Exploiting incomplete information to manage multiprocessor tasks with variable arrival rates

Paolo Dell'Olmo[1] Antonio Iovanella[2] Guglielmo Lulli[3]

Benedetto Scoppola[4]

[1] Dipartimento di Statistica, Probabilità e Statistiche Applicate, University of Roma "La Sapienza"

Piazzale Aldo Moro, 5 - 00185 Rome, Italy

paolo.dellolmo@uniroma1.it

[2] Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata"

Via del Politecnico, 1 - 00133 Rome, Italy.

iovanella@disp.uniroma2.it

[3] Dipartimento di Matematica Pura e Applicata, University of Padova

Via Belzoni, 7 - 35131 Padova, Italy

glulli@math.unipd.it

[4] Dipartimento di Matematica, University of Rome "Tor Vergata"

Via della Ricerca Scientifica - 00133 Rome, Italy

scoppola@mat.uniroma2.it

Abstract

In this paper a semion-line algorithm for scheduling multiprocessor tasks with partial information is proposed. We consider the case in which it is

possible to exploit probabilistic information and use this information to obtain better solutions in comparison with standard non clairvoyant on-line algorithms. A wide computational analysis shows the effectiveness of our algorithm. Moreover, we also consider a test framework with a continuous generation of tasks in order to study the behavior of the proposed approach in real applications, which confirms the efficiency of our approach.

Keywords: Multiprocessor Task Scheduling, Semion-line algorithm, Computational Analysis.

1 Introduction

Several applications in the field of telecommunications, Internet and computer science are calling for new paradigms of decision-making, mathematical models which formalize properly the problems and appropriate algorithms to quickly compute the effective allocation of resources to tasks over time. In particular, a new demand is arising for the development of representative and realistic models of real world scheduling systems. For instance, many real communication services are characterized by a demand which is generally subject to fluctuations. For these problems even though some specific trends might be detected over the long run, fluctuations of arrivals or requests for services generate an overload of the system in the short run. To insure the quality of services, it is fundamental to recover quickly from these fluctuations, keeping the load variations of the system as smooth as possible. For this class of problems, dynamic models which are able to capture this specific aspect can be one key of success.

Note that information on the demand and its trends over time are now available for many real problems. For instance, referring to the telecommunication and Internet applications, dozens of network management tools are available to monitor telecommunications systems and these might be used to detect traffic volume fluctuations. They are designed to provide automated support for some or all network management functions. They enable the network manager to monitor important devices and typically report configuration information, traffic volumes and error conditions for each device. All of this information can be analyzed to diagnose patterns [14]. Combining data bases and data mining techniques with these tools allows us to acquire information on patterns of demand and, in some special cases, to find a good approximation of the complex stochastic processes which provide the information.

To address the above problems, in this paper we study a semion-line multiprocessor task scheduling (MTS) problem. The MTS problem deals with the scheduling of a set $\mathcal{T} = \{1, \ldots, n\}$ of multiprocessor tasks on a set M of m dedicated, independent parallel processors necessary for their execution. In particular, we consider instances where tasks are generated by known stochastic processes (we have information on the process of arriving tasks). To each task is associated a release time (ready time, r_j), i.e., the task's arrival time in the system, and a unitary task processing time ($p_j = 1$, $\forall j \in \mathcal{T}$). We minimize the makespan $C_{\max} = \max_j C_j$, where C_j denotes the completion time of task j under the assumption that preemption is not allowed. Each task requires a subset of processors fix_j drawn from M for its execution. Each processor can work on at most one task at a time, and each task $j \in \mathcal{T}$ must be

simultaneously processed by all the processors in fix_j. We can denote the described problem with $P|semion\text{-}line, fix_j, p_j = 1, r_j|C_{\max}$. To the best of our knowledge, this version of the semion-line multiprocessor task scheduling problem is new in the literature.

To give an example of how the proposed model represents the class of problems here described, we consider the Wavelength Division Multiplexing (WDM) decision problem. In WDM network technology, there is a coordination problem among nodes of the network that wish to communicate with each other. The main feature of broadcast WDM local area networks is the so-called *one-to-many transmission* or *multicasting* ability [1], as depicted in Figure 1. That is, a transmission by a node of the network on a given channel (wavelength) is received by *all* nodes listening simultaneously to that channel at that point in time [19]. Since the number of channels may be less than the number of nodes and two or more nodes may want to send data packets to the same destination node, coordination among nodes that wish to communicate with each other is required.

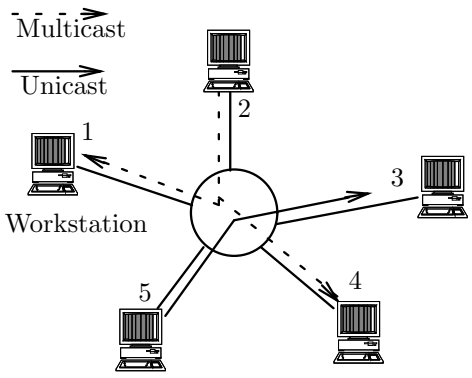

Figure 1: A passive-star-based local optical WDM network

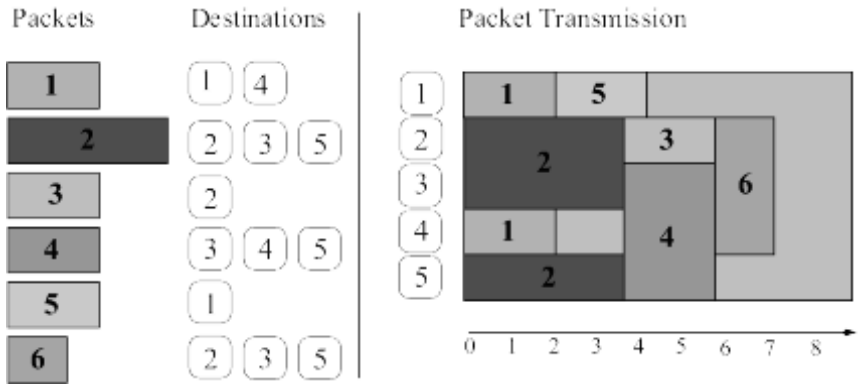

Figure 2: Multi-destination data packets and a transmission schedule.

