Generalized scheduling on a single machine in a real-time systems based on time value functions

Paul Muhlethaler, Ken Chen

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992 25ème anniversaire

N° 1759

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

GENERALIZED SCHEDULING ON A SINGLE PROCESSOR IN REAL-TIME SYSTEMS BASED ON TIME VALUE FUNCTIONS

Paul MUHLETHALER
Ken CHEN

Septembre 1992

RR_1759

Generalized Scheduling on a Single Processor in Real-Time Systems Based on Time Value Functions

Paul MUHLETHALER* and Ken CHEN†

Abstract- Time Value Functions is a recent concept for the description of real-time task timing constraints. In such systems, a Time Value Function (TVF) is completed to every task. The value of this function taken at time t gives the award that the system receives if the corresponding task is achieved by this time. In this paper, we investigate the general scheduling problem which consists in maximizing the sum of the TVFs evaluated at the completion time of the corresponding tasks. Previous studies envision only regular processors (non-idling between tasks). In this paper, we generalize our general approach by allowing for idle intervals between tasks. Except in special cases, this new degree of freedom is likely to improve the global value criterion. For this NP-hard problem, our aim is to find efficient heuristics. First we describe an exact algorithm to solve the problem and we analyze its complexity. Then we define the optimal decomposition : the set of those tasks to be scheduled is divided into a ranked collection of subsets. To achieve optimality, the tasks of a lower rank subset are to be scheduled prior to those of a higher rank. We also introduce polynomial scheduling algorithms which provide sequences matching this optimal decomposition. From a practical point of view, simulation results have shown that these algorithms yield sequences which provide global values close to the optimum.
Keywords: Real-time systems, Time Value Function, Single machine scheduling, Optimization, Decomposition.

Ordonnancement Généralisé Monoprocesseur Pour Des Systèmes Temps Réel Utilisant Des Fonctions De Valeur

Résumé–Les fonctions de valeur sont un nouveau concept pour la description de systèmes temps réel. Dans de tels systèmes une fonction de valeur est associé à chaque tâche. La valeur de cette fonction prise à l'instant t donne la récompense que le système reçoit si la tâche correspondante est terminée à cet instant. Dans ce papier nous étudions le problème général qui consiste à maximiser la somme des fonctions de valeur évaluées à l'instant de terminaison des tâches correspondantes. Alors que les études précédentes n'envisageaient ce problème que dans le cas ou il n'y a pas d'interruption possible entre deux tâches consécutives, notre approche général permet d'inclure des intervalles entre les tâches. Excepté dans des cas particuliers, ce nouveau degré de liberté est succeptible d'améliorer le critère général obtenu. Pour ce problème NP difficile, notre but est de trouver des heuristiques efficaces. D'abord nous décrivons un algorithme exact pour résoudre ce problème et nous analysons sa complexité. Ensuite nous définissons une décomposition optimale : l'ensemble des tâches à ordonnancer est divisé en une collection ordonnée de sous-ensembles. Pour obtenir le critère optimal, les tâches d'un sous-ensemble d'indice inférieur doivent être ordonnancées avant celles d'un sous-ensemble d'indice supérieur.Nous introduisons aussi des algorithme d'ordonnancement polynomiaux qui fournissent des séquences respectant la décomposition optimale. D'un point de vue pratique, des simulations montrent que les algorithnmes proposés fournissent des critères proches de l'optimale.

Mots-clefs: Système temps réel, Fonction de valeur, Ordonnancement mono serveur, Optimisation , Décomposition.

*Projet REFLECS, INRIA, B.P. 105; 78153 Le Chesnay Cedex; France; E-mail: pmu@reflecs.inria.fr
†Network Department, ENST PARIS; 46, rue Barrault; 75634 Paris cedex 13; France; E-mail: chen@res.enst.fr

1 Introduction

Real-time systems are those in which the contribution of a task depends on the time at which this task is finished [LeL83, Sta88]. Thus, the performance of a real-time system deeply depends on its scheduling policy. This paper deals with the problem of task scheduling in real-time systems. Actually task scheduling with deadline has been a popular subject in the past years. In many situations, people only require that tasks to be done prior to a particular instant. A lot of papers investigate scheduling under deadline constraints (*e.g.* [CMM67, Cof76, Sah76]).

