
ELSEVIER

Discrete Applied Mathematics 55 (1994) 259–272

DISCRETE APPLIED MATHEMATICS

Complexity of scheduling multiprocessor tasks with prespecified processor allocations

J.A. Hoogeveen[a], S.L. van de Velde[b], B. Veltman[c,*]

[a] *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands*

[b] *School of Management Studies, Technology and Innovation, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands*

[c] *CWI, P.O. Box 4079, 1009 AB Amsterdam, Netherlands*

Received 24 June 1992; revised 22 January 1993

Abstract

We investigate the computational complexity of scheduling multiprocessor tasks with prespecified processor allocations. We consider two criteria: minimizing schedule length and minimizing the sum of the task completion times. In addition, we investigate the complexity of problems when precedence constraints or release dates are involved.

Key words: Multiprocessor tasks; Prespecified processor allocations; Makespan; Total completion time; Release dates; Precedence constraints

1. Introduction

We address a class of multiprocessor scheduling problems. A collection of n tasks has to be executed by m processors. Task J_j ($j = 1, \ldots, n$) requires processing during a given uninterrupted time p_j. Each task requires the simultaneous use of a set of *prespecified* processors for its execution; each processor can execute at most one task at a time. Such tasks are referred to as *multiprocessor tasks*. Sometimes, for each task J_j a *release date* r_j on which it becomes available for processing or *precedence constraints*, indicating the set of tasks that have to be completed before J_j may start, are specified; we will state explicitly whether this is the case. We have to determine a *schedule*, that is, an allocation of each task J_j to a time interval of length p_j such that no two tasks on the same processor overlap. The completion time of task J_j in schedule σ is denoted by $C_j(\sigma)$ or shortly by C_j, if no confusion is possible as to the

* Corresponding author.

SSDI 0166-218X(93)E0083-B

schedule we refer to. We are interested in two objectives. The first one is to find a schedule that minimizes the *makespan* $C_{\max} = \max_j C_j$. The second objective concerns the minimization of the *total completion time* $\sum C_j = \sum_{j=1}^{n} C_j$.

In this paper, scheduling problems are denoted using the three-field notation scheme that was proposed by Veltman et al. [13] as an extension of the terminology of Graham et al [7]. In the notation scheme $\alpha|\beta|\gamma$, α specifies the processor environment, β the task characteristics, and γ the objective function. Accordingly, the value of γ of a schedule σ and the minimal value with respect to γ are denoted by $\gamma(\sigma)$ and γ^*, respectively. For instance, $P|fix_j, r_j|C_{\max}$ refers to the multiprocessor problem of minimizing the makespan, where for each task a fixed processor allocation and a release date are specified; $Pm|fix_j, p_j = 1|\sum C_j$ denotes the multiprocessor problem of minimizing the total completion time, where all processing times are equal to 1, processor allocations are given, and the number m of processors is specified as part of the problem type.

In the literature, little attention has been devoted to the complexity of scheduling multiprocessor tasks. Krawczyk and Kubale [8] show that $P|fix_j, p_j = 1|C_{\max}$ is NP-hard, even if the instances consist of biprocessor tasks only. Kubale [9] presents a similar proof. Blazewicz et al. [2] show that $P3|fix_j|C_{\max}$ is strongly NP-hard. As to optimization algorithms, two *branch and bound* approaches for $P|fix_j|C_{\max}$ have been proposed. Bozoki and Richard [5] concentrate on *incompatibility*; two tasks are *incompatible* if they have at least one processor in common. Bianco et al. [1] follow a graph-theoretical approach, and they determine a class of polynomially solvable instances that corresponds to the class of comparability graphs. We will investigate the complexity of a class of problems related to $P|fix_j|C_{\max}$. The outline of the paper is as follows.

Section 2 deals with the makespan criterion. The general problem with a fixed number m of processors is polynomially solvable if m is equal to 2, but NP-hard in the strong sense for $m \geqslant 3$. There are two well-solvable cases. The first one concerns the case of unit processing times; the problem is then solvable in polynomial time through an integer programming formulation with a fixed number of variables. The second one concerns the three-processor problem in which all multiprocessor tasks of the same type are decreed to be executed consecutively, the so-called *block-constraint*; this problem is solvable in $O(n\sum p_j)$ time. If the number of processors is part of the problem instance, then the problem with unit processing times is already NP-hard in the strong sense. In general, the introduction of precedence constraints or release dates leads to strong NP-hardness, with one exception: the problem with unit processing times in which both the number of processors and the number of distinct release dates are fixed is solvable in polynomial time through an integer programming formulation with a fixed number of variables. The computational complexity of the problem $Pm|fix_j, r_j, p_j = 1|C_{\max}$ is still open.

