View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

Insertion of a Random Bitask in a Schedule: a Real-Time Approach

Cyril Duron, Jean-Marie Proth, Yorai Wardi

▶ **To cite this version:**

Cyril Duron, Jean-Marie Proth, Yorai Wardi. Insertion of a Random Bitask in a Schedule: a Real-Time Approach. [Research Report] RR-4337, INRIA. 2001, pp.20. inria-00072250

HAL Id: inria-00072250

https://hal.inria.fr/inria-00072250

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Insertion of a Random Bitask in a Schedule : a Real-Time Approach.

Cyril DURON — Jean-Marie PROTH

— Yorai WARDI

N° 4337

Novembre 2001

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4337--FR+ENG

Insertion of a Random Bitask in a Schedule : a Real-Time Approach.

Cyril DURON [*†] , Jean-Marie PROTH [‡ †] , Yorai WARDI [§]

Thème 4 —Simulation et optimisation de systèmes complexes
Projet Sagep

Rapport de recherche n° 4337 —Novembre 2001 —20 pages

Abstract: We consider a set of tasks defined by their duration and their due-date. They are performed by a single resource and scheduled in order to minimize the sum of the delays. This schedule is given.

A task appears in the system at time 0. It is made of two subtasks separated by a fixed period. The duration of the two subtasks and of the period are known only when the task appears. It is also the case for the due date that cannot be violated. The goal is to insert this random task in the schedule while increasing as less as possible the criterion of the initial schedule, that is the sum of the delays. The main difficulty is to insert the task in real-time, which implies that the proposed method should manage to make most of the computation off-line.

Key-words: Real-time, Scheduling, Single resource

[*] E-mail : duron@loria.fr
[†] Shared foot note
[‡] Location : Inria / Sagep, UFR , Scientifique, Université de Metz, Ile du Saulcy, 57000 Metz, France. E-mail : proth@loria.fr
[§] Location : School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA . E-mail : ywardi@ee.gatech.edu

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 —Télécopie : +33 3 83 27 83 19

Insertion d'une bitâche aléatoire dans un ordonnancement : Une approche temps-réel.

Résumé : Nous considérons un ensemble de tâches définies par leur durée et leur délai. Elle sont exécutées par une ressource unique et ordonnancées de façon à minimiser la somme de leurs retards.

Une tâche apparaît dans le système à l'instant 0. Elle est constituée de deux sous-tâches séparées par une période donnée. Les durées des deux sous-tâches et de la période qui les sépare ne sont connues qu'au moment de l'apparition de cette tâche qualifiée d'aléatoire. Il en est de même du délai de cette tâche aléatoire, qui ne peut être violée. L'objectif de ce papier est d'insérer cette tâche aléatoire dans l'ordonnancement existant en augmentant aussi peu que possible le critère associé, c'est-à-dire la somme des retards. La difficulté majeure de ce problème est d'effectuer cette insertion en temps réel, ce qui implique que la méthode proposée devra effectuer la majorité des calculs nécessaires en différé.

Mots-clés : Temps-réel, Ordonnancement, Ressource unique

1 Introduction

In this paper we consider the problem that consists of integrating an unexpected task in a given schedule so as to minimize the increase of the total tardiness in the existing schedule and to complete the unexpected task by a given due date. We assume that the resource in charge of operations is unique and that the tasks are non-preemptives.

The problem was suggested by the management of multifunction radars that performs repetitive tasks and should manage to handle in *real-time* a new task that appears after the detection of an event (a new target for instance).

The unexpected tasks that are performed are in fact composed of two subtasks separated by an idle period. The processing times of the first subtask, the second subtask and the idle period are known only when the new task arises. Our point is to use the idle time for performing initial tasks. The idle period can be used to schedule the initial tasks.

The classical approach for solving this problem is to consider that we have to insert a unique task whose duration is the sum of the durations of both subtasks and the idle period. Doing so, we waste the idle period in the sense that we don't take advantage of it to schedule some of the initial tasks.

Similar works related to the problem at hand are refered as multifunction radar problems. Many authors worked on this important problem. As far as we know, Billeter (see [2]) was the first to publish on multifunction radar. Orman *et al* (see [6] and [3]) described multifunction radars and tried to propose an effective scheduling of the radar jobs under real-time constraint. In their paper, which is a partial abstract of Orman's thesis (see [4]), some scheduling rules are tested and a simulation model is proposed. Some interesting information may be found in [1] and [5].

Unfortunately, none of these works explores the use of the idle periods, which is the goal of the work presented hereafter. The remainings of this publication is divided in two parts. The first one is devoted to the case when the unexpected task is considered as a simple task whose duration is the sum of the duration of the subtasks and the idle period.

In the first part of the paper, the problem is presented in section 2. Section 3 is devoted to the properties of the problem. The real-time strategy is presented in section 4. Section 5 proposes a numerical example.

The second part of the paper explores the case where the goal is to take advantage of the idle period. The problem is formulated in Section 6. Sections 7 and 8 highlight properties of the problem. Section 9 gives an optimal algorithm. Section 10 proposes a first real-time heuristic. A second real-time heuristic is given in section 11. Section 12 presents a numerical example. Section 13.3 contains the complexity of the four algorithms. Section 14 is the conclusion.

PART 1 : The unexpected task is a simple task

2 Formulation of the problem

A schedule that concerns n tasks denoted by $a_1, a_2, \ldots, a_n$ is given. Their duration are $t_1, t_2, \ldots, t_n$ and their starting times are $\mu_1, \mu_2, \ldots, \mu_n$. The ressource is unique.

Indeed, $\mu_i + t_i \leq \mu_{i+1}$ for $i = 1, 2, \ldots, n-1$. The due dates of the tasks are denoted by $d_1, d_2, \ldots, d_n$. We set $\Delta_i = \mu_{i+1} - (\mu_i + t_i)$, $i = 1, \ldots, n-1$. We also set $\Delta_n = +\infty$. Δ_i is the idle period that starts when a_i is completed and ends when a_{i+1} starts.