Indeed, if deadline does reflect the limited lifetime of a task, it does also implicitly yield a binary vision of the task: (*alive, dead*). But, this binary vision certainly makes the deadline approach fail to describe many other real-time systems for which the tasks' behaviors are not simply binary but time-varying. It can be the case in weapon systems or railway transportation. Some recent experimental systems try to include the concept of temporal contribution. For example, the *Alpha* system, a real-time Unix-like operating system, characterize tasks by a combination of some simple form functions, [JLT85, Nor88, TWW87, Wen88].

A general way to characterize a task's contribution is probably to describe it as a temporal function, the so called *Time Value Function* (TVF). From this point of view, a natural performance criterion is to maximize the sum of the Time Value Functions evaluated at the completion of the tasks. This leads to an optimization problem. which is NP-hard.

In this paper we consider the general problem of scheduling n tasks and v idle intervals. A Time Value Functions is associated at each task. The idle intervals which can be intercalated between the tasks. We formalize the problem and give the notations in section 2. In section 3 we describe an exact scheduling algorithm which can solve the maximization problem with a complexity $O((v+1)n2^n)$, a complexity far less than the abrupt try of all the $(n+v)!$ possibilities. In section 4 we have defined an optimal decomposition which is a partition of the initial set of tasks into separate subsets for which an optimal scheduling ranking is established, *i.e.* , to achieve an optimal sequencing tasks of a lower rank subset are to be scheduled prior tasks belonging to a subset of a higher rank. Eventually between tasks (of the same or of different subsets) idle intervals may be intercalated. Then, in section 5 we introduce a family of scheduling heuristics which yield sequences compatible with the optimal decomposition. In section 6 we present results of simulation which are very encouraging. Our heuristics always find a scheduling which produces a high percentage of the optimum criterion (more than 80%).

2 Problem formalization and notations

First, let us formalize the problem as follows. Let $\mathcal{T}$ be the set of n independent tasks, numbered from 1 to n. Task i is characterized by its Time Value Function $F_i()$ and its processing time p_i. We consider an off-line and a single machine scheduling problem, *i.e.* all n tasks are available at time 0 and have the same priority. From this point of view, a natural performance criterion is to maximize the sum of obtained values. The problem is to find a sequence $(i_1, i_2, \ldots, i_n)$ out of all possible permutations, which maximizes the sum:

$$\sum_{k=1}^{n} f_{i_k}(t_{i_k}) \quad \text{where} \quad t_{i_k} = \sum_{j=1}^{k} p_{i_j}.$$

This problem is the frame work of our previous works [ChMu91],[MuCh92]. But we can see that under given circumstances it could increase the criterion to anneal the constraint $t_{i_k} = \sum_{j=1}^{k} p_{i_j}$ for a weaker constraint which is that a task can not be executed since the task prior in the scheduling is not finished. That allows idle intervals between tasks. For example we can see that unless all the TVFs are non increasing, we can expect to increase the criterion with this new degree of freedom. Thus we consider v idle intervals numbered from $n+1$ until $n+v$ and of the same duration that we will note for notation consistency $p_k \equiv cste, k \in \{n+1, \ldots, n+v\}$. For the same reason we introduce the functions $F_k() \equiv 0, k \in \{n+1, \ldots, n+v\}$ which are fictive TVF for the idle intervals. More than one interval may separate two

consecutive tasks. Let $\mathcal{I}$be the set of the v idle intervals numbered from $n+1$ until $n+v$, and $\mathcal{D}$=$\mathcal{I} \cup \mathcal{T}$. Let $\mathcal{S}$ be the set of all the possible sequences. For $\sigma \in \mathcal{S}$, $\sigma(i)$ denotes the number of the task or the number of the interval occupying the i^{th} place in the sequence σ. We are going to deal with the problem which consists in the maximization of the sum of the respective value of the n Time Value Functions evaluated at the completion time of the corresponding task. We can write it :

$$\text{MAXIMIZE}_{\sigma \in \mathcal{S}}\ V(\sigma) \ \text{ with } \ V(\sigma) = \sum_{i=1}^{n+v} F_{\sigma(i)}(t_i) \ \text{ where } \ t_i = \sum_{j=1}^{i} p_{\sigma(j)}$$

We note :

$$C(\sigma) = C(\{1, \ldots, n, n+1, \ldots, n+v\}) = Sup_{\sigma \in \mathcal{S}} V(\sigma)$$

This leads to an optimization problem which is NP-hard. This problem is challenging since it is a very general approach to the scheduling problem on a single machine

