ELSEVIER

Discrete Applied Mathematics 114 (2001) 273–288

DISCRETE APPLIED MATHEMATICS

Linear time approximation scheme for the multiprocessor open shop problem

S.V. Sevastianov[b,*,1], G.J. Woeginger[a,2]

[a] *Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria*
[b] *Sobolev Institute of Mathematics, Novosibirsk-90 630090, Russia*

Abstract

For the r-stage open shop problem with identical parallel machines at each stage and the minimum makespan criterion, an approximation scheme is constructed with running time $O(nrm+C(m,\varepsilon))$, where n is the number of jobs, m is the total number of machines, and $C(m,\varepsilon)$ is a function independent of n.

Keywords: Scheduling; Open shop; Approximation scheme

1. Introduction

Scheduling problems, like the majority of discrete optimization problems, are traditionally attributed to the class of hard tractable problems. Efficient optimization algorithms can normally be constructed for such problems only in simplest cases, when the values of key parameters (such as the number of jobs or the number of machines) are small enough. At the same time, different attempts to extend these results to problems with greater values of those parameters run across an insuperable wall of NP-hardness. That is why we have to renounce the idea of finding the optimal solutions and confine ourselves to searching for approximate ones. But in these cases too, as some recent results for different optimization problems indicate, we run across the so called "approximability threshold", which is different for different problems. For some of them, a polynomial-time approximation scheme (PTAS) exists such that for any fixed $\varepsilon > 0$, a polynomial-time $(1+\varepsilon)$-approximation algorithm $\mathscr{A}_\varepsilon$ can be constructed. (Its running time naturally depends on ε, but for any fixed value of ε it is polynomial on the length of the input.) For other problems and some constants C, C-approximation algorithms can be constructed (with the performance ratio $f(S^{\mathscr{A}})/f(S^*) \leqslant C$, where f is an objective function, S^* is an optimal solution, and $S^{\mathscr{A}}$ is a solution delivered by

* Corresponding author.
E-mail addresses: seva@math.nsc.ru (S.V. Sevastianov), gwoegi@opt.math.tu-graz.ac.at (G.J. Woeginger).
[1] Supported by the Russian Foundation for Fundamental Research (Grant 99-01-00601).
[2] Supported by the START program Y43-MAT of the Austrian Ministry of Science.

0166-218X/01/$ - see front matter
PII: S0166-218X(00)00375-9

the algorithm $\mathscr{A}$), but at the same time, there is no approximation scheme. In such a case, there exists an "approximability threshold", i.e., a constant C such that for any $C' < C$ the existence of a C'-approximation algorithm would imply P = NP. Finally, for problems of the third type, no C-approximation algorithm exists for any constant C, unless P = NP. (For those problems, a further complexity classification is possible. However, we will not discuss these questions here).

Naturally, the very first question normally being raised in the complexity analysis of an NP-hard problem is to determine the complexity class to which the problem belongs. Below, we are interested in this question in connection with a fixed number of machines m and scheduling problems. For the majority of them this question is still open. While numerous C-approximation algorithms have already been constructed for many such problems, for a few scheduling problems only approximation schemes have been found or non-existence of such schemes has been shown. In the area of multi-stage scheduling problems, the open shop problem with the minimum makespan criterion is one of such exceptions. (Using the standard problem classification of Lawler et al. [1], this problem is written as $O_m||C_{\max}$.) In [4] a version of a PTAS for the $Om||C_{\max}$ problem was described, which for any fixed m and ε has running time $O(n \log n)$. On the other hand, it was shown in [6] that there is no such a PTAS if m is not fixed (unless P = NP), because the constant $C = \frac{5}{4}$ is shown to be an "approximability threshold" for this problem. Therefore, a precise separating line between approximable cases (when m is fixed) and non-approximable cases (when m is not fixed) was drawn. There still remains an open question about the existence of a *fully polynomial time approximation scheme* (FPTAS) for the $Om||C_{\max}$ problem, i.e., a scheme whose running time would be polynomial in $1/\varepsilon$.

While the scheme constructed in [4] for the $Om||C_{\max}$ problem provides the answer to an open question of the complexity theory, it does not yield a really efficient means of getting solutions arbitrarily close to the optimum for real-life instances of the problem. Indeed, the additive constant included in the bound on running time of that scheme depends on the running time of constructing the optimal schedule for the so-called "big" jobs. The number of such jobs is bounded above by a double exponential function of $1/\varepsilon$, namely, by $m(m/\varepsilon)^{2^{m/\varepsilon}}$. It is clear that this is a huge amount even for small values of m and $1/\varepsilon$. Thus, the way of $(1+\varepsilon)$-approximating a problem with n jobs proposed by our scheme degenerates to an enumeration algorithm of finding an optimal solution if $n \leqslant m(m/\varepsilon)^{2^{m/\varepsilon}}$. In view of the NP-hardness of the open shop problem, such enumeration algorithms apparently have the running time at least exponential in n. (As an exercise, we offer the reader to take $m=3$, $\varepsilon=\frac{1}{3}$ and try to estimate for which n the scheme does not yield an efficient means even for the $\frac{4}{3}$-approximation of the $O3||C_{\max}$ problem. We compare with this particular problem, because a $\frac{4}{3}$-approximation algorithm with running time linear in n and without huge additive constants is known for this problem [3]).

In the present paper, another version of the approximation scheme is proposed. A new way of dividing the whole set of jobs into subsets of "large", "medium", and "small" jobs enables us, firstly, to reduce considerably the number of large jobs to an "ordinary" exponential function $3.5 \times 2^{m/\varepsilon}$. And secondly, using a "semi-greedy"

scheme of completing a schedule for medium and small jobs (instead of the greedy scheme used in PTAS from [4]), we can replace $O(n \log n)$ by $O(n)$ and get rid of a factor at n depending on ε. Thus, our new scheme is "almost" an FPTAS (except for an additive constant in the bound on running time, which is exponential in m/ε).

Instead of the standard $Om||C_{\max}$ problem, we will apply our scheme to a more general multi-processor r-stage $Or(Pm)||C_{\max}$ problem, where at each stage there is a limited number of identical parallel processors. It is still assumed that the total number m of processors is bounded by a constant.

The remaining part of the paper is organized as follows. Section 2 contains a formal setting of the problem and auxiliary results. In Section 3 we describe a scheme $\mathscr{A}_{GC}$ of a greedy completing of an arbitrary partial schedule; the scheme requires time linear in n (instead of n^2 time required by the direct implementation of the idea of a greedy completing of a schedule). In Section 4, a description and analysis of the approximation scheme is presented. It is specified there how the scheme of the greedy completing should be modified so as to get rid of the factor at n depending on ε. Section 5 contains some thoughts concerning possible ways of further improving the scheme.