We also set : $f_i = [d_i - (\mu_i + t_i)]^+$, $i = 1, 2, \ldots, n$ where $[a]^+$ is the smallest integer greater than or equal to a. f_i represents the time task a_i can be postponed without increasing the criterion of the initial schedule that is the sum C_n of the delays: $C_n = \sum_{i=1}^{n} [\mu_i + t_i - d_i]^+$

The unexpected task, which is a simple task in this case , is denoted by A and defined by :

- D, which is an upper bound of its completion time.
- θ, which is its duration.

3 Properties of the problem

Assume that we insert the unexpected task A after task a_k, $k \in \{1, 2, \ldots, n\}$. The following result holds true.

Result 1:

1. Starting A as soon as a_k ends minimizes the increase of criterion C_n for this position.

2. Furthermore :
   If $\sum_{s=0}^{n_k-1} \Delta_{k+s} < \theta \leq \sum_{s=0}^{n_k} \Delta_{k+s}$, then a_{k+s} , $s = 1,\ldots,n_k$ are the only tasks that will be postponed and the starting time of a_{k+s} becomes : $\mu'_{k+s} = \mu_{k+s} + (\theta - \sum_{i=0}^{s-1} \Delta_{k+i}), s = 1,2,\ldots,n_k$
   If $\theta \leq \Delta_k$, none of the tasks is postponed.

Proof :

1. If A starts at time $\mu_k + t_k + c, c > 0$, it will be completed at time $\mu_k + t_k + c + \theta$ and task a_{k+1} will start at time $\mu^c_{k+1} = \max(\mu_k + t_k + c + \theta, \mu_{k+1})$. Similarly, if A starts at time $\mu_k + t_k$, a_{k+1} will start at time $\mu'_{k+1} = \max(\mu_k + t_k + \theta, \mu_{k+1})$. Thus $\mu^c_{k+1} > \mu'_{k+1}$, and $\mu^c_{k+1} + t_{k+1} > \mu'_{k+1} + t_{k+1}$. As a consequence $(\mu^c_{k+1} + t_{k+1} - d_{k+1})^+ \geq (\mu'_{k+1} + t_{k+1} - d_{k+1})^+$. It is easy to extend this inequality to : $(\mu^c_{k+s} + t_{k+s} - d_{k+s})^+ \geq (\mu'_{k+s} + t_{k+s} - d_{k+s})^+$, for $s = 1,2,3,\ldots$ This completes the first part of the proof.

2. This part of the proof is straightforward.

The second result proposes a formulation of the increase of the criterion C_n when A is inserted as soon as task a_k is completed.

Result 2 :

If A is inserted after a_k, then the increase of the criterion C_n is :

$$L_k(\theta) = \begin{cases} 0 & \text{if} \quad \Delta_k \geq \theta \\ \sum_{s=1}^{n_k} (\theta - \sum_{p=0}^{s-1} \Delta_{k+p} - f_{k+s})^+ & \text{if} \quad \Delta_k < \theta \end{cases} \tag{1}$$

Proof :

The increase of C_n due to the postponement of a_{k+1} is :

$$R_{k+1} = \begin{cases} 0 & \text{if} \quad \theta \leq \Delta_k \\ (\theta - \Delta_k - f_{k+1})^+ & \text{if} \quad \theta > \Delta_k \end{cases} \tag{2}$$

since :

- If $\theta \leq \Delta_k$ then A is completed before μ_{k+1}, and $R_{k+1} = 0$
- If $\theta > \Delta_k$ then a_{k+1} is postponed by $\theta - \Delta_k$.
  If $\theta - \Delta_k \leq f_{k+1}$, the due date of a_{k+1} is not violated, and $R_{k+1} = 0$, otherwise the increase due to the postponement of a_{k+1} is $\theta - \Delta_k - f_{k+1}$.

Similarly, the increase of C_n resulting from the postponement of a_{k+2} is :

$$R_{k+2} = \begin{cases} 0 & \text{if} \quad \theta \leq \Delta_k + \Delta_{k+1} \\ (\theta - \Delta_k - \Delta_{k+1} - f_{k+2})^+ & \text{if} \quad \theta > \Delta_k + \Delta_{k+1} \end{cases} \tag{3}$$

The explanation of (3) is as follows. If $\theta \leq \Delta_k + \Delta_{k+1}$, the starting time of the task a_{k+2} is not modified, as well as the following tasks. If $\theta > \Delta_k + \Delta_{k+1}$ then task a_{k+2} is postponed by $\theta - \Delta_k - \Delta_{k+1}$. If $\theta - \Delta_k - \Delta_{k+1} \leq f_{k+2}$, the due date of task a_{k+2} is not broken and the increase of C_n is equal to 0. Otherwise, the increase of C_n is $\theta - \Delta_k - \Delta_{k+1} - f_{k+2}$.

This result can be easily generalized as follows :

$$R_{k+s} = \begin{cases} 0 & \text{if} \quad \theta \leq \sum_{u=0}^{s-1} \Delta_{k+u} \\ (\theta - \sum_{u=0}^{s-1} \Delta_{k+u} - f_{k+s})^+ & \text{if} \quad \theta > \sum_{u=0}^{s-1} \Delta_{k+u} \end{cases} \tag{4}$$

This completes the proof.

As we can see from (1), $L_k(\theta)$is an increasing, continuous and piecewise linear function. Assume that $z_1, z_2, \ldots$ are the quantities $\sum_{p=0}^{s-1} \Delta_{k+p} + f_{k+s}$ ordered in the increasing order. In this case, $L_k(\theta)$ is linear on each interval (z_i, z_{i+1}), and the slope of $L_k(\theta)$ increases from one interval to the next one. Fig. 3 represents such a function.