Next, we will begin by define some definitions and notations which will be used in the rest of the paper. We recall the classical definition of *partition*: a task set $\mathcal{T}$ is *partitioned* into m subsets $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$, if $\mathcal{T} = \cup_{i=1}^{m} \mathcal{T}_i$ with $\mathcal{T}_i \neq \emptyset$ and $i \neq j \Longrightarrow \mathcal{T}_i \cap \mathcal{T}_j = \emptyset$. The number $Card(\mathcal{T}_i)$, $i = 1, \ldots, m$, will be denoted by n_i. $\tilde{\mathcal{T}}_i$ will be by definition $\mathcal{T}_i \cup J$ where J is some subset of $\mathcal{I}$, thus we have $\mathcal{T}_i = \mathcal{I} \cap \tilde{\mathcal{T}}_i$. Moreover $\mathcal{D} = \cup_{i=1}^{m} \tilde{\mathcal{T}}_i$ and $i \neq j \Longrightarrow \tilde{\mathcal{T}}_i \cap \tilde{\mathcal{T}}_j = \emptyset$. The number $Card(\tilde{\mathcal{T}}_i)$, $i = 1, \ldots, m$, will be denoted by $\tilde{n}_i$.

For a given partition $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$, we say that a sequence σ is of the form $\sigma = \tilde{\sigma}_1 \tilde{\sigma}_2 \ldots, \tilde{\sigma}_m$, where $\tilde{\sigma}_i$ is a sequence of the subset $\tilde{\mathcal{T}}_i$, if

$$\forall l \in \{1, \ldots, m\}, \ \forall j \in \{1, \ldots, n_l\}, \quad \tilde{\sigma}_l(j) = \tilde{\sigma}(d+j) \text{ with } d = \sum_{k=1}^{l-1} \tilde{n}_k$$

A partial sequence $\tilde{\sigma}_i$ relative to a subset $\tilde{\mathcal{T}}_i$ is said to be *locally optimal* if, with the scheduling of tasks of all the other subsets remaining unchanged, $\tilde{\sigma}_i$ makes the minimal value of $V(\sigma)$ ($\sigma = \tilde{\sigma}_1 \ldots \tilde{\sigma}_i \ldots \tilde{\sigma}_m$) among all the sequences of $\tilde{\mathcal{T}}_i$.

For technical reasons we will sometimes assume that $p_i = l_i T, l_i \in \mathbb{N}, t_{tot} = \sum_{k=1}^{n+v} p_k = K(n+v)T$ and no larger T satisfies the same condition. We call that hypothesis : *assumption 1*. With this assumption the scheduling times are on a grid of scale T. Moreover to simply notations we will suppose that the duration of intervals is T therefore $l_i = 1$ $i \in \{n+1, \ldots, n+v\}$.

3 Optimal solutions

3.1 First results

If we consider time intervals as fictive tasks, the resolution of the problem requires that we try all the permutations of $\{1, 2, \ldots, n+v\}$ to find the best schedule. There are $(n+v)!$ such permutations. That is really to much to allow any computation in real cases. Anyhow, we can see in [HK62] that an algorithm based on dynamic programming can solve it with a complexity $(n+v)2^{(n+v)}$. As a matter of fact, let S be a subset of $\{1, 2, \ldots, n+v\}$, we have the following result :

$$if \quad n(S) = 1 \quad S = \{l\} \qquad C(S) = F_l(p_l)$$

$$n(S) \leq 1 \qquad C(S) = \max_{l \in S} \left(C(S - \{l\}) + F_l(\sum_{i \in S} p_l) \right).$$

We can very easily justify these equations. The optimal scheduling of S has to be finished by a task or an interval l. The additional reward is then $F_l(\sum_{i \in S} p_i)$ and we must add this reward to the best scheduling

of $S - \{l\}$ which is $C(S - \{l\})$. Therefore we can see that we must compute the best scheduling for all the subsets of $\{1, 2, \ldots, n+v\}$ which are 2^{n+v} and the obtained complexity is then $(n+v)2^{n+v}$. In fact, we will show in the following that a dynamic programming algorithm of complexity $O(v(n+1)2^n)$ can solve the problem.

