SCHEDULING UNIT – TIME TASKS ON FLOW – SHOPS UNDER RESOURCE CONSTRAINTS

J. BŁAZEWICZ [1], W. KUBIAK [2] and J. SZWARCFITER [3]

[1] *Instytut Automatyki, Politechnika Poznańska, Poznań, Poland*
[2] *Instytut Informatyki, Politechnika Gdańska, Gdańsk, Poland*
[3] *Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil*

Abstract

We consider the problem of scheduling tasks on flow shops when each task may also require the use of additional resources. It is assumed that all operations have unit lengths, the resource requirements are of 0-1 type and there is one type of the additional resource in the system. It is proved that when the number of machines is arbitrary, the problem of minimizing schedule length is NP-hard, even when only one unit of the additional resource is available in the system. On the other hand, when the number of machines is fixed, then the problem is solvable in polynomial time, even for an arbitrary number of resource units available. For the two machine case an $O(n \log_2^2 n)$ algorithm minimizing maximum lateness is also given. The presented results are also of importance in some message transmission systems.

Keywords: Scheduling, flow shops, resource constraints, message transmission

1. Introduction

The paper is concerned with the problem of nonpreemptive scheduling unit time operations on flow-shops when each operation may require additional scarce resources. Three essential elements of the model are: a set of n tasks each consisting of m operations, a set of m machines and a set of s additional resource types required by operations in integer amounts. Tasks may stand e.g. for programs, ships, building operations; machines-for processors, dockyards, building machines, and additional resources may represent primary or mass storage, channels, man-power or additional tools, etc. One of the main issues arising in the context of scheduling theory is the complexity analysis answering the question whether or not a given problem can be solved in polynomial-time. (We refer the reader to [7] where the NP-completeness theory is considered in detail.) The complexity investigation of resource constrained scheduling started in 1975 with a paper by Garey and Johnson [6]. Then, several papers were devoted to this subject [1–5,5,9–12]. However, problem of flow shop scheduling under resource constraints was considered in [5] and [9]. In the first paper, the

NP-hardness of the problem was proved for two machines, arbitrary processing times of the tasks and one resource type available in one unit, when schedule length was to be minimized. The same problem but with unit processing times was solved in $O(n)$ time. Several other results were presented in [9]. These include: an $O(n \log n)$ algorithm for two machines, unit processing time operations, one resource type and the schedule length criterion, and several NP-hardness proofs concerning unit time operations. Among them two problems of minimizing schedule length may be distinguished: scheduling on two machines with two resource types and scheduling on three machines with one resource type (in both cases resource requirements are arbitrary). Using the first of the two results and the theorem given in [4] one may also prove the NP-hardness of the same problem but with many resources each available in the amount of one unit. In [9] the problem of minimizing maximum lateness was also proved to be NP-hard for two machines and one resource type but with arbitrary resource requirements. Most of these results also hold for preemptive as well as no-wait cases. (The latter denotes an additional constraint on the way of task processing such that for any task whenever a processing of its operation on machine M_i, $i = 1, 2, \ldots, m-1$, finishes, immediately a processing of its next operation on machine M_{i+1} starts.) At this point let us also mention an interesting application of a no-wait resource constrained flow-shop scheduling in message transmission systems. Clearly, the first problem with m machines, r units of an additional resource, a set $\mathcal{T}$ of unit processing time tasks, and with schedule length criterion, corresponds to a problem of transmitting a set $\mathcal{T}$ of m-bit words over a line which allows r bits equal to 1 to be sent in parallel only.

Following the above overview we see that several problems still remain open. This is especially true when considering scheduling problems with unit-time operations, one resource type and zero-one resource requirements (i.e. operations may require either zero or one unit of the additional resource). In the present paper, we address ourselves to some of these problems. In particular, we prove that the problem of minimizing schedule length with an arbitrary number of machines is NP-hard even for the case when the additional resource is available in the amount of one unit. This result holds also for any other scheduling criterion. On the other hand, when the number of machines becomes fixed, the above problem may be solved in polynomial time (via a dynamic programming approach) for an arbitrary amount of resource units available. The presented approach has a more general meaning. Whenever the number of different task types (differing by processing times, resource requirements, etc.) is fixed, it will enable to solve the problem of minimizing schedule length. Finally, the problem of minimizing maximum lateness will be considered and an $O(n \log^2 n)$ algorithm for two machines and one unit of the additional resource will be given. The method is also more general and may be applied for some other unit-time task scheduling problems with the maximum lateness criterion. Before doing this we will set up the subject more precisely.

