Performance Evaluation 20 (1994) 361–371

PERFORMANCE EVALUATION
An International Journal

Scheduling preemptive multiprocessor tasks on dedicated processors [+]

L. Bianco [b], J. Błażewicz [a,*], P. Dell'Olmo [b], M. Drozdowski [a]

[a] *Instytut Informatyki, Politechnika Poznańska, Poznań, Poland*
[b] *Istituto Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, Rome, Italy*

(Received 3 August 1992; revised 3 May 1993)

Abstract

In the classical scheduling theory it is widely assumed that any task requires for its processing only one processor at a time. In this paper the problem of deterministic scheduling of tasks requiring for their processing more than one processor at a time, i.e., a constant set of dedicated processors, is analyzed. Schedule length is assumed to be a performance measure. Tasks are assumed to be preemptable and independent. Low order polynomial algorithms for simple cases of the problem are given. Then a method to solve the general version of the problem for a limited number of processors is presented, while the case of an arbitrary number of processors is known to be NP-hard. Finally, a version of the problem, where besides processors every task can also require additional resources, is considered.

1. Introduction

One of assumptions commonly imposed in the classical scheduling theory is that a task can be processed by only one processor at a time [5]. With the progress in technology and parallel algorithm applications, however, this assumption is becoming not so evident. For instance, this is the case for self-testing multi-microprocessor systems where one processor tests others or diagnostic systems in which tested elements are simultaneously stimulated and their outputs analyzed [7,13]. This is also the case in the fault-tolerant systems where multiple copies of the same program are working simultaneously and the final decision is established by voting [10]. This approach may become useful with the development of new parallel algorithms and corresponding future task systems.

* Corresponding author. Tel. +48 61 790 790, fax +48 61 77 15 25, Email: blazewic@ plpotu51.bitnet.
+ The research has been partially supported by grant KBN-3P40600106/PI and by project CRIT.

Optimal scheduling of tasks requiring more than one processor at a time improves performance of a computer system. For example, reduces time consumed by tests performed temporarily by a computer system, reduces the amount of memory necessary to hold intermediate data, reduces time wasted during waiting for synchronization between copies of the program running on different processors. A deterministic approach (i.e. assuming knowledge of some time parameters of a task set) is especially justified in real-time (or hard-real-time) systems. We see that the examples discussed above are often performed in a real-time mode, thus justifying the use of a deterministic model of performance. When analyzing such problems, one tries to construct scheduling algorithms of the lowest possible time-complexity (low order polynomial-time algorithms). However, it is also known for some time that not for all the problems polynomial time algorithms exist. It is especially true for the NP-hard problems, for which rather suboptimal algorithms must be designed (see [9] for the detailed treatment of the subject). In the following we will analyze scheduling problems along these lines.

The problem of deterministic scheduling tasks requiring more than one processor at a time has been studied in many publications. For the schedule length ($C_{\max}$) performance measure and tasks requiring constant *numbers* of identical processors it has been analyzed in [3]. A uniform processor system has been analyzed in [4]. A more general model of multiprocessor task system has been given in [8]. It has been assumed in [8] that a processing time of a task depends on the *number* of processors processing it. Various subcases of the problem of scheduling *nonpreemptable* tasks requiring a *dedicated set* of processors have been analyzed in [1,13,14,2], the case of preemptable tasks being considered in [15]. In this paper, we extend the above results by solving open questions concerning the case of preemptable tasks and dedicated processors.

