Discrete Applied Mathematics 157 (2009) 3625–3630

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints

K. Giaro, M. Kubale, P. Obszarski *

Department of Algorithms and System Modeling, Gdańsk University of Technology, Poland

ARTICLE INFO

Article history:
Received 29 November 2007
Received in revised form 3 February 2009
Accepted 13 February 2009
Available online 3 April 2009

Keywords:
Hypergraph coloring
Dedicated machines
Multiprocessor scheduling
NP-completeness

ABSTRACT

We address a generalization of the classical 1- and 2-processor unit execution time scheduling problem on dedicated machines. In our chromatic model of scheduling machines have non-simultaneous availability times and tasks have arbitrary release times and due dates. Also, the versatility of our approach makes it possible to generalize all known classical criteria of optimality. Under these stipulations we show that the problem of optimal scheduling of sparse tree-like instances can be solved in polynomial time. However, if we admit dense instances then the problem becomes NP-hard, even if there are only two machines.

1. Introduction

In recent years a number of new approaches to the problem of parallel computer systems has been proposed. One of them is scheduling of multiprocessor task systems [4]. According to this model any task may require for its processing more than one processor at a time. There are two main classes of problems in multiprocessor task scheduling. In the first class of problems, it is assumed that the *number* of simultaneously required processors is important [2,11] and each task requires any fixed subset of processors whose cardinality is equal to the prescribed number. In the second class of multiprocessor scheduling problems, the *set* of simultaneously required processors is assumed to be important [1,3,9,10]. In this case either a certain fixed set of processors, or a family of processor sets on which the task can be executed is given. This paper deals with the problem of scheduling tasks assigned to a fixed set of processors.

Since the problem of scheduling 2-processor tasks is NP-hard subject to all classical optimality criteria, in this paper we are concerned with the special case in which the duration of every task is the same. We will call such tasks unit execution time (UET) tasks. Consequently, all data are assumed to be positive integers. Later on we will see that such a restricted version, although remaining NP-hard for some instances in dense systems, does allow for polynomial-time algorithms in numerous special cases of sparse systems.

The third important assumption concerns availability constraints. Namely, we assume that the availability of tasks is restricted and, in addition, some machines are available only in certain intervals called time windows. Time windows may appear due to computer breakdowns or maintenance periods. Moreover, in any multitasking computer system and in hard real-time systems in particular, urgent tasks have high priority and are pre-scheduled in certain time intervals, thus creating multiple time windows of availability. Scheduling in time windows was considered in e.g. [1,2,5,6].

* Corresponding author. Tel.: +48 58 347 17 41; fax: +48 58 347 17 66.
E-mail addresses: giaro@eti.pg.gda.pl (K. Giaro), kubale@eti.pg.gda.pl (M. Kubale), pawel.obszarski@eti.pg.gda.pl (P. Obszarski).

0166-218X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2009.02.024

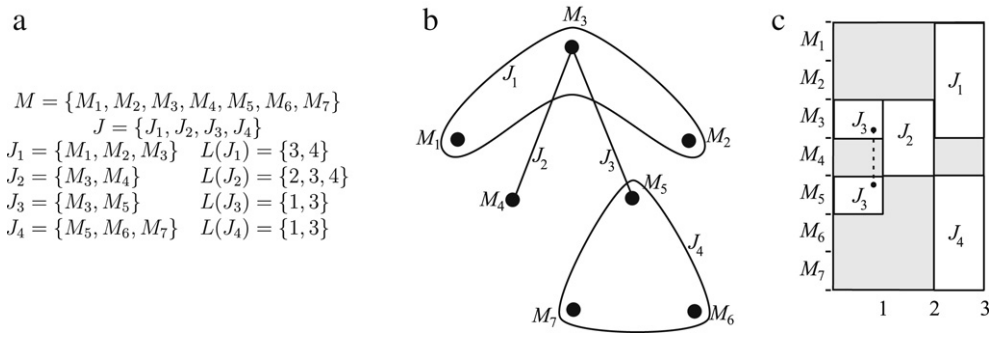

Fig. 1. Example (a) problem instance (b) scheduling hypergraph (c) Gantt diagram.

