WEAK AND STRONG MACHINE DOMINANCE IN A NONPREEMPTIVE FLOWSHOP *

Petr Čáp , Ondřej Čepek and Milan Vlach

Received December 12, 2003

Abstract. Flowshop scheduling deals with processing a set of jobs through a set of machines, where all jobs have to pass among machines in the same order. With the exception of minimizing a makespan on two machines, almost all other flowshop problems in a general setup are known to be computationally intractable. In this paper we study a special case of flowshop defined by additional machine dominance constraints. The main result of this paper shows that the flowshop problem with a dominant machine (where dominance can be defined in several ways) can be solved in a polynomial time for a broad class of objective functions. This result is achieved by providing a general recipe how to obtain polynomial time optimization algorithms for particular objective functions from the corresponding algorithms for single machine problems.

1 Introduction Deterministic flowshop scheduling problems have been studied by both theorists and practitioners since the pioneering work of Johnson [10] from the mid 50's. In [10] Johnson presented a polynomial time algorithm for minimizing the makespan of a flowshop problem on two machines. Following the notation and terminology of the classification for deterministic scheduling problems proposed by Graham et al. [7], we denote this problem as $F2||C_{max}$. Further research revealed that this problem is in some sense an exception, i.e. that most other flowshop problems in a general setup are computationally intractable. Typical results demonstrating this phenomenon include the fact that the $F3||C_{max}$ problem is NP-hard in the strong sense as well as an identical claim for the $F2||\sum C_j$ problem [6]. The $F2||L_{max}$ problem was proved to be strongly NP-hard in [14]. The last two complexity results imply intractability on two machines for many other commonly used objective functions, and therefore justify a search for additional constraints which would define tractable subcases of the general flowshop scheduling problem.

One way to impose such additional constraints is to consider a so called machine dominance. This approach is quite extensively studied in the literature [1, 8, 9, 15, 16, 17]. This paper is intended to integrate and generalize several results about nonpreemptive flowshop scheduling with machine dominance which have appeared in [17] and [3]. Each of these papers uses a different concept of machine dominance. In [3] two cases are considered: when the first machine is a dominant machine and when the last machine is a dominant machine. For the purposes of this paper we shall call that type of machine dominance *strong dominance*. The concept used in [17] is more general: any machine can be dominant, and moreover, the dominance condition in [17] is somewhat weaker than in [3], i.e. if a given machine is dominant under the conditions in [3] it is also dominant using the conditions in [17]. In this paper we shall call that type of machine dominance *weak dominance*. The paper [17] contains two principal results for the flowshop problem with a weakly dominant machine M_ℓ (we shall denote this problem by $F|wM_\ell|\gamma$):

2000 *Mathematics Subject Classification.* 90B35.

Key words and phrases. flowshop scheduling, machine dominance.

*This work was supported by the Grant Agency of the Czech Republic (grant 201/04/1102)

(1) For every regular objective function γ the set of permutation schedules contains an optimal schedule.

(2) Polynomial time optimization algorithms exist (and are presented) for the $f = \sum w_i C_i$ and the $f = L_{max}$ objective functions.

As mentioned above, the paper [3] deals with the cases when the first machine is strongly dominant (we shall denote this problem by $F|sM_1|\gamma$) and when the last machine is strongly dominant (we shall denote this problem by $F|sM_m|\gamma$). The following are the main results of the paper:

(A) For every regular objective function γ the set of permutation schedules contains an optimal schedule, both for the $F|sM_1|\gamma$ and $F|sM_m|\gamma$ problems as well as for the $F|sM_1, nmit|\gamma$ and $F|sM_m, nmit|\gamma$ problems where the additional "no machine idle time" ($nmit$) constraint is imposed.

(B) For a broad class of regular objective functions (which includes both functions in **(2)** and many others) the $F|sM_1|\gamma$ and $F|sM_m|\gamma$ problems are tractable (solvable in polynomial time) if and only if the corresponding single machine problem $1||\gamma$ is tractable.

In this paper we will integrate and generalize the above results. The main new contributions presented here are:

- We define a generalized concept of strong dominance allowing any machine (not just the first or last one) to be the dominant machine. A flowshop problem with this type of strong machine dominance will be denoted by $F|sM_\ell|\gamma$.
- We show that the strong dominance is indeed a special case of the weak dominance, i.e. that the $F|sM_\ell|\gamma$ problem is a special case of the $F|wM_\ell|\gamma$ problem. This immediately proves that the result **(1)** is valid also for $F|sM_\ell|\gamma$, and hence also for the special cases $\ell = 1$ and $\ell = m$ which constitute the first part of the result **(A)**.
- We show that unlike the first part of the result **(A)**, the second part (where the additional $nmit$ constraint is imposed) cannot be carried over to the more general concepts of machine dominance, namely to the $F|sM_\ell, nmit|\gamma$ problem (with the exception of $f = C_{max}$) and to the $F|wM_\ell|\gamma$ problem (even for $f = C_{max}$). This is proved by constructing counterexamples, i.e. instances where no permutation schedule is optimal.
- We prove that the result **(B)** can be extended to work also for the $F|wM_\ell|\gamma$ problem (and thus also for its special case $F|sM_\ell|\gamma$). This is achieved by showing how a polynomial time algorithm for a single machine problem $1||\gamma$ can be turned into a polynomial time algorithm for the corresponding flowshop problem $F|wM_\ell|\gamma$.

The paper is organized as follows. Section 2 starts with a formal definition of the nonpreemptive flowshop scheduling problem. Next, the concepts of strong and weak machine dominance are introduced and compared. After that, we define the notions of semi-active schedules and permutation schedules. In the end of the section we present a formula (from [17]) which is used to compute the completion times of jobs in permutation schedules. In Section 3, we give the proof that the result **(B)** can be carried over to the $F|wM_\ell|\gamma$ problem. Finally, in Section 4, we present the counterexamples which show that the presence of the $nmit$ constraint spoils the properties of permutation schedules for the more general concepts of machine dominace.