Before presenting the results, let us formulate the problem. We are given a set of *tasks* $T=\{T_1,\dots,T_n\}$, a set of processors $\mathscr{P}=\{P_1, P_2,\dots,P_m\}$ and a set of *additional resources* $\mathscr{R}=\{R_1\}$ available in m_1 units. Tasks are *independent* and *preemptable*, i.e. each task can be interrupted and restarted later at no cost. Each task requires for its processing a *dedicated set* of processors and possibly a unit of resource R_1. The set of independent tasks T is partitioned into subsets according to the set of required processors. In particular, let T^D be the set of tasks which require set D of processors. Task $T_i^D \in T^D$ has *processing time* t_i^D and requires set D of processors. For the sake of simplicity, we define by $t^D=\sum_{T_i\in T^D} t_i^D$ the total time required by all tasks which use the set of processors D. Thus, $t^{1,2,3}$, is the total processing time required by tasks which use processors P_1, P_2, P_3. We denote additionally T_R^D tasks requiring additional resources and the same set of processors, the sum of their execution times is denoted t_R^D. Please note that T^D includes in our notation also T_R^D.

The performance measure of the scheduling problem is schedule length $C_{\max}=t^{\mathscr{P}}+C_{\max}^*$, where $t^{\mathscr{P}}$ is the total processing time of tasks requiring all processors.

To denote analyzed problems we will use an extended version of the scheme proposed by Graham et al. [11] with later extensions [6,17]. In this notation the scheduling problem is described by three fields. The first field describes the processor system. In this paper it will be letter P optionally followed by some constant which denotes a number of processors. If there is no constant after P, then the number of processors is not fixed by the formulation of the problem but by the current instance of the problem. The second field describes a task system. We will be using the word "pmtn" to denote that tasks are preemptable, the word "fix$_j$" to denote multiple processor requirement of tasks such that task T_j can be processed on exactly 1 subset of processors [17]. The optional symbol "$p_j = 1$" denotes that tasks are of equal length. Presence of the symbol "res$\rho\sigma\tau$" denotes a requirement of additional resources of ρ types each with σ units available, required by tasks in the number of at most τ units for each task. The third field is the performance measure, in our case it is $C_{\max}$.

The organization of the paper is as follows. Section 2 presents simple algorithms for the case of two, three and four processors. Section 3 presents results for the general case of the problem. In Section 4 usage of additional resources is considered.

2. Polynomial-time algorithms

In this paragraph the problem of scheduling preemptable tasks requiring a set of processors simultaneously, in the case of a processor system consisting of two, three and four processors, respectively, will be considered. Tasks do not require additional resources. Low order polynomial time algorithms will be given.

2.1. $P2 \mid fix_j, pmtn \mid C_{max}$

Everything one can do in this case is to schedule tasks from $T^{1,2}$ in any order at the beginning (or at the end) of the schedule and tasks from T^1, T^2 in any order on appropriate processors. This method is obviously optimal, since preemptions do not decrease the schedule length.

2.2. $P3 \mid fix_j, pmtn \mid C_{max}$

Let us consider graph G(V, E), where $V = \{T^1, T^2, T^3, T^{1,2}, T^{1,3}, T^{2,3}\}$ and an edge is joining two task types if they cannot be executed in parallel. We will name G(V, E) a *competition graph*. There are four cliques in this graph. These are (we enumerate nodes only): $A = \{T^1, T^{1,2}, T^{1,3}\}$, $B = \{T^2, T^{1,2}, T^{2,3}\}$, $C = \{T^3, T^{1,3}, T^{2,3}\}$, $D = \{T^{1,2}, T^{1,3}, T^{2,3}\}$. The following theorem reduces the problem of finding a schedule of minimum length to the analysis of the cliques of graph G.

Theorem 1. *The minimal length of the schedule for the problem $P3 \mid fix_j, pmtn \mid C_{max}$ is equal to the greatest sum of processing times of any clique in graph G, i.e.,*

$$C^*_{\max} = \max_{J \in \{A,B,C,D\}} \left\{ \sum_{\substack{T_i^S \in S \\ S \in J}} t_i^S \right\}.$$

Proof. Obviously the schedule length cannot be smaller than this sum because tasks in any clique cannot be executed in parallel.

Now, we are going to show that if the schedule length $C^*_{\max}$ is a sum of processing times of tasks in a certain clique in graph G, then the schedule is feasible also for tasks not included in this clique.