We are given a set of n *tasks* $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ each of which consists of m operations, i.e. task T_j consists of the set of operations $\mathcal{O}_j = \{O_{1j}, O_{2j}, \ldots, O_{mj}\}$. Operations are to be processed on the set of m *machines* $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ such that for each j, O_{ij} is processed on machine M_i, $i = 1, 2, \ldots, m$. An *additional resource* R with limit r is also available in the system. Each operation O_{ij}, $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$, requires for its processing machine M_i and the additional resource in the amount specified by $R(O_{ij})$ (which may be equal to 0 or to 1). A column vector describing the resource requirements of operations constituting task T_j will be denoted by $\bar{R}(T_j)$. Operations must be processed nonpreemptively, i.e. each operation once assigned to a processor must be processed without any break till its completion. If the same constraint is imposed on the way in which tasks are processed (i.e. each task once started must be processed till the completion of its last operation without any break) the schedule will be said to have the *no-wait* property. All the tasks are independent and the operations of a particular task T_j, $j = 1, 2, \ldots, n$, are processed in such a way that O_{ij} precedes O_{2j} which precedes O_{3j} and so on until O_{mj} is completed. Each operation O_{ij} is characterized by unit *processing time* $p(O_{ij}) = 1$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$. Whenever maximum lateness is to be minimized, we will assume that each task T_j is assigned a *due date* $d(T_j)$, i.e. a time by which the last operation of the task (O_{mj}) is to be completed.

A *feasible schedule* is a function s: $\mathcal{O} \to Z$ (Z is a set of nonnegative integers), which obeys the following conditions:

(a) For each integer t, $t \geqslant 0$,

$$S(t) \leqslant m$$

where $S(t) = \{O_{ij} \in \mathcal{O}: s(O_{ij}) = t\}$ is the set of operations assigned to machines at time t.

(b) For each integer t, $t \geqslant 0$,

$$\sum_{O_{ij} \in S(t)} R(O_{ij}) \leqslant r.$$

(c) For each integer t, $t \geqslant 0$, and $i = 1, 2, \ldots, m$,

$$|\{O_{ij}: O_{ij} \in S(t),\ j = 1, 2, \ldots, n\}| \leqslant 1.$$

(d) For each O_{ij}, $i = 1, 2, \ldots, m-1$; $j = 1, 2, \ldots, n$

$$s(O_{ij}) \leqslant s(O_{i+1,j}) - 1.$$

Whenever maximum lateness $L_{\max}$ is to be minimized we add the following condition to express feasibility for a given value L' of $L_{\max}$:

(e) For each T_j, $s(O_{mj}) + 1 \leqslant d(T_j) + L'$.

For a given schedule, let $c(O_{ij})$ denote the completion time of operation O_{ij}, i.e. $c(O_{ij}) = s(O_{ij}) + 1$.

We will be interested in two criteria when evaluating schedules: the *schedule length* defined as $C_{max} = \max\{c(O_{ij})\}$ and the *maximum lateness* defined as $\max_j\{c(O_{mj}) - d(T_j)\}$.

In the paper, we will use the notation defined in [5,8] for the short and concise classification of scheduling problems. The three problems considered in the paper are denoted as follows. $F\,|\text{res}\,111,\ p_j = 1\,|\,C_{max}$ whose NP-hardness is established in sect. 2. $Fm\,|\text{res}\,1\cdot 1,\ p_{ij} = 1\,|\,C_{max}$ for which a dynamic programming approach is presented in sect. 3. Finally, $F2\,|\text{res}\,111,\ p_{ij} = 1\,|\,L_{max}$ which is solved by an $O(n \log^2 n)$ algorithm in sect. 4.

2. NP-hardness of problem $F\,|\text{res}\,111,\ p_{ij} = 1\,|\,C_{max}$

In this section we will prove that the flow shop scheduling problem with, an arbitrary number of machines, unit-time, nonpreemptive operations, a unit of an additional resource and with C_{max} criterion, is NP-hard, thus unlikely to admit a polynomial time algorithms. The proof is similar to that of theorem 1 in [9], thus we give here only its outline, leaving details to the reader.

