ACS'02 - SCM conference, Poland, October 23-25, 2002

PREEMPTIVE MULTIPROCESSOR TASK SCHEDULING TO MINIMIZE THE SUM OF COMPLETION TIMES

MICHAŁ MAŁAFIEJSKI, ŁUKASZ KUSZNER,
KONRAD PIWAKOWSKI
Gdańsk University of Technology
Foundation of Informatics Department
Narutowicza 11/12, 80-952 Gdańsk, Poland
mima, kuszner, coni@eti.pg.gda.pl

Abstract: In this paper we consider a problem of preemptive scheduling of multiprocessor tasks on dedicated processors in order to minimize the sum of completion times. Using the standard notation this problem is denoted as $P|fix_j, pmtn|\Sigma Cj$. We give a wide class of polynomial cases in terms of conflicting graphs.

Key words: Operational research, task scheduling.

1. INTRODUCTION

This paper is devoted to the complexity of preemptive scheduling of multiprocessor tasks in order to minimize their sum of completion times. Using the standard three field notation this problem is denoted as $P|fix_j, pmtn|\Sigma Cj$. The NP-hardness of this problem has been shown in [7]. Some restricted subproblems have been considered in [5, 6, 8]. The authors of [5] proved strong NP-hardness of subproblem $P2|fix_j|\Sigma Cj$ and constructed a polynomial time exact algorithm for the preemptive version $P2|fix_j, pmtn|\Sigma Cj$. Drozdowski and Dell'Olmo [6] generalized that result and constructed a polynomial time exact algorithm for solving the problem $P||fix_j| \in \{1,m\}, pmtn|\Sigma Cj$, where m is the number of machines. Małafiejski et. al. [8] proposed an exact algorithm solving the problem $P4|fix_j=2,pmtn|\Sigma Cj$ in $O(n \log n)$ time.

Our result is a generalization of the previous work. We introduce a definition of simplified conflicting graph to point out some new subproblems. This idea provides

us to classify polynomial cases precisely considering only the essentials of the problem.

2. MODEL

A collection $J_1, \ldots J_n \in \mathcal{J}$ of n tasks has to be executed on m machines (processors) $M_1, \ldots M_m \in \mathcal{M}$. Each task J_i requires simultaneous use of a dedicated set of processors fix_i during a given time p_i. Every processor performs at most one task at a time. Such tasks are referred to as *multiprocessor* tasks. By a feasible *preemptive schedule* we mean a function $S : \mathcal{J} \to I^*$, where $I^*=\{x: x$ is a finite sum of disjunctive intervals$\}$ and S satisfies the following conditions:

1. $\left|S(J_j)\right| = \Sigma_{i=1}^{k} \left|[a_{ji}, b_{ji}]\right| = \Sigma_{i=1}^{k} \left(b_{ji} - a_{ji}\right) = p_j$
2. $fix_i \cap fix_j = \varnothing \Rightarrow \left|S(J_i) \cap S(J_j)\right| = 0$

By the *completion time* of any task J_i we mean C_i=max $S(J_i)$, and by *starting time* we mean min $S(J_i)$. For a given schedule S we will write $S(J_i) \prec S(J_j)$ if task J_i is completed before task J_j starts, i.e. max $S(J_i) <$ min $S(J_j)$. Our goal is to minimize the sum of completion times ΣCj.