3.2 An improved algorithm

We can notice that a subset of $\{1, 2, \ldots, n+v\}$ can be decomposed in a subset of $\{1, 2, \ldots, n\}$ and a subset of $\{n+1, n+2, \ldots, n+v\}$. To define a subset of $\{1, 2, \ldots, n+v\}$ we must precisely know all the tasks concerned BUT we only have to know the number of idle intervals which are selected in $\{n+1, n+2, \ldots, n+v\}$. Let us define $C(A, j)$ as the reward for the best scheduling for $A \subset \mathcal{T}$ and $j \leq v$ idle intervals. We can compute the reward $C(A, j)$ of i given tasks and j time intervals $i \leq n$, $j \leq v$, step by step with the following formula:

$$C(A,j) = \max\left(\max_{l \in A} C(A - \{l\}, j) + F_l(t_{A,j}) \, , C(A, j-1) + F_{n+j}(t_{A,j})\right)$$

where :

$$t_{A,j} = \sum_{k \in A} p_k + \sum_{k=n+1}^{n+v} p_k \text{ and } C(\emptyset, 0) = 0.$$

In fact since there is no contribution of the idle intervals the formula is equivalent to :

$$C(A,j) = \max\left(\max_{l \in A}(C(A - \{l\}, j) + F_l(t_{A,j})), C(A, j-1)\right)$$

where :

$$t_{A,j} = \sum_{k \in A} p_k + \sum_{k=n+1}^{n+v} p_k \text{ and } C(\emptyset, 0) = 0.$$

The main result is that to solve our problem we have to solve all these problems with all the subsets of $\mathcal{T}$ containing $i \leq n$ tasks and with $j \leq v$ intervals. The result of our problem is given by $C(\{1, 2, \ldots, n\}, v)$. Thus to achieve the knowledge for all the subset of $\mathcal{T}$ containing k elements and a given number of intervals the number of computation is : $\binom{k}{n}(k+1)$. So the complexity is :

$$\sum_{q=1}^{v} \sum_{k=0}^{n} \binom{k}{n}(k+1) = (v+1)2^n$$

Moreover at each step when we increase the number of tasks concerned we must memorize the last task in the scheduling. We need these data when we reach $C(\{1, 2, \ldots, n\}, v)$ to find the best scheduling. But that does not change the complexity of the problem which is thus :$(v+1)n2^n$

4 Analysis and Decomposition

4.1 Task interchange

Consider two sequences $\sigma_1 = \Pi_1 ij \Pi_2$ and $\sigma_2 = \Pi_1 ji \Pi_2$, where Π_1 and Π_2 are two sub-sequences of $\mathcal{D} - \{i, j\}$. Let $t = \sum_{k \in \Pi_1} p_k$, t is the instant at which tasks or intervals i or j are to be scheduled. Let $\Delta_{ij}(t) = V(\sigma_1) - V(\sigma_2)$, we have

$$\begin{aligned} \Delta_{ij}(t) &= F_i(t + p_i) + F_j(t + p_i + p_j) \\ &\quad - F_j(t + p_j) - F_i(t + p_i + p_j). \end{aligned}$$

As we have a maximization problem, σ_1 is selected instead of σ_2 if and only if $\Delta_{ij}(t) > 0$, then we say that *i precedes j* at t, and denote this fact by $i \prec j$: $i \prec j$ at $t \iff \Delta_{ij}(t) > 0$. In fact we must introduce a more sophisticated definition. Consider two sequences $\sigma_3 = \Pi_1 i (id)^k j \Pi_2$ and $\sigma_4 = \Pi_1 j (id)^k i \Pi_2$, where Π_1 and Π_2 are two sub-sequences of $\mathcal{T} - \{i,j\} \cup \mathcal{I}_{v-k}$, where $\mathcal{I}_{v-k}$ is a set of $v-k$ intervals. The previous notations mean that tasks i and j are separated by k intervals in σ_3 and σ_4. We still consider $t = \sum_{k \in \Pi_1} p_k$, let $\Delta_{ij}^k(t) = V(\sigma_3) - V(\sigma_4)$, we have

$$\begin{aligned} \Delta_{ij}^k(t) &= F_i(t+p_i) + F_j(t+k+p_i+p_j) \\ &\quad -F_j(t+p_j) - F_i(t+k+p_i+p_j). \end{aligned}$$

As we have a maximization problem, σ_3 is selected instead of σ_4 if and only if $\Delta_{ij}^k(t) > 0$, then we say that *i precedes j* at t with order k, and denote this fact by $i \prec^k j$; $i \prec^k j$ at $t \iff \Delta_{ij}^k(t) > 0$.