It is sufficient to prove the NP-hardness of the decision counter part (denoted by Π_1) of $F\,|\text{res}\,111,\ p_{ij} = 1\,|\,C_{max}$. As a known NP-complete problem (denoted by Π_2) we will use the *Hamiltonian Path* problem ([7], [9]). For a given instance of Π_2 defined by a graph $G = (V, E)$, let $q = |V|$, $z = 2(q(q-1)+1)$ and let a corresponding instance of Π_1 be defined by assuming:

- A number of $m = 2z$ machines $M_1, \ldots, M_m$. Among them, the odd machines M_i, $i = 3, 5, \ldots, z-1$ are labeled respectively with the pairs (k, l) in lexicographical ordering, $1 \leqslant k,\ l \leqslant q$ and $k \neq l$, that is using labels $(1, 2), (1, 3), \ldots, (2, 1), \ldots, (q, q-1)$. The same labels are applied to the odd machines M_{i+z}; where machine M_{i+z} receives the same label as M_i.
- A number of $n = (q\text{-}1)z + 1$ tasks $T_1, T_2, \ldots, T_n$. The tasks $T_1, T_2, \ldots, T_q$ are called vertices and the resource requirements of their operations are defined as follows:

Machine i, unlabeled	$R(O_{ij})$
1	1
$2, 4, \ldots, z$	1
$z+1$	0
$z+2, z+4, \ldots, 2z$	0
Machine i, labeled (k, l)	**$R(O_{ij})$**
$j \notin \{k, l\}$	0
$j \in \{k, l\}$ and $(k, l) \in E$	0
$j \in \{k, l\}$ and $(k, l) \notin E$	1

The remaining $(q-1)(z-1)$ tasks T_j, $j > q$ have their operations with zero resource requirements $R(O_{ij}) = 0$, for $i = 1, 2, \ldots, m$.

– The threshold value for the schedule length is $(q+1)z$.

Now, it is sufficient to note that in the above construction the resource requirements of the odd operations of vertex i are nothing else but entries of the i-th row and the i-th column of the adjacency matrix of G. Moreover, standard arguments show that each schedule, which can be taken into account in consideration of the threshold value, is a no-wait schedule with z unit times between any two consecutive vertices.

Careful inspection of the above proof shows that for other criteria such as mean flow time, maximum lateness, etc., and for preemptive as well as no-wait modes of processing the above problem remains NP-hard as well. In the next section we will show, however, that fixing the number of machines m, allows one to solve the problem in polynomial time.

3. A dynamic programming approach to solving $Fm|\text{res } 1 \cdot 1, p_{ij}=1|C_{\max}$

Below we will show that if the number of machines is fixed, then the problem considered in the last section may be solved in polynomial time, even for the case when more than one unit of the additional resource is available in the system.We will employ a dynamic programming approach. The problem to be solved is somewhat more general than the formulated flow shop problem. Instead of starting from machine M_1, the operations of T_j constitute any sequence $O_{k_j j}, O_{k_j+1,j}, \ldots, O_{mj}$, where $k_j \in Z_m^+$. * $O_{k_j j}$ is called the *head* of task (sequence) T_j. Now we will explain the idea of the algorithm. The following definitions will be used in order to do that.

Two sequences belong to the same *class of sequences* if they have the same number of operations, and operations processed on the same machine have the same resource requirements in both sequences. If m is the number of machines then the number of classes of sequences c is equal to $2^{m+1}-1$.

Two instances belong to the same *class of instances* if they contain the same number of sequences in each class of sequences. It is not difficult to see that if m is fixed then c is fixed and the number of classes of instances, with at most n sequences, is less than n^c.

Let I be an instance of the more general problem defined above and $H(I)$ the set of heads of I. A *feasible starting set* for I is a nonempty subset of heads $H \subseteq H(I)$, that satisfies

(i) if $O_{ki} \in H$ and $O_{kj} \in H$ then $i=j$.

(ii) $\sum_{O_{ki} \in H} R(O_{ki}) \leqslant r$.

In other words, H is a subset of heads which can start a feasible schedule for I.

* $Z_k^+ = \{1, 2, \ldots, k\}$ and $Z_k = \{0, 1, \ldots, k\}$.

Let H be a feasible starting set for I. Denote by $\alpha_H(I)$ the instance which is obtained by deleting from the sequences of I the heads from H. The instance $\alpha_H(I)$ will be called a *direct successor* of I. Two feasible starting sets for I, H and H' belong to the same *class of feasible starting sets* for I if the instances $\alpha_H(I)$ and $\alpha_{H'}(I)$ belong to the same class of instances. It is easy to see that there is at most $\sum_{1 \leq i \leq m}(c_i) = d$ classes of instances which contain direct successors of I. Obviously, if m is fixed then d is fixed as well.