2. Problem setting, notation, auxiliary results

In a multi-processor r-stage open shop system, there are n jobs $\{J_1,\dots,J_n\}$ and r shops $\mathscr{M}_1,\dots,\mathscr{M}_r$; the ith shop contains m_i identical machines (processors) $M_v \in \mathscr{M}_i$; $m=\sum m_i$ is the total number of machines in all shops. Job J_j, $1 \leqslant j \leqslant n$, consists of r operations $o_1^j,\dots,o_r^j$. Operation o_i^j can be processed by any machine $M_v \in \mathscr{M}_i$ in p_i^j time units. At any time, every job can be processed by at most one machine and every machine can process at most one job. For each job, the order in which its operations have to be processed is *not* fixed in advance but may be chosen arbitrarily; different jobs may get different orders. In processing an operation, preemption is not allowed. To specify the processing of the operations $\mathscr{O}=\{o_i^j \mid j=1,\dots,n;\ i=1,\dots,r\}$ according to a schedule, we should define two functions: $M:\mathscr{O}\to\{M_1,\dots,M_m\}$ and $S:\mathscr{O}\to\mathbb{R}^+$, where $M(o_i^j)$ assigns a processing machine to the operation o_i^j, while $S(o_i^j) \doteq s_i^j$ specifies a starting time for this operation. The schedule is called *feasible* if it satisfies the above requirements. The goal is to compute a schedule with minimal length, i.e., a schedule S with the minimum completion time $C_{\max}(S)$ of the last operation.

Let $C^*_{\max}$ stand for the optimum of the problem, let $L_i=\sum_{j=1}^n p_i^j$ be the total machine load of the ith shop, $\hat{L}_i = L_i/m_i$ be the average machine load of the ith shop, $\hat{L}_{\max} = \max_{i=1,\dots,r}\hat{L}_i$, $d_j=\sum_{i=1}^r p_i^j$ be the length of job J_j, and $d_{\max}=\max_j d_j$. It is clear that for the $O(P)||C_{\max}$ problem, the parameters $\hat{L}_{\max}$ and $d_{\max}$ represent lower bounds on the optimum, i.e.,

$$C^*_{\max} \geqslant \max\{\hat{L}_{\max},\ d_{\max}\}. \tag{1}$$

It was shown in [2] that any greedy algorithm $\mathscr{A}_G$ (including the algorithms completing the empty schedule by the $\mathscr{A}_{GC}$ scheme) enables one to construct a *dense*

schedule S for the $O(P)||C_{\max}$ problem such that

$$C_{\max}(S) \leqslant \hat{L}_{\max} + d_{\max}. \tag{2}$$

Therefore, it follows from (1) and (2) that $\mathscr{A}_G$ is an approximation algorithm of solving the $O(P)||C_{\max}$ problem with the worst-case performance ratio of 2.

3. An $\mathscr{A}_{GC}$ scheme of a greedy completing a schedule for a multi-processor open shop system

Unlike under the scheme $\mathscr{A}_{GP}$ of constructing a dense feasible schedule described in [5], we will not spend time on maintaining permanent priority orders on the sets of operations of each job and each machine, which enables us to decrease the running time of the algorithm. On the other hand, constructing of a schedule will be performed under the more complicated conditions that there already is a partial schedule. (In the scheme $\mathscr{A}_{GC}$ below it is assumed that the set of operations for which a partial schedule is known may be arbitrary.) This yields additional constraints on possible ways of constructing the desired schedule and requires additional calculation. However, the new scheme makes it possible to keep the low running time of the algorithm. In the next section, this scheme is applied to complete a partial schedule specified for so called "large" jobs.

Now, let us start with a detailed description of the $\mathscr{A}_{GC}$ scheme (where "GC" is an abbreviation of "greedily completing"). Let a partial schedule be specified for a given open shop system, i.e., for some operations $\{o_i^j\}$ (that will be called "old") their processing machines $M(o_i^j)$ and starting times $\{s_i^j\}$ are assumed to be known. Our goal is to assign a processing machine and a starting time to each of the remaining ("new") operations (this will be called "to schedule an operation") without changing the existing schedule so as to construct a complete feasible schedule as short as possible. It is sensible to fill in the idle gaps between the old operations, and we do it by the following greedy-type algorithm.

The current (partial) schedule will be stored in two $r \times n$ matrices: matrices M and s will specify the machine processing the operation o_i^j ($1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant n$) and the starting time of the operation, respectively. (If an operation o_i^j is not scheduled yet, we set $M(o_i^j) = nul$.) Furthermore, all currently scheduled operations will be stored as a "list of operations" $\mathscr{L}$. It is assumed that a direct access to any item of the list $\mathscr{L}$ is possible (which requires O(1) time), if its absolute address is known. Also, at each item of the list $\mathscr{L}$ the addresses of some other items of the list will be stored. More precisely, for each operation o_i^j in the list $\mathscr{L}$, we store the index j of its job J_j, the index of its processing machine $M(o_i^j)$, the type of the operation ("old" or "new"), its completion time $c(o_i^j)$, the addresses of the previous and the next operations of job J_j in the list $\mathscr{L}$, the addresses of the previous and the next operations of machine $M(o_i^j)$ in the list $\mathscr{L}$, and finally, the addresses of the previous and the next items of the list $\mathscr{L}$ by key $c(o_i^j)$.

In the course of the algorithm, the list $\mathscr{L}$ is scanned once by key c; at some times new operations are scheduled and added to the list $\mathscr{L}$. The current (being scanned) operation of the list $\mathscr{L}$ is called an *actual* operation. The scanning of the list is accompanied by a spasmodic increasing of the value of a parameter τ that is called a *current time*. The value of τ always coincides with the completion time of the actual operation. The nearest (by key c) operation of job J_j (machine M_i) amongst all operations of the list $\mathscr{L}$ currently not scanned is called a *current operation* of job J_j (machine M_i). (It does not matter, whether this operation is being processed at time τ or it is not started yet.) The nearest (by key c) not scanned old/new operation in the list $\mathscr{L}$ is shortly called *the nearest old/new operation*. The addresses of both operations are stored in the course of the algorithm, as well as the arrays $\Phi[1\ldots n]$ and $\hat{j}[1\ldots m]$ of addresses of the current operations of all jobs J_j $(j=1,\ldots,n)$ and all machines M_i $(i=1,\ldots,m)$.