The correspondence between the WDM decision problem and multiprocessor task scheduling is given: *nodes* correspond to *processors*, and *multi-destination* data packets to *dedicated tasks*(for an illustration see Figure 2). The objective function of minimizing the schedule makespan of dedicated tasks corresponds to minimizing the overall *transmission* time, i.e. the time needed to send all data packets out.

Another field where there is a need for efficient algorithms with the above described characteristics is the replication of data in different storage systems in Data Grid Environments [12]. In fact, in such problems, it could be necessary to make several replicas of a dataset available to the Grid, and such data should be synchronized and consistent, i.e., up to date. An efficient replication system should send, on request, a dataset (i.e., tasks) to a number of sites (i.e., processors), listed in a replica catalogue, maintaining synchronization.

The main point of this paper is to introduce new aspects in the multiprocessor task scheduling model in order to capture the main features of the real applications described above, which are inherently dynamic and stochastic. In particular, we

propose a simple algorithm to handle the fluctuations of the demand for services as efficiently as possible. Especially in Internet applications where customers are spread out all over the world and may request services at any time, the aspect of recovering from these fluctuations is really relevant.

In §2, we describe the different paradigms for MTS decision problems with respect to the information available. The state of the art for all the paradigms we introduce is also given. In §3 we present an algorithm, named $\mathcal{SDSATUR}$, to solve instances of the *semion-line* version of the MTS problem. This algorithm has as its core subroutine a modified version of the DSATUR algorithm proposed by Brelaz [6] to solve the coloring problem. The quality of solutions computed by our algorithm is evaluated by comparing it with a lower bound based on a maximal clique [9]. To verify the advantages of exploiting information, we also provide a comparison with an existing algorithm for *on-line* MTS proposed by Caramia et al. [7] in §4. Finally, in § 5, we give some final remarks.

2 MTS models with respect to available information: state of the art

We here highlight the different strategies and decision-making processes for scheduling tasks in view of the information available over time. Any MTS problem can be either a *one-stage* or a *multi-stage* decision problem. In the former case, all the decisions regarding tasks to be scheduled are taken at a single point in time. This is the case of the deterministic off-line version of the MTS problem. In this version

of the problem, the decision maker knows about all the tasks to be scheduled and their features (processing and ready times and the subset of requested processors) in advance.

There is an extensive literature on multiprocessor task problems with the fix_j option, see [11] for a complete survey on results. Hoogeveen *et al.* [16] proved that the version of the problem $P3|fix_j|C_{max}$ is already strongly $\mathcal{NP}$-hard. Moreover, even in the case of tasks with unitary processing times ($P|fix_j, p_j = 1|C_{\max}$) there is no polynomial approximation algorithm with a performance ratio smaller than 4/3 (unless $\mathcal{P} = \mathcal{NP}$). However, if the number of processors m is not part of the problem, i.e., $Pm|fix_j, p_j = 1|C_{\max}$, then the problem is solvable in polynomial time. Later, Amoura *et al.* [2] proposed a polynomial time approximation scheme for $Pm|fix_j|C_{\max}$ and Bampis *et al.* [4] extended this result to a polynomial time approximation scheme for $Pm|fix_j, r_j|C_{\max}$. In this framework, the best approximation algorithm for the $P3|fix_j|C_{\max}$ is the 9/8-approximation algorithm by Chen and Huang [10]. Recently, Fiskin *et al.* [13] extended the negative results presented in [16] proving that $P|fix_j, p_j = 1|C_{\max}$ cannot be approximated within a factor of $m^{\frac{1}{2}-\epsilon}$, for some $\epsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.

In the *multi-stage* decision problem, the allocation of tasks to processors is decided dynamically at different points in time. At each point in time, the decision maker has complete information only on the tasks available in the scheduler and none on those not available yet. Moreover, decisions on scheduled tasks cannot be reconsidered in later stages. This version of the problem is known as on-line MTS. More formally, given an input instance $\mathcal{T}=\{1, 2, \ldots, n\}$ of tasks to be scheduled,

ordered according to a non decreasing order, an algorithm $\mathcal{A}$ is said to be on-line in processing $\mathcal{T}$ if all the following conditions hold:

1) $\mathcal{A}$ processes tasks in the order given by the list $\mathcal{T}$;

2) $\mathcal{A}$ processes each task i without knowledge of any task j, $j \succ i$;

3) $\mathcal{A}$ never re-processes a task which has already been processed.

Obviously, we cannot expect to minimize the makespan scheduling tasks on-line as in the case of the off-line decision problem.

The quality of an on-line or semi-online algorithm $\mathcal{A}$ is measured by its competitive ratio c, defined as the smallest number such that for every list of tasks $\mathcal{I}$, $\mathcal{A}(\mathcal{I}) \leq c \cdot \mathcal{OPT}(\mathcal{I})$, where $\mathcal{A}(\mathcal{I})$ denotes the makespan achieved by algorithm $\mathcal{A}$ for scheduling the tasks in $\mathcal{I}$ and $\mathcal{OPT}(\mathcal{I})$ denotes the corresponding makespan of some optimal schedule.

Few results are known for on-line MTS. Bampis *et al.* [3] presented a $2\sqrt{m}$-competitive algorithm for $P|\,\text{online}, fix_j, p_j = 1|C_{\max}$ and a $(2\sqrt{m}+1)$-competitive algorithm for $P|\,\text{online}, fix_j, p_j = 1, r_j|C_{\max}$ if the maximum number of processors requested by a task is bounded by a given constant, and Caramia *et al.* proposed some algorithms evaluated by computer simulations [7] and some lower bounds [8].

