JUST-IN-TIME SCHEDULING WITH PERIODIC TIME SLOTS*

Ondřej Čepek[a†] and Shao Chin Sung[b]

Received December May 12, 2003; revised February 5, 2004

Abstract. In this paper we study a problem with periodic time slots in which a processing time and periodically repeating due dates are given for each job, where the period is the same for all jobs. The objective is to minimize the number of periodic time slots in which all jobs can be scheduled just-in-time. We show that when set-up times between jobs are considered, the problem becomes unary NP-hard even in the single machine case. When set-up times are not considered, we study several subproblems which arise by putting additional restriction on processing times. We show that under a rather strict (although natural) restriction the problem is solvable in polynomial time for an arbitrary number of identical parallel machines, and we present a simple polynomial time algorithm solving the problem. We also show, that when the restriction on processing times is lifted, the problem without set-up times becomes binary NP-hard already for two machines and unary NP-hard when the number of machines is a part of the input.

1 Introduction In this paper we study a problem with periodic time slots, in which a processing time and periodically repeating due dates are assigned to each job, where the period (i.e., the time difference between consecutive due dates) is the same for all jobs. The objective is to minimize the number of periodic time slots which are sufficient for scheduling all given jobs just-in-time (i.e. each job is completed exactly on one of its due dates).

This problem is motivated by manufacturing environments with a periodic production cycle (day, week), where a due date of a job corresponds to a periodically repeating event (e.g. arrival of a truck which picks up the production). The problem was first studied by Hiraishi [2] in the environment of identical parallel machines. The following two additional assumptions were considered:

- a nonnegative set-up time is given for every (ordered) pair of jobs, which means that when two jobs are scheduled consecutively, an idle time of (at least) the given length must be scheduled in-between them, and
- for each job, its processing time is less than or equal to its (first) due date, which implies that every job is scheduled within a single time slot (cannot span over several neighboring slots)

In [2] a heuristic algorithm for finding a "good" schedule was proposed. In this note we shall study this problem further (under the same assumptions), and we shall show that the presence of set-up times makes the problem unary NP-hard even in the case of a single machine, unit processing times, and only two different lengths of set-up times. Clearly, two is the smallest possible number of lengths to consider, because when all set-up times are the same, they can be simply added to processing times, which constructs an equivalent

2000 *Mathematics Subject Classification.* 90B35, 90C27.

Key words and phrases. just-in-time scheduling, polynomial time algorithm, NP-hardness.

*This work was supported by the Czech Science Foundation (grant 201/04/1102)

†Corresponding author. Tel: +420 221914246 Email: cepek@ksi.ms.mff.cuni.cz (O. Čepek)

problem without set-up times. Thus, in order to establish polynomially solvable cases of the problem, it is natural to concentrate on the setting with no set-up times (or equivalently with all set-up times equal to zero).

Next we shall show, that when set-up times are removed from the problem studied in [2] (and the restriction on processing times is kept in place), the problem becomes solvable in polynomial time. We present a simple $O(n^2)$ greedy algorithm solving this restricted problem for an arbitrary number of machines, where n is the number of jobs.

Finally, we shall lift the restriction on the processing times (and keep all set-up times zero) and prove, that in such a case the problem becomes binary NP-hard already for two machines, and unary NP-hard when the number of machines is not fixed (i.e., when the number of machines is a part of the input).

2 Problem formulation There are m parallel identical machines $M_1, M_2, \ldots, M_m$ and n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$. The operation time on all machines is divided into slots of length L, and each job is assigned a set of periodically repeating due dates, one per each time slot, namely job J_j has due dates $d_j, L + d_j, 2L + d_j, \ldots$ where $0 < d_j \leq L$. Moreover, for each pair of jobs J_k, J_ℓ, where $k \neq \ell$, a set-up time $0 \leq t_{k\ell} \leq L$ is considered, which means that if J_k and J_ℓ are scheduled on some machine M_i in the order that J_ℓ is started after J_k is completed, then the time difference between the completion of J_k and the start of J_ℓ must be at least $t_{k\ell}$. The problem is to find a feasible nonpreemptive schedule in which each job is completed exactly on one of its due dates, and which minimizes the number of time slots required for all jobs to be processed. The feasibility of schedules and the objective function can be described formally as follows.

