HAL
open science

Linked Task Scheduling : Algorithms for the Single Machine Case

Cyril Duron, Jean-Marie Proth

To cite this version:

Cyril Duron, Jean-Marie Proth. Linked Task Scheduling : Algorithms for the Single Machine Case. [Research Report] RR-4602, INRIA. 2002, pp.12. inria-00071983

HAL Id: inria-00071983

https://inria.hal.science/inria-00071983v1

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.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Linked Task Scheduling : Algorithms for the Single Machine Case

Cyril Duron — Jean-Marie Proth

N° 4602

Octobre 2002

THÈME 4

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4602--FR+ENG

Linked Task Scheduling : Algorithms for the Single Machine Case

Cyril Duron [*†] , Jean-Marie Proth [‡ †]

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Sagep

Rapport de recherche n° 4602 — Octobre 2002 — 12 pages

Abstract: We consider a system made of one resource. The execution of the tasks is non-preemptive on this resource. The tasks we consider are composed of a given number of subtasks, two consecutives subtasks being separated by an idle period. These idle periods may be used for executing other subtasks. We wish to insert a new task in a given schedule. The characteristics of this task are not known before it appears, and its execution must be completed before a given deadline. The criterion is the minimization of the increase of the sum of the delays of the tasks.

Key-words: Scheduling, Linked Tasks, NP-Hardness,Graphs

[*] Corresponding author. E-mail : duron@loria.fr, Phone : 00 33 (0)3 87 31 54 85 Fax : 00 33 (0)3 87 31 54 78
[†] Location : Inria / Sagep, UFR Scientifique, Université de Metz, Ile du Saulcy, 57000 Metz
[‡] E-mail : proth@loria.fr, Phone : 00 33 (0)3 87 31 54 58 Fax : 00 33 (0)3 87 31 54 78

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

Ordonnancement de Trains de Tâches :
Algorithmes pour le Cas d'une Ressource Unique

Résumé :

Nous envisageons un système comprenant une seule ressource, où les tâches s'exécutent de manière non préemptive. Les tâches que nous considérons sont composées d'un nombre donné de sous-tâches, deux sous-tâches successives étant séparées par une période de temps disponible. Les périodes de temps disponibles sont exploitables pour les autres sous-tâches. Nous voulons insérer une nouvelle tâche dans un ordonnancement préexistant. Les caractéristiques de cette tâches ne sont pas connues avant son apparition, et son exécution doit impérativement être terminée avant un délai donné. Le critère considéré est la minimisation de la somme des retards des tâches.

Mots-clés : Ordonnancement, Trains de tâches,NP-Complétude,Graphes

1 Introduction

The problem considered in this paper takes origin in the management of radars (see [3, 7]). A radar is considered as a single resource on which we have to schedule non-preemptive tasks. A basic radar task is composed of two subtasks (emission then reception of a wave) separated by an idle period. The duration values of these subtasks and of the idle period are computed from the distance of the putative target, and cannot be changed. Depending of the kind of task (pursuit, detection, ...) this basic task has to be performed a given number of times. The atmospheric conditions may lead to increase this number.

This problem is a scheduling problem. Formulation and exemples of scheduling problems may be found in [8, 2].

The scheduling of basic radar tasks, that is tasks containing two subtasks separated by an idle period, have been solved, using two approaches. The first approach consists in merging the two subtasks and the idle time in only one processing time (see [5, 9]). The second approach consists in allowing the use of the idle period of a given task by other subtasks (see [4, 1]).

In this paper, we deal with a more general problem. We consider that the tasks are made of a given number of subtasks, separated by idle periods.

Section 2 is dedicated to problem setting. In section 3 we describe an algorithm that solves the problem when the weight of the tasks is an integer value. Section 4 takes advantage of graph theory for inserting subtasks with a non-integer duration. The complexity of the two algorithms is given in section 5. Section 5.2 is the conclusion.

2 Problem Setting

The problem concerns non-preemptive tasks executed on a single resource. The tasks we consider in this paper are made of a given number of subtasks. Each of these subtasks is separated from the next one by an idle period in which other subtasks can be scheduled. This period is equal to zero or to a given number of time units. In the remaining of this paper, we will call such tasks *polytasks*.