The above considerations practically show how to employ a dynamic programming approach in order to get a polynomial time algorithm. A class of instances can be written as a *class vector* from $Z_n \times \cdots \times Z_n - c$ times. The entries of a class vector show how large are classes of sequences in a class of instances. Then a problem state can be defined as a pair consisting of a class vector and a non-negative integer which can be interpreted as the starting moment for the next feasible starting set. The problem states being direct successors of the problem state (I, t) are closely related to the classes of feasible starting sets for I and can be obtained if the classes are generated. Moreover, t is equal to $t' + 1$, where t' is the starting moment for the "earliest" direct predecessor of the problem state (I, t). It is sufficient to generate $O(n^{c+1})$ problem states in order to obtain a problem state with the length of an optimal schedule. Then, backtracking along the best path of decisions allows for the construction of an optimal schedule. The details are left to the reader.

It is easy to see that the time complexity of the algorithm is $O(n^{c+1}\log n)$. It should be noted that the time of generating any problem state is $O(\log n)$. Obviously, it is, a polynomial time algorithm if m is fixed.

4. An $O(n \log^2 n)$ algorithm of $F2 \mid \text{res } 111, p_{ij} = 1 \mid L_{\max}$

In this section we will consider the problem of minimizing maximum lateness in a two-machine flow shop with unit time nonpreemptable operations and a unit of an additional resource available in the system. In the approach presented, two stages can be distinguished. The first stage consists in checking for feasibility, i.e. answering the question whether or not there exists a feasible schedule with no task late for a given set of due dates (the feasibility problem is denoted by $F2 \mid \text{res } 111, p_{ij} = 1, d_j \mid -$ [8,5]). The method here assigns tasks to baskets denoted by particular due dates. Then, starting from the end of the schedule, it tries to assign in consecutive time slots tasks available in corresponding baskets, so that there is no idle time for the machines and the additional resource. The second stage consists in applying a binary search procedure which uses the above method and changes the trial values of due dates and $L_{\max}$, thus producing an optimal schedule. Below we start with the first algorithm which solves the problem $F2 \mid \text{res } 111, p_{ij} = 1, d_j \mid -$.

Let us denote $d_{\min} = \min_{T_j}\{d(T_j)\}$. To avoid pseudopolynomiality of our

algorithm we have to modify the due dates for all the tasks in the following way:

$$d(T_j) := \begin{cases} d_{\min} + 2n, & \text{if } d(T_j) > d_{\min} + 2n, \\ d(T_j), & \text{otherwise.} \end{cases}$$

Then, the value $d_{\max}$ is defined as $d_{\max} = \max_{T_j \in \mathscr{T}}\{d(T_j)\}$. The baskets $\mathscr{T}_j$ are now defined in the following way:

$$\mathscr{T}_j = \{T \in \mathscr{T}: d(T) = j\}, \quad j = d_{\min}, d_{\min}+1, \ldots, d_{\max}.$$

For brevity we will also denote an operation not requiring the additional resource by an operation of type 0 and the one requiring a unit of this resource by an operation of type 1. For the same reasons the four types of tasks will be denoted respectively by 1^0, 0^1,1^1, and 0^0. The first symbol denotes a task T_j with $R(O_{1j}) = 1$ and $R(O_{2j}) = 0$ and the remaining symbols the other corresponding types of tasks.

The following algorithm assigns the tasks to consecutive time slots starting from $d_{\max}$. To make the assignment rules uniform an additional set D of 0 type dummy operations to be processed on M_1, is added. The algorithm uses an additional procedure IDLE TIME which checks whether or not it is possible to assign tasks from $\mathscr{T}$ and dummy operations from D to machines in interval $[0, d_{\max}]$ in such a way that no idle time on M_1 appears. (In the case when such a schedule exists, a Boolean variable IT takes the value "no"). Now the algorithm can be described. Its input is set $\mathscr{T}$ and its output is answer "yes" (plus the schedule) in the case when a schedule with no task late exists for $F2|\text{res } 111, p_{ij} = 1, d_j|-$, or "no" in the opposite case.