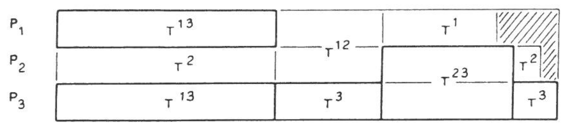


Fig. 1. A schedule corresponding to clique C.

Firstly, let us assume that clique C defined the schedule length. Then the resulting schedule is presented in Fig. 1. This is a feasible schedule because $t^{1,2} \leq t^3$ – otherwise clique D would be chosen for $C^*_{\max}$. Also $t^{1,2} + t^2 \leq t^{1,3} + t^3$ because clique B is not chosen and $t^{1,2} + t^1 \leq t^{2,3} + t^3$ because clique A is not chosen for $C^*_{\max}$. Thus, tasks of types T^1, $T^{1,2}$, T^2 can be scheduled feasibly as in Fig. 1. The same reasoning applies to cliques A and B if we renumber processors appropriately.

Secondly, let us assume that clique D was chosen. Then, the resulting schedule is presented in Fig. 2. This schedule is feasible because $t^{1,2} \geq t^3$, $t^{2,3} \geq t^1$, $t^{1,3} \geq t^2$, otherwise clique D would not have been chosen. □

Following Theorem 1, one can solve optimally problem P3 | fix$_j$,pmtn | $C_{\max}$ in O(n) time, defining firstly an optimal length $C^*_{\max}$ and then scheduling tasks as depicted in Fig. 1 or in Fig. 2.

2.3. *P4 | fix$_j$,pmtn | C_{max}*

Let us consider the competition graph for four processors. Since $T^{1,2,3,4}$ cannot be executed in parallel with any other task type, we have to schedule it at the beginning (or at the end) of the schedule and we can exclude it from further analysis. Consider the competition graph G(V, E) consisting of $V = \{T^1, T^2, T^3, T^4, T^{12}, T^{13}, T^{14}, T^{23}, T^{24}, T^{34}, T^{123}, T^{124}, T^{134}, T^{234}\}$ and edges joining tasks that cannot be executed in parallel. There are twelve cliques in this graph. These are (we enumerate nodes):

$$\begin{aligned}
\mathrm{A} &= \{T^1, T^{12}, T^{13}, T^{14}, T^{123}, T^{124}, T^{134}\},\\
\mathrm{B} &= \{T^2, T^{12}, T^{23}, T^{24}, T^{123}, T^{124}, T^{234}\},\\
\mathrm{C} &= \{T^3, T^{13}, T^{23}, T^{34}, T^{123}, T^{134}, T^{234}\},\\
\mathrm{D} &= \{T^4, T^{14}, T^{24}, T^{34}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{E} &= \{T^{12}, T^{13}, T^{23}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{F} &= \{T^{12}, T^{14}, T^{24}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{G} &= \{T^{13}, T^{14}, T^{34}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{H} &= \{T^{23}, T^{24}, T^{34}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{I} &= \{T^{12}, T^{13}, T^{14}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{J} &= \{T^{12}, T^{23}, T^{24}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{K} &= \{T^{13}, T^{23}, T^{34}, T^{123}, T^{124}, T^{134}, T^{234}\},\\
\mathrm{L} &= \{T^{14}, T^{24}, T^{34}, T^{123}, T^{124}, T^{134}, T^{234}\}.
\end{aligned}$$

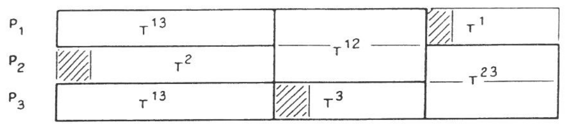


Fig. 2. A schedule corresponding to clique D.

Now, the scheduling problem reduces again to the analysis of the cliques in the competition graph.