Let T be a set of n polytasks, and S an optimal schedule of T, that is a schedule that minimizes the sum of the delays. Let t_i, $i \in \{1,2,\dots,n\}$ be the ith task of T. We assume that the tasks are ordered in their increasing starting time in S. n_i is the number of subtasks that compose t_i. Let t_i^j, $j \in \{1,2,\dots,n_i\}$ be the substasks of t_i. The starting time of t_i^j is given by g_i^j, where $j \in \{1,2,\dots,n_i\}$, and its duration is given by l_i^j. The deadline associated to task t_i is D_i.

At time 0, a new polytask t_* appears and has to be inserted in the schedule S before a given deadline D_*. In the remaining of the paper we refer to this random task as t_*.

The schedule S is such that each task is scheduled as soon as possible. As a consequence, it is possible to postpone some tasks of S, but not to bring them foreward. Some subtasks are not postponable. This is true at least for the first task executed in the order of the schedule, since this task has already been started, but it is also true for all the subtasks belonging to tasks that started before time 0.

The criterion C which is the minimization of the sum of the delays is computed as follows :

$$C = \sum_{i=1}^{n} (g_i^{n_i} + l_i^{n_i} - D_i)^+ \text{ where } (x)^+ = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geq 0 \end{cases} \tag{1}$$

In the first approach developped in section 3, we assume that the duration of a subtask is unitary. In section 4; the second approach considers subtask duration as a non-integer value.

3 Approach using the starting periods of the polytasks.

In this section, we assume that the duration of all the substasks of the polytasks is unitary. In order to modelize the problem, we consider two data structures.

The first one is a table. This table has two columns. The first one contains the name t_i of a task, and the second column contains the list of unit periods where the subtasks of t_i are located. We refer to this data structure as the task table. Table 1 provides the task table that corresponds to the schedule represented in figure 1.

The second structure is a list (see figure 2). Each element of this list contains a pair of data. The first element of the pair is a busy unit period. The second element is the name of the task having one of its subtasks scheduled in this unit period.

The list is built in the increasing order of the busy unit periods. We refer to this data structure as a chronological list. In figure 2, we provide the chronological list corresponding to the schedule of figure 1

The references we introduce in this structure are the following :

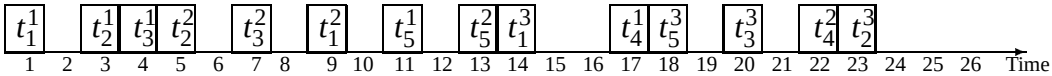

Figure 1: Example of schedule.

- In the task table, the starting time of the subtasks are in fact references of the corresponding nodes in the chronological list.
- In the chronological list, the name of the task is a reference of the corresponding entry in the task table.

Some tasks are impossible to postpone. It is the case of the first task and the postponement of the tasks that started before time 0. We refer to the corresponding subtasks as blocked subtasks.

The algorithm using this structure of data is provided below. It is called Insertion Algorithm, and finds a solution, if any, when we fix the period at which the first subtask of the random task will be performed.

INSERTION ALGORITHM

1. Starting the insertion
   (a) Choose the unit period of the schedule where you want to set the first subtask of the random task.
   (b) Create a new line in the task table. In the first column, we introduce the name of the random task. In the second column we provide the list of the periods that should contain the subtasks of the random task.
   (c) Create in the chronological list the pairs corresponding to those insertion periods such that the resulting list is still chronologically ordered.
2. If all the periods of the chronological list are different from each other then go to 3
   else select in the chronological list the first reference that appears at least twice.
   (a) If none of the tasks corresponding to the reference can be postponed, then there has no solution. END.
   (b) otherwise postpone one of the non-blocked task corresponding to the reference by one unit of time (This may create new conflicts).
   (c) Go to 2
3. END

The following result states about the calculability of the insertion algorithm.

Table 1: The task table for the example schedule.

Task Id	subtasks execution time
t_1	1,9,14
t_2	3,5,23
t_3	4,7,20
t_4	17,22
t_5	11,13,18

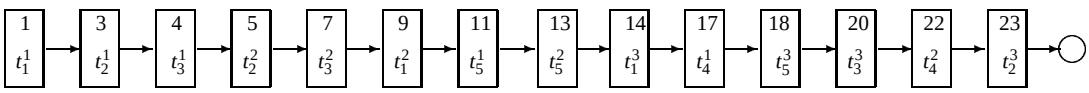

Figure 2: The chronological list for the example schedule.