The rest of the paper is organized as follows. In Section 2 we set up the problem more formally and model it as a cost list edge-coloring (CLEC) problem. Section 3 is devoted to the case when two tasks sharing pairwise at most one machine constitute a hypertree. We show that our problem can be solved efficiently by a tree coloring technique. In Section 4 we generalize this approach to the case when besides the previous tasks there are $O(1)$ additional multiprocessor tasks.

In the second part of the paper we consider the case where any pair of tasks can share an arbitrary number of processors. In particular, we show that the scheduling problem is NP-hard even if the system consists of 2 machines. Since the proof is rather complicated, we postpone it to Section 5 of the paper. The paper is concluded with a description of how the procedure developed in Section 4 can be used to meet the most common criteria of scheduling.

2. Mathematical model

Classical scheduling theory provides various criteria of scheduling like C_{max}, L_{max}, T_{max}, F, $\Sigma w_j C_j$ and many others. All of them can be reduced to the cost list scheduling model which is presented in this section. Let $J = \{J_1, J_2, \ldots, J_m\}$ be a set of tasks which can be executed on a set of processors $M = \{M_1, M_2, \ldots, M_n\}$. All processors are distinct, and they can execute only one specified task at a time. Each task J_i requires the simultaneous use of a nonempty set $fix_i \subseteq M$ of processors for its execution. All tasks are independent, nonpreemptable, and of the same length. For the sake of simplicity we may assume that execution time of J_i, i.e. $p_i = 1$, for each $i = 1, \ldots, m$. Every processor can work on not more than one task at the same time. Time is divided into unit length slots numbered with successive integers. Tasks have availability constraints prespecified by lists $L(J_i) \subseteq \mathbb{N}$ (where $\mathbb{N}$ is the set of nonnegative integers) of available time slots in which J_i can be executed. With each task we associate a function $f_{J_i}(x)$ which assigns to each $x \in L(J_i)$ the cost of executing J_i if it is executed in time slot x. Our aim is to find a schedule with minimum total cost.

Our problem can be described using three-field notation as $P|win, p_j = 1, fix_j|crit$, where win stands for the fact that availability constraints are imposed on tasks. By the word *crit* we mean any of the criteria commonly used in classical scheduling theory.

The cost list scheduling problem can be modeled as the cost list edge-coloring of a hypergraph. Let $H = (V, E)$ be a *hypergraph*. $V = V(H)$ is the set of *vertices* and $E = E(H)$ is a multiset of nonempty subsets of V called *edges*. A d-edge e is an edge that contains exactly d vertices. $\Psi(e) = |e| = d$ denotes the *dimension* of edge e and $\Psi(H) = \max_{e \in E} \Psi(e)$ is the *dimension* of hypergraph H. The *degree* $\Delta(v)$ of a vertex $v \in V$ is the number of edges in which v occurs. $\Delta(H) = \max_{v \in V} \Delta(v)$ is the *degree* of H. The *neighborhood* $N(e)$ of edge e is the set of all edges in H that share at least one vertex with e. $N(H)$ is the cardinality of the maximal neighborhood of an edge in H.

A *proper edge-coloring* of hypergraph H with k colors is a function $c : E(H) \to \{1, \ldots, k\}$ such that no two edges which share a vertex have the same color (number). A *proper list edge-coloring* of a hypergraph requires the function c to satisfy an additional condition $c(e) \in L(e)$ for each $e \in E$, where $L(e)$ is the list of colors available to edge e. Let us consider a function $f_e : L(e) \to \mathbb{N}$ for $e \in E$. If for all e, all elements $x \in L(e)$ are assigned cost $f_e(x)$ and the aim of the coloring is to minimize the total cost, then the problem becomes a *cost list edge-coloring* (CLEC) of the hypergraph.