4.2 Sufficient optimality conditions

Now, we give a set of sufficient conditions for an optimal sequencing:

Proposition 1 *The optimal sequence is* $1, 2, \ldots, n+v$, *if for any* $i < j$ *we have:*

$$\Delta_{ij}(t) \leq 0; \qquad \textit{where } 0 \leq t \leq \sum_{k=1}^{n+v} p_k. \tag{1}$$

Proof:

This trivial result can be proved by iteration from $i = 1$. Details are omitted. ∎

Of course, this set of conditions is very hard to be fulfilled. Nevertheless, for some interesting cases, it leads to some simple scheduling policy.

For example, if all the tasks have the same TVF $F()$, *i.e.* $F_i() = F()$ for $i = 1, \ldots, n$, then it is easy to see that:

- if $F()$ is monotonous decreasing the optimal ordering implies the real tasks are first scheduled according to the SPT (Smallest Processing Time first) rule and then come the v idle tasks.
- similarly, for a monotonous increasing $F()$, the idle tasks are scheduled first, the real tasks are then scheduled according to the LPT (Largest Processing Time first) rule.

If all the tasks share two TVFs $F()$ monotonous increasing and $G()$ monotonous decreasing, then it is also easy to see that:

- the optimal scheduling starts with the tasks whose TVF is $G()$ ranked according to the SPT rule ,then come the v intervals and at last the tasks whose TVF is $F()$ are scheduled according to the LPT rule.

4.3 Precedence relations

The optimal condition we deduced has limited applications. Now, let's return to $\Delta_{ij}()$, in order to investigate the temporal behavior of the precedence relations. Let $t_{tot} := \sum_{j=1}^{n+v} p_j$, t_{tot} is the time required for the processing of all the n tasks and the durations of the v intervals. Feasible scheduling of the couple (i,j) may only occur in the time interval $[0, t_{tot} - p_i - p_j]$. More generally, if a feasible scheduling can only occur between t_b and t_e, the interval $[t_b, t_e]$ will be called the *scheduling scope* of the tasks i and j. Initially, $t_b = 0$ and $t_e = t_{tot} - p_i - p_j$.

If for $t \in [0, t_{tot} - p_i - p_j]$ *then* $\Delta_{ij}(t) > 0$ (resp. $\Delta_{ij}(t) < 0$) holds for all possible scheduling starting instant $t > 0$, then we get always $i \prec j$ (resp. $j \prec i$). we call it an *a strong precedence* at time 0.

On the contrary, the scheduling scope, may be divided into sub-intervals, in which we have successively $i \prec j$ and $j \prec i$, *i.e.* the precedence depends on the time interval in which the current scheduling starting time is located. This leads to the definition of *strong* and *weak* precedences at a given time t:

- if $\forall\ t_1 \geq t$, we have $i \prec j$ at t_1, we say $i \prec j$ *strongly* at t and we denote it by $i\,! \prec j$ at t;
- if $i \prec j$ at t, but $\exists t_1 > t$ such that $j \prec i$ at t_1, we say $i \prec j$ *weakly* at t.

A strong precedence will be preserved when the scheduling time advances, whereas a weak precedence will be eventually inversed.

In fact we need a more general definition since intervals may separate tasks. We must be able to compare the exchange in the scheduling of two tasks separated by idle intervals. If for $t \in [0, t_{tot} - p_i - p_j]$, we have $i \prec^k j$ (resp. $j \prec^k i$) for all possible scheduling starting instant $t_1 \geq t$, we denote that $i\,! \prec^k j$ at t (resp. $j\,! \prec^k i$ at t), If for all $k \leq v$ we have $i\,! \prec^k j$ at t we say that i uniformly precedes j, we denote that by $i \preceq j$ at time t.

Remark: In fact it is possible to use a weaker definition of uniform precedence of order k. The uniform precedence at scheduling points that we still denote by $i \preceq j$ at t_1 means that $\forall\ t_1 \geq t$ and t_1 is a scheduling point, we have $i \preceq j$ at t_1. To simplify the following theorems and demonstrations, we will use the strong precedence although one can easily establish the same results assuming only uniform precedence at scheduling points. The uniform precedence cannot be check numerically whereas it is possible to check uniform precedence at scheduling points. To do it we have to compute : $\Delta_{ij}^k(t)$ where $0 \leq t \leq t_{tot}$, $0 \leq k \leq v$ and t is a scheduling point. We can see than we have at most $2^{n+v-1} - 1$ such points. In this case the required computations are not polynomial. To cope with this problem we can use *assumption 1* and introduce :

