Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Operations Research Letters 31 (2003) 219–224
Operations Research Letters
www.elsevier.com/locate/dsw

The two-machine open shop problem: To fit or not to fit, that is the question

Marjan van den Akker[a], Han Hoogeveen[a,*], Gerhard J. Woeginger[b]

[a] Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands
[b] Department of Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Received 30 September 2002; received in revised form 28 October 2002; accepted 4 November 2002

Abstract

We consider the open shop scheduling problem with two machines. Each job consists of two operations, and it is prescribed that the first (second) operation has to be executed by the first (second) machine. The order in which the two operations are scheduled is not fixed, but their execution intervals cannot overlap. We are interested in the question whether, for two given values D_1 and D_2, there exists a feasible schedule such that the first and second machine process all jobs during the intervals $[0, D_1]$ and $[0, D_2]$, respectively.

We formulate four simple conditions on D_1 and D_2, which can be verified in linear time. These conditions are necessary and sufficient for the existence of a feasible schedule. The proof of sufficiency is algorithmical, and yields a feasible schedule in linear time. Furthermore, we show that there are at most two non-dominated points (D_1, D_2) for which there exists a feasible schedule.

Keywords: Scheduling; Sequencing; Open shop scheduling; Flow shop scheduling; Bicriterion optimization

1. Introduction

We consider the following machine scheduling problem: There are two machines M_1 and M_2 that can process at most one job at a time. Machine M_i $(i = 1, 2)$ is continuously available from time zero to time D_i, where D_1 and D_2 are two given integers. The machines have to process a given job set $\mathcal{J}$ that consists of n jobs $J_1, \ldots, J_n$. Each job J_j consists of two operations, one of which has to be processed by machine M_1, which requires an uninterrupted time period of length a_j, and the other one by M_2, which takes an uninterrupted time period of length b_j. The order in which the two operations should be executed is not prescribed, but the operations of one job are not allowed to overlap in their execution. In the literature such a scheduling environment is known as a two-machine *open shop*, and it is encoded by the entry $O2$ in the first field of the three-field notation scheme of Graham et al. [3]. The open shop can be viewed upon as a relaxation of the *flow shop* environment, in which each job first has to visit M_1 and afterwards M_2; this machine environment is encoded by the entry $F2$ in the first field.

For both the open shop and flow shop problem, there is only one standard optimization problem known that can be solved to optimality in polynomial

* Corresponding author.
E-mail addresses: marjan@cs.uu.nl (M. van den Akker), slam@cs.uu.nl (H. Hoogeveen), g.j.woeginger@math.utwente.nl (G.J. Woeginger).

0167-6377/03/$ - see front matter
doi:10.1016/S0167-6377(03)00018-X

time: Minimizing the makespan, that is, minimizing the time at which the last job is completed. These two optimization problems are denoted by $O2\|C_{\max}$ and $F2\|C_{\max}$. Problem $O2\|C_{\max}$ is solvable through the algorithm by Gonzalez and Sahni [2], which runs in $O(n)$ time. Problem $F2\|C_{\max}$ is solvable through the famous algorithm developed by Johnson [1], which runs in $O(n\log n)$ time. As we will heavily use the algorithm of Gonzalez and Sahni, we will state and recall this algorithm in Section 2. Sloppily speaking, in $O2\|C_{\max}$ one has $D_1 = D_2 = D$ and one wants to determine the minimal value D for which there is a feasible schedule.

In this paper, we are interested in the general feasibility checking problem where D_1 is not necessarily equal to D_2. A schedule that meets both machine deadlines D_1 and D_2 will be called a *schedule that fits*. This feasibility checking problem is closely related to optimization problems, where the goal is to minimize some objective function $f(D_1, D_2)$ and where the values D_1 and D_2 are not given a priori, but have to be determined. The makespan criterion can then be modeled through $f(D_1, D_2) = \max\{D_1, D_2\}$. This approach has been adopted by Shakhlevich and Strusevich [4]. They present eleven schedules which all can be constructed in $O(n)$ time, and they show that for any regular function $f(D_1, D_2)$ this set contains at least one optimal schedule for both the preemptive and the nonpreemptive case. As a consequence of the results in Section 5, this set of eleven schedules can be replaced by a much smaller set with only two schedules.