There is a one-to-one correspondence between cost list edge-coloring of a hypergraph and the cost list scheduling model presented in this section. Vertices of a hypergraph correspond to processors, edges to jobs and colors to time slots. A hypergraph created for an instance of a scheduling problem is called a *scheduling hypergraph*. From now on the terms 'scheduling' and 'edge coloring' will be used interchangeably. In Fig. 1 we give an example of an instance of the cost list scheduling problem, and a solution to it with total value 9. We assume in the example that $f_e(x) = x$ for all e and x.

The hypergraph coloring problem is in the general case NP-hard. For this reason we consider some sparse and specific hypergraphs in this paper. A *hyperforest* H is a hypergraph such that there exists a spanning tree T (simple graph) for which each edge of H induces a connected subtree in T. A connected hyperforest is called a *hypertree*. We say that a hypergraph is *linear* if no two edges share more then one vertex.

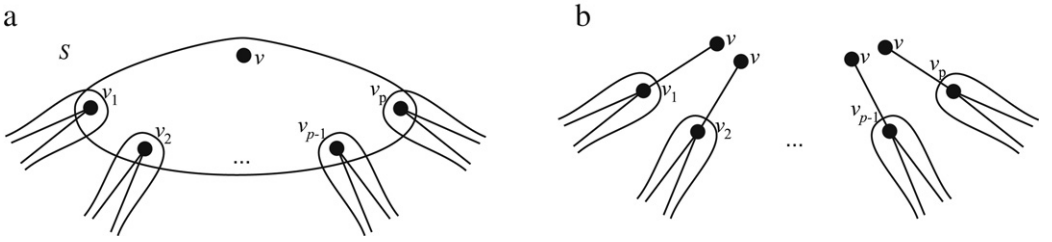

Fig. 2. (a) Hypertree S (b) Hypertrees $S_1, S_2, \ldots, S_p$ created after splitting hypertree S.

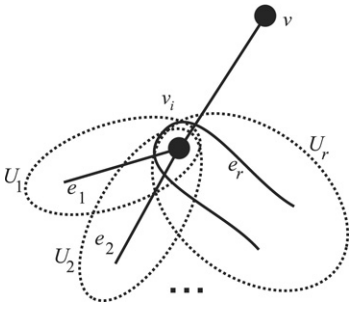

Fig. 3. Hypertree S_i and its subhypertrees $U_1, U_2, \ldots, U_r$.

3. Cost list edge-coloring of linear hypertrees

The stated definition of a hypergraph allows 1-edges (loops) to appear, however, without loss of generality we may consider hypergraphs without 1-edges. Indeed there is a simple procedure that allows us to eliminate all loops. Suppose that $e = \{v\}$ is a 1-edge. Associate one additional vertex v' with e. We can replace e with edge $e' = \{v, v'\}$. Note that such an exchange does not affect the coloring of the hypergraph and if the hypergraph is a hypertree, the new hypergraph still remains a hypertree. In [6] an optimal cost list coloring algorithm for trees was presented. A similar approach was applied in [12] for the edge-chromatic sum of hypertrees. In the following we present an algorithm for cost list coloring of linear hypertrees.

Theorem 3.1. *An optimal cost list coloring of the edges of a linear hypertree T can be found in time $O(nN^2 \log NW)$, where $N = N(T)$, $n = |V(T)|$ and W is a maximal cost of colors among all edges.*

Proof. Without loss of generality we assume that the list of colors $L(e)$ associated with each edge $e \in E(T)$ is of length $|N(e)| + 1$. A hypergraph with such a collection of lists is certainly colorable. All hypergraphs which do not fulfill this condition can be reduced to hypergraphs which do. If the lists are too long we can just delete the most expensive colors from them, because no more than $|N(e)| + 1$ least expensive colors will be used in optimal coloring. On the other hand, if the lists are too short, we can extend them with very expensive colors of cost $C' = |E(T)| \max_{e \in E(T), x \in L(e)} (f_e(x) + 1)$, for example. It is impossible to find the coloring of the initial hypergraph if and only if the cost of coloring of the modified hypergraph exceeds C'.