$$\begin{aligned} \Delta_{ij}^k(t) &= F_i((l + l_i)T) + F_j((l + k + l_i + l_j)T) \\ &\quad -F_j((l + l_j)T) - F_i((l + k + l_i + l_j)T), \end{aligned}$$

where l is such that $\sum_{k \in \Pi_1} p_k = lT$. The function $\Delta_{ij}^k()$ is completely define by the numbers :

$$\Delta_{ij}^k(p) \qquad \text{where } 0 \leq p \leq \sum_{k=1}^{n} l_k + v.$$

To assess a uniform precedence between couple of tasks the complexity is less than $O(Kv(n + v))$.

4.4 Decomposition

We give conditions on a partition $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$ of $\mathcal{T}$, such that every optimal scheduling order is of the form $\tilde{\mathcal{T}}_1 \tilde{\mathcal{T}}_2 \ldots \tilde{\mathcal{T}}_m$. That means that the partition $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$ indicates the order on the tasks but intervals may be intercalated between them. Here, we present a theorem concerning this decomposition:

Proposition 2 *If there exists a partition* $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$ *of* $\mathcal{T}$, *such that*

$$\forall l \in \{1, \ldots, m-1\},\ \ \forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\},\ \ i \preceq j \ \ at\ \ t = \sum_{r=1}^{l-1} \sum_{k \in \mathcal{T}_r} p_k$$

Then all the optimal sequences σ *are of the type* $\tilde{\mathcal{T}}_1 \tilde{\mathcal{T}}_2 \ldots \tilde{\mathcal{T}}_m$ *where* $\tilde{\mathcal{T}}_i$ *contains* $\mathcal{T}_i$ *and eventually idle intervals. The previous notation means that* σ *first schedules the tasks or intervals of* $\tilde{\mathcal{T}}_1$ *and then come the tasks or intervals of* $\tilde{\mathcal{T}}_2$ *and so on.*

Proof: Let us assume the contrary and consider an optimal scheduling which is now fixed for the whole proof. We denote by $\{\mathcal{T}'_i\}_{i\in\{1,\ldots,m\}}$ the partition obtained by successively picking up the tasks from the optimal scheduling tasks according to their rank in the scheduling and by building this way subsets of the same cardinal than the partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ (thus $Card(\mathcal{T}_i) = Card(\mathcal{T}'_i)$. If $\mathcal{T}_1 \neq \mathcal{T}'_1$ then at least a task a of $\mathcal{T}_1$is in another subset $\mathcal{T}'_i$. This means that in our optimal scheduling at least a task of $\mathcal{T}_1$ is preceded by a task b of another subset $\mathcal{T}_j$ with $j > 1$. But since $a \preceq b$ at time 0 we increase the criterion if we exchange the two tasks. As a matter of fact the uniform precedence ensures that at time 0 task a must precede task b whatever can be the number of intervals separating the two tasks. If $\mathcal{T}_1 = \mathcal{T}'_1$ then we can go to $\mathcal{T}_2$ and we can finish by induction since now we have the same property for $\mathcal{T}_2$ at time $t = \sum_{k\in\mathcal{T}_1} p_k$. We are in the same case as $\tilde{\mathcal{T}}_1$ with $t = 0$. ∎

The following proposition is a straightforward application of the previous proposition.

Proposition 3 *If there exists a partition* $\{\tilde{\mathcal{T}}_i\}_{i\in\{1,\ldots,m\}}$ *of* $\mathcal{D}$, *such that*

$$\forall l \in \{1,\ldots,m-1\},\ \ \forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\},\ \ i \preceq j \ \ at\ \ t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k$$

$$with\ \ \forall l \in \{1,\ldots,m\}\ \mathcal{T}_i = \mathcal{T} \cap \tilde{\mathcal{T}}_i$$

then all the optimal sequences σ *are of the type* $\tilde{\mathcal{T}}_1\tilde{\mathcal{T}}_2\ldots\tilde{\mathcal{T}}_m$ *where* $\tilde{\mathcal{T}}_i$ *contains* $\mathcal{T}_i$ *and eventually idle intervals. The previous notation means that* σ *first schedules the tasks or intervals of* $\tilde{\mathcal{T}}_1$ *and then come the tasks or intervals of* $\tilde{\mathcal{T}}_2$ *and so on.*

We have to be careful about the times at which we have to consider the uniform precedences. For example the following proposition is **not correct**.