We can model conflicting requirements between tasks by the so-called *conflicting graph* $G=(\mathcal{J}, \mathcal{E})$, where two jobs J_i and J_j are adjacent ($\{J_i, J_j\} \in \mathcal{E}$) if and only if $fix_i \cap fix_j \neq \varnothing$. Obviously, if G is disconnected then we can reduce the problem of scheduling to easier subproblems corresponding to components of G. So, in the following we suppose that G is connected. We say that $H=(V',E')$ is an *induced subgraph* of $G=(V, E)$ if and only if $V' \subseteq V$ and $E' \subseteq E$ and for all $v_1, v_2 \in V'$ there is $\{v_1, v_2\} \in E' \Leftrightarrow \{v_1, v_2\} \in E$. An induced subgraph H of G is called a *clique* if it is a complete graph [3]. Two tasks J_i and J_j are said to be *related* ($J_i \sim J_j$) if and only if $N(J_i)=N(J_j)$, where $N(J_i)=\{J_i\} \cup \{J_k \in \mathcal{J}: \{J_i, J_k\} \in \mathcal{E}\}$. It is clear that $\sim$ is an equivalence relation, so we can speak about classes $v_1, v_2, \ldots \in V=\mathcal{J}/\sim$ of related tasks. We can observe that each class of tasks forms a clique in G. Let us define a simplified conflicting graph $\hat{G}=(V, E)$, where two vertices v_i and v_j are adjacent if and only if for some $J_{i1} \in v_i$, $J_{j1} \in v_j$ we have $\{J_{i1}, J_{j1}\} \in \mathcal{E}$. It is easy to verify that the definition is correct. Observe, that for any $v, w \in G$ we have $N(v) \neq N(w)$.

Once, we have a simplified conflicting graph with vertices corresponding to classes of tasks, we will continue with the language of conflicting graphs in order to classify scheduling problems, because it is well known, precise and allow us to see the problem in a more general way. Further we will consider some special classes of conflicting graphs, and we give an exact polynomial time algorithm to solve the problem for them.

By a *coloring* of graph $\hat{G}=(V, E)$ we mean a function $c: V \to N$ such that $c(v_1) \neq c(v_2)$ whenever v_1, v_2 are adjacent. A coloring gives us a partition of the vertex set into several independent sets of vertices colored with the same number. The graph $\hat{G}=(V, E)$ is called k-partite if there exists a coloring $c: V \to \{1, ..., k\}$. If, in addition, every two varicolored (colored differently) vertices are adjacent then $\hat{G}$ is called complete k-partite.

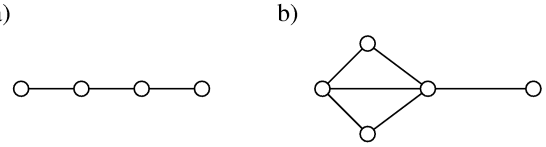

Fig. 1. a) path P_4 and b) dart

Lemma 1 *Any simplified conflicting graph $\hat{G}$ is either complete k-partite or contains at least one of the following: P_4 or the dart as an induced subgraph (see Figure 1).*

Proof: It is clear that P_4 and the *dart* are not induced subgraphs of any complete k-partite graphs, for any $k \in N$. Suppose otherwise, that $\hat{G}$ is a simplified conflicting graph that is not complete k-partite and contains neither P_4 nor the *dart*. Suppose that c colors $\hat{G}$ with minimal number of colors, say k, in such a way that the sum of colors is as small as possible.

If any two varicolored vertices $v, w \in \hat{G}$ are connected then $\hat{G}$ is complete k-partite. So, choose nonadjacent varicolored vertices $v, w \in V$ and suppose that $c(w) < c(v)$. For any two $x,y \in \hat{G}$ by d(x, y) we mean the length of the shortest path from x to y. If d(v, w)≥3 then there is an induced subgraph P_4 of $\hat{G}$, so because $\{v, w\} \notin E$ we have d(v, w)=2. Thus there exists a vertex u such that $\{v, u\} \in E$ and $\{u, w\} \in E$. If in set $N(v)$ there is no vertex colored with $c(w)$ then v could be recolored with $c(w)$, thus we get a coloring with smaller sum of colors. So, there is another vertex $w_1 \in N(v)$ colored with $c(w_1)=c(w)$, hence $\{w_1, w\} \notin E$. Moreover, if $\{w_1, u\} \notin E$ then we have a path (w, u, v, w_1) as an induced subgraph, so $\{w_1, u\} \in E$. Because $\hat{G}$ is a simplified conflicting graph so there exists a vertex x and $v \neq x \in N(v) \setminus N(w_1)$ or $w_1 \neq x \in N(w_1) \setminus N(v)$.