2 Flowshop with a dominant machine The deterministic flowshop scheduling problem can be described as follows. There are n jobs $J_1, J_2, \cdots, J_n$ to be processed through m machines $M_1, M_2, \cdots, M_m$. The technological constraints demand that the jobs pass among the machines in the same order. Without loss of generality we may assume that the machines are numbered according to the technological constraints, that is, we may assume that each job must be processed first on machine M_1, then on machine M_2, and so on, until it is finally processed on machine M_m. The order of jobs on any given machine is not prescribed and may vary from machine to machine. No machine may process more than one job at a time (machine constraint), and no job can be processed by several machines simultaneously (job constraint). Each job consists of m operations, one operation per machine. The operation of job J_j on machine M_i is denoted by O_{ij} and the processing time of operation O_{ij} is denoted by p_{ij}. Once a processing of an operation O_{ij} starts, it cannot be interrupted (nonpreemption constraint), i.e. operation O_{ij} then occupies the machine M_i for the next p_{ij} time units.

A *schedule* S for a nonpreemptive flowshop problem with n jobs to be scheduled on m machines is a set of mn nonnegative numbers C_{ij}^S, $1 \leq i \leq m$, $1 \leq j \leq n$, where each C_{ij}^S denotes the completion time of the operation O_{ij} in the schedule S. Since the operations cannot be preempted, the completion time fully specifies the span in which each operation is processed, namely operation O_{ij} is processed in the interval (B_{ij}^S, C_{ij}^S) where $B_{ij}^S = C_{ij}^S - p_{ij}$. For every job J_j, $1 \leq j \leq n$, the job completion time C_j^S is the completion time of its last operation, i.e. $C_j^S = C_{mj}^S$.

A schedule S is called *feasible* if it fulfills the inequalities $B_{ik}^S \geq 0$ for every machine M_i and job J_k, as well as all the machine and job constraints. These constraints can be stated as follows.

- Machine constraint: each machine may process at most one job at any given time. Formally

$$\forall i \in \{1, \ldots, m\}\ \forall k, \ell \in \{1, \ldots, n\} : (k \neq \ell) \Longrightarrow (B_{ik}^S, C_{ik}^S) \cap (B_{i\ell}^S, C_{i\ell}^S) = \emptyset$$

- Job constraint: no job is processed on more than one machine simultaneously, and moreover the operations of each job are processed in the prescribed order. Formally

$$\forall i, j \in \{1, \ldots, m\}\ \forall k \in \{1, \ldots, n\} : (i < j) \Longrightarrow (C_{ik}^S \leq B_{jk}^S)$$

The quality of a schedule is measured by an *objective function* which assigns to every schedule a real number. Every schedule that attains the minimum value of the objective function among all schedules in a certain set **S** of schedules is called *best* in **S**. Every schedule which is best in the set of all feasible schedules is called *optimal*. Typically, the task of flowshop scheduling is to find an optimal schedule.

In this paper we restrict our attention to objective functions which depend only on the job completion times and furthermore possess an additional property - a so called regularity. An objective function γ is said to be *regular* if for every two distinct schedules S_1 and S_2 the inequality $\gamma(S_1) < \gamma(S_2)$ implies $C_j^{S_1} < C_j^{S_2}$ for at least one j. Examples of commonly used regular objective functions include $C_{max}(S) = max_{j=1}^n C_j^S$, $L_{max}(S) = max_{j=1}^n \{C_j^S - d_j\}$ (so called "minimax" criteria), and $\sum C_j(S) = \sum_{j=1}^n C_j^S$, $\sum T_j(S) = \sum_{j=1}^n \max\{C_j^S - d_j, 0\}$, and $\sum U_j(S) = \sum_{j=1}^n sgn(\max\{C_j^S - d_j, 0\})$ (so called summation criteria), where d_j, $1 \leq j \leq n$, are nonnegative due dates associated with jobs. Regularity is preserved in all of the above cases even if the criteria are made more complex by introducing positive weights associated with jobs.

A feasible schedule is called *semi-active* if no operation O_{ij} can be started earlier without changing the order of operations on some machine or violating the feasibility constraints. It is well-known that every feasible schedule S can be transformed into a unique semi-active schedule S' such that for each machine M_i, the schedules S and S' have identical order of operations on M_i (see e.g. [17] od [3] for details). Moreover, this transformation has the property that the "new" completion times $C_j^{S'}$ are not larger than the "old" completion times C_j^S. Therefore, under an arbitrary regular objective function, the search for an optimal schedule can be restricted to semi-active schedules.

Since in many cases it is very hard to optimize over the entire set of all semi-active schedules, a common approach in flowshop scheduling is to restrict the attention yet further to the set of the so-called permutation schedules. A schedule is a *permutation schedule* if it is semi-active and the jobs are processed in the same order on all m machines. The name "permutation schedule" reflects the fact that a single permutation in this case completely specifies the entire schedule. Let π be an arbitrary permutation of the set $\{1, \ldots, n\}$. Then the permutation schedule specified by π, i.e. a semi-active schedule in which on every machine the first scheduled job is $J_{\pi(1)}$, the second scheduled job is $J_{\pi(2)}$, and so on, will be denoted by T_π. Furthermore, we denote the set of all permutation schedules by $\mathbf{P}$. Unfortunately, unlike with the set of semi-active schedules, it may happen that the set $\mathbf{P}$ of permutation schedules contains no optimal schedule. Although a simple interchange argument shows that $\mathbf{P}$ always contains an optimal schedule for $m = 2$ for all regular objective functions and for $m = 3$ for some "minimax" objective functions [5], already for $m = 4$ one can construct instances in which no permutation schedule is optimal even for the simplest objective function C_{max}.

The above difficulty can be overcome by imposing certain additional conditions on the processing times. Several types of such conditions called *machine dominance constraints* appeared in the literature. It was proved in [17] and [3] that under some of these machine dominance constraints, $\mathbf{P}$ always contains an optimal schedule regardless of the value of m. Let us now define these machine dominance constraints. Let us start with the definition from [3].

We shall say that a machine M_i *dominates* a machine M_j (denoted by $M_i \succ M_j$) if

$$\forall k, \ell \in \{1, \ldots, n\} : (k \neq \ell) \Longrightarrow p_{ik} \geq p_{j\ell}.$$

This definition is slightly more general (by not requiring any relation between the lengths of operations of the same job) than a more commonly used definition which states that M_i dominates M_j if

$$\min_{k=1}^{n}\{p_{ik}\} \geq \max_{k=1}^{n}\{p_{jk}\}.$$

A survey of results using this slightly "stronger" definition of machine dominance can be found in [15].