This paper is organized as follows. In Section 2, we repeat the algorithm by Gonzalez and Sahni. This algorithm is a prerequisite for Section 3 in which we formulate four conditions on D_1 and D_2 that are necessary for the existence of a schedule that fits. These four conditions can be checked in $O(n)$ time. In Section 4, we will show that these conditions are also sufficient, and that the corresponding schedules can be computed in $O(n)$ time. In Section 5, we argue that for each two machine open shop instance there exist at most two non-dominated (or Pareto optimal) points (D_1, D_2) for which there exists a feasible schedule meeting D_1 and D_2. Moreover, we will show how these Pareto optimal points can be determined in $O(n)$ time. Section 6 contains a brief discussion.

2. Gonzalez and Sahni's algorithm

In this section we describe Gonzalez and Sahni's algorithm that solves $O2\|C_{\max}$. Gonzalez and Sahni start with the following three lower bounds on the minimum makespan:

- the total processing time $\sum a_j$ of all jobs on M_1;
- the total processing time $\sum b_j$ of all jobs on M_2; and
- the maximum total processing time $\max(a_j + b_j)$ per job.

Subsequently, they present an $O(n)$ algorithm that finds a feasible schedule with a makespan that is equal to the maximum of the three lower bounds; hence, this makespan is minimal. Their algorithm is as follows.

Gonzalez and Sahni's Algorithm

Step 0: Set $q \leftarrow \operatorname{argmax}(a_j + b_j)$. If $a_q + b_q \geqslant \max\{\sum a_j, \sum b_j\}$, then schedule J_q on M_1 (M_2) in the interval $[0, a_q]$ ($[a_q, a_q + b_q]$) and schedule the remaining jobs in $[a_q, \sum a_j]$ on M_1 and in $[0, \sum b_j - b_q]$ on M_2. Stop.

Step 1: Define $A = \{J_j | a_j \geqslant b_j\}$ and $B = \{J_j | a_j < b_j\}$. Choose any two distinct jobs J_l and J_r such that

$$a_r \geqslant \max_{J_j \in A} b_j \quad \text{and} \quad b_l \geqslant \max_{J_j \in B} a_j.$$

Let $A' = A \setminus \{J_l, J_r\}$ and $B' = B \setminus \{J_l, J_r\}$.

Step 2: If $\sum a_j - a_l \geqslant \sum b_j - b_r$, then construct a feasible schedule where:

- M_1 first executes J_l, then all jobs from B' in any order, all jobs from A' in any order, and finally J_r.
- M_2 executes the jobs in the order J_r, J_l, all jobs from B' in any order, and finally all jobs from A' in random order. The execution intervals are as follows:
  - if $a_r \leqslant \sum b_j - b_r$, then both machines process the jobs contiguously (not necessarily starting at time zero), such that on both machines the last operation ends at time $\max\{\sum a_j, \sum b_j\}$;
  - if $a_r > \sum b_j - b_r$, then M_1 is continuously busy processing in the entire interval $[0, \sum a_j]$, whereas M_2 is idle until time

$\sum a_j - a_r - b_r$ and then processes the jobs contiguously.