First suppose that $\{x, v\} \in E$ and $\{x, w_1\} \notin E$. We can see that $\{w, x\} \notin E$ since otherwise we have a path (w, x, v, w_1). Now, observe that if $\{u, x\} \notin E$, then vertices x, v, u, w induce a path in G, otherwise if $\{u, x\} \in E$ then vertices w, u, x, v, w_1 induce the *dart* as a subgraph of $\hat{G}$. The other case $\{x, v\} \notin E$ and $\{x, w_1\} \in E$ is analogous. □

Theorem 2 *A conflicting graph G is {P_4, dart}-free if and only if $\hat{G}$ is complete k-partite for some $k \in N$.*

Proof: " $\Rightarrow$ " First, we show that $\hat{G}$ is {P_4,dart}-free. Suppose to the contrary that the path (v_1, v_2, v_3, v_4) is an induced subgraph of $\hat{G}$. We can choose $J_1 \in v_1$, $J_2 \in v_2$, $J_3 \in v_3$, $J_4 \in v_4$, hence there is a path (J_1, J_2, J_3, J_4) as an induced subgraph of G, a contradiction. In the similar way we can show that the *dart* is not an induced subgraph of $\hat{G}$. We have that $\hat{G}$ is {P_4, *dart*}-free. By Lemma 1 we have that $\hat{G}$ is a complete k-partite graph.

" $\Leftarrow$ ". Suppose otherwise that the *dart* is an induced subgraph of G. For every two vertices v, w in the *dart* we have $N(v) \neq N(w)$ hence the *dart* is a subgraph of $\hat{G}$, a contradiction. By analogous argument we can show that P_4 is not an induced subgraph of G. □

3. COMPLETE *K*-PARTITE GRAPHS

In this section we assume that G is such a graph that $\hat{G}$ is complete k-partite. If an independent set $U \subseteq V(\hat{G})$ satisfies the following $\forall_{v \in U} : N(v) \setminus \{v\} = V(\hat{G}) \setminus U$ then we say that the elements in U (classes of vertices from G) are *conjugated*.

We have the following easy observations:

Claim 1 *If $\hat{G}$ is a complete k-partite graph then only vertices from each maximal independent set are conjugated.*

Claim 2 *Only tasks from different conjugated classes can be executed simultaneously.*

Claim 3 *For any two related tasks $J_i \sim J_j$ with processing times $p_i < p_j$ we have $S(J_i) < S(J_j)$ in each optimal schedule S.*

Claim 4 *In every class all tasks have to be scheduled according to the SPT rule in each optimal schedule S.*

We call an interval $[a, b]$ a *block* of task J_i in a schedule S if and only if $[a, b] \subseteq S(J_i)$ and for any $\varepsilon > 0$ we have $[a-\varepsilon, b] \not\subset S(J_i)$ and $[a, b+\varepsilon] \not\subset S(J_i)$

Claim 5 *In every optimal schedule beginning of any block is a beginning of a schedule or a completion time of another task.*

Proof: On the contrary, assume that there is an optimal schedule without this property. Among all such schedules we can choose the one with the smallest number of bad blocks starting neither at the beginning of the schedule nor at a completion time of another task. Let t_p be a moment which is the starting time of the first bad block B_1 of some task J_1. Just before the moment t_p at least one task conflicted with J_1 has to be executed, say J_2. Let $t_p - a$ be the beginning of a schedule or the latest completion time of any task performed before t_p and $t_p + b$ be the earliest completion

time of any task completed after t_p or the moment of restarting the task J_2. We can decrease by a all times of all tasks scheduled between t_p and t_p+b and increase by b all times of all tasks scheduled between t_p-a and t_p. If t_p+b is a completion time, we get a schedule with smaller sum of completion times and if t_p+b is the next block of J_2 we get a schedule with smaller number of bad blocks, a contradiction.