There are two common ways how to extend the notion of dominance for two machines to a set of m machines. Both were studied in several papers, e.g. in [1, 3, 9]. Machines $M_1, \ldots, M_m$ are said to form an *increasing series of dominating machines (idm)* if

$$M_m \succ M_{m-1} \succ \cdots \succ M_2 \succ M_1,$$

and to form a *decreasing series of dominating machines (ddm)* if

$$M_1 \succ M_2 \succ \cdots \succ M_{m-1} \succ M_m.$$

For the purposes of this paper we shall use a different notation. In the *idm* case we shall say that the machine M_m is a *strongly dominant machine* and denote the corresponding flowshop problem by $F|sM_m|\gamma$. Similarly, in the *ddm* case we shall say that the machine M_1 is a *strongly dominant machine* and denote the corresponding flowshop problem by $F|sM_1|\gamma$.

This notation suggests a straightforward generalization of the described machine dominance concept. We shall say that the machine M_ℓ, $1 \le \ell \le m$, is a *strongly dominant machine* if

$$M_\ell \succ M_{\ell-1} \succ \cdots \succ M_2 \succ M_1 \quad \text{and} \quad M_\ell \succ M_{\ell+1} \succ \cdots \succ M_{m-1} \succ M_m.$$

The corresponding flowshop problem wil be denoted by $F|sM_\ell|\gamma$.

In [18] and later independently in [17] a yet another concept of machine dominance was defined. We say that machine M_ℓ, $1 \le \ell \le m$, is a *weakly dominant machine* if the processing times satisfy the following conditions:

$$\sum_{k=\ell}^{r} p_{ki} \ge \sum_{k=\ell+1}^{r+1} p_{kj} \quad \forall i \ne j \in \{1,\dots,n\}, \ \forall r \in \{\ell,\dots,m-1\}, \tag{1}$$

and

$$\sum_{k=r}^{\ell} p_{ki} \ge \sum_{k=r-1}^{\ell-1} p_{kj} \quad \forall i \ne j \in \{1,\dots,n\}, \ \forall r \in \{2,\dots,\ell\}. \tag{2}$$

It is not hard to verify that if the machine M_ℓ is strongly dominant it is also weakly dominant. Let us verify conditions (1) by induction on r. In the basic step for $r = \ell$ the required inequality simply reduces to $p_{\ell,i} \ge p_{\ell+1,j}$ for all $i \ne j$. However, that is exactly the definition of $M_\ell \succ M_{\ell+1}$ which holds by our assumption. For the induction step from an arbitrary r, $\ell \le r \le m-2$, to $r+1$ it is sufficient to break the required inequality

$$\sum_{k=\ell}^{r+1} p_{ki} \ge \sum_{k=\ell+1}^{r+2} p_{kj}$$

into two inequalities

$$\sum_{k=\ell}^{r} p_{ki} \ge \sum_{k=\ell+1}^{r+1} p_{kj}$$

and

$$p_{r+1,i} \ge p_{r+2,j}.$$

The first inequality then follows from the induction hypothesis and the second inequality from the assumption $M_{r+1} \succ M_{r+2}$. Conditions (2) can be verified similarly.

Let us now consider the flowshop problem with a regular objective function γ and with the machine M_ℓ being weakly dominant. We shall denote this problem by $F|wM_\ell|\gamma$. The key result in [17] (appearing there as Theorem 5) can be stated using the terminology of this paper as follows.

Theorem 2.1 *Let γ be an arbitrary regular objective function. Then the set $\mathbf{P}$ of all permutation schedules contains an optimal schedule for the $F|wM_\ell|\gamma$ problem. Furthermore,*

if π is an arbitrary permutation of $\{1, 2, \ldots, n\}$ and $T_\pi \in \mathbf{P}$ is the corresponding permutation schedule, then the job completion times in T_π are given by the formula

$$C_{\pi(i)}^{T_\pi} = \sum_{k=1}^{\ell-1} p_{k\pi(1)} + \sum_{k=1}^{i} p_{\ell\pi(k)} + \sum_{k=\ell+1}^{m} p_{k\pi(i)}. \tag{3}$$

Theorem 2.1 guarantees that when searching for an optimal schedule for the $F|wM_\ell|\gamma$ problem, we can restrict ourselves to the set $\mathbf{P}$. Due to the above considerations relating the different concepts of machine dominance, the same is true for the $F|sM_\ell|\gamma$ problem and hence also for its special cases $F|sM_1|\gamma$ and $F|sM_m|\gamma$. The last two results were obtained independently in [3] together with the same results for the $F|sM_1, nmit|\gamma$ and $F|sM_m, nmit|\gamma$ problems, i.e. problems where no machine idle time is allowed. The question whether the set $\mathbf{P}$ also always contains an optimal schedule for the $F|sM_\ell, nmit|\gamma$ and $F|wM_\ell, nmit|\gamma$ problems will be answered (in a negative way) in Section 4.

The second key result in [17] is that formula (3) can be used to find an optimal schedule for the $F|wM_\ell|\gamma$ problem where $f = \sum w_i C_i$ or $f = L_{max}$ in polynomial time. For this result, two polynomial time algorithms solving the above two problems are presented in [17]. In the next section we shall see that formula (3) can be in fact used in a much more general way, yielding algorithms for a broader class of objective functions which includes both $\sum w_i C_i$ and L_{max} as well as many others.

3 Reductions to single machine problems Let us start this section by noting that the single machine scheduling problem is just a special case of the m-machine flowshop scheduling problem for $m = 1$. Moreover, for $m = 1$ all of the considered machine dominance constraints become trivially satisfied. Thus, given any objective function γ for which the single machine scheduling problem $1||\gamma$ is known to be NP-hard, also the corresponding flowshop scheduling problem $F|wM_\ell|\gamma$ is obviously NP-hard (and the same is true for its special cases $F|sM_\ell|\gamma$, $F|sM_1|\gamma$, and $F|sM_m|\gamma$). The simplest examples of such objective functions are $\gamma = \sum T_j$ and $\gamma = \sum w_j U_j$ (proofs of NP-hardness can be found in [4] for the $1||\sum T_j$ problem and in [11] for the $1||\sum w_j U_j$ problem).

