletion step.

Lemma 7 *Consider a job k and let $\alpha = b_k/a_k$. Let $\tilde{\pi}$ and π be permutations of $\{1, \ldots, k-1, k+1, \ldots, n\}$ and $\{1, 2, \ldots, n\}$, respectively, such that $\tilde{\pi}$ differs from π only in the omission of k, and $\pi(k) \neq n$. Let l be the successor of k in π, i.e., $\pi(l) = \pi(k) + 1$, and suppose that $a_k \leq a_l$. Then $P_{\alpha}^{\tilde{\pi}} = P_{\alpha}^{\pi} \backslash \{k\}$.*

Proof. Let $P_{\alpha}^{\pi} = \{i_1, i_2, \ldots, i_R\}$ with $\pi(i_r) < \pi(i_{r+1})$ for $r = 1, \ldots, R-1$. First, suppose that $k \notin P_{\alpha}^{\pi}$. Then, by Lemma 6, the relations (P1) and (P2) are satisfied for all i_r $(r = 1, \ldots, R)$ and $P_{\alpha}^{\tilde{\pi}} = P_{\alpha}^{\pi}$.

Now suppose that $k \in P_{\alpha}^{\pi}$, and let t be such that $k = i_t$. Because $\pi(k) \neq n$ it follows that $t < R$. By Lemma 6, it holds for each pair of jobs that the dominance relation with respect to $\tilde{\pi}$ is the same as the dominance relation with respect to π. The only relations to be validated are that i_{t+1} dominates the jobs in $I = \{i | \pi(i_{t-1}) < \pi(i) < \pi(k)\}$ with respect to $\tilde{\pi}$. If that is the case, then (P2) holds for i_{t+1} and thus $P_{\alpha}^{\pi} \backslash \{k\}$ contains exactly the potentially critical jobs with respect to $\tilde{\pi}$, i.e., $P_{\alpha}^{\tilde{\pi}}$.

From $a_k \leq a_l$ it follows that l dominates each job $i \in I$, because $A^{\tilde{\pi}}(i,l) \geq A^{\pi}(i,k)$ and $B^{\tilde{\pi}}(i,l) = B^{\pi}(i,k)$. The desired result now follows, since either $i_{t+1} = l$ or i_{t+1} dominates l with respect to $\tilde{\pi}$, and dominance is transitive.

□

For insertion we have the following lemma.

Lemma 8 *Consider a job k and let α be such that $b_k = \alpha a_k$. Let $\tilde{\pi}$ and π be permutations of $\{1, \ldots, k-1, k+1, \ldots, n\}$ and $\{1, 2, \ldots, n\}$, respectively, such that $\tilde{\pi}$ differs from π only in the omission of k, and $\pi(k) \neq 1$. Let l be the predecessor of k in π, i.e., $\pi(l) = \pi(k) - 1$ and suppose that $b_k < b_l$. Then P_α^π is either $P_\alpha^{\tilde{\pi}}$ or $P_\alpha^{\tilde{\pi}} \cup \{k\}$.*

Proof. Let $P_\alpha^{\tilde{\pi}} = \{i_1, i_2, \ldots, i_R\}$ with $\pi(i_r) < \pi(i_{r+1})$ for $r = 1, \ldots, R-1$. Note that l dominates k, since $\alpha a_k = b_k < b_l$.

If $\pi(k) = n$, then $i_R = l$. Hence, i_R dominates k with respect to π. It follows from Lemma 3, Lemma 6 and the transitivity property that $P_\alpha^{\tilde{\pi}} \cup \{k\}$ is the set of potentially critical jobs with respect to π, i.e., P_α^π.

Now suppose that $i_u \in P_\alpha^{\tilde{\pi}}$ is such that $\{i | \pi(i_{u-1}) < \pi(k) < \pi(i_u)\}$.

If i_u dominates k with respect to π, then it follows again from Lemma 3, Lemma 6 and the transitivity property that $P_\alpha^\pi = P_\alpha^{\tilde{\pi}}$.

If k dominates i_u, then we claim that $P_\alpha^\pi = P_\alpha^{\tilde{\pi}} \cup \{k\}$. Since l dominates k, we have that l dominates i_u with respect to π. Because of Lemma 6 we have that l also dominates i_u with respect to $\tilde{\pi}$. Hence, using $\pi(k) \neq 1$, we have $l = i_{u-1}$ and $u > 1$. Furthermore, the set $\{i | \pi(i_{u-1}) < \pi(i) \leq \pi(k)\}$ contains only k. Using (P1) and (P2) it can easily be verified that $P_\alpha^\pi = P_\alpha^{\tilde{\pi}} \cup \{k\}$.