A ***schedule*** is a mapping which assigns to every job J_j an ordered pair $(M_{[j]}, C_j)$, where $M_{[j]}$ is a machine on which job J_j is processed, and C_j is its completion time, i.e. J_j is processed on $M_{[j]}$ in the time interval $[C_j - p_j, C_j)$. A schedule is called ***feasible*** if

- for every pair of jobs J_k, J_ℓ, where $k \neq \ell$, if $M_{[k]} = M_{[\ell]}$ (both jobs are assigned to the same machine), then

$$\text{either} \quad C_k \geq C_\ell + t_{\ell k} + p_k \quad \text{or} \quad C_\ell \geq C_k + t_{k\ell} + p_\ell, \tag{1}$$

- for each job J_j there exists a nonnegative integer r_j such that $C_j = r_j \cdot L + d_j$ (i.e. each job is completed exactly on one of its due dates).

Let us denote $R = \max\{r_1, r_2, \ldots, r_n\}$. Then the objective of the problem is to minimize R over the set of all feasible schedules.

An additional constraint on the input data was considered in [2]. It was assumed that $p_j \leq d_j$ holds for every job J_j. This means that if a job is scheduled just-in-time, it completely fits within a single time slot. This constraint is a very natural one in practice. We shall also assume its validity in the next two sections.

In [2], a heuristic algorithm for finding a "good" schedule with respect to the minimization of the value of R was presented. In the next section we shall show, that in fact this problem is unary NP-hard even for the single machine case (i.e. for $m = 1$), where moreover all jobs are assumed to have the same length.

3 Unary NP-Hardness for $m = 1$ with set-up times and unit time jobs In order to prove the above mentioned intractability result, we shall use for the reduction the well-known unary (strongly) NP-complete Hamiltonian path problem (see e.g. [1]), which is defined as follows.

Hamiltonian Path (HP)

Instance: An undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$.

Question: Is there a Hamiltonian path in G, i.e. does there exist a permutation of vertices $v_{[1]}, \ldots v_{[n]}$ such that $(v_{[i]}, v_{[i+1]}) \in E$ for every $1 \leq i \leq n-1$?

Now we shall construct an instance of a decision version of our just-in-time scheduling problem, so that this instance has a solution if and only if the input instance of the Hamiltonian path problem has a solution.

Just-in-time Scheduling with Set-up Time (JSST)

Instance: $m = 1$, $L = 2$, $J_1, \ldots, J_n$ (identify jobs with vertices of the input graph), $p_j = d_j = 2$ for all $1 \leq j \leq n$, and

$$t_{ij} = \begin{cases} 0 & \text{if } (i,j) \in E, \\ 1 & \text{otherwise.} \end{cases}$$

Question: Is there a feasible schedule which occupies at most n time slots, i.e. such that $R \leq n$?

Lemma 1 *The input HP instance has a solution if and only if the constructed JSST instance has a solution.*

Proof. If HP has a solution, then we easily construct a solution of JSST by scheduling the jobs in the order given by the Hamiltonian path. All set-up times between consecutive jobs are in such a case zero, and so the constructed schedule fits into exactly n time slots. On the other hand if JSST has a solution (in such a case exactly n time slots are occupied as it is impossible to schedule n unit time jobs in less than n unit time slots) then the sequence in which jobs are scheduled gives a Hamiltonian path, since all set-up times between consecutive jobs must be in such a case zero, and so this order indeed gives a path in the input graph. ■

Since the above reduction is clearly polynomial, and all processing times, due dates, and set-up times are nonnegative integers bounded by a constant (namely two), we obtain the following theorem.

Theorem 1 *JSST is unary NP-hard.*

Notice that JSST remains unary NP-hard even if the feasibility constraint (1) is required only for pairs of jobs which are scheduled consecutively. This fact follows from the easy observation that the feasibility constraint is fulfilled automatically for all other (non-consecutively scheduled) pairs of jobs in the above presented reduction (the length of every job is larger then the length of any set-up time).