The "new" operations that are already scheduled (and included in the list $\mathscr{L}$) but not yet scanned are called "semi-definite". The information about these operations is stored as a "heap" $\mathscr{H}$ semi-ordered by key c. The latter means that: (1) there is a one-to-one correspondence between the operations and the nodes of a rooted tree; (2) the operation with the smallest value of the key is stored in the root node of the tree; (3) a non-decreasing order of the key values along every path passing from the root to a leaf of the tree is maintained; 4) the length of every such path is maintained to be at most $O(\log n_{\mathscr{H}})$, where $n_{\mathscr{H}}$ is the number of nodes of the tree. At each node of the heap two parameters are permanently stored: the address of the operation in the list $\mathscr{L}$ and the value of the key. The "semi-definiteness" of an operation in the list $\mathscr{L}$ means that for this item of the list all its parameters are defined except for the following two: the references to the previous and to the next items of the list (by key c). Both references become defined after scanning the operation, i.e., after its having been the actual operation.

At each step of the algorithm all new operations currently not scheduled are stored in three families of lists: the lists A_i of operations *allowed* to be scheduled on machine M_i $(i=1,\ldots,m)$, and the lists R_j^J $(j=1,\ldots,n)$ and R_i^M $(i=1,\ldots,m)$ of operations of job J_j (machine M_i) currently not allowed to be scheduled. An operation o' gets into a "job-motivated" ("machine-motivated") list R_j^J (R_i^M) of "not allowed" operations after removing it from a list A_i because of its unavoidable overlap with the current operation of job J_j (current old operation of machine M_i) in the case of starting it at the current time. For each operation o' transferred from a list A_i (of machine M_i) to a list R_j^J, the machine $M(o')=M_i$ is memorized. (This is not necessary for the operations transferred to a list R_i^M, because all operations in the list are taken from the list A_i.) Since the algorithm works on a greedy principle, a machine M_i can be idle during some period of time only when the list A_i is empty for the values of τ from that time period. (This implies that machine M_i cannot be idle at time t if the list A_i corresponding to the value of the parameter $\tau=t$ is not empty.)

The algorithm consists of a "zero" step (or the step of *initialization*) and a cycle of scanning (item by item) the list $\mathscr{L}$ in non-decreasing order of the key c.

At the *Initialization* step, initial values of the lists and variables are defined. Set $\tau:=0$. The list $\mathscr{L}$ initially contains old operations only. Then m additional "old" operations $\tilde{o}_1,\ldots,\tilde{o}_m$ corresponding to dummy jobs $J_{n+1},\ldots,J_{n+m}$ are added to the list $\mathscr{L}$. These operations are processed by machines $M_1,\ldots,M_m$, respectively, and get completion times $c(\tilde{o}_i)=0$. All new operations of each shop $\mathscr{M}_v$ $(v=1,\ldots,r)$ are in the list $\mathscr{O}_v$. For each machine $M_i\in\mathscr{M}_v$, we set $A_i:=\mathscr{O}_v$. (Hence, every operation $o_v^j\in\mathscr{O}_v$ is presented in m_v independent lists A_i.) The lists R_j^J $(j=1,\ldots,n)$, R_i^M $(i=1,\ldots,m)$, and $\mathscr{H}$ are initially empty. Finally, the addresses of the current operations of all jobs (array $\Phi[1\ldots n]$) and all machines (array $\hat{j}[1\ldots m]$), as well as the address of the nearest old operation are found. To get all this information, it is sufficient to scan the whole list $\mathscr{L}$ by key c only once.

The *cycle* of scanning the list $\mathscr{L}$ consists of steps; at each step only one operation in the list (called an *actual* operation) is being scanned. Every step starts with defining a new actual operation which has to be chosen from among two operations: the nearest old and the nearest new. The address of the latter can be found at the root node of the heap $\mathscr{H}$. (At the very first step, when the heap is still empty, the choice is restricted to a single operation: the nearest old one.) The value of τ is changed to the completion time of the current actual operation. If an old operation was chosen, it immediately stops being "the nearest old operation"; for this role, the next (by key c) operation in the list $\mathscr{L}$ is chosen. Alternatively, if a new operation was chosen as an actual one, then first of all we complete defining the remaining two parameters of this item, namely, the references to the previous and to the next operations in the list $\mathscr{L}$. (As such, the previous actual and the nearest old operation are being chosen; at the same time, the references of these two operations are corrected, too.) Next, the new actual operation is removed from the root of the heap $\mathscr{H}$, and the heap is reordered (which can be done in $O(\log n_{\mathscr{H}})$ time, where $n_{\mathscr{H}}$ is the maximum possible cardinality of the heap $\mathscr{H}$). At the same time, a new "nearest new" operation is found. Furthermore, the actual operation stops being the current operation of both its job and its machine, these titles are passed over to the next (in the list $\mathscr{L}$) operation of that job and to the next operation on that machine, respectively.

Furthermore, at each step (at most) two new operations are scheduled so as to be started at time τ. This is performed by two procedures called "job assignment" and "machine load". Let an actual operation $o_{k'}^{j'}$ be processed on machine $M_{i'}$.

Job assignment is intended for scheduling at time τ another (not scheduled yet) operation of job $J_{j'}$. The search for this operation is performed in the list $R_{j'}^J$. (There is no point to use for this purpose the lists A_i, because if a list A_i is not empty, machine M_i is not idle according to the above remark.) The list $R_{j'}^J$ is scanned, and for each operation $o'\in R_{j'}^J$ the following steps are performed.

- If o', being started at time τ, overlaps with the current operation of machine $M_i=M(o')$ and this current operation is new, then o' is transferred to the list A_i.
- If o' overlaps with the current operation of machine $M_i=M(o')$ and this operation is old, then o' is transferred to the list R_i^M.

- If o' does not overlap with the current operation of machine $M_i = M(o')$ but overlaps with the current operation of job $J_{j'}$, then o' remains in the list $R^J_{j'}$.
- Finally, if o' overlaps with neither the current operation of machine $M(o')$ nor the current operation of job $J_{j'}$, then o' is scheduled so as to start at time τ, and the following operations are performed:
  - o' is removed from the list $R^J_{j'}$;
  - o' is assigned to be the current operation of job $J_{j'}$ and machine $M(o')$, and the corresponding references to the previous and to the next operations in both paths are defined in the list $\mathscr{L}$;
  - the completion time of the operation is defined: $c(o') := \tau + p(o')$, where $p(o')$ is the processing time of the operation o';
  - o' is placed to the list $\mathscr{L}$ and to the heap $\mathscr{H}$, the necessary reordering of the heap being performed in time $O(\log n_{\mathscr{H}})$.