□

Note that the lemmas in this subsection do not assume anything about optimality of the respective permutations π and $\tilde{\pi}$.

The results above will be used in Subsection 2.3, where we prove the correctness of our algorithm, which is described in the next subsection.

2.2 Description of the algorithm

We assume that the jobs are numbered such that

$$\frac{b_1}{a_1} \geq \frac{b_2}{a_2} \geq \ldots \geq \frac{b_n}{a_n} \ .$$

Moreover, let the permutations ρ and σ be such that

$a_{\rho(1)} \leq a_{\rho(2)} \leq \ldots \leq a_{\rho(n)}$, and if $a_{\rho(j)} = a_{\rho(j+1)}$ then $\rho(j) < \rho(j+1)$ $(j \in \{1, \ldots, n-1\})$;

$b_{\sigma(1)} \geq b_{\sigma(2)} \geq \ldots \geq b_{\sigma(n)}$, and if $b_{\sigma(j)} = b_{\sigma(j+1)}$ then $\sigma(j) < \sigma(j+1)$ $(j \in \{1, \ldots, n-1\})$.

Note that this amounts to sorting the numbers b_i/a_i, a_i and b_i, and breaking ties properly, which takes $O(n \log n)$ time. As a result of the numbering of the jobs, for any given α, the sets L_1 and L_2 are, respectively, $\{1, \ldots, k\}$ and $\{k+1, \ldots, n\}$, where k is such that $\frac{b_k}{a_k} > \alpha \geq \frac{b_{k+1}}{a_{k+1}}$. The ordering in L_1 and L_2 can be deduced from ρ and σ, respectively. In the sequel, when referring to *the* optimal permutation, we will mean this particular permutation.

In our algorithm, we let α increase from a suitable small value to a suitable large value. Along the way, we keep track of the optimal permutation and the corresponding set of potentially critical jobs.

The range $[\alpha_{min}, \alpha_{max}]$ of interesting values of α is

$$\left[\frac{\min_{i=1,\ldots,n}\{b_i\}}{\max_{i=1,\ldots,n}\{a_i\}}, \frac{\max_{i=1,\ldots,n}\{b_i\}}{\min_{i=1,\ldots,n}\{a_i\}}\right] .$$

For values of α smaller than α_{min} the optimal permutation is ρ, since $L_1 = \{1, \ldots, n\}$ and $L_2 = \emptyset$. Moreover, each job dominates its successor, and therefore $P_\alpha^\rho = \{\rho(1), \ldots, \rho(n)\}$ and $\rho(1)$ is the critical job. For values of α larger than α_{max} the optimal permutation is σ, $L_1 = \emptyset$, $L_2 = \{1, \ldots, n\}$ and $P_\alpha^\sigma = \{\sigma(n)\}$. Thus, P_α^σ contains only the last job in the permutation σ.

Initialization and stopping criterion
We start with α equal to α_{min}. Then we increase α gradually until α_{max} is reached.

The main part of the algorithm consists of a sequence of iterative steps. At the beginning of such a step we know, for the current value of α, the optimal permutation and the corresponding set P_α^π of potentially critical jobs. In the iterative step we increase α to the smallest value for which the optimal permutation or P_α^π changes, and we update the optimal permutation and P_α^π. This updating process may consist of several *minor iterations*. For example, this is the case when several jobs are deleted from L_1 and subsequently inserted into L_2 at the same value of α. Such simultaneous "jumps" are handled sequentially. In each minor iteration one job jumps and we update the permutation and P_α^π. Hence, only at the end of the iterative step we obtain the optimal permutation.

When referring to P, $A(i,j)$, $B(i,j)$ or $\alpha(i,j)$ in the description below, i.e., when we drop the superscript indicating the permutation and the subscript denoting α, these values refer to the current permutation and the current α. Furthermore, P denotes the current set of potentially critical jobs, which is always assumed to consist of $i_1, i_2, \ldots, i_R$, such that $\pi(i_{r-1}) < \pi(i_r)$ for $r = 2, \ldots, R$, where π is the current permutation. For convenience, we let S_r denote the set $\{i | \pi(i_{r-1}) < \pi(i) \leq \pi(i_r)\}$ $(r = 1, \ldots, R)$. This set contains the jobs that are scheduled between i_{r-1} and i_r, where i_{r-1} is not in S_r whereas i_r does belong to the set.

Iterative step
Let π denote the optimal permutation with respect to the current value of α.
Let k be maximal such that $\frac{b_k}{a_k} > \alpha$. Thus, $L_1 = \{1, \ldots, k\}$ and $L_2 = \{k+1, \ldots, n\}$.