Result 1

The insertion algorithm finishes.

Proof 1

In case of conflict, at least one already scheduled task is postponed, and the random task remains at the same position.

As a consequence, the same pair of subtasks cannot be in conflict twice. Since the number of subtasks is limited, the algorithm ends. This completes the proof □

Example : We present a small example of the use of the Insertion Algorithm. We consider the schedule given in figure 1, and we try to insert a new task t_* in the periods 3, 12 and 25. The table 2 provides the differents steps of the computation, and the final result is provided in figure 3.

Table 2: Example of use of the Insertion Algorithm

Step 1	1	3	3	4	5	7	9	11	11	13	14	17	18	19	20	22	23
	t_1^1	t_*^1	t_2^1	t_3^1	t_2^2	t_3^2	t_1^2	t_*^2	t_5^1	t_5^2	t_1^3	t_4^1	t_5^3	t_*^3	t_3^3	t_4^2	t_2^3
Step 2	1	3	4	4	6	7	9	11	11	13	14	17	18	19	20	22	24
	t_1^1	t_*^1	t_2^1	t_3^1	t_2^2	t_3^2	t_1^2	t_*^2	t_5^1	t_5^2	t_1^3	t_4^1	t_5^3	t_*^3	t_3^3	t_4^2	t_2^3
Step 3	1	3	4	5	6	8	9	11	11	13	14	17	18	19	21	22	24
	t_1^1	t_*^1	t_2^1	t_3^1	t_2^2	t_3^2	t_1^2	t_*^2	t_5^1	t_5^2	t_1^3	t_4^1	t_5^3	t_*^3	t_3^3	t_4	t_2^3
Step 4	1	3	4	5	6	8	9	11	12	14	14	17	19	19	21	22	24
	t_1^1	t_*^1	t_2^1	t_3^1	t_2^2	t_3^2	t_1^2	t_*^2	t_5^1	t_5^2	t_1^3	t_4^1	t_*^3	t_5^3	t_3^3	t_4^2	t_2^3
Step 5	1	3	4	5	6	8	9	11	13	14	15	17	19	20	21	22	24
	t_1^1	t_*^1	t_2^1	t_3^1	t_2^2	t_3^2	t_1^2	t_*^2	t_5^1	t_1^3	t_5^2	t_4^1	t_*^3	t_5^3	t_3^3	t_4^2	t_2^3

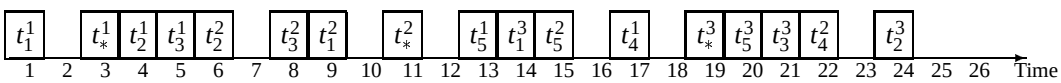

Figure 3: New Schedule made with Insertion Schedule 1

A quality of this algorithm is that there's not always conservation of the original order of the tasks : this order may change during the computation (because of the blocked tasks).

Two obvious ways to reduce the computation burden stands in the following remarks :

- When a task we want to postpone has k subtasks separated by zero periods of time, then we postpone it of k periods of time instead of one.
- When a task t we want to postpone is in conflict with a task (or a bunch tasks that cannot move independently from each other) that is scheduled in several consecutive periods, then it is better to postpone t of more than one period of time.

The Insertion Algorithm may be extented towards integer but non-unitary subtasks duration, considering the fact that a subtask that has a duration of k periods may be subdivided into k unitary subtasks separated by an empty idle time.

4 Approach using a graph

In this section, we represent the schedule by a graph called Flexibility Graph. Its goal is to provide a fast way to compute the maximum postponement of a given task in the schedule S. Each task of S is represented by a node of the graph, labelled with this task's identificator. An arc joins the nodes representing task t_{i_1} to the node representing t_{i_2} if and only if there exists at least a subtask of t_{i_1} that immediatly preceeds a subtask of t_{i_2}. The weight of the arc is the minimal idle period separating a subtask of t_{i_1} from a subtask of t_{i_2}. We denote this weight by $C(i_1, i_2)$. Furthermore, $C(i_1, i_2) = +\propto$ if there is no arc whose origin is t_{i_1} and extremity is t_{i_2}.