We shall show in this section that also the reverse is true for a reasonably broad class of objective functions. By this we mean that given a polynomial time algorithm which solves the single machine problem $1||\gamma$ for a given objective function γ, we can modify this algorithm to solve the corresponding flowshop problem $F|M_\ell|\gamma$ also in polynomial time. However, the complexity of the new algorithm goes up by a factor of n compared with the complexity of the single machine algorithm. It should be noted here that for certain functions γ it may be possible to design "custom made" optimization algorithms which have a better complexity than what is achieved by using our generic approach. For instance Ho and Gupta [9] present a $O(n^2)$ algorithm for the $F|sM_m|L_{max}$ problem (which can be shown to work also for the $F|wM_m|L_{max}$ problem), while the modification of the $O(n \log n)$ EDD algorithm for the $1||L_{max}$ problem by our approach yields an $O(n^2 \log n)$ algorithm.

As we have already indicated, these modifications unfortunately are not applicable to all regular objective functions. However, the class of functions to which the modifications are applicable is quite broad; in particular, it contains both functions used in [17] and also all other commonly used objective functions introduced in Section 2. Let us be more specific. We shall restrict our attention to objective functions of the form

$$\gamma(S) = \max_{k=1}^{n} g_k(C_k^S) \tag{4}$$

and of the form

$$\gamma(S) = \sum_{k=1}^{n} g_k(C_k^S) \tag{5}$$

where in both cases $g_k(C_k^S)$ is a penalty associated with job J_k given by some nondecreasing function $g_k : R \longrightarrow R$. Note, that since the functions g_k are assumed to be nondecreasing, every objective function defined by (4) or (5) is regular.

From now on let us consider a fixed instance $\mathbf{A}$ of the $F|wM_\ell|\gamma$ problem. We shall show how to solve $\mathbf{A}$ in polynomial time, provided we have a polynomial time algorithm for the single machine problem $1||\gamma$. Let us start by rewriting formula (3) as

$$C_{\pi(j)}^{T_\pi} = \sum_{k=1}^{\ell} p_{k\pi(1)} + \sum_{k=2}^{j} p_{\ell\pi(k)} + \sum_{k=\ell+1}^{m} p_{k\pi(j)}. \tag{6}$$

Defining r_k and t_k by

$$r_k = \sum_{i=1}^{\ell} p_{ik}, \quad t_k = \sum_{i=l+1}^{m} p_{ik},$$

we can rewrite the job completion formula (6) as

$$C_{\pi(j)}^{T_\pi} = r_{\pi(1)} + \sum_{k=2}^{j} p_{\ell\pi(k)} + t_{\pi(j)}. \tag{7}$$

Now let us assume that we choose job J_q to be scheduled first. Then we have from (7)

$$C_{\pi(j)}^{T_\pi} = r_q + \sum_{k=2}^{j} p_{\ell\pi(k)} + t_{\pi(j)} \tag{8}$$

for every job J_j. Let us denote by $\mathbf{P}_q$ the set of those schedules T_π in $\mathbf{P}$ for which $\pi(1) = q$, and let us furthermore denote by I_q the index set of jobs that excludes job J_q, that is, $I_q = \{k \mid 1 \leq k \leq n,\ k \neq q\}$. Note that the sum $\sum_{k=2}^{j} p_{\ell\pi(k)}$ gives the job completion time of job $J_{\pi(j)}$, $2 \leq j \leq n$, in the permutation schedule given by π for the single machine problem with $n-1$ jobs J_k, $k \in I_q$, and processing times

$$p_k = p_{\ell k},\ k \in I_q. \tag{9}$$

Let us denote the set of all permutation schedules of the above $1||\gamma$ problem with $n-1$ jobs by $\mathbf{Q}$. Notice, that there exists an obvious bijection $\alpha : \mathbf{Q} \longrightarrow \mathbf{P}_q$ which assigns to every schedule in $\mathbf{Q}$ the corresponding schedule in $\mathbf{P}_q$ with the same permutation of jobs excluding job J_q. Now let $S \in \mathbf{Q}$ be an arbitrary schedule. Then due to (7) we get

$$C_k^{\alpha(S)} = C_k^S + r_q + t_k,\ k \in I_q. \tag{10}$$

We shall see, that for a broad class of objective functions, the task of minimizing γ over $\mathbf{P}_q$ can be reduced to minimizing γ over $\mathbf{Q}$, which is to say that if we find a schedule S which is best in $\mathbf{Q}$ for γ then $\alpha(S)$ is best in $\mathbf{P}_q$ for γ. Let us investigate which objective functions have this property.

Lemma 3.1 *Let γ be given by (5) and let each function g_k, $k \in I_q$, have the following property*

$$\forall t_1, t_2 \ : \ g_k(t_1 + t_2) = g_k(t_1) + g_k(t_2). \tag{11}$$

Then if S is best in $\mathbf{Q}$ for the $1||\gamma$ problem defined by (9) then $\alpha(S)$ is best in $\mathbf{P}_q$ for the problem given by $\mathbf{A}$.

Proof. Let S be the best schedule in $\mathbf{Q}$. Using (5) and (10) we get

$$\gamma(\alpha(S)) = \sum_{k \in I_q} g_k(C_k^{\alpha(S)}) = \sum_{k \in I_q} g_k(C_k^S + r_q + t_k)$$

which can be rewritten using (11) as

$$\sum_{k \in I_q} (g_k(C_k^S) + g_k(r_q) + g_k(t_k)) = \sum_{k \in I_q} g(C_k^S) + (n-1) g_k(r_q) + \sum_{k \in I_q} g_k(t_k).$$

Obviously, the second and third summands are independent of the schedule S, while the first summand attains the minimum possible value over all schedules in $\mathbf{Q}$ due to the choice of S. Therefore, since α is a bijection, $\gamma(\alpha(S))$ also attains a minimum possible value over all schedules in $\mathbf{P}_q$ and thus $\alpha(S)$ is best in $\mathbf{P}_q$. ■

Examples of functions satisfying (11) include e.g. $g_k(t) = t$ or $g_k(t) = w_k t$. On the other hand e.g. the function $g_k(t) = t^2$ does not satisfy (11). This means that Lemma 3.1 is applicable to regular objective functions $\sum C_j$ and $\sum w_j C_j$, but it is not applicable to $\sum C_j^2$ despite the fact that this objective function is also regular.

Lemma 3.2 *Let γ be given by (4) or (5) and let each function g_k, $k \in I_q$, have the form*

$$g_k(t) = \varphi_k(t - d_k) \tag{12}$$

for some nondecreasing function $\varphi_k : R \to R$. Then if S is best in $\mathbf{Q}$ for the $1||\gamma$ problem defined by (9) with due dates given by

$$d'_k = d_k - (r_q + t_k), \ k \in I_q, \tag{13}$$

then also $\alpha(S)$ is best in $\mathbf{P}_q$ for the problem given by $\mathbf{A}$.