Increase α to $\min\{\frac{b_k}{a_k}, \min\{\alpha(i_r, i_{r+1}) | r = 1, \ldots, R-1\}\}$.

For all $r \in \{1, \ldots, R-1\}$ for which $\alpha = \alpha(i_r, i_{r+1})$ delete i_r from P and merge the sets S_r and S_{r+1}.

Move all jobs j with $\alpha = \frac{b_j}{a_j}$ from L_1 to L_2. We will describe this step in detail for k. If k is not the only job that has to be moved, the step should be repeated for the other jobs in order of decreasing index. Job k is added to L_2 at the position where it precedes exactly those jobs $l \in L_2$ with $b_l \leq b_k$.

Denote the permutation after k is moved from L_1 to L_2 by π'. Note that it may happen that k moves from L_1 to L_2 without actually changing its position in the permutation, i.e., $\pi(k) = \pi'(k)$. In that case P remains unchanged. Otherwise, we first delete k from the permutation π and then reinsert it at its new position.

First phase: k is deleted from the permutation π
Let $t \in \{1, \ldots, R\}$ be such that $k \in S_t$.
If $k \neq i_t$, then we only delete k from the set S_t.
If $k = i_t$, then k is deleted from both S_t and P; if $t < R$, the sets $S_t \backslash \{k\}$ and S_{t+1} are merged; if $k = i_1$, then i_2 becomes the critical job.

Second phase: k is reinserted into the permutation $\tilde{\pi}$
If $\pi'(k) = n$, then we only add k to P.
Otherwise, suppose that $\pi'(i_{u-1}) < \pi'(k) < \pi'(i_u)$ for some $u \in \{1, \ldots, R\}$. If i_u dominates k, then we only add k to S_u; if k dominates i_u, then the set $S_u \cup \{k\}$ is split into two sets: $\{i | \pi'(i_{u-1}) < \pi'(i) \leq \pi'(k)\}$ and $\{i | \pi'(k) < \pi'(i) \leq \pi'(i_u)\}$, and k is added to P.

Everytime the set of potentially critical jobs changes, certain values have to be updated.

- If $k \neq i_t$ leaves S_t, then $A(i_{t-1}, i_t)$ and $B(i_{t-1}, i_t)$ are decreased by a_k and b_k, respectively.
- If $k \neq i_u$ enters S_u, then $A(i_{u-1}, i_u)$ and $B(i_{u-1}, i_u)$ are increased by a_k and b_k, respectively.
- If i_t leaves S_t then the sets S_t and S_{t+1} are merged, and $A(i_{t-1}, i_{t+1}) := A(i_{t-1}, i_t) + A(i_t, i_{t+1}) - a_{i_t}$ and $B(i_{t-1}, i_{t+1}) := B(i_{t-1}, i_t) + B(i_t, i_{t+1}) - b_{i_t}$.
- If k enters S_u, and k dominates i_u, then S_u is partitioned into the two sets $\{i | \pi'(i_{u-1}) < \pi'(i) \leq \pi'(k)\}$ and $\{i | \pi'(k) < \pi'(i) \leq \pi'(i_u)\}$. Thus, $A(i_{u-1}, k)$, $B(i_{u-1}, k)$, $A(k, i_u)$ and $B(k, i_u)$ should be calculated.
- If i_1, the critical job, changes or if job k jumps from a position before i_1 to a position after i_1, then the values $A(0, i_1)$ and $B(i_1, n+1)$ must be calculated; this update is needed to keep track of the value of $C_{max}(\alpha)$.

Note that in each case a constant number of the values $A(i, j)$ and $B(i, j)$ must be (re)calculated. How these calculations are implemented efficiently will be shown in Subsection 2.4.

2.3 Correctness of the algorithm

It is obvious that at the end of an iterative step we obtain, for the increased value of α, the optimal permutation. Hence, we only have to show that the set of potentially critical jobs is updated correctly. First note that if the optimal permutation does not change in the iterative step, then it follows from Lemma 5 that P is correctly updated. Now suppose that the optimal permutation does change. It is important to note that, if necessary, P is first updated with respect to the old optimal permutation π. From that point on the permutation will change several times, depending on how many jobs jump from L_1 to L_2. Each jump results in the deletion of a job from the permutation followed by the insertion of that job into the permutation. After each of these changes, we update P with respect to the permutation at hand. Using Lemmas 7 and 8, we are now going to prove that these updates are correct.