Proposition 4 *If there exists a partition* $\{\tilde{\mathcal{T}}_i\}_{i\in\{1,\ldots,m\}}$ *of* $\mathcal{D}$, *such that*

$$\forall l \in \{1,\ldots,m-1\},\ \ \forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\},\ \ i \preceq j \ \ at\ \ t = \sum_{r=1}^{l-1} \sum_{k\in\tilde{\mathcal{T}}_r} p_k$$

$$with\ \ \forall l \in \{1,\ldots,m\}\ \mathcal{T}_i = \mathcal{T} \cap \tilde{\mathcal{T}}_i$$

then all the optimal sequences σ *are of the type* $\tilde{\mathcal{T}}_1\tilde{\mathcal{T}}_2\ldots\tilde{\mathcal{T}}_m$.

As a matter of fact between the end of the tasks of $\mathcal{T}_1$ and the end of the tasks or intervals of $\tilde{\mathcal{T}}_1$ we have no precedence constraint and therefore a task of an other $\mathcal{T}_i$ with $i > 2$ can be introduced without violating the assumptions.

Let us consider the following decomposition where real tasks and intervals are not distinguished.

Proposition 5 *If there exists a partition* $\{\mathcal{D}_i\}_{i\in\{1,\ldots,m\}}$ *of* $\mathcal{D}$, *such that*

$$\forall l \in \{1,\ldots,m-1\},\ \ \forall (i,j) \in \mathcal{D}_l \times \{\cup_{r=l+1}^{m} \mathcal{D}_r\},\ \ i\,! \prec j \ \ at\ \ t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{D}_r} p_k$$

then all the optimal sequences σ *are of the type* $\mathcal{D}_1\mathcal{D}_2\ldots\mathcal{D}_m$.

The proof of this proposition can be obtained in the same way that for proposition 2. Anyhow this decomposition does not occur oftenly since when an interval strongly precedes a given task at time t it means that from this time this task is decreasing.

5 Scheduling algorithms

We introduce two classes of scheduling algorithms which have the nice following property : they match the optimal decomposition. We see further with intensive simulations that these algorithms do yield the sub-optimum scheduling in many cases.

5.1 A first class of algorithms

We have proved that some problems can be optimally decomposed into several smaller problems. In this section, we present polynomial algorithms, which yield a sequence respecting the optimal decomposition.

5.1.1 A General Result

In this section our aim is to present a general algorithm which yields a sequence respecting the optimal decomposition. Let $R(.)$ denote a selection function returning an unique task number (*e.g.* , $R(.) = \max(.)$), $G(t,i)$ a real function from $\mathbb{R} \times \{1,..,n\}$ into $\mathbb{R}$, and σ the algorithmic sequence. The following algorithm selects sequentially the $n+v$ tasks or intervals. At each selection, the chosen task or interval denoted by s maximizes a function evaluated at the completion time of the task or interval scheduled prior. The algorithm uses the local variables t which can be seen as a scheduling time and $\mathcal{D}_r$ which is the set of the unscheduled tasks at time t. The algorithm can be written :

1. *Initialization:* $\mathcal{D}_r := \mathcal{D}$, $t := 0$
2. While $\mathcal{D}_r \neq \emptyset$ Do
   $s := R(\{i \ / \ i \in \mathcal{D}_r \text{ and } G(t,i) = max\{G(t,j) \ / \ j \in \mathcal{D}_r\}\})$;
   $\mathcal{D}_r := \mathcal{D}_r - \{s\}$;
   $\sigma(n + v - Card(\mathcal{D}_r)) := s$;
   $t := t + p_s$;
   End While.
3. Output σ as the algorithmic sequence

Proposition 6 *If $G(t,i)$ is such that for every $\mathcal{D}$ and its optimal decomposition $\{\tilde{\mathcal{T}}_i\}_{i\in\{1,\ldots,m\}}$:*

$$\forall l < m, \ \forall (i,j) \in (\mathcal{T}_l, \cup_{r=l+1}^{m} \mathcal{T}_r) \quad \forall \ t \geq t_b(l) \quad G(t,i) > G(t,j)$$

where $\mathcal{T}_i = \tilde{\mathcal{T}}_i \cap \mathcal{T}$ and $t_b(l) = \sum_{j=1}^{l-1} \sum_{k\in\mathcal{T}_j} p_k$, then the algorithmic sequence σ will respect the optimal decomposition, i.e. σ verifies:

$$\forall l \in \{1,\ldots,m\}, \mathcal{T}_l = \{\sigma(i)\}_{i=l_b..l_e} \ \ with \ \ l_b := (\sum_{r=1}^{l-1} \tilde{n}_r) + 1 \ \ and \ \ l_e := l_b + \tilde{n}_l$$