The procedure terminates as soon as either some operation from the list $R^J_{j'}$ is scheduled or the list $R^J_{j'}$ is completely scanned. Thus, clearly, at most one operation from the list $R^J_{j'}$ can be scheduled during the current step.

Machine load is intended for scheduling at time τ another (not scheduled yet) operation of machine $M_{i'}$. At first, it is checked whether the actual operation is old, and if yes, then the whole contents of the list $R^M_{i'}$ is transferred to the list $A_{i'}$. After this (both in the case of an old actual operation and a new one) the list $A_{i'}$ is scanned, and for each operation $o^j_k \in A_{i'}$ the following steps are performed.

- If o^j_k is already scheduled (which is possible due to a relative independence of lists $\{A_i\}$), then o^j_k is removed from the list $A_{i'}$.
- If o^j_k overlaps with the current operation of machine $M_{i'}$ (clearly, the current operation is "old"), then o^j_k is transferred from $A_{i'}$ to $R^M_{i'}$.
- If o^j_k overlaps with the current operation of job J_j, then o^j_k is transferred from $A_{i'}$ to R^J_j.
- If o^j_k does not overlap with both operations, then it is scheduled so as to start at time τ, and all steps necessary for a just scheduled operation are performed (see their description in the Job assignment procedure). The scanning of the list $A_{i'}$ is terminated.

This completes the description of the algorithm $\mathscr{A}_{GC}$.

Lemma 1. *Suppose that for an m-processor r-stage open shop system, a partial schedule S is given, namely, it is defined for v_k operations of shop $\mathscr{M}_k$ $(k = 1, \ldots, r)$ and it should be defined for the remaining n_k (new) operations of shop $\mathscr{M}_k$. Suppose also that all information about the schedule S is given as required in the description of algorithm $\mathscr{A}_{GC}$. Then algorithm $\mathscr{A}_{GC}$ completes the initial partial schedule for all new operations in time $O(\sum_{k=1}^{r} n_k(rm_k + v_k + \log n_{\mathscr{H}}))$, where $n_{\mathscr{H}}$ is the maximum possible number of new operations that can be performed simultaneously.*

Proof. At every step of the algorithm, we are first looking for a new actual operation, and after it is found, apply the procedures "job assignment" and "machine load". In

the case that the nearest old operation is taken for an actual one, this requires only O(1) time. Alternatively, if the nearest new operation is taken for an actual one, we need $O(\log n_{\mathscr{H}})$ time to reorder the heap $\mathscr{H}$. Hence, the total (over all steps) time required for choosing the actual operations is no more than $O(\sum_{k=1}^{r}(v_k + n_k \log n_{\mathscr{H}}))$.

The total time required by steps within the two procedures includes the time required for transferring new operations between the lists A_i, R_j^J and R_i^M, and the time needed for scheduling new operations. The first time is proportional to the number of transferences. For each machine $M_i \in \mathscr{M}_k$ each of n_k new operations initially included in the list A_i can be removed from the list (to a corresponding list R_j^J) at most $r-1$ times because of its overlap with another operation of the same job, and at most that many times can be returned to the list A_i; the same operation can also be at most s_i times removed from (and returned to) the list A_i because of its overlap with an old operation on machine M_i, where s_i is the number of old operations on machine M_i. Finally, an operation is removed from the list A_i one more time when it is assigned to the schedule. Therefore, the overall number of transferences of each operation in the list A_i is no more than $O(r + s_i)$, which implies that the total number of transferences of all new operations is no more than

$$O\left(\sum_{k=1}^{r}\sum_{M_i \in \mathscr{M}_k} n_k(r+s_i)\right) = O\left(\sum_{k=1}^{r} n_k(rm_k + v_k)\right).$$

Finally, while scheduling a new operation the most running time is required for the procedure of adding the operation to the heap $\mathscr{H}$ and the subsequent reordering of the heap, which makes up $O(\sum_{k=1}^{r} n_k \log n_{\mathscr{H}})$ time. It can be easily seen that summing up the above bounds on running time yields the desired bound on the running time of the algorithm, which completes the proof of Lemma 1. □

4. Polynomial-time approximation scheme for the multi-processor open shop problem with a fixed number of machines

In this section, a multi-processor open shop problem is considered in which the number n of jobs is variable, the number r of stages (shops) is fixed, and the number of machines at stage i, $1 \leqslant i \leqslant r$, is bounded above by a constant m_i. (In the standard notation, this problem is denoted by $Or(Pm)||C_{\max}$.) It will be shown that for this problem there exists an approximation scheme with running time linear in n.

The scheme uses the idea of a "semi-greedy" algorithm of constructing the so-called "semi-dense" schedules. Unlike a dense schedule, in which a machine may be idle only when there is no operation that can be processed on that machine, in a semi-dense schedule a number of "forced" idle time intervals on machines is allowed.

Another crucial idea is dividing the whole set of jobs into three subsets: of "large", "medium", and "small" jobs. For two real numbers $\alpha' > \alpha'' > 0$ we define three subsets of jobs: $L = \{J_j \in \mathscr{J} \mid d_j \geqslant \alpha' \hat{L}_{\max}\}$, $M = \{J_j \in \mathscr{J} \mid \alpha'' \hat{L}_{\max} \leqslant d_j < \alpha' \hat{L}_{\max}\}$, and $S = \{J_j \in \mathscr{J} \mid d_j < \alpha'' \hat{L}_{\max}\}$; these will be called, respectively, *large, medium* and *small.*

Operations of large, medium and small jobs will be also called *large, medium* and *small* (independently of their actual length). The numbers α' and α'' chosen must meet the following requirements:
(a) the number $|L|$ of large jobs must be bounded above by a constant (for a given ε);
(b) the total length of medium jobs cannot exceed the amount $\varepsilon\hat{L}_{\max}$;
(c) the number α' and the ratio α''/α' must be small enough, so as to meet the inequality

$$\alpha' + \alpha'' + \alpha''|L| \leqslant \varepsilon. \tag{3}$$

The scheme represents a family of algorithms $\{\mathscr{A}_\varepsilon \mid \varepsilon > 0\}$ such that for any fixed $\varepsilon > 0$, the corresponding algorithm $\mathscr{A}_\varepsilon$ for any instance of the problem computes (in polynomial time) a schedule with makespan at most $(1+\varepsilon)$ times the optimum makespan. Below a general scheme of the algorithm $\mathscr{A}_\varepsilon$ consisting of four steps is presented.