Among the two paradigms of decision-making described above, many others could be specified according to the information available. As mentioned in § 1, in many real practical problems, we have neither complete knowledge of tasks, as in the *off-line* MTS, nor complete ignorance of tasks as in the *on-line* MTS. Often the decision maker has a certain degree of knowledge on tasks. Indeed, current

technologies allow us to record the arriving tasks and store this large amount of data in databases. Using statistical and data mining techniques is then possible to ascertain probability distributions on the tasks' arrival time. Therefore, with respect to the on-line problem, some further information on tasks is available which allows us to design algorithms which perform better. This class of problems, where at least one of the on-line conditions is relaxed, is usually called "*semion-line*".

Few semion-line algorithms have been considered in the literature. For example, in [18] and [20] the algorithm schedules tasks ordered by processing times, and in [15] some knowledge on the processing times is used. In both the mentioned cases, condition 2 is violated. In order to confirm the vagueness of the semion-line concept, a problem which belongs to neither of the above conditions is considered in [17]. Herein, we violate condition 2, introducing the following hypothesis:

Semion-line hypothesis Probability distributions on the arrival time of tasks are known. We also suppose that for certain tasks such probabilities are negligible, i.e., it is unlikely that they will arrive in the scheduler. $F = \{\tau_1, ..., \tau_{|F|}\}$ is the set of tasks which have a significant probability.

Then we define a probability distribution $p_1, ..., p_{|F|}$ and we assume that each of the tasks arriving in the system in the future has probability p_j to be equal to τ_j, independent of other arrivals. The number n_t of tasks arriving in the system at time t is moreover distributed in a memoryless way, i.e. assuming a Poissonian

distribution with parameter λ for each time t:

$$P(n_t = n) = \frac{\lambda^n}{n!} e^{-\lambda} \quad \forall t < R_{max}$$

where R_{max} is a given fixed parameter, equal to the maximum ready time allowed for the task arrival process. Hence the number of tasks in each instance is not fixed. Note that in many practical contexts (mainly in telecommunications problems) the Poisson distribution is a good approximation of the real distribution of arrivals.

3 An algorithm for the semion-line MTS problem

In this section we present an algorithm to solve the semion-line MTS. The core subroutine of this algorithm is a modified version of the DSATUR algorithm proposed by Brelaz [6] to solve the coloring problem (see Table 1 for a brief description). In MTS problems it is common to represent the relationship between tasks and processors by means of an undirected graph in which each node is associated with a task and an edge between two nodes exists if, and only if, tasks associated with those nodes share at least one (equal) processor. More formally, we introduce a graph $G(V_G, E_G)$, V_G being its node set and E_G its edge set, where $(i, j) \in E_G$ if, and only if, $fix_i \cap fix_j \neq \emptyset$, $\forall i, j \in V_G$. The graph G is usually called an *incompatibility graph*. From now on, we will use $\mathcal{T}$ and V_G (or task and node) interchangeably when no confusion arises.

For the *semion-line* MTS, as well as for the *on-line* version, the incompatibility graph becomes available over time, since we do not know which tasks will come

Procedure DSATUR	
1	Arrange vertices by decreasing order of degrees
2	Color a vertex of maximal degree with color 1
3	Choose a vertex with a maximal saturation degree. If there is an equality, choose any vertex of maximal degree in the uncolored subgraph.
4	Color the chosen vertex with the least possible (lowest numbered) color.
5	If all vertices are colored, stop. Otherwise return to 3.

Table 1: The procedure DSATUR

afterwards. Therefore, we define the incompatibility subgraph G_t of G at each time t, corresponding to those tasks $j \in N$ with $r_j \leq t$ and not yet scheduled. This implies that the set $V_{G_t} \subseteq V_G$ of tasks in the system must be updated dynamically, adding tasks as they arrive and deleting those already scheduled at each point in time.

The Brelaz's algorithm has already been used for the on-line MTS presented in [7]. This algorithm, that we will simply call $\mathcal{DSATUR}$ from now on, finds a feasible coloring of G_t at each iteration, i.e., it finds an admissible partition of G_t in stable sets. At each point in time (iteration), $\mathcal{DSATUR}$ schedules all the tasks which belong to the highest cardinality stable set.

3.1 Algorithm $\mathcal{SDSATUR}$

As mentioned in Section 2, we select in the set of the possible choices $\mathcal{C}$, a subset $F \subset \mathcal{C}$ of tasks with an associated distribution probability for each task and we suppose that their arrivals are independently distributed with the above mentioned probability distribution.

Hence, each task j has a probability p_j, with $\sum_{j \in F} p_j = 1$. Once such values are stated, we can define the weight wm_i of a machine i as the sum of the probabilities that such machine will be requested by the tasks in F:

$$wm_i = \sum_{j \in F} p_j \cdot x_j^{(i)} \tag{1}$$

with $x_j^{(i)}$ a vector of length $|F|$, with $x_j^{(i)} = 1$, if $m_i \in fix_j$, $x_j^{(i)} = 0$ otherwise.

We define the weight w_j of each task j as the sum of the weights of the machines $i \in fix_j$ requested by the task j:

$$w_j = \sum_{i \in fix_j} wm_i \tag{2}$$

Example 3.1. *Let us consider a simple on-line instance composed by tasks A, B and C, with processor requirements as* $fix_A = \{m_1, m_2, m_3, m_5\}$, $fix_B = \{m_1, m_2, m_4\}$ *and* $fix_C = \{m_3, m_6\}$. *If F is composed of only the three tasks, i.e.* $F = \{fix_A, fix_B, fix_C\}$ *and they have a probability* $p_A = 0.35$, $p_B = 0.40$ *and* $p_C = 0.25$ *respectively, then* $wm_1 = (p_A + p_B) = 0.75$, $wm_2 = 0.75$, $wm_3 = 0.60$, $wm_4 = 0.40$, $wm_5 = 0.35$, $wm_6 = 0.25$. *With these values it is possible to compute the weights of the tasks as* $w_A = wm_1 + wm_2 + wm_3 + wm_5 = 2.7$, $w_B = 1.9$ *and* $w_C = 0.85$.