Now we are ready to formulate the most important property of optimal schedules. Since only beginning of a block can be a reason to interrupt the execution of another task, so

Lemma 3 *In every optimal schedule a moment of interruption of a task $J_{i_1} \in v_i$ can be only the moment of completion time for another task J_{j_1} from conjugated class v_j.*

□

Now, we are ready to the next major step. We will transform our problem to $1|out\text{-}tree|\Sigma Cj$, which is polynomially solvable [4]. Having an instance I of $P|fix_j, pmtn|\Sigma Cj$ with $\hat{G}=K_{n1,n2, \ ..., \ nk}$ and a set $A^i = \{v_1^i, v_2^i, ..., v_{n_i}^i, \}$ of n_i conjugated classes with $v_l^i = \{J_{1,1}^i, J_{1,2}^i, ...\}$, we assume that tasks in each class are ordered according to the SPT rule (see Claim 3) in each optimal schedule S, so $p_{1,a}^i \le p_{1,b}^i$ if and only if $a<b$. Let us consider a set of all tasks from classes in A^i : $B^i = \{J_{1,1}^i, J_{1,2}^i, ..., J_{2,1}^i, J_{2,2}^i, ..., J_{n_i,1}^i, J_{n_i,2}^i, ...\}$. Let us permute B^i with τ in such a way that $r_1 = \tau(l_1,k_1) < \tau(l_2,k_2) = r_2$ if and only if $f_{r_1} = \sum_{q=1}^{k_1} p_{l1,q}^i < f_{r_2} = \sum_{q=1}^{k_2} p_{l_2,q}^i$. If $f_{r_1}=f_{r_2}$ then precedence is irrelevant. From all the tasks in B^i we construct a chain of nonpreemptive 1-processor tasks with precedence constraints $J_{r_1} \rightarrow J_{r_2} \rightarrow ... \rightarrow J_{r_q}$ and processing times $p_1=f_{r_1}$, $p_2=f_{r_2}-f_{r_1}$, ..., $p_q=f_{r_q}-f_{r_{q-1}}$, ... Similarly we can create chains of new tasks for other sets of conjugated classes. (See the appropriate construction for Example 1 in Figure 3). Additionally, we add one "dummy" task J_0 with processing time $p_0=0$ and force it to be executed before all other jobs. We have constructed an instance I' of $1|out\text{-}tree|\Sigma Cj$.

Now we can find an optimal schedule S' for I' [1] in $O(n \log n)$ time. Then we can construct a schedule S for I in the following way. If in schedule S' a task J_{r_k} is processed at time t then in S we have that tasks from the corresponding sets of conjugated classes A_i are executed in parallel (cf. Fig. 4). Observe that completion times of the corresponding tasks J_{r_1} and J_{r_1} are equal, so we have

Lemma 4 *For every legal schedule S' for instance I' there exists a schedule S for instance I with the same sum of completion times.*

Lemma 5 *The sum of completion times of each optimal schedule S' for instance I' is equal to the sum of completion times of the optimal schedule for instance I.*

Proof: Let *S* be any optimal schedule for an instance *I*. Considering Claim 4 and Lemma 3we conclude that there exists a corresponding schedule *S*' for an instance *I*' with the same sum of completion times. Suppose that *S*' is not optimal. Having an optimal schedule *S*' for *I*', by Lemma 4 we can construct a schedule *S* for *I* with the same sum of completion times, hence better then optimal *S*, a contradiction. □

Concluding, for an instance *I* of problem P|fix$_j$, pmtn|Σ*Cj* we can construct a simplified conflicting graph in time $O(n^2)$. Then we have to check if that graph is complete k-partite and if so we can find a solution as follows:

Step 1 Split tasks into classes in $O(n^2)$ time.
Step 2 Sort all tasks in each class according to the SPT rule in $O(n \log n)$ time.
Step 3 Construct an instance *I*' of 1|out-tree|Σ*Cj* in $O(n)$ time.
Step 4 Find an optimal solution for *I*' in $O(n \log n)$ time [1].
Step 5 Construct a solution for *I* in $O(n)$ time.