Algorithm $\mathscr{A}_\epsilon$

Step 1. If $\varepsilon \geqslant 1$ or $d_{\max} \leqslant \varepsilon\hat{L}_{\max}$, then apply the greedy algorithm $\mathscr{A}_G$ and take the constructed schedule for the desired output. Otherwise, go to Step 2.

Step 2. Find a partition of the set of jobs into three subsets: L, M, and S (of large, medium, and small jobs) so as to meet the requirements (a)–(c).

Step 3. Construct an optimal schedule S^L for the set of jobs L.

Step 4. Complete the schedule S^L for the remaining jobs (from the subsets M and S), using a "semi-greedy" modification of the algorithm $\mathscr{A}_{\mathrm{GC}}$.

Theorem 1. *For any real number $\varepsilon > 0$, integer r and m and any instance of the problem $Or(Pm)||C_{\max}$ with n jobs, the algorithm $\mathscr{A}_\varepsilon$ constructs a schedule S with makespan*

$$C_{\max}(S) \leqslant (1+\varepsilon)C^*_{\max}. \tag{4}$$

The algorithm has running time $O(n)$, *the multiplicative constant at n being polynomial in r and m and independent of ε.*

Proof. If $\varepsilon \geqslant 1$ or $d_{\max} \leqslant \varepsilon\hat{L}_{\max}$, then due to (2) the desired schedule S can be constructed at the first step of the algorithm $\mathscr{A}_\varepsilon$. Now, let $\varepsilon \in (0,1)$, $d_{\max} > \varepsilon\hat{L}_{\max}$, and assume that numbers α' and α'' satisfying the requirements (a)–(c) are already found at the second step of the algorithm $\mathscr{A}_\varepsilon$. (The step will be described in more detail a little bit later.) Step 3 needs no detailed description. It is clear that constructing an optimal schedule S^L for the jobs from the set L can be performed by any enumeration algorithm. The running time of such an algorithm can be counted in the overall bound on running time of the whole algorithm as an additive amount independent of n but depending on the number of large jobs and the number of machines. Since the last two parameters are bounded by constants, this adds only an additive constant to the overall bound on running time.

To complete the schedule for the medium and small jobs at Step 4, we could apply the scheme $\mathscr{A}_{\mathrm{GC}}$. In this case, due to Lemma 1, the running time of Step 4 would be estimated as $O(\sum_{k=1}^{r} n_k(rm_k + v_k + \log n_{\mathscr{H}})) \leqslant O((rnm + rn|L|)$, and therefore, the multiplicative constant at n would depend on ε. Now, we show how to avoid this dependence having somewhat sacrificed the density of the schedule.

To this end, we slightly change the "machine load" procedure described above. Now before scanning the list $A_{i'}$, we estimate the length of the idle time gap on machine $M_{i'}$, i.e., the length of the interval from time τ to the starting time of the current (old) operation on that machine. If the length is less than $\alpha''\hat{L}_{\max}$, the procedure terminates and we pass on to the next step of the cycle of algorithm $\mathscr{A}_{\mathrm{GC}}$ (even if some operations in the list $A_{i'}$ fit in the gap and could be scheduled at time τ). This guarantees that no operation of a small job will be transferred from the list $A_{i'}$ to the list $R_{i'}^{M}$. Therefore, the total number of transferences of each small operation in shop i cannot exceed $O(rm_i)$ (we remind the reader that every operation is presented in m_i independent lists A_i). Hence, the total number of transferences over all operations of a small job is $O(rm)$, and the number of transferences over all small jobs is $O(nrm)$. For medium operations, the old bound $O(r|M|(m + |L|))$ on the number of transferences remains valid. Since both the number of large jobs and the number of medium jobs are bounded by constants (depending on ε), the number of transferences of their operations contributes to the overall bound on running time of the algorithm as an additive constant. Thus, the running time of Step 4 for the new version of algorithm $\mathscr{A}_{\mathrm{GC}}$ does not exceed $O(nrm)$.

Next, we show that the performance ratio of the algorithm is indeed at most $1 + \varepsilon$. Let S^A be the schedule constructed, $C_{\max}^A$ be its length, and $C_{\max}^L$ be the length of the schedule S^L. Clearly, $C_{\max}^* \geqslant C_{\max}^L$. Without loss of generality, we can assume that the *critical machine*, which completes its work at time $C_{\max}^A$, is machine $M_1 \in \mathscr{M}_1$.

If the last operation on machine M_1 is large, then we have $C_{\max}^A = C_{\max}^L \leqslant C_{\max}^*$, and therefore, the schedule S^A is optimal. Let us further assume that the operation on machine M_1 completed last in schedule S^A belongs to a job J_k which is not large. It follows that $d_k \leqslant \alpha'\hat{L}_{\max}$. Let t be the starting time of operation o_1^k, t_i^* be the maximum completion time of a large operation in schedule S^L on machine $M_i \in \mathscr{M}_1$, and $t^* = \max_{M_i \in \mathscr{M}_1} t_i^*$. It is clear that

$$t^* \leqslant C_{\max}^* \leqslant C_{\max}^A = t + p_1^k,$$

and no large operation is being processed in the time interval $[t^*, C_{\max}^A]$ on a machine $M_i \in \mathscr{M}_1$. Let us first assume that no small operation is performed (even partially) on machine M_1 in the time interval $[t^*, C_{\max}^A]$ (and hence, J_k is a medium job). Since all idle time of machine M_1 in the time interval $[t^*, C_{\max}^A]$ is caused by processing other operations of job J_k (being processed in other shops), the total idle time of machine M_1 in the time interval $[t^*, C_{\max}^A]$ does not exceed $d_k - p_1^k$, whereas the total load of machine M_1 in the same interval does not exceed $p_1^k + \sum_{J_j \in M \setminus \{J_k\}} d_j$. Therefore, the length of the interval $[t^*, C_{\max}^A]$ is no more than the total length of medium jobs, which, due to the requirement (b), is at most $\varepsilon\hat{L}_{\max}$. This implies that

$$C_{\max}^A \leqslant t^* + \varepsilon\hat{L}_{\max} \leqslant (1 + \varepsilon)C_{\max}^*.$$

Now, let us consider the case when there is a small operation o_1^j, which is completed on machine M_1 at time $t' > t^*$. An idle time on machine $M_i \in \mathscr{M}_1$ within the time intervals $[0, t_i^*]$ and $[t_i^*, t]$ will be called *interior* and *exterior*, respectively. Let us estimate the total amount of the interior and exterior idle time (I_1 and I_2, respectively) over all machines $M_i \in \mathscr{M}_1$. First of all, we observe that any interior idle time on a machine $M_i \in \mathscr{M}_1$ precedes the starting time of the operation o_1^j. Therefore, any interior idle time is either covered by the interval of processing another operation of job J_j (the total length of such intervals over all machines $M_i \in \mathscr{M}_1$ is no more than $m_1 d_j \leqslant m_1 \alpha'' \hat{L}_{\max}$), or it is a "forced" idle time interval that precedes processing a large operation on that machine and has length at most $\alpha'' \hat{L}_{\max}$. The total length of such "forced" idle time intervals is at most $|L|\alpha'' \hat{L}_{\max}$. Therefore, $I_1 < (m_1 + |L|)\alpha'' \hat{L}_{\max}$.