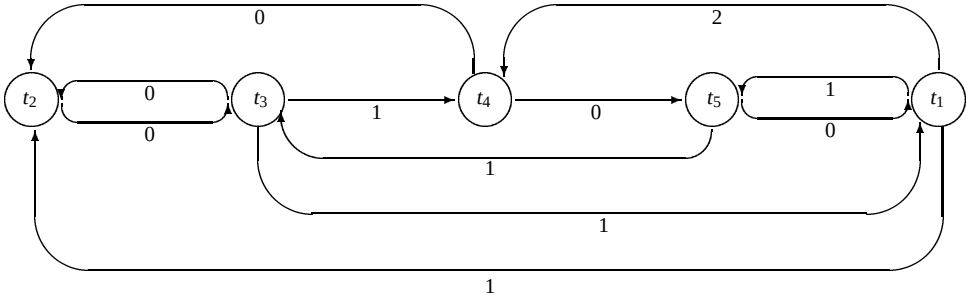

Figure 4: The flexibility graph.

In figure 4, we provide the flexibility graph corresponding to S, denoted by $G(S)$.
Indeed, if all the tasks are fixed except t_{i_1}, then t_{i_1} can be postponed by at most $\theta = \min_{i_2 \neq i_1} C(i_1, i_2)$.

4.1 Graph simplifications

A first simplification of the flexibility graph is made because of the blocked tasks : it is useless to keep the arcs that start from the nodes representing these tasks.
The first simplification of the flexibility graph of figure 4 is provided in the left part of figure 5.
The following result allows a second simplification of the flexibility graph :

Result 2

Let t_1 and t_2 be two nodes of $G(S)$. If $C(t_1,t_2) = C(t_2,t_1) = 0$ then t_1 and t_2 are mutually blocked and cannot be postponed independently from each other.

Proof 2

1. If $C(t_2,t_1) = 0$ then the smallest amount of time t_1 may be postponed without postponing t_2 is 0. So, it is not possible to postpone A without postponing t_2.
2. If $C(t_2,t_1) = 0$ then the smallest amount of time t_2 may be postponed without postponing t_1 is 0. So, it is not possible to postpone t_2 without postponing t_1.

This completes the proof □

So, if several tasks are mutually blocked (see result 2), we can merge them. Merging those tasks consists in :

1. representing these tasks by one node.
2. removing the arcs that join those tasks, and
3. taking the new node as the extremity (resp. the origin) of each arc that had one of the previous two nodes as an extremity (resp. origin).

The reader notices that the resulting graph may be divided into subgraphs that have no connections with each other (non-connex graph).
The construction algorithm of the simplified flexibility graph (SFG for short) is given below.

SFG ALGORITHM

Let BT be the set of blocked tasks.
Let PT be the set of postponable tasks. pt_i is the ith element of PT.
We suppose this set of task ordered in the increasing starting time of the tasks.
Let G be the set of tasks under consideration. g_i is the ith element of G.
Initially, $G = \emptyset$.

1. If $G = \emptyset$ then
   (a) If $PT = \emptyset$ then go to 3
   (b) else $G = \{pt_1\}$, $PT = PT \setminus \{pt_1\}$.
2. Let $succ(g_1)$ be the set of tasks that are the successors of g_1.
   (a) $G = G \cup (succ(g_1) \cap PT)$
   (b) $PT = PT \setminus succ(g_1)$
   (c) $BT = BT \setminus succ(g_1)$
   (d) $G = G \setminus \{g_1\}$
   (e) For each t, $t \in succ(g_1)$, create an arc oriented from g_1 to t, and weight it with the minimum flexibility between the two tasks.
   (f) Go to 1
3. The nodes that are linked with a bidirectionnal arc weighted with a value of 0 must be fusionned. The arcs starting from or arriving to the original nodes must be merged too, using their minimum value.

The second simplification of the flexibility graph of figure 4 is provided in the right part of figure 5.