Section 3 deals with the total completion time criterion. In general, this criterion leads to severe computational difficulties. The problem is NP-hard in the ordinary sense for $m = 2$ and in the strong sense for $m = 3$. The weighted version and the problem with precedence constraints are already NP-hard in the strong sense for $m = 2$. The problem with unit time processing times is NP-hard in the strong sense if

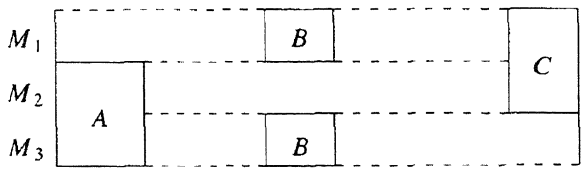

Fig. 1. A schedule satisfying the block-constraint.

the number of processors is part of the problem instance, but still open in case of a fixed number of processors. Another open problem is $Pm|fix_j, r_j, p_j = 1|\sum C_j$.

2. Makespan

In this section, we investigate the computational complexity of minimizing the makespan. If no precedence relation is specified, then we may discard the tasks that need all the processors for execution, since they can be scheduled ahead of the other ones. Hence, the two-processor problem without precedence constraints is simply solved by scheduling each single-processor task on its processor without causing idle time.

2.1. The block-constraint and pseudopolynomiality on three processors

The block-constraint decrees that all biprocessor tasks of the same type are scheduled consecutively. As this boils down to the case that there is at most one biprocessor task of each type, we replace all biprocessor tasks of the same type by one task of this type with processing time equal to the sum of the individual processing times. The biprocessor task that requires M_2 and M_3 is named a task of type A and its processing time is denoted by p_A. Correspondingly, the biprocessor task that requires M_1 and M_3 and the biprocessor task that requires M_1 and M_2 are said to be of type B and C, respectively; their processing times are denoted by p_B and p_C (Fig. 1).

Theorem 2.1. *The problem $P3|fix_j|C_{\max}$ subject to the block-constraint is* NP*-hard in the ordinary sense.*

Proof. We will show that $P3|fix_j|C_{\max}$ subject to the block-constraint is NP-hard by a reduction from the NP-complete problem Partition.

Partition

Given a multiset $N = \{a_1, \ldots, a_n\}$ of n integers, is it possible to partition N into two disjoint subsets that have equal sum $b = \sum_{j \in N} a_j/2$?

Given an instance of Partition, define for each $j \in N$ a task J_j that requires M_1 for execution and has processing time $p_j = a_j$. In addition, we introduce five *separation*

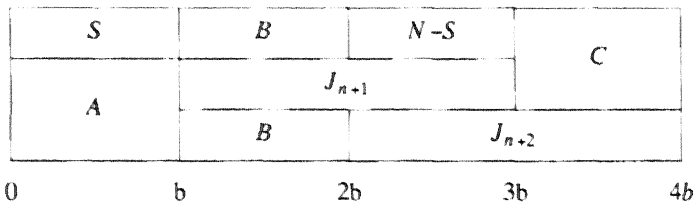

Fig. 2. A schedule with partition sets S and $N - S$.

tasks that create two time slots of length b on M_1. The tasks J_A, J_B, and J_C, each with processing time b, are of the type A, B, and C, respectively. The two single-processor tasks J_{n+1} and J_{n+2}, each with processing time $2b$, have to be executed by M_2 and M_3, respectively.

Note that each processor has a load of $4b$, which implies that $4b$ is a lower bound on the makespan of any feasible schedule. We will show that Partition has a solution if and only if there exists a schedule for the corresponding instance of $P3|fix_j|C_{\max}$ with $C_{\max} \leqslant 4b$.

Suppose that there exists a subset $S \subset N$ such that $\sum_{j\in S} a_j = \sum_{j\in N-S} a_j = b$. A schedule of length $C_{\max} = 4b$ then exists, as is illustrated in Fig. 2.

Conversely, suppose there exists a schedule with makespan at most $4b$. Notice that only four possibilities exist to schedule the tasks $J_{n+1}, J_{n+2}, J_A, J_B$, and J_C in a time interval of length $4b$. Each of these possibilities leaves two separated idle periods of length b on processor M_1, in which the tasks J_j with $j \in N$ must be processed. Thus, if there exists a schedule of length $C_{\max} = 4b$, then there is a subset $S \subset N$ such that $\sum_{j\in S} a_j = \sum_{j\in N-S} a_j$.