We shall sketch a procedure for determining the minimum cost of a list coloring of a hypertree T. One can easily extend this procedure to find an optimal coloring.

In every linear loopless hypertree we can find an edge which contains at least one vertex of degree 1. Let e be such an edge of a linear hypertree S and let e contain a vertex v such that $\Delta(v) = 1$. We define a function $Cost_S(e) : L(e) \to \mathbb{N}$ which to every element $k \in L(e)$ assigns the minimal total cost of coloring of the hypertree S under the assumption that e is colored with color k. Values of the function $Cost_S(e)$ will be obtained recursively. If S consists of only one edge e then $Cost_S(e)(k) = f_e(k)$. Otherwise, let $e = \{v, v_1, v_2, \ldots, v_p\}$ and let $\Delta(v) = 1$. We decompose S into p hypertrees $S_1, S_2, \ldots, S_p$. Each S_i consists of vertex v, vertex v_i, an edge $\{v, v_i\}$ with the same list and cost function as e and the whole connected component (which contains vertex v_i) established after deletion of the edge e from S. This is presented in Fig. 2.

To count the values of the function $Cost_{S_i}(\{v, v_i\})$ we shall create a bipartite graph K^{v,v_i} with weights on the edges. The vertices in the first partition represent subhypertrees $U_1, U_2, \ldots, U_r$ formed after splitting hypertree S_i (each U_j has only one edge e_j containing v_i). This is presented in Fig. 3.

The second partition of K^{v,v_i} consists of vertices representing colors $L(e_1) \cup \cdots \cup L(e_r)$. There is an edge $\{U_j, x\}$ in K^{v,v_i} if and only if $x \in L(e_j)$. The weight of $\{U_j, x\}$ is $Cost_{U_j}(e_j)(x)$. The value of $Cost_{S_i}(\{v, v_i\})(x)$ is equal to the minimal weighted matching in $K^{v,v_i} - \{x\}$ plus $f_e(x)$. Using the values of all functions $Cost_{S_i}(\{v, v_i\})$ it is possible to count $Cost_S(e)$ with the following formula $Cost_S(e)(x) = Cost_{S_1}(\{v, v_1\})(x) + \cdots + Cost_{S_p}(\{v, v_p\})(x) - (p-1)f_e(x)$. The result of the algorithm is

the minimum value of function $Cost_T(e)(x)$, where e is an arbitrary edge of T which contains at lest one vertex of degree 1. In this way we obtain the optimal value of a cost list coloring of the hypertree.

In [7] the authors present an algorithm for finding the minimal weighted matchings in all bipartite graphs $G - \{v\}$, $v \in V(G)$ with complexity $O(m'\sqrt{n'}\log n'W)$, where W is the maximal weight, n' is the number of vertices and m' is the number of edges in G. In our case $n' \leq \Delta(v_i) + \Delta(v_i)(N+1)$ and $m' \leq \Delta(v_i)(N+1)$. The complexity of calculating all the values of the function $Cost_{S_i}(\{v, v_i\})(x)$ is $O(\Delta^{3/2}(v_i)N^{3/2}\log \Delta NW)$. So the total complexity of the whole algorithm is bounded by $\sum_{v_i} O(\Delta^{3/2}(v_i)N^{3/2}\log \Delta NW) = O(n\Delta^{1/2}N^{3/2}\log \Delta NW)$ since our hypergraph is a linear hypertree. The final complexity result is $O(nN^2 \log NW)$ as $\Delta(H) \leq N(H)$. ■

If our set of tasks generates a hypergraph which is a set of disconnected hypertrees, then we can color such a hyperforest by coloring each connected component separately since their colorings do not interfere.