ALGORITHM 1

1. If $\min_{T \in \mathscr{T}}\{d(T)\} \geqslant 2n$, then the answer is "yes". Schedule the tasks one by one in the time interval $[0, 2n]$. End the algorithm. Otherwise go to step 2.
2. If $d_{\max} \leqslant n$, then the answer is "no". End the algorithm. Otherwise go to step 3.
3. Set $D := \{d_{\max} - n$ dummy operations of type 0, to be processed on $M_1\}$.
4. Call IDLE TIME $(D, \mathscr{T})$. If IT = "yes", then the answer is "no". End the algorithm.

Otherwise the answer is "yes". Remove dummy operations from the obtained schedule. End the algorithm. □

The procedure IDLE TIME takes as the input the sets D and $\mathscr{T}$ and its output is "yes" if there is an idle time on M_1 when scheduling D and $\mathscr{T}$, and "no" otherwise. Its description is as follows.

PROCEDURE IDLE TIME $(D, \mathscr{T})$

1. Set $t := d_{\max}$, $B := \mathscr{T}_t$.

Construct a list ZERO, consisting of the operations of type 0, ZERO := D.

Construct a list ONE consisting of the operations of type 1, initially empty.

2. If either ONE or ZERO is not empty, then choose an operation from any list and a task from set B in the following order (first the operation appears, then the task chosen): $1\ 1^0 \to 0\ 0^1 \to 0\ 1^1 \to 1\ 0^0 \to 0\ 1^0 \to 0\ 0^0 \to 1$ (a task is not chosen) $\to 0$ (B is empty). Go to step 3.

Otherwise, i.e. if ONE and ZERO are empty, go to step 4.

3. Remove the chosen operation from its appropriate list and the chosen task (if any) from set B. Assign at time $t-1$ the chosen operation to M_1 and the second operation of the chosen task (if any) to M_2. The first operation of that task is added to the beginning of the appropriate list (i.e. if $R(O_{ij}) = 1$, then O_{ij} is added to ONE). Set $t := t-1$, $B := B \cup \mathscr{T}_t$ and go to step 2.

4. If all tasks have been assigned, set IT := "no", otherwise set IT := "yes". End the procedure. □

The uniqueness of the choices made by the procedure stems from the following rules.

(i) Operations are taken from the beginning of the lists, except for the case when no task is chosen. In the latter case an operation from the end of the list is chosen.

(ii) Tasks of the same type are chosen in nonincreasing order of their due dates.

Before presenting some lemmas that prove the correctness of algorithm 1 we need some more definitions. For a given schedule S and time moment $t = 1, 2, \ldots, d_{\max} - 1$, let $C(t, s)$ [ZERO(t, S)] denote a set of operations [a set of operations of type 0] to be processed on M_1, not assigned in interval $[t, d_{\max}]$, being dummy or belonging to tasks whose second operation has been assigned in interval $[t, d_{\max}]$. Moreover, let $\mathscr{T}(t, S) := \{T \in \mathscr{T}: d(T) > t$ and t has not been assigned in interval $[t, d_{\max}]$ in $S\}$. Now we may give the lemma. The first lemma contains some quite simple observations and we present it omitting a proof.

LEMMA 1

Let $\mathscr{T}$ and D be respectively a set of tasks and a set of dummy operations. Then for any schedule for $\mathscr{T}$ and D with no task late, and without an idle time on M_1 in time interval $[0, d_{\max}]$, there exists schedule S with the same properties, which in addition fulfils the following conditions:

(i) tasks of the same type are processed on M_2 in increasing order of their due dates;

(ii) if x and y are operations of type 0 to be processed on M_1 and $s(x) > s(y)$ in S and $y \notin D$, then $s(z) \leqslant s(x)$, where y and z compose one task;

(iii) if for some time t no machine processes an operation of type 1, then $d(T) \leqslant t$ for each task of type 1^1 or 0^1 completed in S before t, and $C(t+1, S)$ contains no operation of type 1;

(iv) if there exists time t in which M_2 performs a task of type 1^1 (resp. 0^0) (resp.

has an idle time), then $d(T) \leqslant t$ for each task of type 0^1 (resp. 1^0) (resp. 0^0 and 1^0) which has been completed in S before t. □

The second lemma relates the schedules produced by procedure IDLE TIME to other schedules with no task late.

LEMMA 2

If the Boolean variable IT = "yes" after performing procedure IDLE TIME for sets $\mathcal{T}$ and D, then there is no schedule for $\mathcal{T}$ and D with no task late and without and idle time on M_1 in time interval $[0, d_{\max}]$.