Proof. Let S be an arbitrary schedule in $\mathbf{Q}$ and let us first assume that γ is given by (4). Then if we use (12) we obtain

$$\gamma(S) = \max_{k \in I_q} \ g_k(C_k^S) = \max_{k \in I_q} \ \varphi_k(C_k^S - d'_k)$$

which, using (13), can be rewritten as

$$\max_{k \in I_q} \ \varphi_k(C_k^S - (d_k - (r_q + t_k))) = \max_{k \in I_q} \ \varphi_k(C_k^S + r_q + t_k - d_k)$$

and furthermore, using (10) we get

$$\max_{k \in I_q} \ \varphi_k(C_k^{\alpha(S)} - d_k) = \max_{k \in I_q} \ g_k(C_k^{\alpha(S)}) = \gamma(\alpha(S)).$$

If γ is given by (5), an almost identical computation to the one above can be made. We leave this to the reader as an easy exercise. Hence, if $\gamma(S)$ is the minimum value of γ over $\mathbf{Q}$ then $\gamma(\alpha(S))$ is the minimum value of γ over $\mathbf{P}_q$ which completes the proof. ■

Examples of functions satisfying (12) include e.g. $g_k(t) = t - d_k$, $g_k(t) = \max\{t - d_k, 0\}$, and $g_k(t) = \text{sign}(\max\{t - d_k, 0\})$. This means that among other objective functions, Lemma 3.2 is applicable to functions L_{max}, $\sum T_j$, and $\sum U_j$. It is easy to see that Lemma 3.2 remains applicable for all of the above functions even if we add job related positive weights.

The only regular objective function from those introduced in Section 2 which fits neither Lemma 3.1 nor Lemma 3.2 is C_{max}. However, minimizing C_{max} over the set $\mathbf{P}_q$ is trivial, since C_{max} attains the same value for every schedule in $\mathbf{P}_q$, and hence every schedule in $\mathbf{P}_q$ is best in $\mathbf{P}_q$.

Now let us assume that γ fulfills the assumptions of Lemma 3.1 or Lemma 3.2 and moreover that there is a polynomial time algorithm finding an optimal schedule for the corresponding $1||\gamma$ problem defined by (9) (and by (13), if applicable). Then Lemma 3.1 and Lemma 3.2 give a recipe how to find the best schedule in the set $\mathbf{P}_q$. Indeed, if we use the given algorithm to obtain an optimal schedule S for the single machine problem, then $\alpha(S)$ is the best schedule in $\mathbf{P}_q$ for the problem given by $\mathbf{A}$.

Thus we now have a tool how to find the best schedule in $\mathbf{P}_q$ (for an arbitrary $1 \leq q \leq n$) for the problem given by $\mathbf{A}$, as long as γ belongs to the above specified broad class of regular objective functions which includes all of the commonly used functions introduced in Section 2. It is obvious, that to find an optimal schedule for the problem given by $\mathbf{A}$, it is now enough to find the best schedule in every $\mathbf{P}_q$, $1 \leq q \leq n$, and then compare these n candidates. It then follows that the complexity of finding the optimal schedule for problem given by $\mathbf{A}$ is n times the complexity of the algorithm which solves the single machine problem (unless this complexity is dominated by the time necessary for reading the input data). Notice, that this last claim is not influenced by the preprocessing phase of computing the "constants" r_q and t_k, as this computation clearly takes just $O(nm)$ time while $\Omega(nm)$ time is in any case necessary to read the input data.

Let us now give several concrete examples of how the general approach described above can be used. The polynomial time algorithm for the $1||\sum w_jC_j$ problem [12] yields a polynomial time algorithm for the $F|wM_\ell|\sum w_jC_j$ problem, and the polynomial time algorithm for the $1||L_{max}$ problem [2] yields a polynomial time algorithm for the $Fm|wM_\ell|L_{max}$ problem. This gives alternative proofs of polynomial time solvability for these two problems (polynomial time algorithms for them were presented in [17]), which are the only $F|wM_\ell|\gamma$ type problems known so far to be solvable in polynomial time. However, we can now extend this class by deriving new results. As an example let us take the polynomial time algorithm for the $1||\sum U_j$ problem [13]. This algorithm yields a polynomial time algorithms for the $F|wM_\ell|\sum U_j$ problem, a problem whose complexity was not known yet. The same approach as for polynomial time algorithms works also for pseudopolynomial time ones. Indeed an existence of a pseudopolynomial time algorithm for the $1||\gamma$ problem yields (by the same argument) a pseudopolynomial time algorithm for the corresponding $F|wM_\ell|\gamma$ problem (the complexity increases again by a factor of n). Let us give two examples demonstrating this phenomenon. The pseudopolynomial algorithm for the $1||\sum T_j$ problem [4] yields a pseudopolynomial algorithm for the $Fm|wM_\ell|\sum T_j$ problem, and the pseudopolynomial algorithm for the $1||\sum w_jU_j$ problem [13] yields a pseudopolynomial algorithm for the $Fm|wM_\ell|\sum w_jU_j$ problem.

In the next section we shall investigate, how do the so far discussed flowshop problems change, and what results can be derived for them, if we add the no machine idle time ($nmit$) constraint.

4 Problems with the no machine idle time constraint The no machine idle time (*nmit*) constraint means that each machine, once it commences its work, has to process all operations assigned to it without any interruption. Such a restriction may be very natural in some real life situations, e.g if machines represent expensive pieces of equipment which have to be rented only for the duration between the start of the first operation and the completion of the last one.

Flowshop problems with machine dominance and the additional *nmit* constraint were studied in [1], where polynomial time algorithms finding the best permutation schedule were presented for the $F|sM_1, nmit|\sum C_j$ and $F|sM_m, nmit|\sum C_j$ problems. In [3] it was shown, that the above algorithms in fact find an optimal schedule. This was achieved by proving that the set of permutation schedules always contains an optimal schedule for both of the above problems, moreover not just for $\gamma = \sum C_j$ but for every regular objective function γ. In this section we shall investigate, whether the above described property of permutation schedules carries over to the more general concepts of machine dominance, namely to the $F|sM_\ell, nmit|\gamma$ and $F|wM_\ell, nmit|\gamma$ problems. We shall see that unfortunately with one slight exception, the answer is negative.