4. Cost list edge-coloring of hypergraphs with few cycles

Let us consider a class of hypergraphs which is a generalization of hypertrees. We define *k-hypertrees* as hypertrees with k additional edges, deletion of which reduces a k-hypertree to a linear hypertree.

Lemma 4.1. *A hypergraph $H = (V, E)$ is a linear hypertree if and only if it is connected and $|E| = |V| - 1 - \sum_{e\in E}(\Psi(e) - 2)$.*

Proof. A simple graph is a tree if and only if it is connected and $m = n - 1$, where m is the number of edges and n is the number of vertices. Notice that hypergraph H is a linear hypertree if and only if the graph obtained by changing all the edges of H into trees is a tree. Hence each interchange of a connected subtree into a hyperedge in a certain tree reduces the number of edges by $\Psi(e) - 2$. ■

Theorem 4.1. *For every fixed k there is an algorithm with complexity $O(nN^{2+k}\log NW + n^{k+1})$ for an optimal cost list edge-coloring of a k-hypertree.*

Proof. We will use analogous method as in [6] for cost list coloring of graphs with few cycles. At the very beginning we have to find out which k edges are the edges whose deletion changes a k-hypertree into a linear hypertree. In linear time we can check whether a hypergraph is connected. By Lemma 1 we can conclude that checking if a connected hypergraph is a linear hypertree can also be done in linear time. With a brute force method we need to examine $\binom{m}{k} = O(m^k) = O(n^k)$ different k-sets of edges. Assume that deletion of edges $e_1, e_2, \ldots, e_k$ form hypergraph $H = (V, E)$ makes it hypertree. We can list precolor those edges in $O(N(H)^k)$ different ways. Let us change each of those edges $e_i = \{v_1, v_2, \ldots, v_{\Psi(e_i)}\}$ into $\Psi(e_i)$ pendant edges $\{v_j, v'_j\}$, for $j = 1, \ldots, \Psi(e_i)$, where v'_j is a new additional vertex, and assign to each new edge a 1-element list $\{c(e_i)\}$ of cost 0, where $c : \{e_1, e_2, \ldots, e_k\} \to \mathbb{N}$ is a precoloring function. Now we need to call the cost list edge-coloring algorithm for hypertrees. The total cost must include the costs $f_{e_i}(c(e_i))$ of the precolored edges $e_1, e_2, \ldots, e_k$. Since this step must be executed $O(N^k)$ times the complexity of the algorithm follows. ■

5. NP-completeness proof

The reader may ask: what is the complexity of the problem on non-linear hypergraphs? In this section we answer this question by showing that the problem is NP-hard even for hypergraphs with 2 vertices. We shall discuss a list edge-coloring problem for which the CLEC problem is a generalization.

Theorem 5.1. *The problem of list edge-coloring is NP-complete even for general hypergraphs with 2 vertices.*

Proof. We will reduce to our problem the classical 3-SAT problem [8]. Let Φ be a Boolean formula. Let $C = \{c_1, \ldots, c_m\}$ be a set of clauses and $X = \{x_1, \ldots, x_n\}$ the set of variables. By $X^* = \{x, \bar{x} : x \in X\}$ denote the set of all literals. Our aim is to construct a hypergraph F with only 2 vertices that has its list coloring if and only if Φ is satisfied. Let $g : \{1, \ldots, m\} \times X^* \to \{1, \ldots, 2mn\}$ be an arbitrary one-to-one function, we assume that $g(m+1, x) = g(1, x)$. For each $x \in X$ we construct auxiliary hypergraph F_x on 2 vertices u, v. F_x has $2m$ edges: between u and v there are parallel edges $e_{i,x}^{\{u,v\}}$ and 1-edges containing $ve_{i,x}^{\{v\}}$ for $i = 1, \ldots, m$. We assign the two-element lists to the edges as follows (see Fig. 4):

$$L(e_{i,x}^{\{u,v\}}) = \{g(i, x), g(i, \bar{x})\}$$
$$L(e_{i,x}^{\{v\}}) = \{g(i, \bar{x}), g(i+1, x)\} \quad (1)$$