Any exterior idle time can only be caused by processing operations of job J_k in other shops. So, the total exterior idle time over all machines $M_i \in \mathscr{M}_1$ is at most $I_2 \leqslant m_1(d_k - p_1^k)$. Since every machine $M_i \in \mathscr{M}_1$ at any time in the interval $[0, t]$ is either busy or idle, and all machines in the shop $\mathscr{M}_1$ can be busy in the time interval $[0, t]$ at most L_1 time units (in total), it follows that

$$tm_1 \leqslant L_1 + I_1 + I_2 < L_1 + (m_1 + |L|)\alpha'' \hat{L}_{\max} + m_1(d_k - p_1^k).$$

This implies that

$$\begin{aligned} C_{\max}^A = t + p_1^k &< L_1/m_1 + d_k + (1 + |L|/m_1)\alpha'' \hat{L}_{\max} \\ &\leqslant (1 + \alpha' + \alpha'' + \alpha''|L|)\hat{L}_{\max} \leqslant (1 + \varepsilon) C_{\max}^*. \end{aligned}$$

Thus, the algorithm $\mathscr{A}_\varepsilon$ always guarantees a $(1 + \varepsilon)$-approximation.

It remains to describe the algorithm of finding the numbers α' and α'' that provide a partition of the set of jobs into three subsets, of large, medium, and small jobs, with conditions (a)–(c) satisfied.

Compute the values of parameters $\{d_j \mid j = 1, \ldots, n\}$, $d_{\max} = \max d_j$ and $\hat{L}_{\max}$. We remind the reader that the inequalities $d_{\max} > \varepsilon \hat{L}_{\max}$ and $\varepsilon < 1$ are assumed to be valid. We can also assume that $\varepsilon = m/N$ for some integer N ($N > m$). At first, an algorithm with running time $O(n \log n)$ will be presented; then we show how to obtain the same result in $O(n)$ time.

- Number the set of all jobs in nonincreasing order of d_j.
- Set $\alpha_1 = \varepsilon/2$; $\alpha_0 = \infty$; $E_1 = [\alpha_1 \hat{L}_{\max}, \alpha_0 \hat{L}_{\max})$.
  Scan the list of jobs until the inequality $d_j < \alpha_1 \hat{L}_{\max}$ is satisfied;
  compute $D_1 \doteq \sum_{d_j \in E_1} d_j$ and set $n_1' = (D_1/\alpha_1 \hat{L}_{\max}) - 1$
- For $k = 2, 3, \ldots$ do the following:
  - find α_k from the equation

$$\alpha_{k-1} + \alpha_k(1 + n_{k-1}') = \varepsilon; \tag{5}$$

  - define $E_k = [\alpha_k \hat{L}_{\max}, \alpha_{k-1} \hat{L}_{\max})$; proceed with scanning the list of jobs until the inequality $d_j < \alpha_k \hat{L}_{\max}$ is satisfied; compute $D_k \doteq \sum_{d_j \in E_k} d_j$ and put

$$n_k' = n_{k-1}' + \frac{D_k}{\alpha_k \hat{L}_{\max}}; \tag{6}$$

- if $D_k \leqslant \varepsilon\hat{L}_{\max}$, then $\{\alpha'' := \alpha_k;\ \alpha' := \alpha_{k-1};\ \textbf{stop}\}$.

• End of the cycle on k.

We first show that for any $i \geqslant 2$ the inequality $\alpha_i < \alpha_{i-1}$ is valid (i.e., the numbers $\{\alpha_i\}$ define nonempty and *non-overlapping* intervals $\{E_i\}$). Suppose the contrary, i.e., for some $i \geqslant 2$ we have got $\alpha_i \geqslant \alpha_{i-1}$. Then in the case $i = 2$ we obtain

$$\varepsilon = \alpha_1 + \alpha_2(1 + n_1') \geqslant \alpha_1 + \frac{\alpha_1 D_1}{\alpha_1 \hat{L}_{\max}} \geqslant \alpha_1 + \frac{d_{\max}}{\hat{L}_{\max}} \geqslant \frac{3}{2}\varepsilon.$$

A contradiction. In the case $i \geqslant 3$ we obtain

$$\varepsilon = \alpha_{i-1} + \alpha_i(1 + n_{i-1}') \geqslant \alpha_{i-1}\left(2 + n_{i-2}' + \frac{D_{i-1}}{\alpha_{i-1}\hat{L}_{\max}}\right) > \alpha_{i-1}(2 + n_{i-2}') + \varepsilon > \varepsilon.$$

A contradiction.

Set $n_i = |\{J_j \mid d_j \geqslant \alpha_i \hat{L}_{\max}\}|$. By induction on i, we now prove the inequality $n_i \leqslant n_i'$.

Since the sum D_1 includes the length of job J_{j^*} as a summand, $(d_{j^*} = d_{\max} \geqslant \varepsilon\hat{L}_{\max})$, and the length of each of the remaining $n_1 - 1$ jobs presented in the sum D_1 can be bounded below as $d_j \geqslant \frac{\varepsilon}{2}\hat{L}_{\max}$, we derive

$$D_1 \geqslant \varepsilon\hat{L}_{\max} + (n_1 - 1)\tfrac{\varepsilon}{2}\hat{L}_{\max} = (n_1 + 1)\tfrac{\varepsilon}{2}\hat{L}_{\max} = (n_1 + 1)\alpha_1\hat{L}_{\max}.$$

Hence, we obtain $n_1 \leqslant n_1'$.