We conclude that $P3|fix_j|C_{\max}$ is NP-hard in the ordinary sense. □

Theorem 2.2. *The problem $P3|fix_j|C_{\max}$ subject to the block-constraint is solvable in pseudopolynomial time.*

Proof. We propose an algorithm for this problem that requires $O(n\sum_{j\in N} p_j)$ time and space. For $i = 1, 2, 3$, let T_i denote the set of indices of tasks that require only M_i for processing, and $n_i = |T_i|$. In addition, we define $p(S) = \sum_{j\in S} p_j$.

Using an interchange argument, we can transform any optimal schedule into an optimal schedule with some biprocessor task scheduled first and some other biprocessor task scheduled last. Suppose for the moment that these tasks are of type A and C, respectively; a B-type task is then scheduled somewhere in between. Any feasible schedule of this type, referred to as an ABC-schedule, is completely specified by the subsets $Q_1 \subseteq T_1$ and $Q_3 \subseteq T_3$ scheduled before the B-type task; see Fig. 3.

For an ABC-schedule with given subsets Q_1 and Q_3, the earliest start time of the task of type B is

$$S_B(Q_1, Q_3) = \max\{p(Q_1), p_A + p(Q_3)\}.$$

The earliest start time of the task of type C is then

$$S_C(Q_1, Q_3) = \max\{S_B(Q_1, Q_3) + p_B + p(T_1 - Q_1), p_A + p(T_2)\}.$$

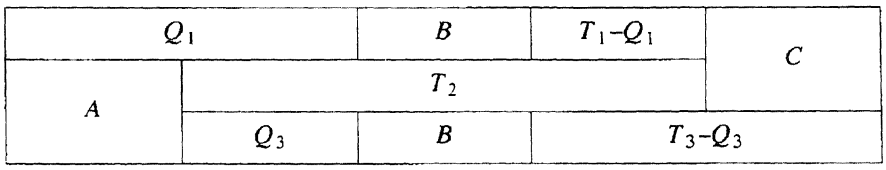

Fig. 3. Structure of an ABC-schedule.

The minimal length of such a schedule is therefore

$$C_{\max}(Q_1, Q_3) = \max\{S_C(Q_1, Q_3) + p_C, S_B(Q_1, Q_3) + p_B + p(T_3 - Q_3)\}. \quad (1)$$

Hence, the minimal length of an ABC-schedule is determined by $p(Q_1)$ and $p(Q_3)$. In other words, the length of an optimal ABC-schedule is equal to the minimum of $C_{\max}(Q_1, Q_3)$ over all possible values of $p(Q_1)$ and $p(Q_3)$. Due to symmetry, we can transform any ABC-schedule into an CBA-schedule of the same length. The only other types of schedules of interest to us are therefore the BAC- and ACB-schedules. Similar arguments show that the length of an optimal BAC-schedule is equal to the minimum of $C_{\max}(Q_2, Q_3)$ over all possible values of $p(Q_2)$ and $p(Q_3)$, and that the length of an optimal ACB-schedule is equal to the minimum of $C_{\max}(Q_1, Q_2)$ over all possible values of $p(Q_1)$ and $p(Q_2)$.

For $i = 1, 2, 3$, we compute all possible values that $p(Q_i)$ can assume in $O(n_i p(T_i))$ time and space by a standard dynamic programming algorithm of the type also used for the knapsack and the subset-sum problems; see e.g. [12]. If these values are put in sorted lists, then all possible values that $S_B(Q_1, Q_3)$ can assume are computed in $O(p(Q_1) + p(Q_3))$ time and space. The minimum of $C_{\max}(Q_1, Q_3)$ over $p(Q_1)$ and $p(Q_3)$ is then determined by evaluating expression (1) for each possible combination of $p(Q_1)$ and $p(Q_3)$; this takes $O(p(T_1) + p(T_3))$ time.

The lengths of the optimal BAC- and ACB-schedules are determined similarly. The overall minimum then follows immediately, and an optimal schedule is determined by backtracing. Since $n_i \leqslant n$ and $p(T_i) \leqslant \sum_{j \in N} p_j$ for each i, it takes $O(n \sum_{j \in N} p_j)$ time and space to find an optimal schedule. □

2.2. *Strong* NP-*hardness for the general 3-processor problem*

The computational complexity of $P3|fix_j|C_{\max}$ has already been addressed by Blazewicz et al. [2], but we include our own version of the reduction for sake of completeness.

Theorem 2.3. *The problem* $P3|fix_j|C_{\max}$ *is* NP-*hard in the strong sense.*

Proof. The proof is based upon a reduction from the strongly NP-complete problem 3-Partition.