Considering this set of values, we could imagine a scheduler that knows F and the distribution of probability for each task. The scheduler computes the r stable sets C_l, with $l = 1, \ldots, r$ at each point in time and evaluates the weights $W(C_l)$ for each of such sets. The set of tasks with the highest value of such weight among all the stable sets is scheduled.

Note that, in order to avoid unnecessarily idle machines, we compare only the color classes with maximum cardinality, so among classes with the same maximum cardinality, it selects the heaviest. The idea underlying this procedure is the fact that a task with a high weight requires machines which will be requested by future tasks with a higher probability, and therefore it is better to schedule it first.

The algorithm we introduce to solve the semion-line MTS, called $\mathcal{SDSATUR}$ (*Semion-line DSATUR*), comes from $\mathcal{DSATUR}$ plus the probabilistic knowledge of the arrival process. Algorithm details are described in Table 2.

Algorithm $\mathcal{SDSATUR}$	
0	Load the weights of the tasks in F;
1	$t = 0$, $S_t = \emptyset$;
2	If $S_t = V(G)$ then stop, i.e., all the tasks have been scheduled;
3	Compute G_t where $V(G_t) = j \in V(G) \vert r_j \leq t \setminus S_t$ is its node set;
4	Color G_t by means of the procedure DSATUR, obtaining p color classes $C_1, \ldots, C_p$ and compute the weights $W(C_l)$ for $l = 1, ..., p$ (weighted stable sets);
5	Define C_t as the maximum weighted stable set C_l, with $l = 1, \ldots, p$. Schedule C_t at time t;
6	$t = t + 1$, $S_t = S_{t-1} \cup N_{t-1}$, go to 2.

Table 2: The Algorithm $\mathcal{SDSATUR}$

The main differences between the two algorithms $\mathcal{DSATUR}$ and $\mathcal{SDSATUR}$ are in *Step* 4 and 5 in which instead of computing and selecting the maximal cardinality color class, we compute and select the highest weighted color class, where weights are loaded at the first step.

4 Computational Experience

In this section, the computational results of the algorithm on instances of different sizes of MTS problem are given.

We generated different sets of instances by varying the following problem input parameters: $n \in \{50, 100, 200\}$, $m \in \{20, 30\}$, $|F| = 40$, $R_{max} \in \{6, 12, 20, 25\}$ and $k \in \{6, 8, 12, 16\}$, where R_{max} and k are respectively the maximum ready time for a task, and the maximum number of processors that can be associated with a task. In particular, we implemented the following instance generator, which works as described in Table 3.

Algorithm *GEN*	
1	Set the four parameters n, m, R_{max} and k;
2	Create the tasks in F and generate probabilities and weights;
3	For $i = 1$ to 100 do
	- extract n tasks, randomly chosen in F, according to the probability distribution p_j;
	- assign to each task j a ready time r_j, with $0 \leq r_j \leq R_{max}$, according to a Poisson distribution;
	- build the incompatibility graph G_i.

Table 3: The instance generator

For every choice of n, m, R_{max} and k, we generated one hundred graphs (see 4.1 for a discussion on this choice) using only tasks extracted from F and we assigned to each of them a ready time with independently distributed values, which means that they are generated according to a classical Poisson distribution. The parameter λ (the average rate of arrivals of the Poisson distribution) introduced in Section 2 is clearly related to R_{max} by $\lambda = \frac{n}{R_{max}}$.

When all the data are collected, the program generates the one hundred graphs, stored in the well known DIMACS format files. As we said at the beginning of this Section, we considered several test problems, and for a fixed data set, the algorithms were all tested on the same instances. In particular, for its computation $\mathcal{DSATUR}$ considers only the incompatibility graph, instead, $\mathcal{SDSATUR}$, considers also the weights of each task.

The algorithms were implemented in the C language, compiled with the GNU compiler GCC 3.4.1 with the -o3 option switched on and tested on a PC Pentium 2 GHz with Linux OS.

4.1 Computational results

In this section we present the results of the $\mathcal{SDSATUR}$ algorithm. We compare our results with those provided by the $\mathcal{DSATUR}$ algorithm and the first fit algorithm ($\mathcal{FFS}$)) presented in Bampis *et al.* in [3], which is a greedy on-line algorithm and schedules tasks following a FIFO rule. The authors proved that the $\mathcal{FFS}$ algorithm has a guaranteed performance when $|fix_j|$ is bounded by some constant k, more precisely is k-competitive for $P\,|\,\text{online, fix}_j,\ p_j = 1|C_{\max}$. Moreover, for each instance, we compute a lower bound LB based on the algorithm of Carraghan and Pardalos [9] with a stopping criterion based on halting the algorithm at 10,000 iterations. Then, LB is taken as the largest maximal clique computed within these iterations. The lower bound is considered in order to make the comparison more meaningful.

Selected experiments on the values achieved by the four algorithms are reported

from Table 4 to Table 7, where in the first three columns the makespan achieved by the algorithms $\mathcal{FFS}$, $\mathcal{DSATUR}$ ($C^{\mathcal{D}}_{\max}$) and $\mathcal{SDSATUR}$ ($C^{\mathcal{SD}}_{\max}$) is reported. LB is the lower bound and in the next two columns we show the absolute differences $C^{\mathcal{D}}_{\max} - C^{\mathcal{SD}}_{\max}$ and, finally, the relative improvement achieved with respect to the lower bound.

As general remarks about the numerical results we obtained, we want to outline that the makespan achieved by $\mathcal{SDSATUR}$ is slightly better than the makespan of $\mathcal{DSATUR}$, but the absolute difference is in general small. However, also the absolute differences between $\mathcal{DSATUR}$ and the two bounds ($\mathcal{FFS}$ and LB) are already quite small. Therefore, we listed the relative improvement due to $\mathcal{SDSATUR}$. This improvement is in some cases really dramatic. Such improvement is bigger when the number of maximum value k of requested machines in each task is bigger. This has a quite evident intuitive counterpart: the more the single task tends to require a large number of machines, the more effective a strategy that takes into account the probability of the future arrivals. In many cases the performance of $\mathcal{SDSATUR}$ are really near to the lower bound when k is large.