Suppose the permutation changes from π to π' because job k jumps from L_1 to L_2. Hence, $b_k = \alpha a_k$ and $\pi'(k) > \pi(k)$. Each of the two phases (deletion and insertion) will be analyzed separately. Therefore, in addition to π and π' we will also consider the permutation $\tilde{\pi}$ that results from π after deleting k.

First phase. Deletion of k from the permutation π.
Note that $\pi(k) \neq n$, because $\pi'(k) > \pi(k)$. Hence, the correctness follows from Lemma 7 if we can show that $a_k \leq a_l$, where l is the immediate successor of k in π. If $l \in L_1$, then this follows from the definition of L_1. If $l \in L_2$, then l is the first job in L_2 and k is the last job in L_1. Since $\pi'(k) > \pi(k)$, it follows that $b_k \leq b_l$. Moreover, since $l \in L_2$ we have $b_l \leq \alpha a_l$, and thus $\alpha a_k = b_k \leq b_l \leq \alpha a_l$. This implies $a_k \leq a_l$.

Second phase. Insertion of k into the permutation $\tilde{\pi}$.
Because $\pi'(k) > \pi(k)$, it holds that $\pi'(k) \neq 1$. Therefore, correctness follows from Lemma 8 if we can show that $b_k < b_l$, where l is the immediate predecessor of k in π'. If $l \in L_2$ then $b_l > b_k$, otherwise k is inserted into L_2 before l. If $l \in L_1$, then it is the last job in L_1 (since $k \in L_2$). It follows that $\pi(k) < \pi(l)$ since $\pi' \neq \pi$, and therefore $a_k \leq a_l$. Since $l \in L_1$, we also have $\alpha a_l \leq b_l$. If at least one of these inequalities is strict, then the desired result follows, since $\alpha a_k = b_k$. Otherwise, we have $a_k = a_l$ and $\alpha a_l = b_l$. Because of the definition of the permutation ρ and the fact that k preceded l in L_1, the first equality implies $k < l$. However, jobs that jump at the same value of α are handled in order of decreasing index. Hence, $k > l$, a contradiction.

From the analysis above we conclude that P is updated correctly every time the permutation changes. It follows that at the end of the iterative step, we obtain the set of potentially critical jobs corresponding to the new optimal permutation.

We will now turn to the issue of implementing the algorithm efficiently.

2.4 Complexity and data structures

As mentioned before, every iterative step consists of one or several minor iterations. In each minor iteration either one job is deleted from P while the permutation remains the same, or one job jumps from L_1 to L_2 and P is updated accordingly. We have seen that $|P|$ can only increase if the permutation changes. Moreover, in a certain minor iteration only the job that jumps from L_1 to L_2 may be added to P. Hence, $2|L_1| + |P|$ decreases by at least one in every minor iteration. Since $2|L_1| + |P| \leq 3n$ at the beginning of the algorithm, the total number of minor iterations is bounded by $3n$.

The data structures are chosen such that the amount of work per minor iteration is $O(\log n)$. From the previous description of the algorithm, the reader can check that the following operations must be performed a constant number of times per minor iteration.

(a) Calculate $A(i,j)$ and $B(i,j)$ for given i and j.

(b) Calculate $\alpha(i,j)$ for given i and j.

(c) For a given job k find $i_{r-1}, i_r \in P$ such that $\pi(i_{r-1}) < \pi(k) \leq \pi(i_r)$.

(d) Add a job to P or delete a job from P.

(e) Find $i_r \in P \backslash \{i_1\}$, with r minimal, such that $\alpha(i_{r-1}, i_r)$ is minimal.

We keep track of the current permutation by storing it implicitly in two 2-3 trees T_1 and T_2. This data structure supports the operations SEARCH, ADD and DELETE in $O(\log n)$

time, n being the number of leaves (cf. [1]). The trees T_1 and T_2 facilitate the execution of (a) and (b) as follows. Both trees contain n leaves, numbered 1 to n, and each node has two labels. When $i \in L_1$, the first label of the leaf numbered $\rho(i)$ in T_1 is equal to a_i. The first labels of the other leaves in T_1 are 0. Intermediate nodes in T_1 have a first label equal to the sum of the first labels of the leaves in the subtree rooted at the node. Analogously, a leaf $\sigma(i)$ in T_2 has first label a_i when $i \in L_2$, otherwise the label is 0. With these labels the value $A(j,k)$ can be calculated in $O(\log n)$ time for given j and k. Thus, $A(i_r, i_{r+1})$ for $i_r \in P \backslash \{i_R\}$ can be calculated within this time bound. The $B(j,k)$ values can be calculated analogously by storing the values b_i in the same trees, using the second labels of the nodes. Therefore, the value $\alpha(i_r, i_{r+1})$ can be calculated in $O(\log n)$ time for any given i_r and i_{r+1}. Finally, updating T_1 and T_2, when a given job k jumps from L_1 to L_2, is easily seen to take $O(\log n)$ time also.