F_x can be colored in only two ways:

1. $c(e_{i,x}^{\{v\}}) = g(i, \bar{x})$ and $c(e_{i,x}^{\{u,v\}}) = g(i, x)$ for $i = 1, \ldots, m$—then on the edges incident with u we have missing colors $g(i, \bar{x})$,
2. $c(e_{i,x}^{\{v\}}) = g(i+1, x)$ and $c(e_{i,x}^{\{u,v\}}) = g(i, \bar{x})$ for $i = 1, \ldots, m$—then on the edges incident with u we have missing colors $g(i, x)$.

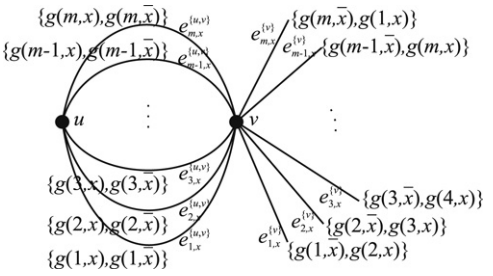

Fig. 4. Auxiliary hyperpath F_x with the number and list of available colors on each edge.

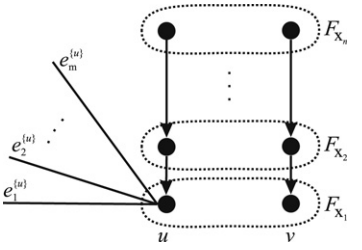

Fig. 5. Construction of hypergraph F.

The first case represents the situation when variable x has value "false", the second when x is "true". The whole of F is constructed in the following way:

1. We unify all the vertices u, v in all F_x. In this way we obtain one hypergraph on two vertices. Note that as g is one-to-one, the colors on the edges of F_x and F_y do not interfere.
2. For each clause $c_i \in C$ we associate with vertex u one additional 1-edge $e_i^{\{u\}}$ with list of colors

$$L(e_i^{\{u\}}) = \{g(i, p_i), g(i, q_i), g(i, r_i)\} \tag{2}$$

where p_i, q_i, r_i are the literals of clause c_i.

In Fig. 5 we show a construction of hypergraph F.

Notice that we can assign a color to $e_i^{\{u\}}$ if at least one color among $L(e_i^{\{u\}})$ is available. Assignment of a color to $e_i^{\{u\}}$ means that we assign the value "true" to one of the literals of c_i. Hence Φ is satisfied if and only if the hyperpath F is colorable. ■

Corollary 5.1. *Cost list edge-coloring of hypergraphs is NP-hard for hypergraphs with* 2 *vertices.*

Proof. Let us consider an instance of list edge-coloring problem. We can easily solve it using CLEC with lists of length $N(H) + 1$. For each hyperedge e it is enough to extend its lists to $N(e) + 1$ elements so that old colors get cost 0 and the new colors get cost 1. The minimal total cost is 0 if and only if a list edge-coloring exists. ■

Notice that every hypergraph with 2 vertices is also a hypertree, so we can conclude that problem of list edge-coloring of general hypertrees is NP-hard.

6. Final remarks

Notice that the CLEC problem is a very versatile one. First, all the problems without lists can be easily solved by adding lists consisting of $N(H) + 1$ cheapest colors to all edges. The costs of colors depend on the criterion. Let us come back to the language of task scheduling. The size of lists of available time slots for tasks is linear in terms of the number of edges of the scheduling hypergraph. Therefore, if we know a polynomial algorithm for cost list edge-coloring for a certain class of scheduling hypergraphs (in our case: linear hypertrees), we can polynomially satisfy many scheduling criteria.