So we have

Theorem 6 *There exists an exact algorithm to solve P|fix$_j$, pmtn, G={P_4,dart}-free|ΣCj problem in $O(n^2)$ time.*

Corollary 7 [5] *The problem P2|fix$_j$, pmtn|ΣCj can be solved in O(n* log *n) time.*

Proof: For this problem $\hat{G} = P_3 = K_{1,2}$. The first step is already done, we have only three classes of tasks, so the complexity is $O(n \log n)$. □

Corollary 8 [6] *The problem P| |fix$_j$| ∈ {1,m}, pmtn|ΣCj can be solved in O(n* log *n) time.*

Proof: In this case *G* is a star $K_{1,m}$. □

Corollary 9 [8] *The problem P4| |fix$_j$|=2, pmtn|ΣCj can be solved in time O(n* log *n).*

Proof: In this case *G* is a 3-partite graph $K_{2,2,2}$.

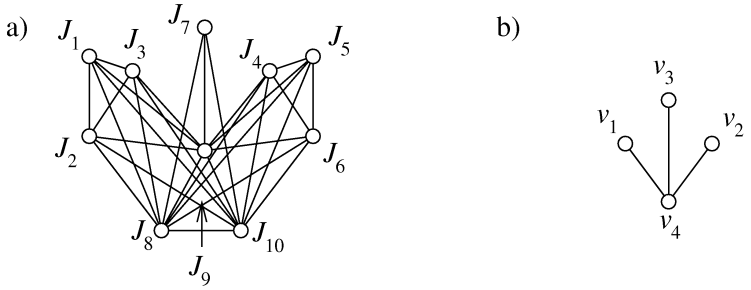

Fig. 2. a) conflicting graph and b) simplified conflicting graph

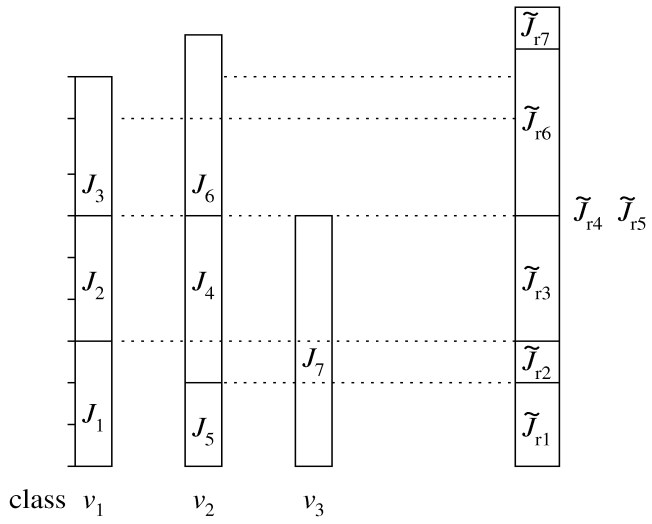

Fig. 3. Transformation of a set of conjugated classes of tasks into a chain of new tasks

Let M be a set of machines M_1,..., M_5 and J=J_1,..., J_{10} a set of tasks with fix_1={M_1}, p_1=3, fix_2={M_1}, p_2=3, fix_3={M_1}, p_3=4, fix_4={M_2}, p_4=4, fix_5={M_2, M_3}, p_5=2, fix_6={M_2, M_3}, p_6=5, fix_7={M_4, M_5}, p_7=6, fix_8={M_1, M_2, M_3, M_4}, p_8=1, fix_9={M_1, M_2, M_5}, p_9=2, fix_{10}={M_1, M_2, M_5}, p_{10}=8.