Step 3: If $\sum a_j - a_l < \sum b_j - b_r$, then construct a feasible schedule where:

- M_1 starts with all jobs from B' in any order, then all jobs from A' in any order, J_r, and finally J_l;
- M_2 first executes J_l, then all jobs from B' in any order, then the jobs from A' in any order, and finally J_r. The execution intervals are as follows:
  - if $a_l \leqslant \sum b_j - b_l$, then both machines process the jobs contiguously (not necessarily starting at time zero), such that on both machines the last operation ends at time $\max\{\sum a_j, \sum b_j\}$;
  - if $a_l > \sum b_j - b_l$, then M_1 is continuously busy processing in the entire interval $[0, \sum a_j]$, whereas M_2 is idle until time $\sum a_j - a_l - b_l$ and then processes the jobs contiguously.

Theorem 1. *Gonzalez and Sahni's algorithm finds a feasible schedule with makespan equal to the maximum of the three given lower bounds.*

3. The four necessary conditions

In this section we consider the problem of determining for two given values D_1 and D_2 whether there exists a feasible schedule such that M_1 (M_2) can process all tasks in the interval $[0, D_1]$ ($[0, D_2]$). We present necessary and sufficient conditions for D_1 and D_2. We present an algorithm that constructs in $O(n)$ time a feasible schedule that meets D_1 and D_2 given that D_1 and D_2 obey the conditions.

The case $D_1 = D_2$ boils down to the problem of minimizing the makespan, which is solved through Gonzalez and Sahni's algorithm presented in the previous section. We proceed with the case that $D_1 \neq D_2$. Since the role of the machines is interchangeable (there is no prescribed ordering in the execution of the operations), we assume without loss of generality that $D_1 < D_2$. Gonzalez and Sahni's lower bounds immediately lead to three necessary Conditions 1–3 on D_1 and D_2: We find that D_1 and D_2 must satisfy

$$\sum_{j=1}^{n} a_j \leqslant D_1. \tag{1}$$

$$\sum_{j=1}^{n} b_j \leqslant D_2. \tag{2}$$

$$\max_{1 \leqslant j \leqslant n} (a_j + b_j) \leqslant D_2. \tag{3}$$

Unfortunately, Conditions 1–3 are not sufficient for the existence of a feasible schedule. Consider the following two-job example where both jobs have processing time 1 on machine M_1 and processing time 2 on M_2. The combination $D_1 = 2$ and $D_2 = 4$ satisfies all conditions, but there is no feasible schedule that fits. It is easily checked that, either $(D_1, D_2) \geqslant (3, 4)$ or $(D_1, D_2) \geqslant (2, 5)$ is required for a fitting feasible schedule. This example leads to the following lower bound on D_2 for a given value D_1. Define S as the subset of $\mathscr{J}$ that contains all jobs J_j with $a_j + b_j > D_1$. In any feasible schedule, these jobs must be executed by M_1 first and then by M_2. Ignoring the jobs that are not in S, we obtain a two-machine flow shop problem, which can be solved through Johnson's algorithm in $O(n \log n)$ time. Since the jobs in S constitute a special instance, we can solve it in $O(n)$ time, however. The specialty of the instance lies in the fact that we can complete each job on M_1 at time $\sum a_j \leqslant D_1$, whereas the first job in any feasible schedule is completed after time D_1 on M_2 by definition of S. Hence, we only care about selecting the right job to be processed first, after which we can process the remaining jobs in any order on M_1 and M_2 without any idle time. If we start with some job J_0, then we find a schedule with makespan equal to $a_0 + \sum_{j \in S} b_j$; hence, we must select the job J_j in S with minimum a_j to be processed first. This gives us the following Condition 4 on D_2:

$$D_2 \geqslant \min_{J_j \in S(D_1)} a_j + \sum_{J_j \in S(D_1)} b_j \equiv F^*(D_1). \tag{4}$$

We have added the argument $S(D_1)$ to show the dependence between S and D_1. If $S(D_1) = \emptyset$, then $F^*(D_1) = 0$. In our example above, we have for $D_1 = 2$ that $S(2) = \{J_1, J_2\}$ and we find $F^*(2) = 5$. If $D_1 = 3$, then $S(3) = \emptyset$ and hence $F^*(3) = 0$.