Let us start with the $F|sM_\ell, nmit|\gamma$ problem. We shall construct instances of this problem with $\gamma = L_{max}$ and $\gamma = \sum C_j$ where the set of permutation schedules fails to contain an optimal schedule.

Lemma 4.1 *The set of permutation schedules for the $F|sM_\ell, nmit|L_{max}$ problem does not always contain an optimal schedule.*

Proof. Let us define an instance of the $F|sM_\ell, nmit|L_{max}$ problem with $m = n = 3$ (three jobs on three machines) by

machine/job	J_1	J_2	J_3
M_1	1	4	4
M_2	5	5	5
M_3	3	2	1

It is easy to verify that this instance fulfils the sM_2 constraint, or in other words that $M_1 <\cdot M_2 \cdot> M_3$. Furtermore let us define the due date for job J_2 by $d_2 = 13$, and let us set the due dates for jobs J_1 and J_3 sufficiently large so that they cannot influence the objective function L_{max} (e.g. $d_1 = d_3 = 100$).

Now let us check Figure 1 (for now ignore the numbers on the right margin). In the topmost non-permutation schedule all three jobs meet their due dates (job J_2 tightly) and thus this schedule attains the value $L_{max} = 0$. On the other hand, in all six permutation schedules depicted below the job J_2 is late, thus forcing in all six cases $L_{max} > 0$. Therefore, no permutation schedule is optimal. ∎

Lemma 4.2 *The set of permutation schedules for the $F|sM_\ell, nmit|\sum C_j$ problem does not always contain an optimal schedule.*

Proof. Let us define an instance of the $F|sM_\ell, nmit|\sum C_j$ problem in the same way as in the previous proof (except that no due dates are needed now). Once again, let us check Figure 1. For each schedule the job completion times C_1, C_2, and C_3 as well as their sum are presented on the right margin in the form $C_1 + C_2 + C_3 = \sum C_j$. We can see that the topmost non-permutation schedule attains a smaller value of $\sum C_j$ than any of the six permutation schedules, and hence no permutation schedule is optimal. ∎

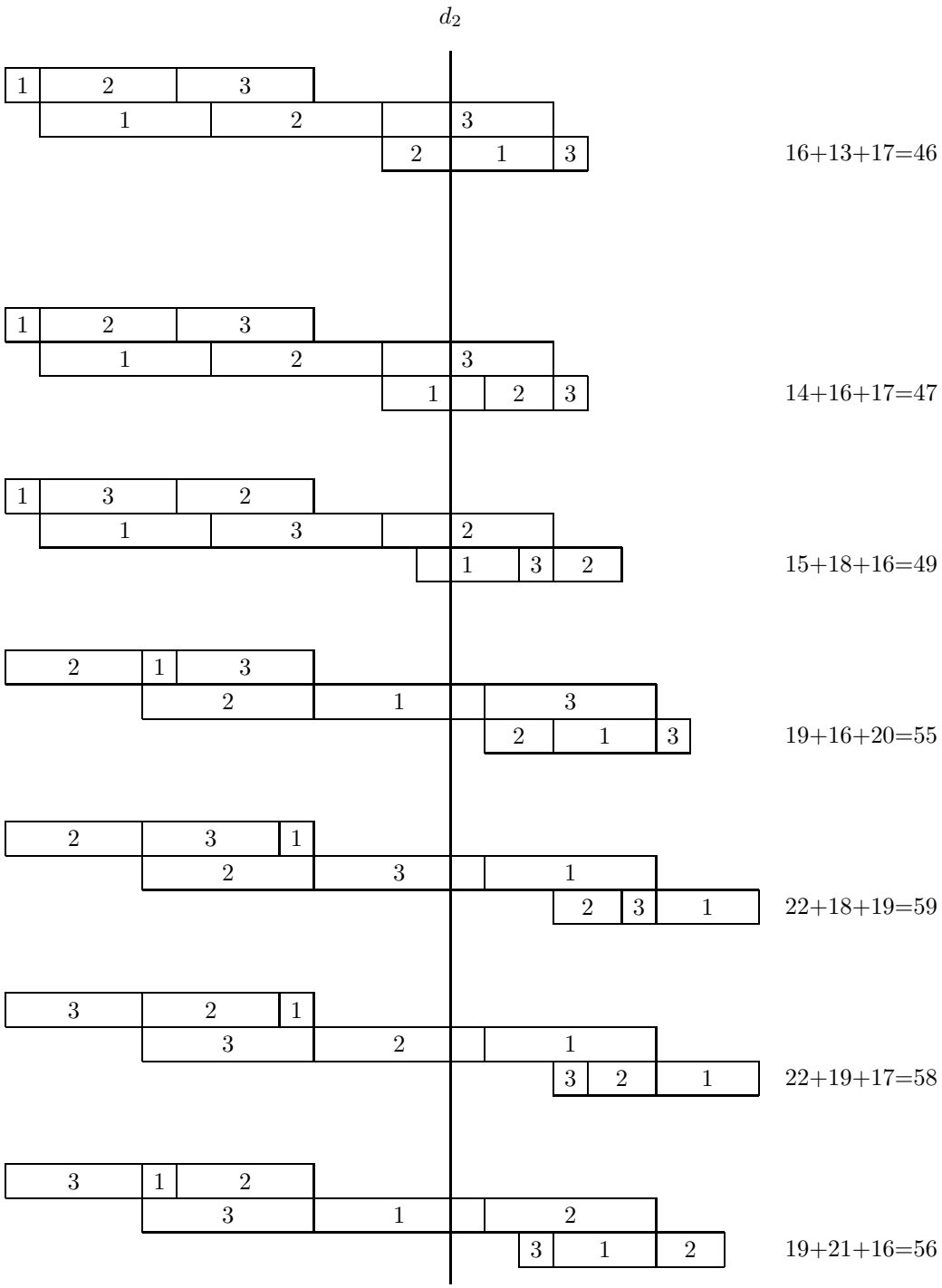

Figure 1: Counterexample for $F|sM_\ell, nmit|L_{max}$ and $F|sM_\ell, nmit|\sum C_j$ problems

It is easy to see, that the counterexample for $\gamma = L_{max}$ works also for $\gamma = \sum U_j$ and $\gamma = \sum T_j$ (the topmost non-permutation schedule has $\sum U_j = \sum T_j = 0$ while all six permutation schedules have $\sum U_j = \sum T_j > 0$). Of course, the same counterexamples work also for problems with added job related weights, i.e. for objective functions $\sum C_j, \sum U_j$, and $\sum T_j$ (if we take all weights equal to one). Thus the statement of Lemma 4.1 and Lemma 4.2 is true for all commonly used objective functions except for $\gamma = C_{max}$. This objective function really is an exception, as the following lemma shows.