In this case we have four classes of tasks v_1={J_1, J_2, J_3}, v_2={J_4, J_5, J_6}, v_3={J_7}, v_4={J_8, J_9, J_{10}}. The simplified conflicting graph for this case is $K_{1,3}$. Classes v_1, v_2, v_3 are conjugated. Appropriate conflicting graph and simplified conflicting graph are shown in Figure 2 Next step is to transform the instance of the problem to an instance of $P|out\text{-}tree|\Sigma Cj$. Transformation shown in Figure 3 will lead to chains $J_{r1} \rightarrow J_{r2} \ldots \rightarrow J_{r7}$ with processing times 2, 1, 3, 0, 0, 4, 1 and $J_{s1} \rightarrow J_{s2} \rightarrow J_{r3}$ with processing times 1, 2, 8 preceded by additional task Z_0. The optimal schedule for this instance is $Z_0 \rightarrow J_{s_1} \rightarrow J_{r_1} \rightarrow J_{r_2} \rightarrow J_{r_3} \rightarrow J_{r_4} \rightarrow J_{r_5} \rightarrow J_{s_2} \rightarrow J_{r_6} \rightarrow J_{r_7} \rightarrow J_{s_3}$. Now we can give the final solution which is shown in Figure 4.

4. CONCLUSIONS

An idea of simplified conflicting graph introduced in the paper seems to be useful to compare different subcases of the multiprocessor task scheduling. We have shown that $P|fix_j,\ G=\{P_4,dart\}\text{-}free,\ pmtn|\Sigma Cj$ problem is solvable in a polynomial time. Our solution to that problem is a generalization of some previously known results.

The complexity of the problem remains open even when the simplified conflicting graph is P_4 or *dart*. These should be interesting topic for further research. Another interesting unsolved case is $P|fix_j, G=\text{path}, pmtn|\Sigma Cj$.

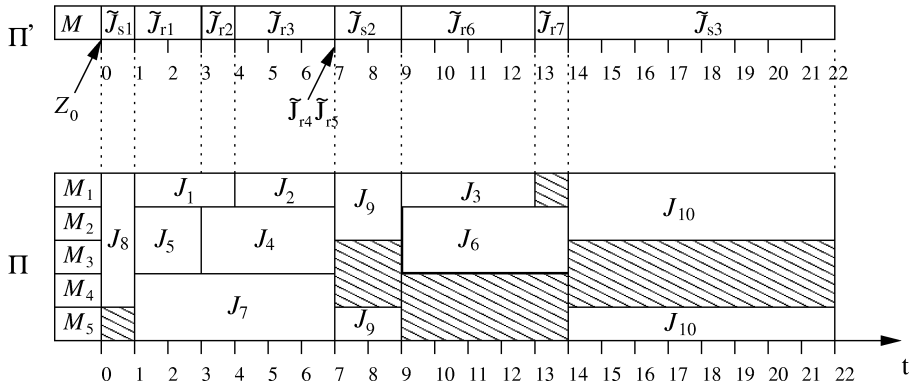

Fig. 4. Ultimate schedule obtained by the algorithm.

5. REFERENCES

[1] Adolphson D., Hu T. C. 1973. 'Optimal linear ordering'. SIAM J. Appl. Math. 25, pp. 403–423.

[2] Błażewicz J., Ecker K., Pesch E., Schmidt G., Węglarz J. 2001. Scheduling in Computer and Manufacturing Processes, Springer-Verlag,.

[3] Brandstädt A., Van Bang L. Spinrad J.P. 1999. 'Graph Classes: a Survey'. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[4] Brucker P. 1995. Scheduling Algorithms, Springer-Verlag, Berlin.

[5] Cai X., Lee C.-Y., Li C.-L. 1998. 'Minimizing total completion time in two-processor task systems with prespecified processor allocations'. Naval Res. Logist. 45, pp. 231–242.

[6] Drozdowski M., Dell'Olmo P. 'Scheduling multiprocessor tasks for mean flow time criterion'. Comput. Oper. Res. 27, pp. 571–585.

[7] Hoogeveen J.A., van de Velde S.L., Veltman B. 1994 'Complexity of schedulingmultiprocessor tasks with prespecified processor allocations' Disc. Appl. Math. 55, pp. 259–272.

[8] Małafiejski M., Kuszner Ł., Kubale M. 2002. 'Preemptive scheduling of biprocessor tasks on dedicated machines to minimize sum of completion times' (in Polish), Zeszyt Nauk. Politechniki Śląskiej nr 1554, Ser. Automatyka, z. 134, pp. 313–325.