4. The proof of sufficiency

In this section we will show that Conditions 1–4 not only are necessary but sufficient as well. We present an algorithm that constructs a schedule in which M_1 and

M_2 are finished at times D_1 and D_2 if the conditions are satisfied. The basis is formed by adapting the schedule determined by Gonzalez and Sahni's algorithm.

Suppose that we are given values D_1 and D_2 that satisfy Conditions 1–4. Without loss of generality, we assume that we cannot decrease D_1 or D_2 without violating at least one of these conditions. We will design an algorithm that constructs a feasible schedule meeting D_1 and D_2. The algorithm breaks up the instance set in a set of categories, and for each category we describe how a feasible algorithm can be found.

We start with instances in which there exists a job J_q with $a_q+b_q \geqslant \max\{\sum a_j, \sum b_j\}$. This corresponds to Step 0 in Gonzalez and Sahni's algorithm, and the cure is the same: we process job J_q in $[0, a_q]$ on M_1 and in $[a_q, a_q+b_q]$ on M_2, after which we schedule the remaining jobs in the remaining gaps. The feasibility of this schedule follows immediately from Conditions 1–3.

We proceed as described in Gonzalez and Sahni's algorithm and find the jobs J_l and J_r. Suppose that $\sum a_j - a_l \geqslant \sum b_j - b_r$, which leads us to Step 2 of their algorithm. We increase a_r by $D_2 - D_1$ (we denote the adjusted a_j-values by $\bar{a}_j$) and construct the schedule as described by the algorithm. In both cases M_1 and M_2 finish processing at time $\max\{\sum \bar{a}_j, \sum b_j\} \leqslant D_2$ and J_r is processed last on M_1. If we replace the processing time $\bar{a}_r$ by a_r, then we have a feasible schedule meeting D_1 and D_2.

Now suppose that $\sum a_j - a_l < \sum b_j - b_r$, which leads us to Step 3 of Gonzalez and Sahni's algorithm. If $a_l + b_l \leqslant D_1$, then Gonzalez and Sahni's algorithm finds a schedule in which M_2 completes its jobs at time $\sum b_j$, which is no more than D_2. Moreover, M_1 completes its jobs at time $\max\{a_l + b_l, \sum a_j\}$, which is no more than D_1 by assumption. Hence, we are done, unless $a_l + b_l > D_1$. For this case, we need Condition 4.

Let S be a subset of $\mathscr{J}$ that satisfies the following conditions:

1. S contains all jobs J_j with $a_j + b_j > D_1$;
2. Solving $F2\|C_{\max}$ for S yields a makespan that is smaller than or equal to D_2.

We start with the subset S that contains only the jobs J_j with $a_j+b_j > D_1$; as we have just observed, $J_l \in S$. We construct our schedule through an iterative process, where we augment S in each iteration until we have a feasible schedule that fits. We use $a(S)$ and $b(S)$ as a short hand notation for the total processing time of the jobs in S on M_1 and M_2, respectively. We first check whether $a(S) + b(S) \geqslant D_2$. If this is the case, then we can find a feasible schedule by processing the jobs in S in random order but starting with the one with minimum a_j-value among the jobs with $a_j + b_j > D_1$ in the intervals $[0, a(S)]$ and $[D_2 - b(S), D_2]$ on M_1 and M_2, after which we put the remaining jobs in the gaps. If this is not the case, then we check whether there exists a job J_v with $J_v \notin S$ such that $b_v \geqslant a(S)$. If J_v exists, then we construct a feasible schedule in the following way:

- M_1 starts at time zero with the jobs in S followed by all jobs not in S except for J_v; finally J_v is executed in the interval $[D_1 - a_v, D_1]$;
- M_2 starts at time zero with J_v, then executes the jobs in S, and finally the remaining jobs.