Theorem 2. *The minimal length of the schedule for the problem $P4 \mid fix_j, pmtn \mid C_{\max}$ is equal to the greatest sum of processing times of any clique in graph G, i.e.,*

$$C^*_{\max} = \max_{J \in \{A,B,C,D,E,F,G,H,I,J,K,L\}} \left\{ \sum_{\substack{T_i^S \in S \\ S \in J}} t_i^S \right\}.$$

Proof. Note that there is only a limited set of task types that can be executed in parallel (cf. Figs. 3–5). One can only schedule three-processor tasks with appropriate types of uni-processor tasks; duo-processor tasks in appropriate pairs or with at most two appropriate uni-processor tasks; finally, uni-processor tasks with different requirements can be scheduled in parallel. The schedule length cannot be smaller than the weight of the above cliques (i.e., the sum of execution times of tasks included in the clique).

Let us first analyze cliques A, B, C and D including uni-processor tasks. Assume A is the clique with the greatest weight. Then, combining cliques A and E we can derive the inequality:

$$t^1 + t^{12} + t^{13} + t^{14} + t^{123} + t^{124} + t^{134} \geqslant t^{12} + t^{13} + t^{23} + t^{123} + t^{124} + t^{134} + t^{234}$$

and from the above:

$$t^1 \geq t^{23} - t^{14} + t^{234}.$$

In a similar way we obtain, combining A with F, A with G and A with H, respectively, the following set of inequalities:

$$t^1 \geq t^{234} + t^{24} - t^{13},$$
$$t^1 \geq t^{234} + t^{34} - t^{12},$$
$$t^1 \geq t^{234} + t^{34} - t^{12} + t^{24} - t^{13} + t^{23} - t^{14}.$$

Combining clique A with cliques I till L we derive the inequalities:

$$t^1 \geq t^{234},$$
$$t^1 \geq t^{234} + t^{24} - t^{13} + t^{23} - t^{14},$$
$$t^1 \geq t^{234} + t^{23} - t^{14} + t^{34} - t^{12},$$
$$t^1 \geq t^{234} + t^{34} - t^{12} + t^{24} - t^{13}.$$

From the above eight inequalities we can conclude that T^1 tasks cannot be fully executed when assigned only in parallel with $T^{2,3,4}$. Moreover, they cannot be executed fully neither with any one duo-processor task type and $T^{2,3,4}$ nor with any pair of duo-processor tasks and $T^{2,3,4}$.

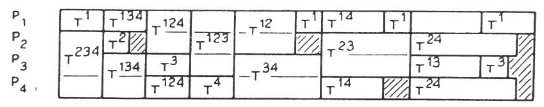


Fig. 3. A schedule corresponding to clique A.

Finally, T^1 will not be executed fully with all feasible duo-processor tasks and $T^{2,3,4}$. From this we can conclude that tasks of type T^1 must be executed as the last in the schedule. Other uni-processor tasks cannot be executed after completion of T^1, otherwise clique A would not be maximal. Thus, the length of the schedule is equal to the sum of processing times of tasks using processor P_1. The optimal schedule looks like in Fig. 3. Note, that this reasoning applies also to cliques B, C and D, respectively, if we renumber the processors.

Now, let us assume that one of cliques E, F, G or H is maximal. Without loss of generality we may assume E to be the one. According to the above method one can combine cliques E and I, E and J, E and K, E and L and obtain from this, respectively, a set of inequalities:

$$t^{23} \geq t^{14},$$
$$t^{13} \geq t^{24},$$
$$t^{12} \geq t^{34},$$
$$t^{12} + t^{13} + t^{23} \geq t^{14} + t^{24} + t^{34}.$$

From these inequalities one can conclude, that all the duo-processor tasks can be executed feasibly during an execution of tasks of the types T^{12}, T^{13}, T^{23} and the schedule can look like in Fig. 4. No uni-processor task can be processed after the execution of the tasks of this clique, otherwise E would not be maximal. This reasoning can be applied to cliques E, F, G and H, respectively, if renumbering of processors is applied.