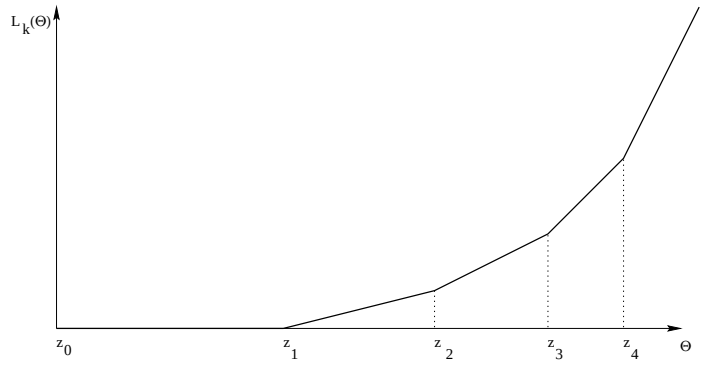

Figure 1: Increase of the criterion.

The next result shows that it is not necessary to try all the possible locations of A to find the optimal one.

Result 3 :

Assume that $\Delta_{k+i} = 0$ for $i = 1, 2, \ldots, h$ and $\Delta_{k+h+1} > 0$. Then inserting the random task A as soon as a_{k+h+1} ends is never worse than inserting A as soon as a_{k+i} is completed, $i \in \{1, 2, \ldots, h\}$.

Proof :

Assume that we insert A after $a_{k+i}, i \in \{1, 2, \ldots, h\}$. Then, tasks $a_{k+i+r}, r = 1, \ldots, h-i+1$ are postponed by A, and the following tasks are postponed so as to absorb a task of duration $(\theta - \Delta_{k+h+1})^+$.

If we insert A after a_{k+h+1}, only the following tasks are postponed so as to absorb a task of duration $(\theta - \Delta_{k+h+1})^+$.

This completes the proof.

From result 3, we deduct an interesting corollary that will allow us to minimize the load of computing in most of the cases.

Corollary 1 :

Let 0 be the instant when a random task of duration θ and deadline D appears, and let $a_1, a_2, \ldots, a_Q$ be the tasks already scheduled, where Q is the greater integer such as $\mu_Q + t_Q + \theta \leq D$

1. Let $W_Q = \{k / k \in \{1, 2, \ldots, Q\} \text{ and } \Delta_k > 0\} \cup \{Q\}$. Then, the optimal position for the random task is just after one element of W_Q. In other words, it is not useful to take into consideration tasks a_k such as $\Delta_k = 0$, except may be for a_Q.
2. If $W_Q = \emptyset$ starting task A just after the end of a_Q is optimal.

This corollary is a direct consequence of result 3.

4 Real-Time Strategy

The approach proposed in this paper aims at making most of the computation off-line, and to reduce as much as possible the on-line computation. The algorithm presented hereafter consists of :

- Computing off-line the functions $L_k(\theta)$ for tasks a_k, $k = 1, \ldots, n$.
- Using these functions when the random task A appears. This is done in two steps :
  - Computation of W_Q (see corollary 1).
  - Computation of $\min_{k \in W_Q} L_k(\theta)$ where θ is the duration of the random task.

This computation is illustrated in Fig. 2and detailed in *algorithm 1*.

Algorithm 1.

1. Off-line computation.
   (a) Compute $L_k(\theta)$ for $k = 1, \ldots, n$ and $\theta \in [0, +\infty)$
2. On-line computation.

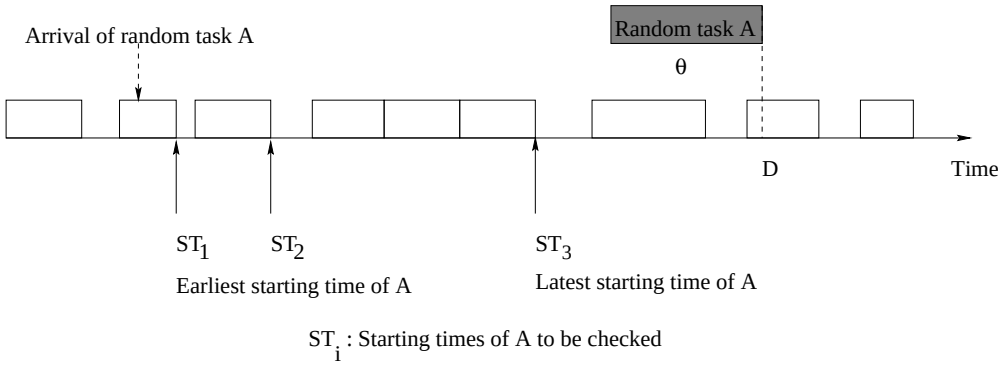

Figure 2: Limits of the starting times of A.

(a) Compute the greater integer Q such that $a_Q + t_Q + \theta \leq D$

(b) Compute $W_Q = \{k/k \in \{1,2,\ldots,Q\} \text{ and } \Delta_k > 0\} \cup \{Q\}$

(c) Compute the increase $z_A(\theta) = \min_{k \in W_Q} L_k(\theta)$ Let k^* be the value of $k \in W_Q$ that leads to the minimum.

(d) Set the schedule up to date :

$$\mu'_{k^*+s} = \mu_{k^*+s} + (\theta - \sum_{i=0}^{s-1} \Delta_{k^*+i})^+ \text{ for } s = 1,\ldots,Q$$

According to the previous results, Algorithm 1 is optimal.

5 A Numerical Example

In this section we propose a numerical example on which we will apply algorithm 1. First, we provide the numerical features of an existing schedule (see Table 1). This table provides Δ_i and f_i for $i = 1,\ldots,30$.

The first step of algorithm 1 (that is off-line computation) provides $L_k(\theta)$ for $k = 1,2,\ldots,n$ and $\theta \in [0,+\infty)$. Remember that these functions are piecewise linear and continuous.