This schedule is feasible, as $a_v + b_v \leqslant D_1$, since $J_v \notin S$, and the first one of the remaining jobs starts at time $b_v + b(S) \geqslant a(S) + b(S) > D_1$ on M_2. If such a job J_v does not exist, then we construct the following, currently infeasible schedule. On M_1, we first process the jobs in S and then the other jobs in arbitrary order in the interval $[0, \sum a_j]$; on M_2 we process the jobs in the same order as on M_1 in the interval $[a(S), a(S) + \sum b_j]$. If $a(S) + \sum b_j \leqslant D_2$, then we have discovered a feasible schedule, since $a(S) + b(S) > D_1$, which implies that the jobs that do not belong to S do not overlap in their execution. If $a(S) + \sum b_j > D_2$, then we check whether there exists a job J_w that is started before time D_2 and completed after time D_2. If there is no such job J_w, then we move the part scheduled in $[D_2, a(S) + \sum b_j]$ to the interval $[0, a(S)]$ on M_2. Since $D_2 \geqslant \sum b_j$, we know that this fits, and hence we are done, as there is no overlap.

Suppose that we face the unlucky event that J_w does exist. We denote the set of jobs that are scheduled in between S and J_w by E and the jobs that succeed J_w by T. We adjust our schedule on M_2 by moving J_w and the jobs in T forward in front of the jobs in S, such that J_w starts at time zero and is followed by the jobs in T, whose relative order remains unchanged. If necessary, the jobs in S and E are shifted to the right; see Fig. 1 for a schematic illustration. Since

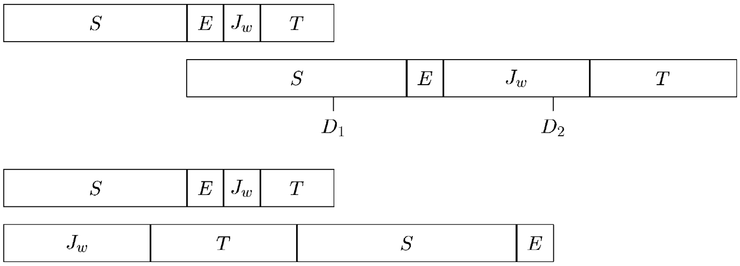

Fig. 1. Augmenting S.

$a(S) > b_w$ (otherwise J_w would have qualified as the job J_v), the only jobs that may overlap are the jobs in T. Obviously, this can only be the case if the last job in T is completed on M_2 after the first job in T is started on M_1, that is, $b_w + b(T) > a(S) + a(E) + a_w$. Hence, if the current schedule is not feasible, then we must have that $a(S) + a_w < \sum b_j - b(S)$, from which we deduce that $a_w + a(S) + b(S) < D_2$. Since $b_w < a(S)$, we find that the makespan of the optimum schedule for $F2\|C_{\max}$ applied to the job set $\{J_w\} \cup S$ is smaller than D_2. Hence, we can augment S by J_w and apply the same analysis again. This either leads to a feasible schedule meeting D_1 and D_2 or suggests a job that can be added to S. As the number of jobs is bounded, we will eventually find a feasible schedule that fits. Using an appropriate datastructure this can be implemented to run in $O(n)$ time.

Theorem 2. *There exists a feasible schedule for the two machine open shop problem in which machines M_1 and M_2 are finished at times D_1 and D_2 if and only if D_1 and D_2 satisfy Conditions* 1–4.

5. Finding the non-dominated points (D_1, D_2)

Our example in Section 3 shows that there can be two points (D_1, D_2) that lead to a feasible schedule, which are incomparable to each other. In this section we show that there are at most two such points and that they can be determined in O(n) time. The properties of these D_1 and D_2 values depends on which of the lower bounds (1)–(4) is tight. We partition the instances in the following way, where as before $q = \mathrm{argmax}_{1 \leqslant j \leqslant n}\{a_j + b_j\}$:

- $a_q + b_q \geqslant \max\{\sum a_j, \sum b_j\}$;
- $a_q + b_q \leqslant \min\{\sum a_j, \sum b_j\}$;
- $\max\{\sum a_j, \sum b_j\} > a_q + b_q > \min\{\sum a_j, \sum b_j\}$.