For the sake of completeness, we report in Table 8 the CPU times achieved for each run performed. As was predictable, harder instances take more CPU time to complete the whole set of tasks. Nevertheless, each decision stage is always performed instantaneously, so, such long times refer only to the length of the experiments.

Since we are evaluating the performance of the algorithm on random generated instances, one question remains open: do the completion times $C^{\mathcal{D}}_{\max}$ and $C^{\mathcal{SD}}_{\max}$ have

R_{max}	k	$\mathcal{FFS}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	LB	$C^{\mathcal{D}}_{\max}$- $C^{\mathcal{SD}}_{\max}$	$\frac{C^{\mathcal{SD}}_{\max}-LB}{C^{\mathcal{D}}_{\max}-LB}$
6	6	38.66	37.26	37.14	36.68	0.12	0.21
12	6	37.63	36.01	35.66	34.38	0.35	0.21
20	6	40.07	38.52	38.15	36.85	0.37	0.22
25	6	36.90	35.78	35.22	32.77	0.56	0.19
6	8	46.51	44.52	44.29	43.63	0.23	0.26
12	8	44.17	43.17	42.75	42.08	0.42	0.39
20	8	47.17	45.65	44.66	43.17	0.99	0.40
25	8	44.68	43.85	43.13	41.63	0.72	0.32
6	12	58.63	57.42	57.26	56.97	0.16	0.36
12	12	58.60	57.45	57.10	56.83	0.35	0.56
20	12	67.04	66.67	66.11	65.58	0.56	0.51
25	12	72.16	71.64	71.09	70.68	0.55	0.57
6	16	86.64	86.57	86.43	86.38	0.14	0.74
12	16	71.17	70.22	69.64	69.51	0.58	0.82
20	16	76.34	75.96	75.15	75.06	0.81	0.90
25	16	76.58	76.54	75.68	75.65	0.86	0.97

Table 4: Results for makespan for $n = 100$, $m = 20$ (average value of 100 instances).

the same probability distribution? If this is the case, then the observed discrepancies between the two completion times is the effect of a statistical fluctuation. We show how the choice of the number of tasks we made (one hundred, see above) is such that the probability of observing our results assuming the same probability distribution for $C^{\mathcal{D}}_{\max}$ and $C^{\mathcal{SD}}_{\max}$ is negligible for all the values of the parameters. Hence, we give evidence that the two algorithms have different completion times.

To do this, observe that if $C^{\mathcal{D}}_{\max}$ and $C^{\mathcal{SD}}_{\max}$ have the same probability distribution we have to assume that there exists a probability p defined as

$$P(C^{\mathcal{D}}_{\max} > C^{\mathcal{SD}}_{\max}) = P(C^{\mathcal{SD}}_{\max} > C^{\mathcal{D}}_{\max}) = p$$

R_{max}	k	$\mathcal{FFS}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	LB	$C^{\mathcal{D}}_{\max}$- $C^{\mathcal{SD}}_{\max}$	$\frac{C^{\mathcal{SD}}_{\max}-LB}{C^{\mathcal{D}}_{\max}-LB}$
6	6	29.43	28.04	27.94	27.22	0.10	0.12
12	6	32.55	31.21	30.66	28.73	0.55	0.22
20	6	33.59	32.70	32.40	31.06	0.30	0.18
25	6	30.60	29.56	29.14	25.84	0.42	0.11
6	8	34.79	33.19	33.01	32.06	0.18	0.16
12	8	33.39	32.21	32.07	31.10	0.14	0.13
20	8	43.40	42.09	41.57	40.20	0.52	0.28
25	8	45.10	44.43	43.98	42.56	0.45	0.24
6	12	54.16	52.56	52.36	51.96	0.20	0.33
12	12	58.00	57.30	56.99	56.76	0.31	0.57
20	12	56.81	56.00	55.44	54.64	0.56	0.41
25	12	49.44	48.53	47.82	46.57	0.71	0.36
6	16	55.87	54.69	54.44	54.05	0.25	0.39
12	16	68.84	68.28	67.95	67.49	0.33	0.42
20	16	77.96	77.81	77.03	76.90	0.78	0.86
25	16	68.14	67.67	67.09	66.87	0.58	0.73

Table 5: Results for makespan for $n = 100$, $m = 30$ (average value on 100 instances).

and therefore

$$P(C^{\mathcal{SD}}_{\max} = C^{\mathcal{D}}_{\max}) = 1 - 2p$$

Assuming that the different tasks are independent, we can easily write the probability of seeing in a series of 100 experiments, that n_0 times the two algorithms have the same completion time, n_D times $\mathcal{SDSATUR}$ outperforms $\mathcal{DSATUR}$, i.e. $C^{\mathcal{D}}_{\max} > C^{\mathcal{SD}}_{\max}$, and n_S times $\mathcal{DSATUR}$ outperforms $\mathcal{SDSATUR}$, i.e. $C^{\mathcal{D}}_{\max} < C^{\mathcal{SD}}_{\max}$. This probability has the following form