Finally, let one of the cliques I through L be maximal. Without loss of generality we may assume I to be the one. In the same way combining I and E, I and F, I and G, I and H we obtain:

$$t^{14} \geq t^{23},$$
$$t^{13} \geq t^{24},$$
$$t^{12} \geq t^{34},$$
$$t^{12} + t^{13} + t^{14} \geq t^{23} + t^{24} + t^{34}.$$

The same arguments as for clique E can be applied. The schedule is presented in Fig. 5. □

Again, let us sum up this section with the observation that the schedule for each of the above cases can be built in time O(n). This is a result of the fact that the optimal schedules have a

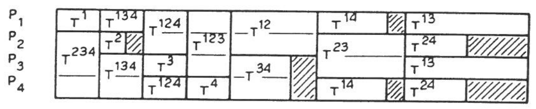


Fig. 4. A schedule corresponding to clique E.

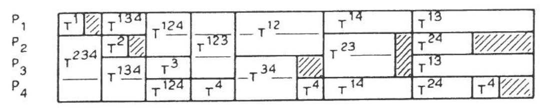


Fig. 5. A schedule corresponding to clique I.

fixed structure (cf. Figs. 3 through 5) and each task is taken into account during the scheduling process either only a limited number of times if it is interrupted or only once otherwise.

3. A general case

In this paragraph a general case of the problem will be considered. It is known [15] that the problem is NP-hard for the case of an arbitrary number of processors m and it will be shown that the problem is solvable in polynomial time if m is limited.

Theorem 3. *[15] $P \mid fix_j,pmtn,p_j = 1 \mid C_{\max}$ is NP-hard.*

We would like to comment this theorem. From the computational complexity point of view, problem $\mathrm{P} \mid \mathrm{fix}_j,\mathrm{pmtn},\mathrm{p}_j = 1 \mid C_{\max}$ is (for m not fixed) computationally difficult as is the problem $\mathrm{P} \mid \mathrm{fix}_j \mid C_{\max}$ where nonpreemptable tasks are considered [2]. To demonstrate the difference between those problems, note that in the nonpreemptable case any permutation of feasible sets could change the schedule length $C_{\max}$, while in the preemptive one, any permutation of feasible sets does not change $C_{\max}$. One can see that the complexity of both problems follows (among others) from a simultaneous competition of tasks for the processors (which directs us to coloring problems).

This complexity result is obviously valid for $\mathrm{P} \mid \mathrm{fix}_j,\mathrm{pmtn} \mid C_{\max}$ $\mathrm{P} \mid \mathrm{fix}_j,\mathrm{pmtn},\mathrm{res}111 \mid C_{\max}$ and then for $\mathrm{P} \mid \mathrm{fix}_j,\mathrm{pmtn},\mathrm{res}\ldots \mid C_{\max}$. It follows that none of these problems can be solved in polynomial time. However, if the number of processors m in the system is limited (fixed), the problem can be solved via linear programming approach.

Let us consider now the case in which either m – the number of processors – is limited or the set of possible sets of processors required by tasks has a limited cardinality.

By a *processor feasible set* we mean here a set of tasks which can be processed simultaneously. Let there be Z different processor feasible sets with processing time x_i for set i. Let Q_j^D be a set of indices of feasible sets containing T_j^D. Now, our problem can be formulated as a linear programming problem as follows:

$$\min \sum_{i=1}^{Z} x_i, \tag{1}$$

subject to

$$\sum_{i \in Q_j^D} x_i = t_j^D, \; j = 1 \ldots n, \tag{2}$$

$$x_i \geq 0, \; i = 1 \ldots Z. \tag{3}$$

For m fixed, Z is limited and is $O(n^m)$. For a fixed cardinality of the set of possible sets of processors required by tasks, Z is also limited. Since linear programming problems can be solved in the time bounded by polynomial in the number of variables and constraints, then our problem can be solved in polynomial time, too (cf. [16] for a discussion of the two-phase linear programming approach to the resource-constrained scheduling).