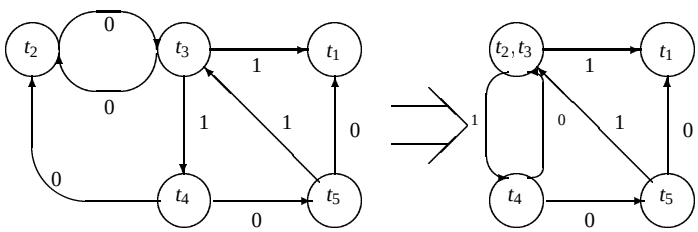

Figure 5: The simplified flexibility graph of $G(S)$.

The reader will notice that the only task that may be postponed in this graph is the pair $t_2 - t_3$, for up to 1 period of time.

4.2 Basic result

The following result provides information about the maximum amount of time we may postpone a non-blocked task of the schedule.

Result 3

The maximum value we may postpone a non-bloked task t of S is given by the minimum of the shortest paths whose origin is t and whose extremities are blocked tasks. If no such path exists, then t can be postponed to infinity.

Proof 3

Let BT be the set of blocked tasks of S extremities of a path whose origin is t.
Let $C_{min}(a,b)$ be the weight of the shortest path between task a and task b.

1. If $BT = \emptyset$ then there exists no constraint on t, and t can be postponed to infinity.

2. Let bt_1 and bt_2 be two elements of BT. If $C_{min}(t,bt_1) > C_{min}(t,bt_2)$ (resp. $C_{min}(t,bt_2) > C_{min}(t,bt_1)$) then postponing t by $C_{min}(t,bt_1)$ (resp. $C_{min}(t,bt_2)$) would lead to postpone bt_2 (resp. bt_1), which is impossible by assumption. So, the maximum value we may postpone t is given by $\min(C(t,bt_1),C(t,bt_2)))$. Thus, the maximum postponement of task t is given by $\min_{bt \in BT} C_{min}(t,bt)$.

This completes the proof □

In our example, if we wish to postpone the task t_2, the maximum value we may postpone it is 1, and the resulting graph is given figure 6.

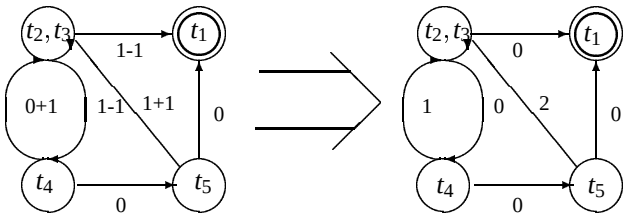

Figure 6: Postponement of task t_2 by one period of time.

In result 3, we have to compute the shortest paths (in the simplified flexibility graph) between the node t associated to the task we want to postpone, and the blocked nodes extremities of paths originated in t, if any. The maximal postponement is then given by the path with the smallest weight.

The computation of the shortest path is made using the Dijkstra Algorithm (see [2, 6]).

4.3 Insertion algorithm 2

In this subsection, we provide an algorithm inserting a new polytask in a given schedule S. First, we use the SFG algorithm to create the graph $SFG(S)$. Then, we compute the set of possible insertion instants for the new polytask in S. The possible insertion periods are such that it does not lead the new polytask to violate its due date D_*. We test these possible instants of insertion in their chronological order. A test consists in evaluating for each subtask of the new polytask if the time available at the wished insertion instant is big enough to allow the insertion of the considered subtask. If yes then we iterate with the next subtask. If no, we use the result 3 to know whether postponing the task that should follow the considered subtask in S would solve the problem. If no then the insertion is not possible here, and we have to consider another starting instant for the insertion, if any.

These requires adjustements in $SFG(S)$. If we perform a postponement of tasks, we do not need to start over the computation of $SFG(S)$.

Let z be the maximum postponement of t_i and $k_i \leq z$. It is easy to define the new simplified flexibility graph obtained when postponing a task t_i by k_i :

- k_i is substracted from the weight of the arcs issued from the node associated to t_i.
- k_i periods of time are added to the arcs whose extremity is the node associated with t_i.

If the weights of the arcs are negative, we proceed as follows.

For each node u that is the extremity of at least one arc whose weight is negative, we compute $a = \min_{v \in Pred(u)} (C(v,u)) < 0$ where $Pred(u)$ is the set of nodes origin of the arcs whose extremity is u. We then add $-a$ to each arc whose origin is u.

We continue until the weight of all the arcs is greater than or equal to zero.