Table 1
Separation tasks for $P3|fix_j|C_{max}$

Number	Allocation	Processing time
n	M_2 & M_3 (type A)	p_A
n	M_1 & M_3 (type B)	p_B
n	M_1 & M_2 (type C)	p_C
1	M_1	$p_A + b$
$n-1$	M_1	$p_A + b + p_z$
n	M_1	p_y
$n-1$	M_2	p_z
n	M_2	$p_B + b + p_y$
1	M_3	$p_C + p_y$
$n-1$	M_3	$p_C + p_y + p_z$

3-Partition

Given an integer b and a multiset $N = \{a_1, \ldots, a_{3n}\}$ of $3n$ positive integers with $b/4 < a_j < b/2$ and $\sum_{j=1}^{3n} a_j = nb$, is there a partition of N into n mutually disjoint subsets $N_1, \ldots, N_n$ such that the elements in N_j add up to b, for $j = 1, \ldots, n$?

Given an instance of 3-Partition, we construct the following instance $P3|fix_j|C_{max}$. There are $3n$ single-processor tasks J_j that correspond to the elements of 3-Partition; these tasks have to be executed by M_3 and their processing time is equal to a_j, for $j = 1, \ldots, 3n$. In addition, there are $3n$ biprocessor *separation tasks* and $5n - 1$ single-processor *separation tasks*; their processing times and processing requirements are defined in Table 1. Here we define

$$p_B = (n+1)b,$$

$$p_y = (n+1)(b + p_B),$$

$$p_z = (n+1)(b + p_B + p_y),$$

$$p_C = (n+1)(b + p_B + p_y + p_z),$$

$$p_A = (n+1)(b + p_B + p_y + p_z + p_C).$$

Note that each processor has a processing load equal to $\gamma = n(p_A + p_B + p_C + p_y + p_z + b) - p_z$, which implies that γ is a lower bound on the makespan of any schedule. Any schedule with makespan γ should have the form as displayed in Fig. 4, or its mirror image. We assert, without proof, that 3-Partition has an affirmative answer if and only if there exists a schedule with makespan at most γ for the corresponding instance of $P3|fix_j|C_{max}$. □

2.3. Unit execution times, release dates, and precedence constraints

In this section, we show that the $Pm|fix_j, p_j = 1|C_{max}$ problem is solvable in polynomial time by providing an integer linear programming formulation with a fixed

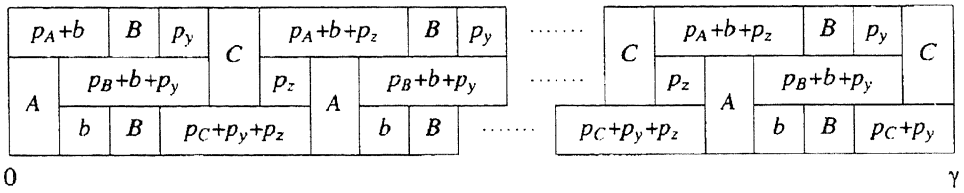

Fig. 4. Structure for $P3|fix_j|C_{max}$: $ABCAB...CABC$.

number of variables; a problem that allows such a formulation is solvable in polynomial time (Lenstra Jr [10]). A similar approach is given by Blazewicz, et al. [3].

Consider an arbitrary instance of the problem. There are at most $M = 2^m - 1$ tasks of a different type; let these types be numbered $1, \ldots, M$. We can denote the instance by a vector $b = (b_1, \ldots, b_M)$ in which component b_j indicates the number of tasks of type j. A collection of tasks is called *compatible* if all these tasks can be executed in parallel; hence, a compatible collection of tasks contains at most one task of each type. A compatible collection is denoted by a $\{0,1\}$-vector c of length M with $c_j = 1$ if the collection contains a task of type j and zero otherwise. There are at most $K = 2^M - 1$ different compatible collections; this number is fixed, as M is fixed. Let the collections be numbered $1, \ldots, K$; let the vectors indicating the collections be denoted by $c_1, \ldots, c_K$. The problem of finding a schedule of minimal length is then equivalent to the problem of finding a decomposition of this instance into a minimum number of compatible collections. Formally, we wish to minimize $\sum_{j=1}^{K} x_j$ subject to $\sum_{j=1}^{K} c_j x_j = b, x_j$ integer and nonnegative. As the number of variables in this integer linear programming formulation is fixed, we have proven the following lemma.