- L_{max} criterion—Let d_j be a due-date of the task represented by edge e_j. For certain C let us establish all the costs $f_{e_j}(x) = 0$ for $x \leq d_j + C$ and $f_{e_j}(x) = 1$ for $x > d_j + C$. Notice that the procedure to solve the CLEC problem will give a total cost of 0 if and only if $L_{max} \leq C$. Hence L_{max} can be calculated using the binary search algorithm.
- C_{max} and T_{max} criterion—The algorithm which finds L_{max} can be easily applied to find C_{max} and T_{max}, since $T_{max} = \max\{0, L_{max}\}$ and C_{max} is a special case of T_{max}.
- $\Sigma w_j T_j$ criterion—The cost of all the elements of lists with $x \leq d_j$ is 0, other elements have cost $f_{e_j}(x) = (x - d_j)w_j$, where w_j is a priority factor from the $\Sigma w_j T_j$ criterion for task represented by edge e_j.
- $\Sigma w_j C_j$ criterion—This is a special case of $\Sigma w_j T_j$.

- $\Sigma w_j U_j$—Let us set the costs of tasks as follows: $f_{e_j}(x) = 0$ for $x \leq d_j$ and $f_{e_j}(x) = w_j$ for the remaining elements, where w_j is again the priority factor.
- bicriterial problem, eg. C_{max}, ΣC_j—In this case at the beginning we search for the optimal scheduling with respect to C_{max} and then we remove from the lists all the time slots exceeding the optimal length. Next we run the algorithm for ΣC_j on the modified instance.

The main result of our paper can be expressed in the form of two corollaries.

Corollary 6.1. *The* $P\left|win, p_j = 1, fix_j\right|$ *crit problem with a scheduling hypergraph which is a k-hypertree for* $k \in O(1)$ *is solvable in polynominal time.*

Corollary 6.2. *The* $P2\left|win, p_j = 1, fix_j\right|$ *crit problem is NP-hard.*

References

[1] L. Bianco, J. Błażewicz, P. Dell'Olmo, M. Drozdowski, Preemptive multiprocessor task scheduling with release times and time windows, Ann. Oper. Res. 70 (1997) 43–55.
[2] J. Błażewicz, P. Dell'Olmo, M. Drozdowski, P. Mączka, Scheduling multiprocessor tasks on parallel processors with limited availability, European J. Oper. Res. 149 (2003) 377–389.
[3] J. Błażewicz, P. Dell'Olmo, M. Drozdowski, M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors, Inform. Process. Lett. 41 (1992) 275–280; Inform. Process. Lett. 49 (1994) 269–270. (corrigendum).
[4] M. Drozdowski, Scheduling multiprocessor tasks—An overview, European J. Oper. Res. 94 (1996) 215–230.
[5] A. Gharbi, M. Haouari, Optimal parallel machines scheduling with availability constraints, Discrete Appl. Math. 148 (2005) 63–87.
[6] K. Giaro, M. Kubale, Chromatic scheduling of 1- and 2-processor UET tasks on dedicated machines with availability constraints, Lecture Notes in Comput. Sci. 3911 (2006) 855–862.
[7] M. Kao, T. Lam, W. Sung, H. Ting, All-cavity maximum matchings, Proc. ISAAC'97, Lecture Notes in Comput. Sci. 1350 (1997) 364–373.
[8] R.M. Karp, Reducibility Among Combinatorial Problems, Complexity of Computer Computations, Plenum Press, New York, 1972, 85–103.
[9] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors, Inform. Process. Lett. 24 (1987) 141–147.
[10] M. Kubale, Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated prosessors, European J. Oper. Res. 94 (1996) 242–251.
[11] E.L. Lloyd, Concurrent task systems, Oper. Res. 29 (1981) 189–201.
[12] P. Obszarski, J. Dąbrowski, Hypergraph model for sparse multiprocessor task scheduling problem, Zesz. Nauk. Wydz. ETI Politech. Gdań. 10 (2006) 499–506 (in Polish).