4 Polynomiality for arbitrary m under $p_j \leq d_j$ constraint In the following we shall consider the case when all set-up times are zero, and hence do not in any way influence the feasibility of schedules.

In the absence of set-up times the $p_j \leq d_j$ constraint, i.e. the constraint that in every feasible schedule no job spans over more than one time slot, causes that the schedule in each individual time slot on each individual machine constitutes a "block" which can be moved around freely in the schedule (both by permuting blocks on one machine and by moving blocks from one machine to another) without spoiling the feasibility of the resulting schedule.

This observation leads to the following idea. Rather than solving the original problem of minimizing the number of used time slots on a fixed number of machines, we shall solve an equivalent problem of scheduling all jobs in a single time slot on a minimum number of machines. The correspondence between these two problems is obvious. Given a one-slot schedule on a minimum possible number, say q, of machines, one can easily construct a schedule on a fixed number m of machines that occupies $\lceil q/m \rceil$ time slots, and this is the best possible value.

So by solving the new minimization problem (minimizing the number of machines with one slot) we also solve the original one (minimizing the number of used time slots on a fixed number of machines). The new problem can be solved by the following greedy algorithm.

Algorithm MinNM**:**

Phase 1. Sort jobs increasingly by $s_i = d_i - p_i$ and renumber them accordingly, i.e., after renumbering we have

$$s_1 \leq s_2 \leq \ldots \leq s_n.$$

Phase 2. Take jobs one by one from the sorted sequence and when processing job J_j find the first available machine M_i, i.e. a machine with the smallest index such that no job (from the set of previously scheduled ones) is processed on it at time s_j, and schedule J_j on M_i in the time interval $[s_j, d_j)$.

Obviously, placing each job takes time bounded by the number of machines used up until that moment, and moreover the total number q of all machines used up by the algorithm clearly fulfills $q \leq n$. Thus, we have the following proposition.

Proposition 1 *Algorithm* MaxNM *runs in* $O(n^2)$ *time.*

Notice that the running time of algorithm MinNM does not depend on the number of machines m in the original problem which minimizes number of time slots. The correctness of the algorithm is given by the following theorem.

Theorem 2 *Algorithm* MinNM *produces an optimal schedule, i.e. a schedule on a minimum possible number of machines.*

Proof. Let q be the number of machines used by the algorithm, and let job J_j be the first one to be scheduled on machine M_q. Since MinNM puts J_j on M_q, it means that every machine $M_1, \ldots, M_{q-1}$ processes some job in the time interval $[s_j, s_j + \varepsilon)$ for some suitably small $\varepsilon > 0$. However, this set of $q-1$ jobs together with job J_j force every feasible schedule to use at least q machines because no pair of jobs from this set can be scheduled on the same machine. ■

In the last two sections we shall relax the $p_j \leq d_j$ constraint and consider arbitrarily long processing times.

5 Binary NP-hardness for $m = 2$ and arbitrary p_j's In this section we shall show that relaxing the $p_j \leq d_j$ constraint causes the problem to become binary NP-hard already in the two machine case. For the reduction, we shall use the well-known binary NP-complete PARTITION problem (see e.g. [1]), which is defined as follows.

PARTITION

Instance: n positive integers $a_1, a_2, \ldots, a_n$.

Question: Is there a subset S of the index set $I = \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} a_i = \sum_{i \in I \setminus S} a_i \quad ?$$

Now we shall construct an instance of a decision version of the currently considered just-in-time scheduling problem on two machines, so that this instance has a solution if and only if the input instance of the PARTITION problem has a solution.

Just-in-time Scheduling on Two Machines (JS2M)

Instance: $m = 2$, $L = 1$, $J_1, \ldots, J_n$ (identify jobs with the input numbers a_i), $p_j = a_i$, $d_j = 1$ for all $1 \leq j \leq n$.