Lemma 2.4. *The $Pm|fix_j, p_j = 1|C_{max}$ problem is solvable in polynomial time.*

If the number of processors is specified as part of the problem type, implying that this number is no longer fixed, then things get worse from a complexity point of view. This is stated in the following theorem.

Theorem 2.5. *The problem of deciding whether an instance of $P|fix_j, p_j = 1|C_{max}$ has a schedule of length at most 3 is* NP*-hard in the strong sense.*

Proof. The proof is based upon a reduction from the strongly NP-complete problem Graph 3-Colorability. A similar approach is used by Blazewicz et al. [4].

Graph 3-Colorability

Given a graph $G = (V, E)$, does there exist a 3-coloring, that is, a function $f: V \to \{1,2,3\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

Given an arbitrary instance $G = (V, E)$ of Graph 3-Colorability, we construct the following instance of $P|fix_j, p_j = 1|C_{max}$. There are $|V|$ tasks $J_1, \ldots, J_{|V|}$ and $|E|$ processors $M_1, \ldots, M_{|E|}$. A task J_u has to be processed by M_e if $u \in e$. We claim that there exists a 3-coloring for G if and only if there exists a schedule of length at most 3. □

Corollary 2.6. *For $P|fix_j, p_j = 1|C_{max}$, there exists no polynomial approximation algorithm with performance ratio smaller than 4/3, unless* P = NP.

The introduction of precedence constraints leaves little hope of finding polynomial-time optimization algorithms. Even the two-processor problem with unit execution times and the simplest possible precedence relation structure, a collection of vertex-disjoint chains, is already NP-hard in the strong sense.

Theorem 2.7. *The $P2|chain, fix_j, p_j = 1|C_{max}$ problem is* NP*-hard in the strong sense.*

Proof. The proof is based upon a reduction from 3-Partition and follows an approach of Blazewicz et al. [4]. Given an arbitrary instance of 3-Partition, we construct the following instance of $P2|chain, fix_j, p_j = 1|C_{max}$. Each element a_j corresponds to a chain K_j of $2a_j$ tasks; the first part consists of a_j tasks that have to be executed by M_1 and the second part also consists of a_j tasks that have to be executed by M_2. In addition, there is a chain L of $2nb$ tasks; groups of b tasks have to be alternately executed by M_2 and M_1.

We assert that 3-Partition has an affirmative answer if and only if there exists a schedule of length at most $2nb$ for the corresponding instance of $P2|chain, fix_j, p_j = 1|C_{max}$. □

The introduction of release dates has a similar inconvenient effect on the computational complexity.

Theorem 2.8. *The $P2|fix_j, r_j|C_{max}$ problem is* NP*-hard in the strong sense.*

Proof. The proof is again based upon a reduction from 3-Partition. Given an arbitrary instance of 3-Partition, we construct the following instance $P2|fix_j, r_j|C_{max}$. For each element a_j, we define a task J_j with $p_j = a_j$ and $r_j = 0$ that has to be executed by M_1. Furthermore, there are n tasks K_k with processing time b and release date $r_k = (k-1)(b+\varepsilon)$, for $k = 1, \ldots, n$ and ε sufficiently small; these tasks have to be executed by M_2. Finally, there are $n-1$ biprocessor tasks L_l with processing time ε and release date $r_l = lb + (l-1)\varepsilon$, for $l = 1, \ldots, n-1$. It is easy to see that 3-Partition has an affirmative answer if and only if there exists a feasible schedule for $P2|fix_j, r_j|C_{max}$ with $C_{max} \leqslant nb + (n-1)\varepsilon$. □

Consider the case $Pm|fix_j, r_j, p_j = 1|C_{max}$ where the number of distinct release dates is fixed. Analogously to our analysis of $Pm|fix_j, p_j = 1|C_{max}$, we can transform any

instance of $Pm|fix_j, r_j, p_j = 1|C_{max}$ into an integer linear programming problem with a fixed number of variables. We have proven the following theorem that extends Lemma 2.4.

Theorem 2.9. *The $Pm|fix_j, r_j, p_j = 1|C_{max}$ problem with a fixed number of distinct release dates is solvable in polynomial time.*

3. Sum of completion times

In this section, we investigate the computational complexity of our type of scheduling problems when we wish to minimize total completion time. Our main result is establishing NP-hardness in the ordinary sense for $P2|fix_j|\sum C_j$. The question whether this problem is solvable in pseudopolynomial time or NP-hard in the strong sense still has to be resolved. The weighted version, however, is NP-hard in the strong sense. We start with an easy observation. Given an instance, let the maximal processing time be denoted by $p_{max} = \max_j p_j$.