Lemma 4.3 *The set of permutation schedules for the $F|sM_\ell, nmit|C_{max}$ problem always contains an optimal schedule, and moreover such an optimal schedule can be constructed in a polynomial time.*

Proof. A trivial lower bound for the makespan of any feasible schedule is given by

$$\min_{1\le a\le n, 1\le b\le n, a\neq b} \Big\{ \sum_{i=1}^{\ell-1} p_{ia} + \sum_{i=1}^{n} p_{\ell i} + \sum_{i=\ell+1}^{m} p_{ib} \Big\}, \tag{14}$$

where J_a is the first job processed on machine M_ℓ, J_b is the last job processed on machine M_ℓ, and the formula takes the minimum over all possible pairs J_a, J_b. Notice that the middle sum which represents the sum of lengths of all operations on machine M_ℓ is a constant independent of the choice of J_a and J_b.

A permutation schedule with a makespan equal to the lower bound (14), i.e. an optimal permutation schedule, can be constructed as follows. Compute for each job J_j the expressions $r_j = \sum_{i=1}^{\ell-1} p_{ij}$ and $t_j = \sum_{i=\ell+1}^{m} p_{ij}$. Then select the two smallest r_j's, i.e. let $r_a = \min_{j\in N} r_j$ and $r_{a'} = \min_{j\in N\setminus\{a\}} r_j$ where $N = \{1, \dots, n\}$. Similarly let $t_b = \min_{j\in N} t_j$ and $t_{b'} = \min_{j\in N\setminus\{b\}} t_j$ (in both cases break ties arbitrarily). Now if $a \neq b$ then select J_a to be the first job and J_b to be last job. If $a = b$ then compare $r_a + t_{b'}$ with $r_{a'} + t_b$ and instead of the pair J_a, J_b select either $J_a, J_{b'}$ or $J_{a'}, J_b$, whichever attains the smaller value. Once the first and last jobs are chosen, finalize the permutation of jobs by putting the remaining jobs inbetween the first and last one in an arbitrary order.

Now let us construct the desired optimal schedule. Starting at time zero schedule the operations of job J_a (or $J_{a'}$, whichever was selected) on machines $M_1, \dots, M_{\ell-1}$ one immediately after another (in a no-wait fashion). Then schedule all operations on M_ℓ in the chosen order with no machine idle time inbetween operations, and follow with the operations of job J_b (or $J_{b'}$) on machines $M_{\ell+1}, \dots, M_m$ (again in a no-wait fashion). Finally, schedule all remaining operations in the order given by machine M_ℓ : on machines $M_1, \dots, M_{\ell-1}$ immediately after the corresponding operation of job J_a (or $J_{a'}$) and on machines $M_{\ell+1}, \dots, M_m$ immediately before the corresponding operation of job J_b (or $J_{b'}$), of course in both cases without any machine idle time. It is easy to check, that due to the strong dominance of machine M_ℓ, the schedule constructed above is feasible (no two operations of the same job overlap). ∎

Now let us turn our attention to the weak form of machine dominance, namely to the $F|wM_\ell, nmit|\gamma$ problem. Since the $F|sM_\ell, nmit|\gamma$ problem is just a special case of this problem (as was shown in Section 2), it is obvious that whenever the set of permutation schedules for the $F|sM_\ell, nmit|\gamma$ problem fails to always contain some optimal schedule for a given objective function γ, the same is true for the $F|wM_\ell, nmit|\gamma$ problem. Thus, because of the results derived for the $F|sM_\ell, nmit|\gamma$ earlier in this section, the only open case for $F|wM_\ell, nmit|\gamma$ is the case $\gamma = C_{max}$.

We shall show now that for the $F|wM_\ell, nmit|C_{max}$ problem no statement that would be similar to Lemma 4.3 can be proved. The reason is that the weak machine dominance