When A arises, that is when θ and D are known, we can :

- Select W_Q
- Compute $L_k(\theta)$ for each $k \in W_Q$
- Find $k^* \in W_Q$ such that $L_{k^*}(\theta) = \min_{k \in W_Q} L_k(\theta)$

Thus, the optimal location of A is after a_{k^*}.

Here, we will assume that we want to insert a task whose $\theta = 59$ and $D = 361$.

We obtain

$W_Q = \{2,3,5,7,8,10,11,15,16\}$

The values of $L_k(\theta)$ for $k \in W_Q$ are given in table 2.

Thus, A should start as soon as a_{k^*} is completed, and $k^* = 15$.

According to step 2.d of the algorithm, the starting times of some tasks were postponed. See Table 3.

Table 1: Example of Schedule

Task number i	Starting time	Processing time	Due date	Δ_i	f_i
1	0	14	24	0	10
2	14	19	43	1	10
3	34	9	53	1	10
4	44	13	53	0	0
5	57	3	50	19	0
6	79	18	89	0	0
7	97	20	127	8	10
8	125	8	143	5	10
9	138	1	131	0	0
10	139	18	167	6	10
11	163	13	182	12	6
12	188	7	193	0	0
13	195	20	211	0	0
14	215	20	225	0	0
15	235	14	259	19	10
16	268	20	279	0	0
17	288	16	306	0	2
18	304	20	334	19	10
19	343	9	345	19	0
20	371	20	384	0	0
21	391	8	409	0	10
22	399	10	413	0	4
23	409	19	432	0	4
24	428	9	447	0	10
25	437	19	466	0	10
26	456	18	467	0	0
27	474	19	503	19	10
28	512	20	536	0	4
29	532	13	555	0	10
30	545	20	575	0	10

Table 2: Values of $L_k(\theta)$ of $k \in W_Q$

Task number k	$L_k(\theta)$
2	322
3	285
5	178
7	275
8	313
10	258
11	259
15	**133**
16	266

PART 2 : The unexpected task is a bitask

6 Problem formulation

A bitask is composed of two subtasks denoted by A_1 and A_2, separated by an idle period of duration L. A bitask is represented in figure 3.

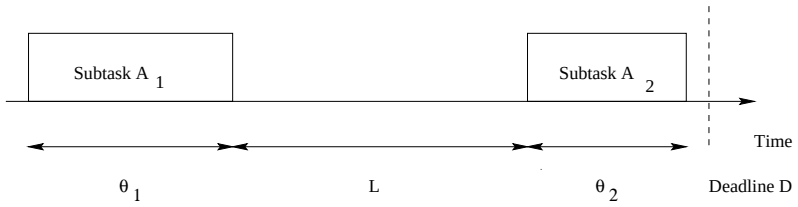

Figure 3: Representation of a bitask A.

The following notations are used :

n : number of already scheduled tasks denoted by 1 to n.
They are arranged in the increasing order of their starting time.
μ_i: initial starting time of task i
$\mu_i^{'}$: new starting time of task i
t_i : processing time of task i
d_i: due date of task i
θ_1: processing time of subtask 1
θ_2: processing time of subtask 2
L : idle time between subtasks 1 and 2
D: due date of the bitasks. This due date cannot be violated.

It's easy to compute the increase C_k of the criterion, resulting of the insertion of the bitask right after the end of task k :

$$C^k = \sum_{i=1}^{r} ((\mu_{k+i} + t_{k+i} - d_{k+i})^- - \mu_{k+i} + \mu_{k+i}^{'})^+$$

where $(A)^+ = Max(0,A)$, $(A)^- = \min(0,A)$ and r is the number of tasks of the initial schedule that are postponed due to the insertion of the bitask. The formulaes that give the new starting times $\mu_i^{'}$, $i \in [k+1, k+r]$, for the postponed tasks are given in section 8

In the case of a bitask, and when we decide to insert A_1 after a_k, it may happen that the optimal solution does not consist in starting A_1 as soon as a_k is completed. The reason is that the location of A_2 is constrained by the location of A_1 and that postponing A_1 slightly may allow t insert one more task between the end of A_1 and the beginning of A_2, which may decrease the criterion. Thus, we introduce the starting time z of A_1. If we want to insert A_1 after a_k, then $z \geq \mu_k + t_k$.

The goal of this problem is to find the value of z that minimizes the increase of the criterion if we decide to insert A_1 after a_k. The results presented in section 7 reduces significantly the computational burden.

7 An upper bound for z

Result 4 :
If we decide to insert A_1 after a_k, the optimal solution is to start A_1 at time z. z is such that $\mu_k + t_k \leq z \leq z_1$ with

Table 3: New schedule

Task number k	6	7	8	9	10	11	12	13	14	15
Prev. starting time	79	97	125	138	139	163	188	195	215	235
New starting time	119	137	157	165	166	184	197	204	224	244

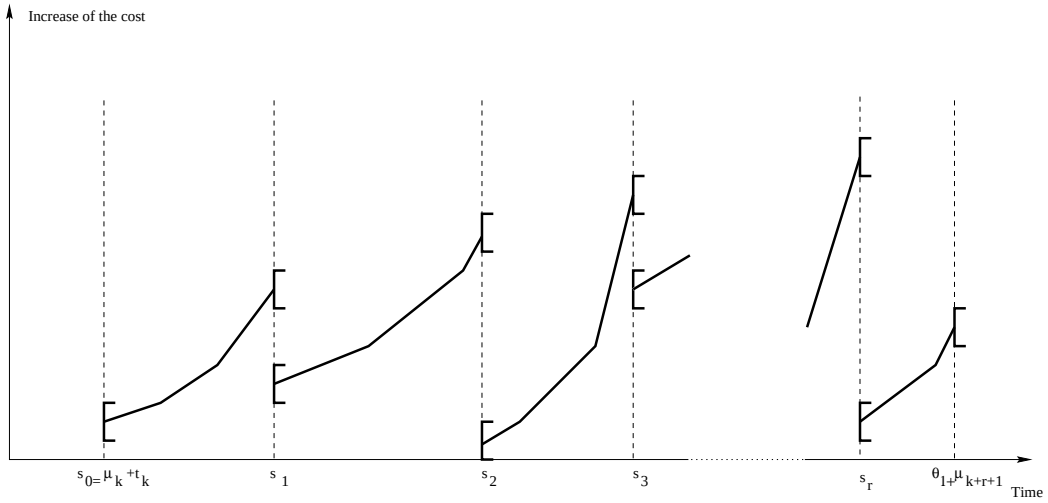

Figure 4: Increase of the criterion.