The Insertion Algorithm using the graph approach is summarized below in the Insertion Algorithm 2 (IA2 for short)

Insertion Algorithm 2

1. The new polytask t_* appears.
2. Choose an insertion period for t_* that is not already checked, and that does not lead to violate the deadline D_*. If there has none, then it is not possible to insert t_*. END.
3. For $i = 1$ to n_* do :
   (a) If it is not possible to insert t_*^i in the known schedule without postponing at least one task of S, then :
      i. Create (if needed) SFG(S) using the *SFG* Algorithm.
      ii. Compute the maximal amount of time m we may postpone the task of S that is in conflict with t_*^i, using result 3.
      iii. If postponing the task scheduled after t_*^i by a period m does not allow the insertion of t_*^i, then go to 2
      iv. Else postpone the task scheduled after t_*^i of the exact time needed to insert t_*^i.
      v. Update SFG(S). Go to 3.
   (b) Else insert t_*^i. Update SFG(S). Go to 3.
4. The insertion of t_* is done : END

The following result states about the calculability of the Insertion Algorithm 2.

Result 4

The Insertion Algorithm 2 finishes.

Proof 4

1. The known schedule S contains a finite number n of polytasks, and then a finite number of idle periods between the subtasks of these tasks. So the set of possible starting instants for the new polytask in S is finite.
2. The random polytask has a finite number of subtasks.

As a consequence the number of iterations of the algorithm is finite. This completes the proof □

4.4 Example

We consider a schedule of five tasks t_i, described in table 3. *IP* stands for Idle Period. These events are given chronologically.

Table 3: Example schedule 2.

Task Id	t_1^1	*IP*	t_2^1	*IP*	t_1^2	*IP*	t_2^2	t_5^1	*IP*	t_3^1
Execution time	5.0	0.1	2.2	0.2	3.0	0.8	2.0	1.0	2.1	2.0
Task Id	*IP*	t_4^1	*IP*	t_3^2	*IP*	t_5^2	t_4^2	t_5^3	*IP*	t_1^3
Execution time	2.5	1.0	3.0	3.0	2.0	1.0	2.0	1.3	5.0	2.5

We want to insert a new polytask, that has three subtasks, in the known schedule. The first subtask of this polytask has a duration of 1.0. The second subtask starts 4 units of time after the end of the first one, and lasts 2 units of time. The third subtask starts 2.2 units of time after after the end of the second one and lasts 0.1. The deadline D_* is fixed at time 26.

We compute SFG(S), and we obtain the graph in figure 7.

We then compute the set of possible insertion points. The first one is between task t_1 and task t_2. The idle period is 0.1, which is not enough for inserting a task that lasts 1.0. We run the result 3, and know that the maximal postponement

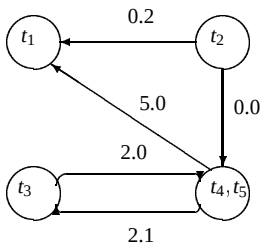

Figure 7: Original SFG(S) for example 2.

we may apply to t_2 is 0.3. It is then not possible to set the first subtask here, neither that it is possibble in the second idle time for the same reason. The third idle period has a duration of 0.8, and the maximal postponement we may apply (0.2) allows us to insert the first subtask here. We update SFG(S), and we find one negative arc. We have to correct it by adding 0.2 to the incoming arcs of node t_4, t_5 and to remove 0.2 of the outcoming arcs of node t_4, t_5. The new SFG(S) is given in figure 8.

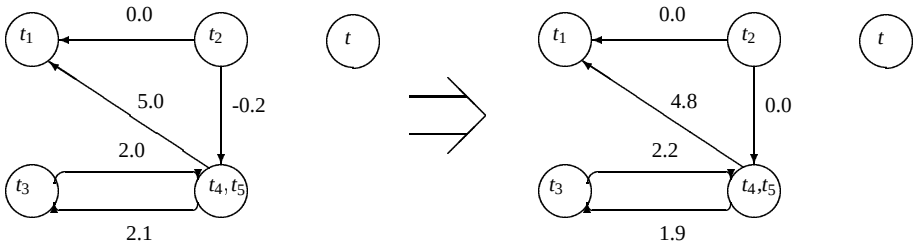

Figure 8: Corrections to SFG(S).