In the same way one can solve $Pm\,|\,\text{fix}_j,\text{pmtn},\text{res}\ldots\,|\,C_{\max}$ taking into account also resources during generation of feasible sets.

4. Scheduling on dedicated processors with additional resources

In this section we consider the case in which a task may require (besides a set of processors) one unit of an additional resource (eg. communication medium) available in m_1 units.

We have to consider different situations:

(1) $m_1 = 1$: in this case it is possible to represent a problem as if the additional resource was a dedicated processor. The problem $Pk\,|\,\text{fix}_j,\text{pmtn},\text{res}111\,|\,C_{\max}$ is then equivalent to $P(k+1)\,|\,\text{fix}_j,\text{pmtn}\,|\,C_{\max}$. For example $P4\,|\,\text{fix}_j,\text{pmtn}\,|\,C_{\max}$ is equivalent to $P3\,|\,\text{fix}_j,\text{pmtn},\text{res}111\,|\,C_{\max}$.

To consider other subcases, let us denote by q the maximum size of a set of tasks which can be executed in parallel while not taking into account additional resources.

(2) $m_1 \geq q$: In this case all resource requirements can be easily satisfied and a schedule can be built without considering additional resources.

(3) $m_1 < q$: In this case we may have a set of tasks, feasible with respect to the processor usage, which are not feasible with respect to resource usage. This is possible because the maximum number of tasks that can be executed in parallel on the dedicated processors is given by q. So, we could have q tasks all requiring the use of additional resource and since $m_1 \leq q$, conflicts on the use of resource would arise. A simple case here is $m_1 = q - 1$. If $m_1 = q - 1$, there is only a limited number of feasible sets requiring more resources than available. In this case a schedule feasible with respect to the processors is also feasible with respect to resources if there is at least one type of tasks included in maximal feasible set for which all the tasks requiring an additional resource can be executed fully outside the maximal feasible set. In other words, if there is enough time in the schedule feasible with respect to the processors to accommodate all the tasks requiring the resource from at least one type of tasks included in the maximal feasible set. Following this way of reasoning we will analyze the problem $P3\,|\,\text{fix}_j,\text{pmtn},\text{res}121\,|\,C_{\max}$. In this case $m_1 = 2$ and $q = 3$. Thus, $m_1 = q - 1$ and the schedule feasible with respect to the processors is also feasible with respect to the processors if one of the types of the uni-processor tasks requiring an additional resource can be fully scheduled outside the feasible set $\{T_R^1, T_R^2, T_R^3\}$. Otherwise the schedule must be longer. The following theorem summarizes our observations for the problem.

Theorem 4. *The minimal length of a schedule for $P3\,|\,\text{fix}_j,\text{pmtn},\text{res}121\,|\,C_{\max}$ is equal to*

$$C_{\max} = \sum_{\substack{T_i^S \in S \\ S \in D}} t_i^S + \max\left\{0,\ \max_{1 \leq i \leq 3}\{\Delta_i\},\ \min_{\substack{i,j,k=\{1,2,3\} \\ i \neq j \neq k}} \left\{\max\left(\Delta_i^R + \Delta_j^R,\ \Delta_k^R\right)\right\}\right\},$$

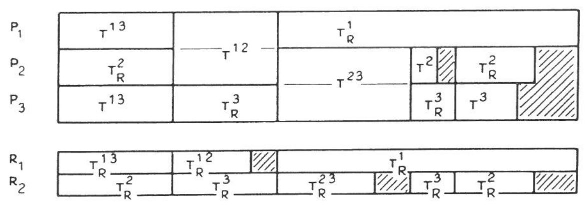


Fig. 6. A schedule in which two sets of tasks are executed sequentially.

where D is a clique defined in Theorem 1, $\Delta_1 = t^1 - t^{23}$, $\Delta_2 = t^2 - t^{13}$, $\Delta_3 = t^3 - t^{12}$ *and* $\Delta_1^R = t_R^1 - t^{23}$, $\Delta_2^R = t_R^2 - t^{23}$, $\Delta_3^R = t_R^3 - t^{12}$.