Question: Is there a feasible schedule which occupies at most $T = \frac{1}{2} \sum_{i \in I} a_i$ time slots, i.e. such that $R \leq T$?

Lemma 2 *The input PARTITION instance has a solution if and only if the constructed JS2M instance has a solution.*

Proof. If PARTITION has a solution, then we easily construct a solution of JS2M by scheduling the jobs selected into the subset S on the first machine and the remaining jobs on the second machine. The number of slots used on each machine will be exactly T, and all jobs will be scheduled just-in-time, because every end of a slot is a due date for every job, and the lengths of jobs are integral.

On the other hand if JS2M has a solution, then exactly T time slots are occupied on each of the two machines and so the subset of jobs scheduled on any of the two machines constitutes a solution to the input PARTITION problem. ■

Since the above reduction is clearly polynomial, i.e. the number of bits needed to encode the constructed instance of JS2M is bounded by a polynomial in the number of bits needed to encode the input instance of PARTITION, we get the following theorem.

Theorem 3 *JS2M is binary NP-hard.*

In the last section of this paper we shall strengthen the NP-hardness result to unary coding. In order to obtain this result we shall sacrifice the fixed number of machines, i.e we shall assume that the number of machines is a part of the input.

6 Unary NP-hardness for input m and arbitrary p_j's In this section we shall mimic the proof for the binary NP-hardness using the well-known unary NP-complete 3-PARTITION problem (see e.g. [1]), which is defined as follows.

3-PARTITION

Instance: $3m+1$ positive integers b and $a_1, a_2, \ldots, a_{3m}$, such that $\frac{b}{4} < a_i < \frac{b}{2}$ for every $1 \leq i \leq 3m$, and

$$\sum_{i=1}^{3m} a_i = mb.$$

Question: Can the set $I = \{1, 2, \ldots, n\}$ be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that

$$\sum_{i \in S_i} a_i = b \quad \text{for all } 1 \leq i \leq m?$$

Now we shall construct an instance of a decision version of the considered just-in-time scheduling problem on m machines (where m is the input value from the instance of 3-PARTITION), so that this instance has a solution if and only if the input instance of the 3-PARTITION problem has a solution.

Just-in-time Scheduling on m Machines (JSmM)

Instance: $L = 1$, $J_1, \ldots, J_{3m}$ (identify jobs with the input numbers a_i), $p_j = a_i$, and $d_j = 1$ for all $1 \leq j \leq 3m$.

Question: Is there a feasible schedule which occupies at most b time slots, i.e. such that $R \leq b$?

Lemma 3 *The input 3-PARTITION instance has a solution if and only if the constructed JSmM instance has a solution.*

Proof. If 3-PARTITION has a solution, then we easily construct a solution of JSmM by scheduling the jobs selected into the subset S_i on the machine M_i for every $1 \leq i \leq m$. The number of slots used on each machine will be exactly b, and all jobs will be scheduled just-in-time, because every end of a slot is a due date for every job, and the lengths of jobs are integral.

On the other hand if JSmM has a solution, then exactly b time slots are occupied on each machine and so the subset of jobs scheduled machine M_i gives the set S_i for every $1 \leq i \leq m$, which constitutes a solution to the input 3-PARTITION problem. ■

As in the previous case, the above reduction is polynomial, i.e. the number of bits needed to encode the constructed instance of JSmM is bounded by a polynomial in the number of bits needed to encode the input instance of 3-PARTITION. Hence we get the following theorem.

Theorem 4 *JSmM is unary NP-hard.*

References

[1] Garey MR, Johnson DS. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[2] Hiraishi K. *Just-In-Time Scheduling of Parallel Identical Machines with Multiple Time Slots.* Proceedings of the 4th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, Jindrichuv Hradec, September 2001.

[a] *Department of Theoretical Informatics and Mathematical Logic, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic*
(also teaching at the Institute of Finance and Administration - VŠFS)
Email: cepek@ksi.ms.mff.cuni.cz

[b] *School of Information Science,*
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, Japan
Email: son@jaist.ac.jp