Proposition 3.1. *There is an optimal schedule for $P|fix_j|\sum C_j$ in which the tasks that require all processors for execution during p_{max} time are scheduled last, if they exist.*

Proof. Consider a schedule σ for $P|fix_j|\sum C_j$ in which the task J that needs all processors for execution during time p_{max} is not scheduled last. The interchange illustrated in Fig. 5 generates a schedule σ^* with $\sum C_j(\sigma^*) \leqslant \sum C_j(\sigma) + p(B) - \lceil p(B)/p_{max} \rceil p_{max} \leqslant \sum C_j(\sigma)$, where $p(B) = \sum_{J \in B} p_j$. □

3.1. NP-hardness for the 2-processor problem

Theorem 3.2. *The $P2|fix_j|\sum C_j$ problem is* NP-*hard in the ordinary sense.*

Proof. Our proof is based upon a reduction from the NP-complete problem Even–Odd Partition.

Even–Odd Partition

Given a set of $2n$ positive integers $A = \{a_1, \ldots, a_{2n}\}$ such that $a_i < a_{i+1}$ $(i = 1, \ldots, 2n - 1)$, is there a partition of N into two disjoint subsets A_1 and A_2 with equal sum $b = \sum_{i=1}^{2n} a_i/2$ and such that A_1 contains exactly one of $\{a_{2i-1}, a_{2i}\}$, for each $i = 1, \ldots, n$?

Fig. 5. The interchange.

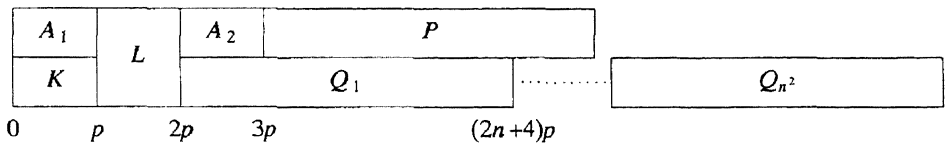

Fig. 6. The schedule σ^* with partition sets A_1 and A_2.

Given an instance of Even–Odd Partition, define $p = (n^2 + 1)b$, $q = n^2(n^2 + 1)(n + 1)p$, and $r = \sum_{j=1}^{n}(n + j - 1)(a_{2j-1} + a_{2j}) + n^2(n + 1)b$. We construct the following instance of $P2|fix_j|\sum C_j$. Each element a_j corresponds to a partition task J_j with processing time $p_j = nb + a_j$ that has to be executed by M_1. In addition, we define $n^2 + 3$ extra tasks. There are n^2 identical tasks Q_i $(i = 1, \ldots, n^2)$ with processing time $2p(n + 1)$ that have to be executed by M_2, a task K with processing time p that has to be executed by M_2, a biprocessor task L with processing time p, and a task P with processing time $2p(n + 1)$ that has to be executed by M_1. We will show that Even–Odd Partition is answered affirmatively if and only if there exists a schedule for the corresponding instance of $P2|fix_j|\sum C_j$ with total completion time no more than the threshold

$$y = (2n^2 + 4n + 8)p + q + r.$$

Suppose that there exist subsets A_1 and A_2 that lead to an affirmative answer to Even–Odd Partition. Then there exists a schedule σ^* with total completion time no more than y, as is illustrated in Fig. 6. The completion times of the extra tasks add up to $(2n^2 + 2n + 8)p + q$, the sum of the completion times of the partition tasks is equal to $2np + r$.

Conversely, suppose that there exists a schedule σ with total completion time no more than y. We first show that the extra tasks in σ must be scheduled according to the pattern of Fig. 6.

A straightforward computation shows that task P and the Q-tasks must be completed after all other tasks in σ. Suppose that task L precedes task K, and that m partition tasks are completed before L starts. Note that $m \leqslant n$; otherwise, task K could be scheduled parallel to the m partition tasks, without increasing the completion time of any other job. If we compare this schedule with σ^*, then task L turns out to be the only task with smaller completion time; this gain is more than offset by the increase of completion time of task K. Hence, in order to satisfy the threshold, the extra jobs must be scheduled according to the pattern of Fig. 6.

We now show that, if $\sum C_j(\sigma) \leqslant y$, then the partition tasks must constitute an affirmative answer to Even–Odd Partition. First, suppose that the partition tasks before L in σ have total processing time smaller than p, implying that at most n partition tasks are scheduled before L. Then the total completion time of the partition jobs amounts to at least $r + 2np$, the total completion time of the Q-tasks,

task K, and task L is equal to the total completion time of these tasks in σ^*, and the completion time of task P is greater than $3p + (2n + 2)p$, implying that the threshold is exceeded. Hence, the total processing time of the partition tasks before task L amounts to at least p.