$z_1 = \min(\mu_{k+r} + t_{k+r} - \theta_1 - L, D - \theta_1 - L - \theta_2)$ where r is the greater integer such that $\sum_{i=1}^{r} t_{k+i} \leq L$

Proof :

1. $\mu_k + t_k \leq z$ is obvious since A_1 is inserted after a_k.
2. When $z = z_1 = \mu_{k+2} + t_{k+2} - \theta_1 - L$, tasks $a_{k+1}, a_{k+2}, \ldots, a_{k+r}$ are located in the time interval of length L between the end of A_1 and the beginning of A_2, and it is impossible to insert one more task in this interval. Thus, if $z > z_1$, tasks $a_{k+1}, a_{k+2}, \ldots, a_{k+r}$ are postponed by $z - z_1$, and tasks $a_{k+r+1}, a_{k+r+2}, \ldots$ may be postponed also, which means that the criterion of the problem cannot be lower than the one obtained for $z = z_1$.
3. When $z = z_1 = D - \theta_1 - L - \theta_2$, then A_1 cannot start after z_1, otherwise the deadline D would be violated.

To simplify the presentation, we still denote by r the number of tasks that are inserted between A_1 and A_2 in this case.

Result 5 :

The optimal solutions for z is one of the values $\{\mu_{k+i} + t_{k+i} - \theta_1 - L\}_{i=1,2,\ldots,r}$. If $z = \mu_{k+i} + t_{k+i} - \theta_1 - L$, then $a_{k+1}, a_{k+2}, \ldots, a_{k+i}$ are inserted between A_1 and A_2, and $a_{k+i+1}, a_{k+i+2}, \ldots$ are performed after A_2. If $r = 0$, that is if $t_{k+1} > L$, then $z = \mu_k + t_k$ is optimal.

Proof :

1. If $z = \mu_{k+i} + t_{k+i} - \theta_1 - L$, $i \in \{1, 2, \ldots, r\}$, then a_{k+i} ends when A_2 starts.
   If $z \in (\mu_{k+i} + t_{k+i} - \theta_1 - L, \mu_{k+i+1} + t_{k+i+1} - \theta_1 - L)$, only $\{a_{k+1}, \ldots, a_{k+i}\}$ are scheduled between A_1 and A_2, and the increase of the criterion is an increasing function of z in this interval.
   If $z = \mu_{k+i} + t_{k+i} - \theta_1 - L - \varepsilon$, wher ε is as small as we want, only $\{a_{k+1}, \ldots, a_{k+i-1}\}$ can be scheduled between A_1 and A_2.Thus, the increase o the cost is greater than the increase of the cost for $\varepsilon = 0$.
   Let us set $s_i = \mu_{k+i} + t_{k+i} - \theta_1 - L$. Figure 7 provide the increase of the cost as a function of z.
2. If $t_{k+1} > L$, a_{k+1} cannot be inserted between A_1 and A_2, and thus the minimum of the criterion is obtained for A_1 starting as soon as possible, that is as soon as a_k is completed.

8 The new starting times

We first consider the tasks inserted between A_1 and A_2. Assume that $z = \mu_{k+i} + t_{k+i} - \theta_1 - L$, $i \in \{1, 2, \ldots, r\}$ According to result 5, $\{a_{k+1}, \ldots, a_{k+i}\}$ are inserted between A_1 and A_2.

$$\begin{cases} \mu'_{k+1} = \max\{\mu_{k+1}, z + \theta_1\} \\ \mu'_{k+j} = \max\{\mu_{k+j}, \mu'_{k+j-1} + t_{k+j-1}\} \quad \text{if} \quad j = 2, \ldots, i-1 \\ \mu'_{k+i} = \mu_{k+i} \end{cases} \tag{5}$$

Let us now consider the tasks that are performed after A_2. For these tasks, that is a_{k+i+s} for $s = 1,2,\ldots$,

$$\begin{cases} \mu'_{k+i+1} = \max\{\mu_{k+i+1}, z+\theta_1+L+\theta_2\} \\ \mu'_{k+i+s} = \max\{\mu_{k+i+s}, \mu'_{k+i+s-1}+t_{k+i+s-1}\} \quad \text{for} \quad j = 2,\ldots \end{cases} \tag{6}$$

9 Optimal Algorithm

To obtain the optimal solution, we test each one of the possible locations of the random task. For a location of the random task after a task a_k, we compute the increase of the criterion if we start A_1 as soon as a_k is completed. We then check each one of the positions given in Result 5.

Algorithm 2 (*n*,μ,t_i,d_i,θ_1,*L*,θ_2,*D*)

Data : The data required are those given in section 6, plus the following temporary variables.