$$P(n_0, n_D, n_S) = p^{n_D+n_S}(1-2p)^{n_0}\frac{100!}{n_0!n_D!n_S!}$$

R_{max}	k	$\mathcal{FFS}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	LB	$C^{\mathcal{D}}_{\max}$- $C^{\mathcal{SD}}_{\max}$	$\frac{C^{\mathcal{SD}}_{\max}-LB}{C^{\mathcal{D}}_{\max}-LB}$
6	6	71.44	69.09	68.72	67.71	0.37	0.27
12	6	84.12	81.14	80.92	79.84	0.22	0.17
20	6	71.03	69.04	68.56	67.55	0.48	0.32
25	6	86.94	85.37	84.74	83.82	0.63	0.41
6	8	99.08	96.64	96.52	96.14	0.12	0.24
12	8	88.25	85.95	85.52	85.05	0.43	0.48
20	8	75.33	72.84	72.47	71.39	0.37	0.26
25	8	98.96	95.60	95.20	94.18	0.40	0.28
6	12	126.07	125.01	124.75	124.69	0.26	0.81
12	12	109.01	107.51	107.22	107.12	0.29	0.74
20	12	123.08	122.03	121.69	121.60	0.34	0.79
25	12	135.19	134.15	133.55	133.45	0.60	0.86
6	16	139.03	138.20	137.96	137.94	0.24	0.92
12	16	174.44	174.39	174.25	174.25	0.14	1.00
20	16	167.64	167.28	166.92	166.87	0.36	0.88
25	16	158.40	157.44	156.87	156.80	0.57	0.89

Table 6: Results for makespan for $n = 200$, $m = 20$ (average value on 100 instances).

As can be seen by our results, the value of n_S is always zero, while $n_D > \bar{n} = 48$. We will use this value in the following discussion since the probability of obtaining this kind of result manifestly decreases with n_D. Hence, the probability that our results are the consequence of a statical fluctuation, can be bounded by

$$P(100 - \bar{n}, \bar{n}, 0) = p^{\bar{n}}(1 - 2p)^{100-\bar{n}} \frac{100!}{(100 - \bar{n})!\bar{n}!}$$

It can immediately be shown that this probability has its maximum in p for $p = \bar{p} = \bar{n}/200$. Observing now that, by Newton binomial formula, for all p, n_D, n_0 we have

$$p^{n_D}(1 - p)^{n_0} \frac{100!}{n_0!n_D!} \leq 1$$

R_{max}	k	$\mathcal{FFS}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	LB	$C^{\mathcal{D}}_{\max}$- $C^{\mathcal{SD}}_{\max}$	$\frac{C^{\mathcal{SD}}_{\max}-LB}{C^{\mathcal{D}}_{\max}-LB}$
6	6	54.63	52.42	52.06	50.66	0.36	0.20
12	6	53.88	52.12	51.65	50.01	0.47	0.22
20	6	51.51	49.80	49.23	46.70	0.57	0.18
25	6	48.59	47.94	47.53	46.05	0.41	0.22
6	8	75.18	71.94	71.74	70.45	0.20	0.13
12	8	72.46	69.12	68.41	65.40	0.71	0.19
20	8	76.91	73.94	73.51	71.21	0.43	0.16
25	8	70.19	68.34	67.92	66.83	0.42	0.28
6	12	101.14	99.84	99.67	99.47	0.17	0.46
12	12	127.08	126.60	126.48	126.45	0.12	0.80
20	12	143.39	143.16	142.93	142.83	0.23	0.70
25	12	105.11	104.01	103.74	103.32	0.27	0.39
6	16	140.04	139.33	139.08	138.93	0.25	0.62
12	16	118.21	116.91	116.68	116.60	0.23	0.74
20	16	136.86	136.16	135.97	135.92	0.19	0.79
25	16	153.94	153.03	152.42	152.38	0.61	0.94

Table 7: Results for makespan for $n = 200$, $m = 30$ (average value on 100 instances).

we can write

$$P(100 - \bar{n}, \bar{n}, 0) \leq \left[\frac{1 - 2\bar{p}}{1 - \bar{p}}\right]^{100-\bar{n}}$$

and this probability is absolutely negligible ($< 1.5 \cdot 10^{-9}$) for the observed value of $\bar{n}$.

Using this result, we could easily compute the expected values of the makespan $C^{\mathcal{SD}}_{\max}$ for a certain test problem, say $E[C^{\mathcal{SD}}_{\max}]$, as the difference of the $E[C^{\mathcal{D}}_{\max}]$ and the value of the difference $C^{\mathcal{D}}_{\max} - C^{\mathcal{SD}}_{\max}$ obtained experimentally. For instance, if we consider $n = 100$, $m = 30$, $R_{max} = 25$ and $k = 12$ in Table 5, we obtain the difference $C^{\mathcal{D}}_{\max} - C^{\mathcal{SD}}_{\max} = 1.38$ and, finally, that $E[C^{\mathcal{SD}}_{\max}] = E[C^{\mathcal{D}}_{\max} - 0.71]$

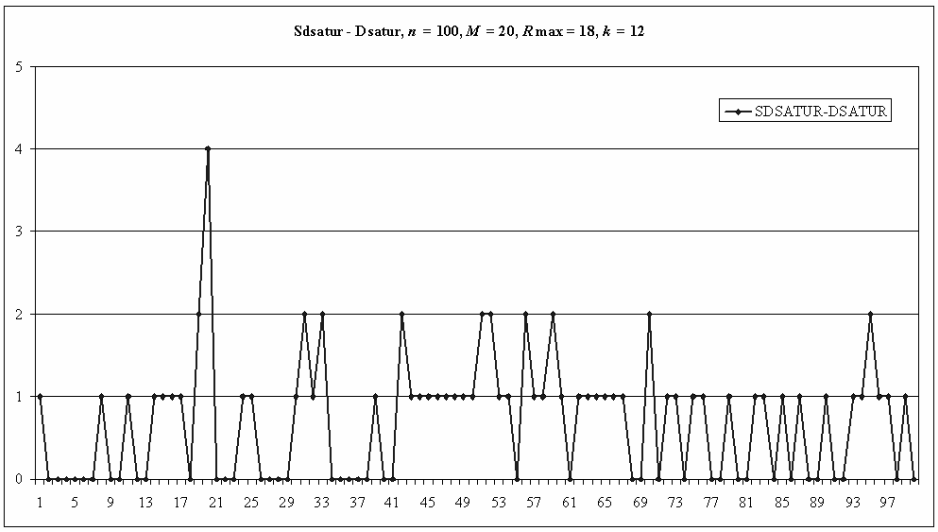

Figure 3: Difference between $\mathcal{DSATUR}$ and $\mathcal{SDSATUR}$ values along the 100 instances, for $n = 100$, $M = 20$, $R_{max} = 20$ and $k = 12$.