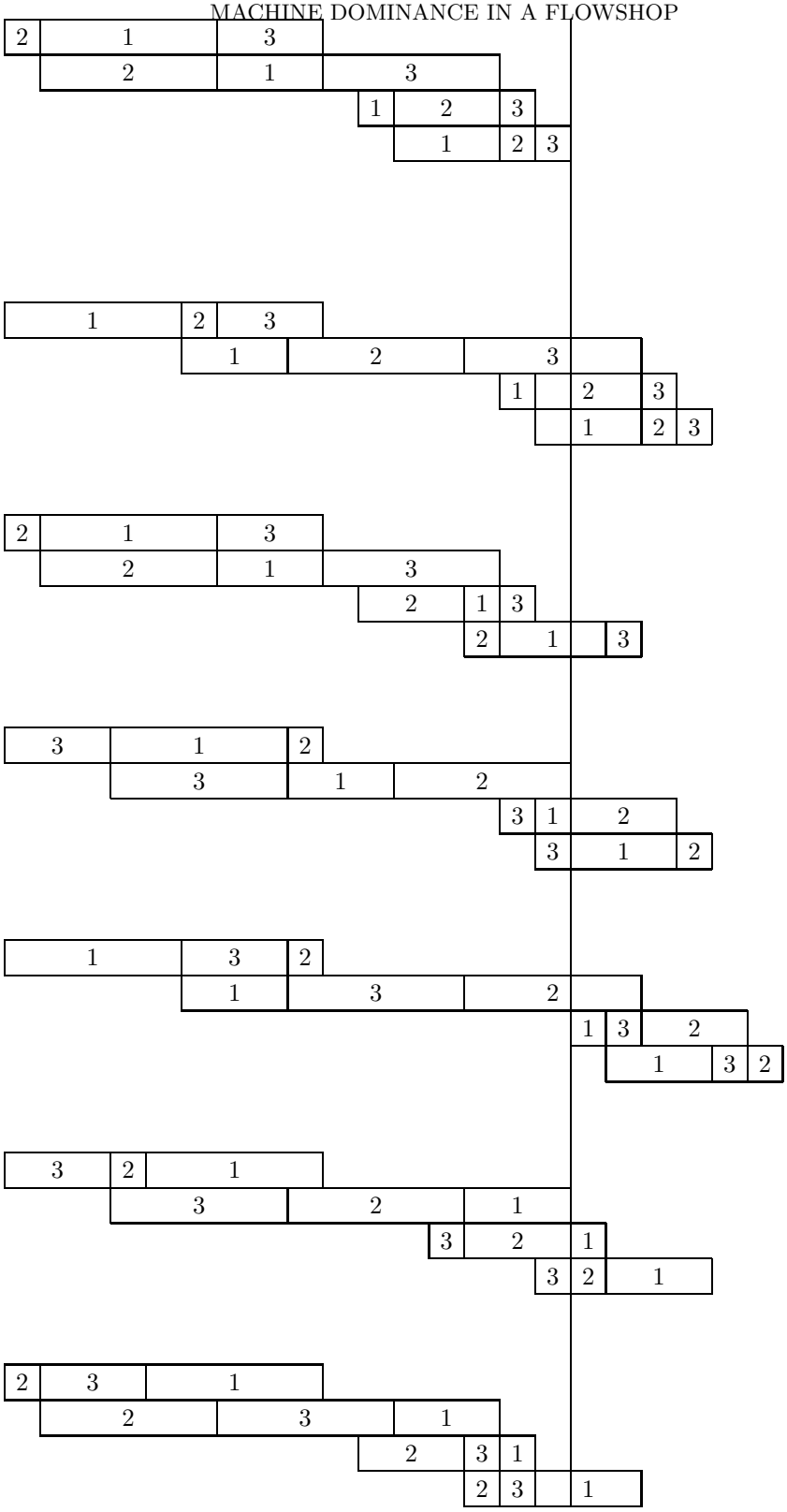

Figure 2: Counterexample for the $F|wM_\ell, nmit|C_{max}$ problem

is not enough to guarantee the feasibility of the permutation schedule constructed in the proof of Lemma 4.3. In fact, just the opposite of Lemma 4.3 is true.

Lemma 4.4 *The set of permutation schedules for the $F|wM_\ell, nmit|C_{max}$ problem does not always contain an optimal schedule.*

Proof. Let us define an instance of the $F|wM_\ell, nmit|C_{max}$ problem with $m = 4$ and $n = 3$ (three jobs on four machines) by

machine/job	J_1	J_2	J_3
M_1	5	1	3
M_2	3	5	5
M_3	1	3	1
M_4	3	1	1

It is easy to verify that this instance fulfils the wM_2 constraint, i.e. that machine M_2 is weakly dominant. To check this fact it is enough to verify the inequalities (1) and (2) in the definition of weak dominance, namely the inequality (1) for $r = 2$ and $r = 3$, and the inequality (2) for $r = 2$. We leave this to the reader as an easy excercise.

Now let us check Figure 2. The topmost non-permutation schedule is clearly optimal as it attains the lower bound (14) because both r_2 and t_3 are clearly minimal. On the other hand, all six permutation schedules have a strictly longer makespan than the topmost schedule, and hence no permutation schedule is optimal. ■

References

[1] Adiri, I. and Pohoryles, D. *Flowshop/No-idle or No-wait scheduling to minimize the sum of completion times*, Naval Research Logistics Quarterly 29, 495-504 (1982)

[2] K.R. Baker, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan [1983], *Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints*, Operations Research 31, (1983) 381-386.

[3] Čepek, O., Okada, M., and Vlach, M. *Nonpreemtive Flowshop Scheduling with Machine Dominance*, Research report IS-RR-98-0007F, Japan Advanced Institute of Science and Technology (1998).

[4] J. Du, J.Y.-T. Leung, *Minimizing total tardiness on one machine is NP-hard*, Mathematics of Operations Research 15, 3, (1990) 483-495.

[5] Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling*, Addison-Wesley, Reading, Mass.(1967)

[6] Garey, M. R. , Johnson, D. S., and Sethi, R. *The complexity of flowshop and jobshop scheduling*, Mathematics of Operations Research 1, 117-129 (1976)

[7] Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G. *Optimization and approximation in deterministic sequencing and scheduling*, a survey. Annals of Discrete Mathematics 5, 287-326 (1979)

[8] Gupta, J. N. D. *Optimal Schedules for Special Structure Flow-shops*, Naval Research Logistics Quarterly, Vol.22, 1975, pp. 255-269

[9] Ho, J. C. and Gupta, J. N. D. *Flowshop Scheduling with Dominant machines*, Computers Ops. Res., 22, No. 2, 237-246 (1995)

[10] Johnson, S. M. *Optimal Two-and Three-Stage Production Schedules with Setup Times Included*, Naval Research Logistics Quarterly, Vol. 1, No. 1 (March, 1954)

[11] R.M. Karp, *Reducibility among combinatorial problems*, in R.E. Miller, J.W. Thatcher: Complexity of Computer Computations, Plenum Press, New York, (1972) 85-103.

[12] E.L. Lawler, *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, Annals of Discrete Mathematics 2, (1978) 75-90.

[13] E.L. Lawler, *A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs*, Annals of Operations Research 26, (1990) 125-133.

[14] Lenstra, J. K., Rinnooy Kan A. H. G., and Brucker, P. *Complexity of machine scheduling problems*, Ann. Discrete Math., Vol.1, pp.343-362 (1977)

[15] Monma, C. L. and Rinnooy Kan, A. H. *A Concise Survey of Efficiently Solvable Special Cases of the Permutation Flow-Shop Problem*, RAIRO, Vol. 17, No. 2 105-119(1983)

[16] Nabeshima, I. *The Order of n Items Processed on m machines*, J. Operations Research Society Japan, Vol.3, 1961, pp. 170-175.

[17] van den Nouweland, A., Krabbenborg, M., and Potters, J. *Flow-shops with a dominant machine*, European Journal of Operational Research 62 (1992) 38-46.

[18] Tanaev, V.S., Sotskov, Y.N., and Strusevich, V.A. *Teoriia raspisanii. Mnogostadiinye sistemy*, Nauka, Moscow, 1989. English translation *Scheduling theory. Multi-stage systems*, Kluwer, Dordrecht, 1994.

Authors' affiliations:
Petr Čáp: Charles University, Praha, Czech Republic
Ondřej Čepek: corresponding author, Department of Theoretical Informatics and Mathematical Logic, Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic (e-mail: cepek@ksi.ms.mff.cuni.cz, phone: +420-221-914-246, fax: +420-221-914-323)
Milan Vlach: Charles University, Praha, Czech Republic