Temporary variables :

overcost	: integer, overcost we are computing
k	: integer, index of the task after which we insert A_1
k_{max}	: integer, index of the last task after which we may insert A_1
r	: integer
r_{prime}	: integer, index of the first task processed after A_2
$finished$	: boolean, regulation variable
beginning	: boolean, regulation variable
μ'_i	: integer , new starting time of task i

Results :

$overcost^*$	: integer, smaller overcost found
μ^*_i	: integer, starting times leading to the smallest increase of the criterion

BEGIN

1. Algorithm Initialisation
   k = 1
   $k_{max} = 1$
   $overcost^*$ = $+\infty$
2. Computation of k_{max}
   (a) WHILE $((\mu_{k_{max}} + t_{k_{max}} + \theta_1 + L + \theta_2 \leq D)$ AND $(k_{max} < n))$ DO $k_{max} = k_{max} + 1$
   (b) IF $(\mu_{k_{max}} + t_{k_{max}} + \theta_1 + L + \theta_2 > D)$ THEN $k_{max} = k_{max} - 1$
3. Initialisation of the computation of the new starting times and overcost when inserting the bitask after task $k+r-1$
   r = 1
   $finished = false$
   overcost = 0
4. Computation of the overcost and the new starting times for the tasks whose new location is between the two subtasks.
   (a) IF $((finished = true)$ OR $(\mu_{k+r} + t_{k+r} > \mu_k + t_k + \theta_1 + L))$ GO TO 5
   (b) IF $(r = 1)$ THEN $\mu'_{k+1} = \max(\mu_{k+1}, \mu_k + t_k + \theta_1)$
   ELSE $\mu'_{k+r} = \max(\mu_{k+r}, \mu'_{k+r-1} + t_{k+r-1})$
   (c) IF $(\mu'_{k+r} + t_{k+r} > \mu_k + t_k + \theta_1 + L)$ THEN
      i. $finished = true$
   ELSE
      i. $overcost = overcost + ((\mu_{k+r} + t_{k+r} - d_{k+r})^- - \mu_{k+r} + \mu'_{k+r})^+$

ii. $r = r + 1$
iii. IF $(k + r > n)$ GO TO 5

(d) GO TO 4

5. Keeping trace of the index of the first task following the bitask
$r_{prime} = r$
$beginning = true$
$finished = false$

6. Computing of the overcost and of the new starting times of the tasks that are located after the bitask.

(a) IF $finished = true$ GO TO 8

(b) IF $(beginning = true)$ THEN

i. $\mu'_{k+r} = \max(\mu_{k+r}, \mu_k + t_k + \theta_1 + L + \theta_2)$
ii. $beginning = false$

ELSE $\mu'_{k+r} = \max(\mu_{k+r}, \mu'_{k+r-1} + t_{k+r-1})$

(c) IF $(\mu'_{k+r} = \mu_{k+r})$ THEN $finished = true$
ELSE

i. $overcost = overcost + ((\mu_{k+r} + t_{k+r} - d_{k+r})^- - \mu_{k+r} + \mu'_{k+r})^+$
ii. $r = r + 1$
iii. IF $(k + r > n)$ THEN $finished = true$

(d) GO TO 6

7. Optimization of the criterion.

(a) $nbtache = 1$
$temps = 0$

(b) IF $((temps + \mu_{k+nbtache} < L))$ THEN

i. $temps = temps + \mu_{k+nbtache}$
$nbtache = nbtache + 1$
ii. GO TO 7b

(c) $nbtache = nbtache - 1$

(d) This part is activated only if it is possible to insert an additional task between A_1 and A_2 by postponing A_1. IF $(nbtache > origine)$ THEN

i. $ajout = nbtache - origine$
ii. FOR $cpt = 1$ TO $ajout$ STEP 1

A. $maxy = origine + cpt$
$\mu'_{k+1} = \max(\mu_{k+1}, \mu_{k+maxy} + t_{k+maxy} - L)$
$ovcost2 = ((\mu_{k+1} + t_{k+1} - d_{k+1})^- - \mu_{k+1} + \mu'_{k+1})^+$
$r = 2$

B. $\mu'_{k+r} = \max(\mu_{k+r}, \mu_{k+r-1} + t_{k+r-1})$
$ovcost2 = ovcost2 + ((\mu_{k+r} + t_{k+r} - d_{k+r})^- - \mu_{k+r} + \mu'_{k+r})^+$
$r = r + 1$
IF $r \leq maxy$ GO TO 7(d)iiB

C. $finished = false$
$r = nbtache + 1$
$\mu'_{k+r} = \max(\mu_{k+r}, \mu_{k+origine+1} + t_{k+origine+1} + \theta_2)$

D. IF ($finished$ = true) GO TO 7(d)iiF

E. $\mu'_{k+r} = \max(\mu_{k+r}, \mu'_{k+r-1} + t_{k+r-1})$
IF $\mu'_{k+r} = \mu_{k+r}$ $finished = true$
ELSE $ovcost2 = ovcost2 + ((\mu_{k+r} + t_{k+r} - d_{k+r})^- - \mu_{k+r} + \mu'_{k+r})^+$
$r = r + 1$
$finished = finished$ OR $(k + r \geq n)$

F. SI ($ovcost2 < overcost$) $overcost = ovcost2$

8. If the overcost obtained when the bitask is inserted after a_k is the smallest computed so far, we keep record of it.
   (a) IF ($overcost < overcost^*$) THEN
      i. $overcost^* = overcost$
      ii. FOR $i = 1$ TO k DO $\mu_i^* = \mu_i$
      iii. $\mu_{k+1}^* = \mu_k + t_k + \theta_1$
      iv. FOR $i = k+1$ TO $k + r_{prime} - 1$ DO $\mu_{i+1}^* = \mu_i^{'}$
      v. $\mu_{k+r_{prime}+1}^* = \mu_k + t_k + \theta_1 + L + \theta_2$
      vi. FOR $i = k + r_{prime}$ TO $k + r$ DO $\mu_{i+2}^* = \mu_i^{'}$
      vii. FOR $i = k + r + 1$ TO n DO $\mu_{i+2}^* = \mu_i$
   (b) IF $k < k_{max}$ THEN
      i. $k = k + 1$
      ii. GO TO 3

END

10 First heuristic Algorithm

The first real time heuristic is based on the following idea : when the bitask arrives, we try to schedule a single task whose length is equal to $\theta_1 + L + \theta_2$. We use Algorithm 1 to find the optimal insertion. Then we insert the bitask taking advantage of its idle time at the position provided by algorithm 1.