Let the inequality $n_{i-1} \leqslant n_{i-1}'$ be valid. Then

$$n_i \leqslant n_{i-1} + \frac{D_i}{\alpha_i\hat{L}_{\max}} \leqslant n_{i-1}' + \frac{D_i}{\alpha_i\hat{L}_{\max}} = n_i',$$

as required.

Next, we show that the command **stop** has worked and the parameters α', α'' has been defined. It follows from $D_1 \geqslant d_{\max} \geqslant \varepsilon\hat{L}_{\max}$ and relations

$$\sum_{k=1}^{N} D_k \leqslant \sum_{j=1}^{n} d_j = \sum_{i=1}^{r} L_i \leqslant \hat{L}_{\max} \sum_{i} m_i = m\hat{L}_{\max} = N\varepsilon\hat{L}_{\max}$$

that for at least one of the numbers $D_2, \ldots, D_k$, the inequality $D_k \leqslant \varepsilon\hat{L}_{\max}$ holds, which is exactly the condition that causes the execution of the command **stop**.

It remains to verify that the parameters α', α'' defined in the described above procedure meet the requirements (a)–(c). Requirement (b) is the condition of execution of the command **stop**; as already shown, it is satisfied. Requirement (c) follows from the relations

$$\alpha_{k-1} + \alpha_k(1 + n_{k-1}) \leqslant \alpha_{k-1} + \alpha_k(1 + n_{k-1}') = \varepsilon.$$

Requirement (a) will be a straightforward corollary of an upper bound on n_{k-1} to be derived now.

For the convenience of calculation, let us take $\varepsilon\hat{L}_{\max}$ for the unit and denote $\beta_i = \alpha_i/\varepsilon$. Then for $i = 3, \ldots, k$, from (5) and (6) we have

$$1 - \beta_{i-1} = \beta_i(1 + n_{i-1}') = \beta_i(1 + n_{i-2}' + D_{i-1}/\beta_{i-1}) = \beta_i(1 - \beta_{i-2} + D_{i-1})/\beta_{i-1},$$

or $\beta_i(1 - \beta_{i-2} + D_{i-1}) = \beta_{i-1}(1 - \beta_{i-1})$.

For $i=2$, we obtain a similar relation $1-\beta_1=\beta_2(1+n'_1)=\beta_2 D_1/\beta_1$, or $\beta_2(1-\beta_0+D_1)=\beta_1(1-\beta_1)$, where $\beta_0=1$. Therefore, the numbers β_i can be found from the recurrent relations

$$\beta_0=1; \quad \beta_1=\tfrac{1}{2}; \tag{7}$$

$$\beta_i(1-\beta_{i-2}+D_{i-1})=\beta_{i-1}(1-\beta_{i-1}), \quad i=2,\ldots,k, \tag{8}$$

as functions of parameters $\{D_i\}$ satisfying the relations

$$D_i\geqslant 1; \quad \sum_{i=1}^{k-1} D_i\leqslant N.$$

Since the number n_{k-1} of large jobs meets the inequality

$$n_{k-1}\leqslant n'_{k-1}=\frac{1-\beta_{k-1}}{\beta_k}-1\leqslant\frac{1}{\beta_k}-2,$$

it follows that while deriving an upper bound on n_{k-1}, it suffices to bound from below the amount β_k for $k\in[2,N]$. Since D_{k-1} is presented only in the last recurrent relation from (8), i.e.,

$$\beta_k(1-\beta_{k-2}+D_{k-1})=\beta_{k-1}(1-\beta_{k-1}),$$

we deduce that β_k takes its minimum value when D_{k-1} is maximum possible. Therefore, we may assume the equality $\sum_{i=1}^{k-1} D_i=N$ to be valid. For every $k\in\{2,\ldots,N\}$, we now derive a lower bound on β_k under conditions (7), (8), and

$$D_i\geqslant 1; \quad \sum_{i=1}^{k-1} D_i=N. \tag{9}$$

At first, we show that for every $i=0,1,2,\ldots$, the amount β_i meets the inequality

$$\beta_i\leqslant\tfrac{1}{2^i}. \tag{10}$$

For β_0 and β_1 the inequality is valid. Let it be valid for $i\leqslant k$, where $k\geqslant 1$. From (8) and $D_i\geqslant 1$ we derive

$$\beta_{k+1}=\frac{\beta_k(1-\beta_k)}{1-\beta_{k-1}+D_k}\leqslant\frac{\beta_k(1-\beta_k)}{2-1/2^{k-1}}.$$

The expression $\beta_k(1-\beta_k)$ as a function of β_k increases in the interval $[0,\frac{1}{2}]$, and due to the inequality $\beta_k\leqslant 1/2^k$, it takes its maximum value when $\beta_k=1/2^k$. Hence,

$$\beta_{k+1}\leqslant\frac{(1/2^k)(1-1/2^k)}{2(1-1/2^k)}=\frac{1}{2^{k+1}},$$

i.e., (10) is valid for $i=k+1$.

Consecutively using the recurrent formula (8), we obtain

$$\beta_k = \frac{(1-\beta_{k-1})(1-\beta_{k-2})\cdots(1-\beta_1)\beta_1}{(1-\beta_{k-2}+D_{k-1})(1-\beta_{k-3}+D_{k-2})\cdots(1-\beta_1+D_2)D_1}$$

$$= \left(\text{by } \beta_1 = \frac{1}{2}\right) = \frac{1-\beta_{k-1}}{2D_1}\prod_{i=1}^{k-2}\frac{1-\beta_i}{1-\beta_i+D_{i+1}} \geqslant \text{(by (10))}$$

$$\geqslant \frac{1-2^{-k+1}}{2D_1}\prod_{i=1}^{k-2}\frac{1-2^{-i}}{1-2^{-i}+D_{i+1}} = \frac{1}{2}\prod_{i=1}^{k-1}\frac{1-2^{-i}}{1-2^{-i+1}+D_i}. \tag{11}$$

First let us estimate the amount $\prod_{i=1}^{k-1}(1-2^{-i})$ from below. We have

$$\ln\prod_{i=1}^{k-1}(1-2^{-i}) > \ln\prod_{i\geqslant 1}(1-2^{-i}) = \sum_{i\geqslant 1}\ln(1-2^{-i})$$

$$= -\sum_{i\geqslant 1}\sum_{j\geqslant 1}\frac{1}{j(2^i)^j}$$

$$= -\sum_{j\geqslant 1}\sum_{i\geqslant 1}\frac{1}{j(2^j)^i} = -\sum_{j\geqslant 1}\frac{1}{j(2^j-1)}.$$