In the first subcase, we have that $\max\{D_1, D_2\} = a_q + b_q$. Due to the lower bounds (1) and (2), we arrive at the points $(\sum a_j, a_q + b_q)$ and $(a_q + b_q, \sum b_j)$. A fitting schedule is easily derived in both cases. In the first case, we execute J_q in the intervals $[0, a_q]$ and $[a_q, a_q + b_q]$ on M_1 and M_2 and subsequently put the remaining operations in the gaps. A feasible schedule for $(a_q + b_q, \sum b_j)$ is derived in a similar fashion.

The second subcase is even easier: the point $(\sum a_j, \sum b_j)$ is clearly as small as possible in both components, and it is easily checked that in this case it satisfies Conditions 1–4. Hence, there is only one point of interest now.

In the third subcase, we assume without loss of generality that $\sum b_j > a_q + b_q > \sum a_j$. We first check whether $(\sum a_j, \sum b_j)$ satisfies Conditions 1–4. If this is the case, then this is the only non-dominated point. Otherwise, we must have that

$$\sum b_j < \min_{J_j \in S(\sum a_j)} a_j + \sum_{J_j \in S(\sum a_j)} b_j \equiv F^*\left(\sum a_j\right).$$

Hence, we find that the first non-dominated point is $(\sum a_j, F^*(\sum a_j))$. To be able to arrive in another non-dominated point, we must increase the D_1-value such that the set $S(D_1)$ becomes smaller. The smallest D_1-value for which $S(D_1)$ changes is

$$\min_{J_j \in S(\sum a_j)} (a_j + b_j) \equiv Q.$$

But if D_1 is put equal to Q, then we can find a feasible schedule in which M_2 has finished all jobs at time $\sum b_j$ in the following way: If J_0 is the job that leads to Q, then execute J_0 in the intervals $[0, b_0]$ and $[b_0, a_0 + b_0]$ on M_2 and M_1, and put the remaining jobs into the gaps. Since the minimum D_2-value has been determined, there is no need to increase the D_1-value any further, which implies that $(\sum a_j, F^*(\sum a_j))$ and $(Q, \sum b_j)$ are the only non-dominated points for the third subcase if $F^*(\sum a_j) > \sum b_j$.

Theorem 3. *For any instance of the two machine open shop problem, there exist at most two Pareto optimal points* (D_1, D_2) *for which there exists a feasible schedule meeting* D_1 *and* D_2.

6. Conclusions

We have shown that there are at most two non-dominated points (D_1, D_2) for the open shop problem with two machines, and that these points can be determined in $O(n)$ time together with the schedule that realizes these values.

An interesting question occurs when we look at the number of such non-dominated points in case of $m \geqslant 3$ machines. It is easy to construct instances with $m!$ non-dominated points: There is only one job with unit processing time operations on each machine; each permutation of $1, \ldots, m$ corresponding to the order in which the operations are executed leads to a non-dominated point. *We conjecture that this bound is tight*. Observe, however, that since the problem $O3\|C_{\max}$ is already $\mathcal{NP}$-hard in the ordinary sense, there is not much hope to compute all non-dominated points in polynomial time.

Acknowledgements

The authors thank an anonymous referee for his/her comments. The first and second author were partially supported by EC Contract IST-1999-14186 (Project ALCOM-FT).

References

[1] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Res. Logist. Quart. 1 (1954) 61–68.

[2] T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, J. Assoc. Comput. Mach. 25 (1976) 92–101.

[3] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.

[4] N.V. Shakhlevich, V.A. Strusevich, Two machine open shop scheduling problem to minimize an arbitrary machine usage regular penalty function, European J. Oper. Res. 70 (1993) 391–404.