The off-line part of the algorithm is the same than the one of algorithm 1. The on-line part of the algorithm is the same than the one of algorithm 1, followed by the computation of the new starting times of the tasks, taking avantage of the idle time of the bitask.

This algorithm may be summarized as follows :

Algorithm 3

1. Off-line computation.
   (a) Compute $L_k(\theta)$ for $k = 1, \ldots, n$ and $\theta \in [0, +\infty)$
2. On-line Computation.
   (a) Compute the greater integer Q such that $a_Q + t_Q + \theta_1 + L + \theta_2 \leq D$
   (b) Compute $W_Q = \{k / k \in \{1, 2, \ldots, Q\} \text{ and } \Delta_k > 0\} \cup \{Q\}$
   (c) Compute the increase $z_A(\theta) = \min_{k \in W_Q} L_k(\theta)$ Let k^* be the value of $k \in W_Q$ that leads to the minimum.
   (d) Set the schedule up to date using the equations (5) and (6).

11 Second heuristic Algorithm

The second real time algorithm works as the optimal one (*Algorithm 2*), except that only the first possible starting time of A_1 is tested for each location. The difference with *Algorithm 2* is that we do not execute step 7. In the remaining of this paper we call this heuristic *Algorithm 4*.

12 Numerical examples

In this section we test the performance of the heuristic described in *algorithm 3* versus the performance of *algorithm 4* in term of quality of results. We compare with the optimal results provided by *Algorithm 2*.

We generate a schedule at random using the following rules :

- Processing times of the tasks are generated at random as follows :

- Interval [1,8] is selected with a 0.3 probability.
- Interval [9,13] is selected with a 0.2 probability.
- Interval [14,20] is selected with a 0.5 probability.

As soon as the interval is selected, the processing time is chosen at random in the interval.

- Each task is followed by an idle period the length of which is generated as follows :
  - 0 with a 0.5 probability.
  - chosen at random in [1,20] with a 0.5 probability.
- The deadline of a task is selected at random in the interval $[z-1, z+10]$ where z is the completion time of the task.

This schedule is given in Table 6

We try to insert random bitasks in that schedule. These bitasks are generated at random. θ_1, L, θ_2 are generated respectively on [1,30], [1,50] and [1,20], using uniform repartition. The deadline $D = 600$ is the same for each bitask.

We consider 10 cases. They are described in table 4

We apply *Algorithm 1*, *Algorithm 2*, *Algorithm 3* and *Algorithm 4* on each one of these cases. The results are presented in table 7

13 Complexity

In this section, we compute the complexity of each proposed algorithm. We assume that any event (jump, operation, test) has the same complexity. We only consider the worst case. We assume that there are n tasks in the schedule where we want to insert the random task

13.1 Complexity of *Algorithm 1*

13.1.1 Off-line complexity.

The better algorithm we can use to compute (1) is given straightforward : FOR i = 1, n DO

FOR j = 1, i DO

$X_{i,j} = 0$
$Y_{i,j} = 0$

FOR j = i+1, i DO

$X_{i,j} = X_{i,j-1} - \Delta_{j-1}$
$Y_{i,j} = X_{i,j-1} - f_{j-1}$

The complexity associated to this algorithm is $C^1_{off} = \sum_{i=1}^{n}(\sum_{j=1}^{i} 2 + \sum_{j=i+1}^{n} 4) \leq 3n^2$

13.1.2 On-line complexity.

The *algorithm 1* may be detailled as follows for a better understanding

1. Q = 1
   complexity : 1
2. IF $a_Q + t_Q + \theta \leq D$ THEN
   Q=Q+1
   GO TO 2
   complexity : $\sum_{i=1}^{6} 6 = 6n$
3. $Q = Q - 1$
   $W_Q = \emptyset$
   complexity : 3

4. FOR k=1, Q-1

   IF (Δ_k>0) THEN $W_Q = W_Q \cup \{a_k\}$

   complexity : $\sum_{i=1}^{n-1} 4 = 4(n-1)$

5. $W_Q = W_Q \cup \{a_Q\}$
   $Min = L_{W_Q^1}(\theta)$
   $k^* = 1$
   complexity : 3

6. FOR k=2, $CARD(W_Q)$

   IF ($Min > L_{W_Q^k}(\theta)$) THEN

   $Min = L_{W_Q^k}(\theta)$

   $k^* = k$

   complexity : $\sum_{k=2}^{n} 3 = 3(n-1)$

7. FOR $s = 1, Q$

   $\mu'_{k^*+s} = \mu_{k^*+s} + (\theta - \sum_{i=0}^{s-1} \Delta_{k^*+i})^+$

   complexity : The summations on the Δ_k have already been computed off-line so there will only have $\sum_{i=1}^{n} 4 = 4n$

The total on-line complexity will be $C_{on}^1 \leq 17n$

13.2 Complexity of *Algorithm 2*

There has no off-line complexity to compute. The complexity of algorithm 2 is detailed step by step.

- Step 1 : 3
- Step 2 : 10n+8
- Step 3 : 3
- Step 4 : 32
- Step 5 : 3
- Step 6 : 29
- Step 7 : 8+(n-k)(60+20n)
- Step 8 : 2n+10

We only provide the basic complexity for each step, the number of time we will execute each step is function of the case. In our case, the worst complexity arises when there is no task that can be located between the two subtasks of the bitask (due to the completion time) and that the processing time of all the tasks that are after a_k is less than or equal to L.