The second subtask has to be inserted 4.0 units of time after completion of the first one. It enters in conflict with task t_3, and requires 1.1 extra units of time. The result 3 indicates that it is possible to postpone t_3 by 2.2 units of time. SFG(S) modifications are given in figure 9

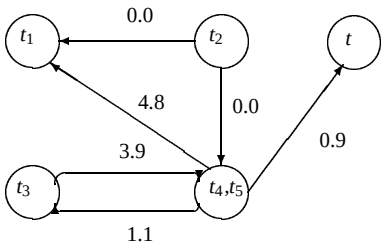

Figure 9: Corrections to SFG(S).

The final subtask of the new polytask has to be placed 2.2 units of time after completion of the second subtask. It lasts 0.1 units of time, and so it may be inserted in the schedule without postponing any other tasks.

5 Complexity

This section is devoted to the computation of the complexity of IA1 and IA2.

5.1 Complexity of IA1

In this subsection, we will assume that the average number of subtasks of a polytask is m.
The average complexity of the initialization part of this algorithm (i.e. Step 1.) is $(1+m)+m = 1+2m$ operations.

The second step starts with a test whose average complexity is : $\frac{\sum_{i=1}^{n} m + m - 1}{2} = \frac{(n+1)m-1}{2}$ The inner part of step 2 requires m operations, and has to be performed $(n-1)m$ times.

The total complexity is then in $O(\frac{m^2n^2}{2})$.

5.2 Complexity of IA2

The complexity of the Dijkstra algorithm is in $O(n * \log(\frac{m}{n}))$ (see [6]), where n is the number of nodes, and m the number of arcs.

The complexity of the SFG algorithm is computed from :

- Step 1 : 4 operations.
- Step 2 : $\sum_{k=1}^{n} (\frac{m}{n} * (n-k) + 3mn + n) = n + \frac{(7n-1)m}{2}$
- Step 3 : $3m$ operations per couple of nodes to merge.

The complexity of the SFG algorithm is then in $O(nm)$

The insertion algorithm 2 complexity is computed as follows :

- Step 1 : at most $n-1$ iterations.
- step 2 : its first execution costs $nm + nln(\frac{m}{n}) + 3m + 1$, and all the others $ln(\frac{m}{n}) + 3m + 1$

So, the total complexity is $((n-1)\sum_{i=1}^{m}) * (nln(\frac{m}{n}) + 3m + 1) + nm + nln(\frac{m}{n}) + 3m + 1$

which is in $O(mn(n\ln(\frac{m}{n}) + m))$

sectionConclusion

In this article, we provide two algorithms. The first one allows to insert polytasks with unitary subtask duration in a known schedule of polytasks. This algorithm may be easily extended to the case of subtask that has integer duration.

The second algorithm considers the case of subtasks with non-integer duration.

These algorithms are a first approach, because they just provide a possible insertion, if any, without optimization of a criterion. The complexity of the algorithms does not allow to use them in a real-time environment.

These two points will be the subject of further work.

References

[1] Milevic D.J. Popovic B.M. Improved algorithms for the interleaving of radar pulses. *IEE Proceedings F 139, pp98-104*, 1992.

[2] Proth J.-M. Chu C. *L'ordonnancement et ses applications.* Masson, 1996.

[3] Proth J.-M. Duron C. Multifunction radar : Task scheduling. *Journal of Mathematical Modelling and Algorithms*, 1, num. 2, pp 105-116, ISSN 1570-1166, 2002.

[4] Proth J.-M. Duron C. Insertion of a random bitask in a schedule: a real-time approach. INRIA Research report 4193, Nov. 2001.

[5] Wardi Y. Duron C., Proth J.-M. Insertion of a random task in a schedule : a real-time approach. *IEEE-ETFA Proceedings, Antibes-Juan les Pins, France*, Vol. 1 pp 559-565, ISBN 0-7803-7241-7, and Inria Research Report 4337, 2001.

[6] Dijskstra E. A note on two problems in connexion with graphs. *Num. Maths.*, 1, pp 269-271, 1959.

[7] 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.

[8] Brucker P. *Scheduling Algorithms*. Springer, 2001.

[9] Shapiro R.D. Scheduling coupled tasks. *Naval. Res. Logist. Quart.*, 27, pp 477-481, 1980.

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