Since $2j(2^j-1) < (j+1)(2^{j+1}-1)$ for any $j\geqslant 5$, we obtain

$$-\sum_{j\geqslant 5}\frac{1}{j(2^j-1)} > -\frac{2}{5(2^5-1)}.$$

Therefore,

$$\ln\prod_{i=1}^{k-1}(1-2^{-i}) > -\sum_{j=1}^{4}\frac{1}{j(2^j-1)} - \frac{2}{5(2^5-1)} > -1.25,$$

which implies

$$\prod_{i=1}^{k-1}(1-2^{-i}) > \mathrm{e}^{-1.25}. \tag{12}$$

Now, let us estimate the amount

$$\prod_{i=1}^{k-1}(1-2^{-i+1}+D_i) \tag{13}$$

from above. It is easily seen that for a fixed sum of amounts D_i, namely, equal to N, the maximum of expression (13) is attained when all its factors are equal, i.e., $D_1 = 1-2^{-1}+D_2 = 1-2^{-2}+D_3 = \cdots = 1-2^{-k+2}+D_{k-1}$, or $D_{i+1} = D_1-1+2^{-i}$, $i = 1,\ldots,k-2$. (We ignore here the restriction $D_i\geqslant 1$.) For these $D_1,\ldots,D_{k-1}$, we have

$$N = \sum_{i=1}^{k-1}D_i = D_1(k-1)-(k-2)+\sum_{i=1}^{k-2}2^{-i}.$$

Therefore,

$$D_1 = \left(N + k - 2 - \sum_{i=1}^{k-2} 2^{-i} \right) /(k-1) < \frac{N+k-2}{k-1},$$

and expression (13) does not exceed $D_1^{k-1} \leqslant ((N+k-2)/(k-1))^{k-1}$.

It can be easily checked that the function $((N-1+x)/x)^x$ of x increases for $x > 0$, because the derivative of its logarithm, equal to $(\ln(N-1+x)/x) - (N-1)/(N-1+x)$, is positive. (This follows from the inequality $\ln(1/(1-\Delta)) > \Delta$ for $\Delta \in (0,1)$.) This implies that the amount $((N+k-2)/(k-1))^{k-1}$ attains the maximum for $k = N$, i.e., amount (13) is no more than 2^{N-1}. Substituting this bound, as well as bound (12) into (11), we obtain $\beta_k > e^{-1.25} \cdot 2^{-N}$. Therefore, the number of large jobs is no more than $n_{k-1} < 1/\beta_k < e^{1.25} \cdot 2^{m/\varepsilon}$.

The described above algorithm of running time $O(n \log n)$ can be easily transformed into a linear time algorithm. Since for any given instance the bound $\alpha'' > \varepsilon/(e^{1.25} \cdot 2^N)$ is guaranteed, we can first select (in linear time) the jobs J_j that are obviously "small", namely, those for which the inequality $d_j \leqslant (\varepsilon/e^{1.25} \cdot 2^N)\hat{L}_{\max}$ holds. Then for the remaining jobs (whose number is bounded above by a constant $\frac{m}{\varepsilon} \cdot e^{1.25} \cdot 2^{m/\varepsilon}$), using the described procedure, we find the numbers α', α'' and the partition of the remaining jobs into subsets L, M, and S.

Therefore, the running time of Step 2 is $O(rn)$, whereas the running time of the whole algorithm $\mathscr{A}_\varepsilon$ can be bounded by the amount $O(nrm)$, in which only an additive constant is ε-dependent. Theorem 1 is proved. □

5. On the efficiency of the approximation scheme

Let us consider an instance with parameters $m = 3$, $\varepsilon = \frac{1}{3}$. The algorithm described in [3] guarantees a $\frac{4}{3}$-approximation of the problem $O3||C_{\max}$ in $O(n)$ time. The latter amount includes (as an additive constant) the time needed to find for an instance with five jobs an approximate solution with an absolute approximation bound

$$C_{\max}(S) \leqslant \tfrac{4}{3} L_{\max}.$$

(It is proved that such a schedule exists for any instance of the problem $O3||C_{\max}$.) For comparison, in the described above approximation scheme, where we also have to add an additive constant to a linear in n bound on running time, this constant is an upper bound on the time needed for constructing an optimal (!) schedule for a significantly greater number of jobs (in the case that $m = 3$ and $\varepsilon = \frac{1}{3}$, the number of large jobs may be about a thousand). Thus, from the practical point of view our approximation scheme proves to be inefficient.

The way out of this situation lies in a further improvement of the scheme. For constructing a more perfect scheme, a deeper knowledge about the properties of optimal schedules of the open shop problem should be used. For example, it is known that in any dense schedule the number of inner idle time intervals on any machine is at most

$r-1$. And although for some instances there may be no dense optimal schedules (one such instance, with three jobs and three machines, is presented in [4]), this does not exclude that for any instance there may exist an optimal schedule with a similar property when the number of inner idle time intervals is bounded above by a constant independent of the number of jobs. If we proved this property, we could suggest another, much more efficient approximation scheme for the open shop problem, because in this case we had no need to bound the number of inner idle time intervals by the number of large jobs.

Acknowledgements

The authors would like to thank Dr. A.D. Korshunov who suggested a more elegant proof of the upper bound on the number of large jobs.

References

[1] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: Handbooks in Operations Research and Management Science, Vol. 4, S. Graves, A.H.G. Rinnooy Kan and P. Zipkin, eds., Logistics of Production and Inventory, Elsevier, North-Holland, Amsterdam, 1993, pp. 445–522.

[2] P. Schuurman, G.J. Woeginger, Approximation algorithms for the multiprocessor open shop problem, Report Woe-13, October 1997, TU Graz, Austria.

[3] S.V. Sevastianov, I.D. Tchernykh, Computer-aided way to prove theorems in scheduling, in: Algorithms — ESA'98. Proceedings of the sixth Annual European Symposium, Venice, Italy, August 1998. Lecture Notes in Computer Science, Vol. 1461, Springer, Berlin, 1998, pp. 502–513.

[4] S.V. Sevastianov, G.J. Woeginger, Makespan minimization in open shops: A polynomial time approximation scheme, Math. Programming 82 (1998) 191–198.

[5] S.V. Sevast'yanov, Efficient scheduling in open shops, in: A.D. Korshunov (Ed.), Discrete Analysis and Operations Research, Kluwer, Dordrecht, 1996, pp. 235–255.

[6] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov, D.B. Shmoys, Short shop schedules, Oper. Res. 45 (1997) 288–294.