So, the complexity will be :

$$\begin{aligned} C_{on}^2 &= 3 + \sum_{i=1}^{n-1} 32 + 3 + \sum_{k=1}^{n} (8 + (n-k)(60+20n)) + 2n + 10 \\ &= 34n^2 - 6n + 11 + 30n^3 + 20n^2 + 38n \\ &= 30n^3 + 54n^2 + 32n + 11 \end{aligned}$$

13.3 Complexity of *Algorithm 3*

This algorithm has the same off-line computations to perform than *Algorithm 1* so, its off-line complexity will be : $C^3_{off} = C^1_{off} \leq 3n^2$

The on-line complexity is given by the complexity of steps 2.a to step 2c of the on-line part of *Algorithm 1*, that is $13n$. Then, we have to run for one value of k the steps 3,4,5,6 and 8 of *Algorithm 2*. The complexity is $3 + \sum_{i=1}^{n-1} 32 + 3 + 2n + 10 = 34n - 19$

And, finally $C^3_{on} = 13n + 34n - 19 = 47n - 19 \leq 47n$

13.4 Complexity of *Algorithm 4*

There is no off-line complexity to compute. The values of the complexity are given in section 13.2. Step 7 of the algorithm is ignored.

$C^4_{on} = 3 + 10n + 8 + \sum_{k=1}^{n} (3 + \sum_{i=1}^{n-k} 32 + 3 + 2n + 10)) = 34n^2 - 6n + 11$

The results on complexity are displayed in Table 5

14 Conclusions

The conclusions of this paper are the following :

- Algorithm 3 provides a value of the criterion that is always significantly better than the value provided by algorithm 1. It means that it is always better to take advantage of the idle period.
- The results provided by Algorithm 4 are slightly better than the results provided by algorithm 3.

A good trade-off between the complexity and the quality of the solution is *Algorithm 3*.
Further research concern the insertion of a bitask in a schedule of bitasks.

Table 4: The parameters of the bitasks.

Bitask number i	θ_1	θ_2	L
1	30	19	13
2	30	9	55
3	1	17	36
4	5	12	41
5	30	12	10
6	6	12	31
7	22	11	9
8	26	12	16
9	30	10	41
10	14	10	47
11	30	1	23

Table 5: Complexity of the algorithms.

	Algorithm 1	*Algorithm 2*	*Algorithm 3*	*Algorithm 4*
Complexity *Off-line*	$3n^2$	0	$3n^2$	0
Complexity *On-line*	$17n$	$30n^3+54n^2+32n+11$	$47n$	$34n^2-6n+11$

Table 6: Example of Schedule

Task #*i*	Start. time	Proces. time	Due date	Idle time		Task #*i*	Start. time	Proces. time	Due date	Idle time
1	0	17	11	0		26	465	20	495	0
2	17	8	32	0		27	485	20	514	15
3	25	20	41	0		28	520	3	531	0
4	45	13	53	0		29	523	13	543	0
5	58	15	83	0		30	536	15	547	11
6	73	8	76	0		31	562	20	592	19
7	81	17	105	0		32	601	12	606	15
8	98	14	122	19		33	628	8	639	1
9	131	17	158	3		34	637	8	643	8
10	151	13	165	8		35	653	19	672	19
11	172	13	194	0		36	691	20	707	0
12	185	8	186	0		37	711	13	717	10
13	193	20	220	14		38	734	17	742	14
14	227	13	236	0		39	765	15	781	0
15	240	7	257	15		40	780	13	794	16
16	262	20	278	15		41	809	9	825	3
17	297	13	305	19		42	821	20	836	15
18	329	10	344	0		43	856	12	875	19
19	339	8	337	0		44	887	15	903	0
20	347	20	377	15		45	902	13	912	18
21	382	15	407	0		46	933	5	939	0
22	397	8	415	0		47	938	13	952	0
23	405	2	403	19		48	951	12	973	0
24	426	8	424	18		49	963	13	976	0
25	452	13	468	0		50	976	8	994	0

Table 7: Results

	Algorithm 1			Algorithm 2		
bitask Id	Inserted After	Starting time	Criterion	Inserted After	Starting time	Criterion
1	15	247	103	23	407	42
2	15	247	344	8	112	14
3	23	407	66	6	81	0
4	15	247	84	12	194	0
5	23	407	56	23	407	15
6	23	407	44	6	81	0
7	23	407	25	23	407	3
8	23	407	66	23	407	13
9	15	247	229	15	247	15
10	15	247	155	17	310	0
11	23	407	66	23	412	16

	Algorithm 3			Algorithm 4		
bitask Id	Inserted After	Starting time	Criterion	Inserted After	Starting time	Criterion
1	15	247	103	23	407	42
2	15	247	36	8	112	14
3	23	407	6	6	81	0
4	15	247	8	20	367	0
5	23	407	15	23	407	15
6	23	407	1	6	81	0
7	23	407	3	23	407	3
8	23	407	13	23	407	13
9	15	247	15	15	247	15
10	15	247	23	17	310	0
11	23	407	17	23	407	17

References

[1] Huizing A. and Bossé E. A high-level multifunction radar simulation for studying the performance of multisensor data fusion systems. *Part of the SPIE conference on signal processing, Sensor fusion, and target recognition VII, Orlando, Florida*, April 1998.

[2] Billeter D.R. *Multifunction Array Radar*. Artech House, 1989.

[3] Orman A. J. Modelling for the control of a complex radar system. *Computers Ops Res.*, Vol. 25, n°3, pp 239-249, 1998.

[4] Orman A. J. *Models for scheduling a multifunction phased array radar system.* PhD thesis, University of Southampton, 1998.

[5] Weinberg L. Scheduling multifunction radar systems. *RCA Government Systems, Division Missile and Surface Radar, Moorestown, N.J., 08057*, <10-4A-EASCON77 to 10-4J-EASCON77>.

[6] Shahani A.K. Moore A.R. Orman A.J., Potts C.N. Scheduling for a multifunction array radar system. *European Journal of operational research*, 90, pp 13-25, 1996.

Contents

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399