4.2 Further results with a continuous generation of tasks

We here consider instances where tasks are generated in a continuous fashion with a wider time horizon.

In order to test this situation, we implemented a simulation framework in which a continuous process of task generation is created and the three algorithms $\mathcal{SDSATUR}$, $\mathcal{DSATUR}$ and $\mathcal{FFS}$ are compared on the same generated tasks. Throughput, average lengths of the queue and average waiting times in the queue are considered as performance indicators. Moreover, it has the capability of changing the value of λ during the simulation, i.e., to generate the fluctuation of the arrival rate of tasks.

On the following figures we show the better performances of $\mathcal{SDSATUR}$ in terms of the queue length and mean waiting time of tasks before being processed. The advantage of the $\mathcal{SDSATUR}$ algorithm in terms of queue length is evident.

		$n = 100$				$n = 200$			
		$m = 20$		$m = 30$		$m = 20$		$m = 30$	
R_{max}	k	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$	$C^{\mathcal{D}}_{\max}$	$C^{\mathcal{SD}}_{\max}$
6	6	0.055	0.075	0.031	0.053	1.305	1.425	0.503	0.619
12	6	0.042	0.064	0.030	0.051	1.578	1.721	0.272	0.400
20	6	0.029	0.049	0.017	0.041	0.636	0.819	0.188	0.305
25	6	0.017	0.037	0.011	0.033	1.030	1.163	0.093	0.210
6	8	0.131	0.173	0.055	0.077	3.292	3.340	1.363	1.492
12	8	0.065	0.089	0.026	0.047	1.655	1.818	1.349	1.475
20	8	0.058	0.083	0.046	0.075	0.728	0.887	0.845	0.990
25	8	0.001	0.012	0.039	0.070	1.714	1.851	0.335	0.498
6	12	0.242	0.271	0.235	0.255	5.717	6.098	3.070	3.331
12	12	0.164	0.193	0.184	0.220	3.182	3.674	3.874	4.188
20	12	0.177	0.217	0.122	0.143	3.566	3.853	5.012	4.882
25	12	0.172	0.248	0.060	0.088	3.949	4.761	1.781	1.834
6	16	0.599	0.824	0.246	0.284	7.012	8.653	6.989	7.097
12	16	0.286	0.422	0.339	0.396	10.160	14.858	3.916	4.156
20	16	0.246	0.402	0.319	0.485	7.701	12.005	4.569	5.072
25	16	0.209	0.321	0.168	0.228	5.913	8.767	5.431	8.053

Table 8: CPU time (average value on 100 instances).

Differences in queue length between the two models increase steadily over time as it is possible to see in Figure 4. A similar trend is shown by the waiting times, see Figure 5. The same statistics are reported in for the case of the variable arrival rate. In particular, we changed the parameter λ (from 4 to 8) at two different time instants, $t = 300$ and $t = 600$ for 25 time periods. Obviously, the number of tasks in queue increases significantly during this arrival time peek period (see Figure 6). Moreover, it is possible to infer from the figure that $\mathcal{SDSATUR}$ is able to reduce the negative effect of the fluctuation rapidly. With regards to the mean waiting time of tasks (see Figure 7), there is a reduction during the peek periods because of the increased number of tasks in queue. However, similar to the queue length statistics,

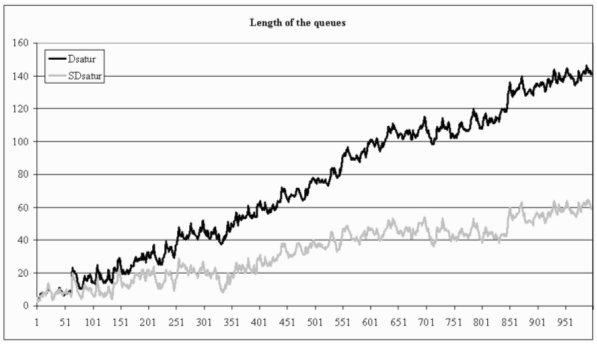

Figure 4: Length of the queues for $M = 100$, $k = 16$, $|F| = 50$, $\lambda = 4$

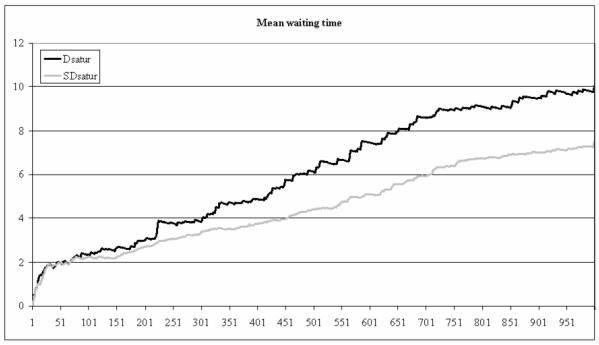

Figure 5: Waiting time of the queues for $M = 100$, $k = 16$, $|F| = 50$, $\lambda = 4$

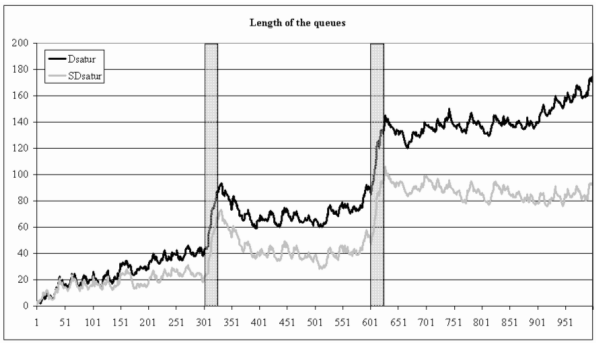

Figure 6: Length of the queues for $M = 100$, $k = 16$, $|F| = 50$, and variable lambda