Now suppose that m partition tasks with total processing time $p + x$ precede task L. Comparing σ with σ^* shows that the total completion time of the extra jobs in σ is $x(n^2 + 1)$ greater and that the difference in total completion time of the partition tasks is no more than $2p(n - m) + x(2n - m)$ in favor of σ. If $m = n$, then the difference in total completion time between σ^* and σ is at least equal to $x(n^2 + 1) - xn$ in favor of σ^*; $x > 0$ then clearly implies that the threshold will be exceeded. In case $m > n$, we wish to show that $x(n^2 + 1) > 2p(n - m) + x(2n - m)$, which boils down to showing that $x(n^2 + 1 - 2n + m) > 2p(n - m)$. As the left-hand side of the inequality is positive and the right-hand side negative, we have that the case $m > n$ leads to an excess of the threshold. Hence, exactly n partition tasks with total processing time equal to p must precede task L in σ. The total completion time of the partition tasks is equal to $2np + n(p_{[1]1} + p_{[1]2}) + \cdots + (p_{[n]1} + p_{[n]2})$, where $p_{[i]1}$ and $p_{[i]2}$ denote the processing time of the $[i]$th partition task before L and after L, respectively. It is easy to see that the threshold can only be met if $\{p_{[i]1} + p_{[i]2}\} = \{p_{2i-1}, p_{2i}\}$, for $i = 1, \ldots, n$. Define A_1 and A_2 as the set of partition tasks before L and after L in σ, respectively. As the total processing time of the tasks in A_1 amounts to $n^2b + \sum_{A_1} a_j = p = (n^2 + 1)b$, we have that the corresponding subset of partition elements has sum equal to b. Furthermore, A_1 contains exactly one element from every pair $\{a_{2i-1}, a_{2i}\}$; hence, the subsets A_1 and A_2 lead an affirmative answer to Even–Odd Partition. □

Theorem 3.3. *The $P2|fix_j|\sum w_j C_j$ problem is* NP*-hard in the strong sense.*

Proof. The proof is based upon a reduction from 3-Partition. Given an arbitrary instance of 3-Partition, we construct the following instance of $P2|fix_j|\sum w_j C_j$. Each element a_j corresponds to a task J_j with processing time a_j and unit weight that has to be executed by M_1. In addition, there are n tasks K_j with processing time b and weight $2(j + \alpha - 1)\beta$ that have to be executed by M_2, and n_L biprocessor tasks L_j with processing time b and weight $(2j - 1)\beta$, where $\alpha = 3n(2n - 1)$, $\beta = \alpha b$, and $n_L = \alpha + n - 1$.

Suppose that there exists a partition of N into $N_1, \ldots, N_n$ that yields an affirmative answer to 3-Partition. A feasible schedule with sum of weighted completion times no more than $y = \beta + \sum_{k=1}^{n} w_k(2(n - k) + 1)b + \sum_{l=1}^{\alpha} w_l(2n + \alpha - l)b + \sum_{l=\alpha+1}^{n_l} w_l(2n - 2(l - \alpha))b$ is then obtained by scheduling the tasks as illustrated in Fig. 7.

Conversely, suppose that there exists a schedule σ with sum of weighted completion times no more than y. Straightforward computations show that the K-tasks and the L-tasks have to be scheduled as indicated in Fig. 7 and that the tasks J_j have to be scheduled in the time slots parallel to the K-tasks. Let N_j denote the set of J-tasks that are scheduled parallel to K_j; the sets $N_1, \ldots, N_n$ constitute a solution to 3-Partition. □

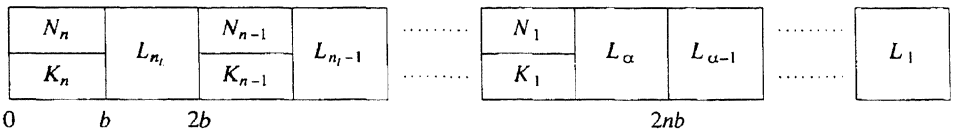

Fig. 7. A schedule for $P2|fix_j|\sum w_jC_j$ with $\sum w_jC_j \leqslant y$.

3.2. *Strong* NP*-hardness for the general 3-processor problem*

Theorem 3.4. *The* $P3|fix_j|\sum C_j$ *problem is* NP*-hard in the strong sense.*

Proof. The proof is based upon a reduction from the decision version of the $P3|fix_j|C_{\max}$ problem, which was shown to be NP-complete in Section 2.2. The decision version of $P3|fix_j|C_{\max}$ is defined as the following question: given an instance of $P3|fix_j|C_{\max}$ and a threshold b, does there exist a schedule σ with makespan no more than b?