Proof

Suppose on the contrary that there exists a schedule S for $\mathcal{T}$ and D with no task late and without an idle time on M_1, despite the fact that IT = "yes". Using this assumption, we will change S step by step so that no idle time will appear on M_1 and after k steps S and the partial schedule P obtained by procedure IDLE TIME, will be consistent in time interval $[d_{\max} - k, d_{\max}]$, thus leading to a contradiction.

According to lemma 1, we may assume that S obeys conditions (i) through (iv). Without loss of generality we may also assume that in P if M_1 performs an operation x of type 1 and M_2 has an idle time, then the second operation of the same task is performed immediately after the completion of x.

Now taking into account the above assumptions and the way in which tasks are assigned in procedure IDLE TIME, we have to consider only two cases in which P and S may differ.

CASE 1

If in S $s(x) \leqslant s(y)$ (see fig. 1), where x is the first operation to be assigned out of those composing set ZERO (τ, S) then the way of changing S is obvious. Hence, suppose $s(x) > s(y)$. According to condition (i) in lemma 1, in time

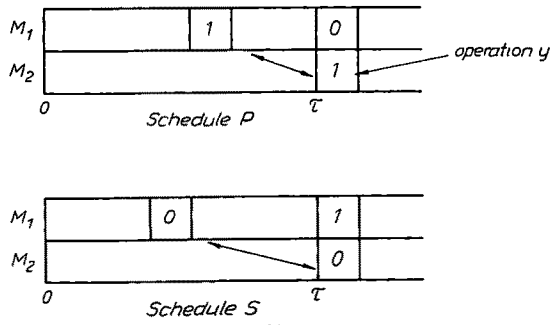

Fig. 1.

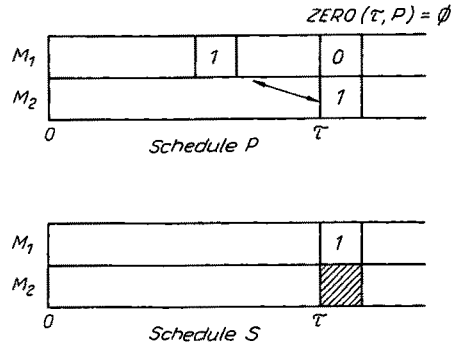

Fig. 2.

interval $[s(y)+1, \tau]$ in S machine M_2 cannot perform an operation of a task of type 1^1 (cf. schedule P and the fact that in time interval $[\tau+1, d_{\max}]$ both schedules are consistent). Taking additionally into account condition (iii), it follows that any operation of type 0 performed on M_1 in time interval $[s(y)+1, \tau]$ is performed in parallel with the second operation of a task of type 0^1. Thus, following condition (ii) we get for each t, $s(y)+1 \leqslant t \leqslant \tau$, that set ZERO (t, S) contains at least one element from set ZERO (τ, S), which leads to a contradiction. Hence, we get that P and S are consistent in time interval $[\tau, \tau+1]$. Moreover, taking into account the above considerations, we assume that in S if at time t M_1 performs an operation of type 1 and M_2 a task of type 0^0 and ZERO $(t+1, S) \neq \emptyset$, then there is no task of type 1^1 such that $d(T) \geqslant t$.

CASE 2

Let $\tau_i < \tau$ be the latest time in S in which M_2 performs a task of type 1^1 (cf. fig. 2). Since ZERO $(\tau, S) \neq 0$, then for each t, $\tau_1 < t \leqslant \tau$, ZERO $(t, S) \neq \emptyset$. Thus, in S M_2 performs only tasks of types 1^0 or 0^1 at each time t, $\tau_1 + 1 \leqslant t \leqslant \tau$. Moreover, since ZERO $(\tau, P) = \emptyset$ and $|C(\tau, P)| \geqslant 2$, then in P at each time moment t, $\tau^1 < t < \tau$, M_2 performs tasks of type 1^0, where τ^1 is the latest moment, $\tau^1 < \tau$, in which M_2 performs a task of type 0^0 in P (if such a moment exists). It is not hard to see that there must be such a time moment, if there exists time moment τ^2, $\tau^2 < \tau$, in which M_2 performs in P a task of type 0^1 or 1^1 (in this case $\tau^2 < \tau^1$). Hence, at each moment t, $\tau_1 \leqslant t \leqslant \tau$, M_2 performs in P only tasks of types 1^0, 0^0 or 0^1. Moreover, there is no idle time on either machine in time interval $[\tau_1, \tau)$ in S. It follows from the above considerations that if for each task T of type 1^0 or 0^1 completed in S by τ_1, $d(T) \leqslant \tau_1$, then $\mathscr{T}(\tau_1, S) = \mathscr{T}(\tau_1, P) \cup \{T\}$ where T is of type 0^0 and $|\text{ZERO}(\tau_1, P)| = 1$. Let us consider the opposite case. Then set $\mathscr{T}(\tau_1, S)$ contains a task of type 1^0 and does not contain a task of type 0^1.