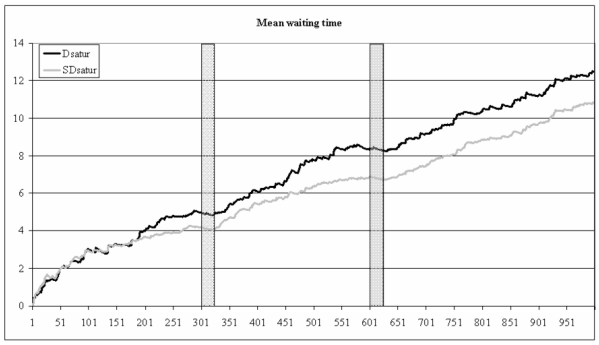

Figure 7: Waiting time of the queues for $M = 100$, $k = 16$, $|F| = 50$, and variable lambda

with $\mathcal{SDSATUR}$, tasks wait for a shorter time, on average.

For the sake of completeness, note that the statistics for $\mathcal{FFS}$ have not been reported because they are too poor with respect to the corresponding statistics of the other two algorithms.

5 Conclusions

In this paper, we show the advantage of exploiting available information in decision problems. In particular, we give an example with the semion-line multiprocessor

task problem (MTS) for which we provide a new paradigm for decision-making. We implement an algorithm ($\mathcal{SDSATUR}$) which is a modified version of the DSATUR algorithm proposed by Brelaz [6] to solve the semion-line MTS problem. We showed the advantage of using the additional information that is available, comparing the results with those provided by the on-line algorithm, which by definition is an algorithm which does not exploit all the available information.

$\mathcal{SDSATUR}$ provides better performance in terms of makespan, queue length and mean waiting time. Especially the last two statistics, in real applications, are considered as measures of efficiency. It is important to note that the makespan reduction gained with the $\mathcal{SDSATUR}$ algorithm is even more significant if measured in terms of the percentage deviation of the solution from the lower bound computed by the Carraghan and Pardalos algorithm [9].

The quality of the results shows the possible benefits that can be acquired formalizing new models and paradigms for decisions which exploit all the available information.

References

[1] M. Ammar, G. Polyzos, S. Tripathi (Eds). *Special issue on network support for multipoint communication*, IEEE Journal Selected Areas in Communications, 15 (1997).

[2] A. K. Amoura, E. Bampis, C. Kenyon, Y. Manoussakis, *Scheduling independent multiprocessor tasks*, in Proceedings of 5^{th} European Symposium on Algorithms, LNCS 1284 (1997), 1-12.

[3] E. Bampis, M. Caramia, J. Fiala, A. V. Fishkin, A. Iovanella, *Scheduling of indipendent dedicated multiprocessor tasks*, in Proceeding of 13^{th} Annual International Symposium on Algorithms and Computation, LNCS, 2518, (2002), 391-402.

[4] E. Bampis, A. Kononov, *On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements*, in Proceedings of 15^{th} International Parallel and Distribute Processing Symposium, San Francisco (2001).

[5] P. Brucker, A. Krämer, *Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems*, European Journal of Operations Research 90 (1996), 214-226.

[6] D. Brelaz, *New methods to color the vertices of a graph*, Communications of the ACM, 22, (1979), 251-256.

[7] M. Caramia, P. Dell'Olmo, A. Iovanella, *On-Line Algorithms for Multiprocessor Task Scheduling with Ready Times*, Foundations of Computing and Decision Sciences, 26 (3), (2001), 197-214.

[8] M. Caramia, P. Dell'Olmo, A. Iovanella, *Lower Bound Algorithms for Multiprocessor Task Scheduling with Ready Times*, submitted.

[9] R. Carraghan and P. M. Pardalos *An exact algorithm for the maximum clique problem*, Operations Research Letters 9 (1990), 375-382

[10] J. Chen, J. Huang. Semi-normal scheduling: Improvement on Goemans' Algorithm. In *Proceedings 12th International Symposium on Algorithms and Computation*, LNCS 2223 (2001), 48-60.

[11] M. Drozdowski, *Scheduling multiprocessor task. An overview*, European Journal of Operations Research 94 (1996), 215-230.

[12] D. Düllmann, W. Hoschek, J. Jean-Martinez, A. Samar, B. Segal, H. Stockinger, and K. Stockinger. *Models for Replica Synchronisation and Consistency in a Data Grid*, in 10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10), San Francisco, California, August 2001.

[13] A. Fishkin, K. Jansen, L. Porkolab, *On minimizing average weighted completion time of multiprocessor tasks with release dates*, in Proceeding of 28^{th} International Colloquium on Automata, Languages and Programming, LNCS 2076 (2001), 875-886.

[14] J. Fitzgerald, A. Dennis, *Business Data Communications and Networking*, J. Wiley & S., New York, (1999).

[15] Y. He and G. Zhang, *Semi on-line scheduling on two identical machines*, Computing, 62 (1999), 179-187.

[16] J. A. Hoogeveen, S. L. van de Velde and B. Veltman *Complexity of scheduling multiprocessor tasks with prespecified allocations*, Discrete Applied Mathematics 55 (1994), 259-272.

[17] H. Kellerer, V. Kotov, M. G. Speranza and Z. Tuza, *Semi on-line algorithms for the partition problem*, Operations Research Letters, 21 (1997), 235-242.

[18] W. P. Liu, J. B. Sidney, A. van Vliet, *Ordinal algorithms for parallel machine scheduling*, Operations Research Letters, 18 (1996), 223-232.

[19] K. M. Sivalingam, S. Subramniam (Editor), *Optical WDM networks: principles and practice*, Kluwer Academic Publishers, 2000.

[20] G. Zhang, *A note on on-line scheduling with partial information*, Computers & Mathematics with Applications, 44 (2002), 539-543.