The data structures used for the execution of (c), (d) and (e) are as follows. Consider the pairs $(i_r, \alpha(i_r, i_{r+1}))$ for $i_r \in P$. We define three 2-3 trees which contain these pairs as their leaves. One 2-3 tree is used to store the pairs ordered according to the values $\alpha(i_r, i_{r+1})$. A second tree contains the pairs for which $i_r \in L_1 \cap P$, ordered according to a_{i_r}, the processing times on M_1. Analogously, the third tree contains the pairs for which $i_r \in L_2 \cap P$, ordered according to b_{i_r}, the processing times on M_2. This suffices to perform (c), (d) and (e) in $O(\log n)$ time.

The following theorem states our main result.

Theorem 9 *The breakpoints of the piece-wise linear function $C_{max}(\alpha)$ ($\alpha \in (0, \infty)$) can be determined in $O(n \log n)$ time.*

3 Concluding remarks

In Section 2 the speed of M_2 was fixed. We may introduce a speed factor β for this machine as well. The problem is then to minimize a function $f(\alpha, \beta, C_{max})$. However, $C_{max}(\alpha, \beta)$ has the same shape for any fixed β as follows from the formula

$$C_{max}(\alpha, \beta) = \beta \left\{ \min_{\pi} \max_{i} \{\delta A^{\pi}(0,i) + B^{\pi}(i, n+1)\} \right\} \quad ,$$

where $\delta = \alpha/\beta$. Intuitively this is clear: speeding up both machines by the same factor reduces C_{max} by the same factor. Ishii et al. [3] show that this property can be used to find the optimal speeds for cost functions of the form

$$f(C_{max}, v_1, v_2) = w_1 C_{max}^{q_1} + w_2 v_1^{q_2} + w_3 v_2^{q_2} \qquad (w_1, w_2, w_3 > 0, q_1, q_2 \geq 1) \ .$$

The complexity of the algorithm to determine optimal speeds and the corresponding optimal schedule with respect to this class of objective functions has now been reduced to the same complexity as Johnson's algorithm. Sorting is an essential part of Johnson's algorithm. In fact, it can be shown that any algorithm that solves the two-machine flow shop correctly in $O(f(n))$ time, is capable of sorting numbers in $O(f(n))$ time. Therefore, in any comparison based model, one may not hope for any improvement on the running time of the algorithm.

A similar result has been proved by Van Vliet [7] for the two-machine open shop scheduling problem. For the no-wait flow shop the complexity gap lies between $n \log n$ and n^3. The $O(n \log n)$ time bound for the original problem is proved by Gilmore et al. [2], whereas the time bound for the problem with speed-up of machines is given by Strusevich [6]. There is evidence that the gap between the running times of the constant speed problem and the variable speed problem can be reduced. However, it is an open problem whether the gap can be closed.

Acknowledgement
We would like to thank an anonymous referee whose detailed comments on several earlier versions of this paper have led to immense improvements of the presentation.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, 1983. *Data structures and algorithms*, Addison Wesley.

[2] P.C. Gilmore, E.L. Lawler, and D.B. Shmoys, 1985. Well-solved special cases, in *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds), Wiley, New York.

[3] H. Ishii, T. Masuda, and T. Nishida, 1987. Two machine mixed shop scheduling problem with controllable machine speeds, *Discrete Applied Mathematics* 17, 29–38.

[4] S.M. Johnson, 1954. Optimal two- and three stage production schedules with setup times included, *Naval Research Logistics Quarterly* 1, 61–68.

[5] C.L. Monma and A.H.G. Rinnooy Kan, 1983. A concise survey of efficiently solvable special cases of the permutation flow shop problem, *RAIRO Recherche Operationnelle* 17, 105–119.

[6] V.A. Strusevich, 1991. Two machine flow shop scheduling problem with no wait in process: controllable machine speeds, report 9122/A, Econometric Institute, Erasmus University Rotterdam, The Netherlands (to appear in *Discrete Applied Mathematics*).

[7] M. Van Vliet, 1991. Optimization of Manufacturing System Design, PhD thesis, Econometric Institute, Erasmus University Rotterdam, The Netherlands.

[8] M. Van Vliet, 1993. A multi-machine flow shop problem with controllable processing times: worst-case and robustness results, report 9309/A, Econometric Institute, Erasmus University Rotterdam, The Netherlands (to appear in *European Journal of Operational Research*).