Let τ_2, $\tau_2 < \tau_1$, be the latest time moment in S such that M_2 performs a task of type 1^0 and set $\mathscr{T}(\tau_2, S)$ does not contain a task of type 1^0. Since ZERO

$(\tau_1, S) = \emptyset$, then at each moment t, $\tau_2 \leqslant t < \tau_1$, M_2 performs a task of type 1^0 in S. It follows that at each moment t, $\tau_2 \leqslant t < \tau$, M_2 performs in P only tasks of types 1^0, 0^1 or 0^0 (exactly one task of that type is processed). Thus, $\mathscr{T}(\tau_2, S) = \mathscr{T}(\tau_2, P) \cup \{T\}$ where T is of type 0^0 and $|\text{ZERO}(\tau_2, P)| = 1$.

Now, it is not hard to show the way in which one has to change the assignment of tasks in S in time interval $[\tau_1, \tau]$ ($[\tau_2, \tau]$, respectively), to obtain the desired consistency. □

Finally, the following lemma establishes the correctness of algorithm 1.

LEMMA 3

Algorithm 1 constructs a schedule with no task late for a given set of tasks, whenever such a schedule exists.

Proof

It is sufficient to note that if IT = "yes" for $\mathscr{T}$ and D such that $|D| = d_{\max} - n$, then lemma 2 assures us that there is no schedule without delayed tasks for set $\mathscr{T}$. □

Now we may proceed to the minimization problem $F2|\text{res } 111, p_{ij} = 1|L_{\max}$. A binary search procedure conducted on the value $L_{\max}$ (and the adjusted values of task due dates) will give us an optimal schedule after $O(\log_2 n)$ repetitions of algorithm 1. The overall complexity is thus $O(n \log_2^2 n)$ since algorithm 1 can be easily implemented to run in $O(n \log_2 n)$ time.

Let us note that the presented approach is of more general nature, since its idea may also be used in solving problems with parallel machines and in open shops whenever resource requirements are of 0-1 type and tasks (or operations) are of unit length.

Acknowledgment

The authors gratefully acknowledge helpful comments made by Hans Röck on the first draft of this paper.

References

[1] J. Błażewicz, Deadline scheduling of tasks with ready times and resource constraints, Inform. Process. Lett. 8 (1979) 60–63.

[2] J. Błażewicz, J. Barcelo, W. Kubiak and H. Röck, Scheduling tasks on two processors with deadlines and additional resources. European J. on the Oper. Res. 26 (1986) 364–370.

[3] J. Błażewicz, W. Cellary, R. Słowiński, J. Weglarz, *Scheduling under Resource Constraints: Deterministic Models*, Annuals of Operations Research 7 (J.C. Baltzer, Basel, 1986).

[4] J. Błażewicz, W. Kubiak, H. Röck and J. Szwarcfiter, Minimizing mean flow time under resource constraints on parallel processors, Acta Informatica 24 (1987) 513–524.
[5] J. Błażewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. 5 (1983) 11–24.
[6] M.R. Garey and D.S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. on Comput. 4 (1975) 397–411.
[7] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman, San Francisco, 1979).
[8] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[9] H. Röck, Some new results in flow-shop scheduling, Zeitschrift für Oper. Res. 28 (1984) 1–16.
[10] R. Słowiński, L'ordonnancement des tâches préemptives sur les processeurs indépendants en présence de réssources supplémentaires, RAIRO Informat. 15 (1981) 155–156.
[11] J.D. Ullman, Complexity of sequencing problems, in: *Computers and Job/Shop Scheduling Theory*, ed. E.G. Coffman, Jr. (J. Wiley, New York, 1976).
[12] D. de Werra, Preemptive scheduling linear programming and network flows, Siam J. Algebr. and Discrete Meth. 5 (1984) 11–20.