Given an arbitrary instance of $P3|fix_j|C_{\max}$ and a threshold b, we construct the decision instance of $P3|fix_j|\sum C_j$ by adding $nb+1$ identical triprocessor tasks K_j with processing time $p_{\max}$. The corresponding threshold is equal to $y = nb + \sum_{k=1}^{nb+1}(b + kp_{\max})$.

Application of Proposition 3.1 shows that there is an optimal schedule with the K-tasks executed last. The number of K-tasks is such that the threshold will be exceeded if the first K-task starts later than b. Hence, the decision variant of $P3|fix_j|\sum C_j$ has an affirmative answer if and only if the decision variant of $P3|fix_j|C_{\max}$ has an affirmative answer.

Note that, the number of tasks needed in our reduction is pseudopolynomially bounded. We conclude that $P3|fix_j|\sum C_j$ is NP-hard in the strong sense. □

3.3. *Unit execution times and precedence constraints*

In this section, we address the complexity of minimizing total completion time in case of unit processing times. We show that $P|fix_j, p_j = 1|\sum C_j$ is NP-hard in the strong sense; the complexity of this problem with a fixed number of processors is still open.

Theorem 3.5. *The* $P|fix_j, p_j = 1|\sum C_j$ *problem is* NP*-hard in the strong sense.*

Proof. The proof of this theorem is based upon a reduction from $P|fix_j, p_j = 1|C_{\max}$; it proceeds along the same lines as the proof of the previous theorem. Given an instance of $P|fix_j, p_j = 1|C_{\max}$, we add w tasks that require all processors for execution; application of Proposition 3.1 shows that these tasks can be assumed to be executed after all other tasks. By choosing w suitably large, we obtain the situation

that the threshold of $P|fix_j, p_j = 1|\sum C_j$ is exceeded if and only if the threshold of $P|fix_j, p_j = 1|C_{max}$ is exceeded. As the decision variant of $P|fix_j, p_j = 1|C_{max}$ is NP-complete in the strong sense and as w is polynomially bounded, we conclude that $P|fix_j, p_j = 1|\sum C_j$ is NP-hard in the strong sense. □

As could be expected, the addition of precedence constraints does not have a positive effect on the computational complexity. We show that even the mildest non-trivial problem of this type, with two processors and chain-type precedence constraints, is NP-hard in the strong sense.

Theorem 3.6. *The $P2|chain, fix_j, p_j = 1|\sum C_j$ problem is* NP*-hard in the strong sense.*

Proof. The proof is based upon the same reduction as used in the proof of Theorem 2.7, only the threshold differs. As the number of tasks is equal to $2nb$, and as each task has unit processing time, an obvious lower bound on the total completion time is equal to $y = 2nb(2nb + 1)$; this bound can only be attained by a schedule without idle time in which both processors execute nb tasks. Hence, there exists a schedule with total completion time no more than y if and only if there exists a schedule with makespan no more than b. We conclude that $P2|chain, fix_j, p_j = 1|\sum C_j$ is NP-hard in the strong sense. □

Acknowledgement

The authors wish to express their gratitude towards Jan Karel Lenstra for his helpful comments.

References

[1] L. Bianco, P. Dell'Olmo and M.G. Speranza, On scheduling independent tasks with dedicated resources. Program and Abstracts, 14th International Symposium Mathematical Programming, Amsterdam (1991).
[2] J. Blazewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors, Inform. Process. Lett. 41 (1992) 275–280.
[3] J. Blazewicz, M. Drabowski and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans. Comput. 35 (1986) 389–393.
[4] J. Blazewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. 5 (1983) 11–24.
[5] G. Bozoki and J.P. Richard, A branch-and-bound algorithm for the continuous-process task shop scheduling problem, AIIE Trans. 2 (1970) 246–252.
[6] M.R. Garey and D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).
[7] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[8] H. Krawczyk and M. Kubale, An approximation algorithm for diagnostic test scheduling in multiprocessor systems, IEEE Trans. Comput. 34 (1985) 869–872.

[9] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors, Inform. Process. Lett. 24 (1987) 141–147.
[10] H.W. Lenstra Jr, Integer programming with a fixed number of variables, Math. Oper. Res. 8 (1983) 538–548.
[11] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.
[12] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations (Wiley, Chichester, 1990).
[13] B. Veltman, B.J. Lageweg and J.K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Comput. 16 (1990) 173–182.