Proof. Let us note that for the case of tasks requiring no additional resources, the schedule length is equal to that given in Theorem 1.

The set $F = \{T_R^1, T_R^2, T_R^3\}$ can be scheduled feasibly with respect to the processor set but it is the only one which is not feasible with respect to the resources. If at least one $\Delta_i^R \le 0$ $(i = 1 \dots 3)$, then tasks requiring resources can be executed fully in the schedule feasible with respect to processors. Since $\Delta_i^R \le \Delta_i$ (because $T_R^i \subseteq T^i$), then in this case the schedule length is the same as in Theorem 1.

Finally, if tasks requiring resources cannot be fully executed with duo-processor tasks, it is necessary to execute them in such a way that no more than two units of the resource are simultaneously required. This means that two sets of tasks requiring the resource have to be executed sequentially. There are three such sequences. Such a group of tasks can be executed in time $\max(\Delta_i^R + \Delta_j^R, \Delta_k^R)$. This case is presented in Fig. 6. The case for which $\Delta_i^R + \Delta_j^R$ is maximal can also be expressed in terms of graph theory. One can build a competition graph with nodes $V = \{T^{13}, T^{12}, T^{23}, \Delta_1^R, \Delta_2^R, \Delta_3^R\}$, where Δ_1^R, Δ_2^R, Δ_3^R represent tasks (or parts of tasks) which cannot be executed together with T^{23}, T^{13}, T^{12}, respectively. Because Δ_1^R, Δ_2^R, Δ_3^R cannot be scheduled all in parallel, then there is a pair Δ_i^R, Δ_j^R connected by an edge. This leads to a conclusion that such a competition graph must have clique with nodes $\{T^{13}, T^{12}, T^{23}, \Delta_i^R, \Delta_j^R\}$. □

Using the above result, one can build in O(n) time an optimal schedule in which uni-processor tasks are scheduled in parallel with appropriate duo-processor tasks and uni-processor tasks requiring an additional resource are given priority over tasks not requiring it.

Other subcases of the problem can be solved optimally following the lines the general case of the problem without additional resources was solved.

5. Conclusions

In this paper, low order polynomial algorithms for deterministic scheduling of preemptable independent tasks requiring for processing a set of processors simultaneously (from among a set of three or four), have been presented. The general version of the problem is known to be NP-hard but is solvable in polynomial time if the number of processors is limited. A simple algorithm for three processors and tasks requiring one unit of the additional resource is also given.

Further research in this area may include an analysis of the problem for other performance measures such as lateness (L_{max}) or flow time (F), or the case in which a task may be processed (alternatively) by more than one set of processors simultaneously.

References

[1] 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.

[2] J. Błażewicz, 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. Błażewicz, M. Drabowski and J. Węglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Trans. Comput.* **35** (1986) 389–393.

[4] J. Błażewicz, M. Drozdowski, G. Schmidt and D. de Werra, Scheduling independent two processor tasks on a uniform duo-processor system, *Discrete Appl. Math.* **28** (1990) 11–20.

[5] J. Błażewicz, K. Ecker, G. Schmidt and J. Węglarz, *Scheduling in Computer and Manufacturing Systems* (Springer, Berlin, 1993).

[6] J. Błażewicz, J.K. Lenstra and A.H.G. Rinnoy Kan, Scheduling subject to resource constraints: classification and complexity, *Discrete Appl. Math.* **5** (1983) 11–24.

[7] M. Dal Cin and E. Dilger, On the diagnosibility of self-testing multiprocessor systems, *Microprocessing Microprogramming* **7** (1981) 177–184.

[8] J. Du and J.Y-T. Leung, Complexity of scheduling parallel task systems, *SIAM J. Discrete Math.* **2** (1989) 473–487.

[9] M.R. Garey and D.S. Johnson, Computers and Intractability, *A Guide to the Theory of NP-completeness* (Freeman, San Francisco, CA, 1979).

[10] E. Gehringer, D. Siewiorek and Z. Segall, Parallel Processing, The Cm* Experience (Digital Press, 1987).

[11] R.I. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnoy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.* **5** (1979) 287–326.

[12] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thather, eds. *Complexity of Computer Computation* (Plenum, New York, 1972).

[13] H. Krawczyk and M. Kubale, An approximation algorithm for scheduling diagnostic tests in multicomputer systems, *IEEE Trans. Comput.* **34** (1985) 869–872.

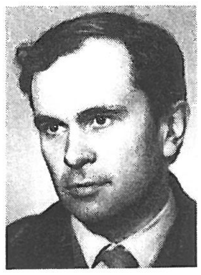
[14] M. Kubale, The Complexity of Scheduling Independent Two-Processor Tasks on Dedicated Processors, *Inform. Process. Lett.* **24** (1987) 141–147.

[15] M. Kubale, Preemptive scheduling of two-processor tasks on dedicated processors (in Polish), *Zeszyty Naukowe Politechniki Śląskiej, Seria: Automatyka* **100** (1082) (1990) 145–153.

[16] D. de Werra, Preemptive scheduling, linear programming and network flows, *SIAM J. Algebraic Discrete Meth.* **5** (1984) 11–20.

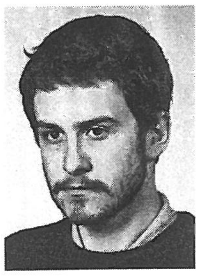
[17] B. Veltman, B.J. Lageweg and J.K. Lenstra, Multiprocessor scheduling with communication delays, Report BS-R9018, June 1990, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

Lucio Bianco is full professor of Operations Research at the University of Rome "Tor Vergata". He is also the director of the Istituto di Analisi dei Sistemi ed Informatica and of the Research Project on Transportation of the Italian National Research Council. He has extensively worked on mathematical modelling and combinatorial optimization area (project scheduling, vehicle scheduling and routing problems, etc.) and applications of operations research methodologies to transportation and related fields. He is the author of about 70 scientific papers.



Jacek Błażewicz, born in (M.Sc. in control engineering 1974, Ph.D. and Dr. habil. in computer science in 1977 and 1980, respectively), is a professor of computer science at the Faculty of Electrical Engineering of the Technical University of Poznań. Presently he is a deputy director of the Institute of Computing Science. He has been abroad several times as a visiting professor. His research interests include algorithms design, and complexity analysis of algorithms especially in scheduling theory and in data transmission systems. He has published widely in the above mentioned fields in many outstanding journals. He is also the author of four and co-author of three monographs and a co-author of three textbooks for graduate students. Dr. Błażewicz is a member of several scientific societies, among others: the Polish Computer Science Society, the American Mathematical Society, the Society for Industrial and Applied Mathematics and the Mathematical Programming Society.

Paolo Dell'Olmo received his degree in electronic engineering from the University of Rome "La Sapienza", Italy. He is a fellow of Istituto di Analisi dei Sistemi ed Informatica – C.N.R., Rome, Italy. His primary research areas include combinatorial optimization and control theory. In particular, he has recently published papers on project scheduling, multiprocessor scheduling and scheduling in CIM systems. Dr. Dell'Olmo is also a member of the Operations Research Society of America.



Maciej Drozdowski was born in Poznań, Poland on . He received the M.Sc. degree in control engineering and a Ph.D. in computer science from the Technical University of Poznań, Poznań, Poland, in 1987 and 1992 respectively. Since 1987 he has been with the Technical University of Poznań, where he is presently the assistant professor. His research interests include design and complexity analysis of algorithms especially in the scheduling theory. He also took part in designing some software packages for cutting of irregular shapes. Dr. Drozdowski is a member of Polish Informatics Society.