Proof:

Assume the contrary, *i.e.* the set $\mathcal{E} = \{k \ / \ \tilde{\mathcal{T}}_k \neq \{\sigma(i)\}_{(i=k_b..k_e)}\}$ is not empty, then $l = \min \mathcal{E}$ exists. As $\tilde{\mathcal{T}}_l$ and $\{\sigma(i)\}_{i=l_b..l_e}$ have the same size (n_l), necessary, $\exists j \notin \{l_b..l_e\}$ and $\exists i \in \{l_b..l_e\}$ such that $\sigma(j) \in \tilde{\mathcal{T}}_l$ and $\sigma(i) \notin \tilde{\mathcal{T}}_l$.

On the one side, we have necessary, $j > l_e$ and our algorithm is such that

$$\sigma(i) := \max\{r \ / \ r \in \mathcal{R}, \ \text{ and } \ p_r = \max\{G(t,r) \ / \ r \in \mathcal{R}\}\}$$

with $\mathcal{R} = \mathcal{D} - \{\sigma(h)\}_{h=1..i-1}$ and $t = \sum_{h=1}^{i-1} p_{\sigma(h)}$. As $j > l_e \geq i$, we have $\sigma(j) \in \mathcal{R}$, and so $G(t, \sigma(i)) \geq G(t, \sigma(j))$.

On the other side, always due to the condition $l = \min \mathcal{E}$, we have necessarily $\sigma(i) \in \tilde{\mathcal{T}}_r$ with $r > l$. As we have $i \geq l_b$, so $t = \sum_{h=1}^{i-1} p_{\sigma(h)} \geq \sum_{h=1}^{l_b-1} p_{\sigma(h)} = t_b(l)$. Consequently, our hypothesis on $G(t,i)$ implies that $G(t, \sigma(j)) > G(t, \sigma(i))$. We have thus a contradiction which is due to the initial assumption. Consequently, $\mathcal{E}$ must be empty, and that completes the proof. ∎

Remark:

- In this proof, we use only the fact that the selected task has the highest precedence number, no constraint is supposed concerning $R()$, so in case of multiple choices, any one of the tasks having maximum $G(.,.)$ can be returned by $R()$
- If $v = 0$ (no idle interval) the previous algorithm is the same as the one described in [ChMu91]. The decomposition used in this case relies on the notion of strong precedence instead of uniform precedence.

We have the following proposition about the time complexity of this algorithm :

Proposition 7 *If the computation of* $s := \max\{i \ / \ i \in \mathcal{T} \ \text{ and } G(t,i) = \max\{G(t,j) \ / \ j \in \mathcal{D}_r\}\}$ *is of time complexity* $O((n+v)^a)$ *$(n + v = Card(\mathcal{D}))$ the algorithm is of time complexity* $O((n+v)^{(a+1)})$.

Proof:

In each step of the scheduling, let r be the remaining task's number, the computation of $G(t,i)$ requires $O(r^a)$, since we must compare each remaining task i with each other remaining task. The selection of s is $O(r)$. So, each step of scheduling is of complexity $O(r^2)$. As r is reduced by one at each step, the complexity is $O(\sum_{r=1}^{n+v} r^a)$ which is upper bounded by $\int_0^{n+v} (x+1)^a dx \simeq (n+v+1)^{(a+1)}/(a+1)$, consequently, the complexity of the whole loop is $O((n+v)^{(a+1)})$.

∎

5.1.2 Precedence among tasks

Let us denote by $P(t,i)$ the number of real tasks that the task or interval i precedes at time t and by $UP(t,i)$ the number of real tasks that the task or interval i uniformly precedes at time t. We have the obvious properties :

$$\begin{aligned} \forall t \quad & P(t,i) \geq UP(t,i) \\ \forall t_1, t \quad & t_1 \geq t \Longrightarrow UP(t_1,i) \geq UP(t,i) \end{aligned}$$

But $P(t,i)$ is not increasing with t, since the weak precedence is not time-conserving. We have another very interesting result: when an optimal decomposition exists, the decomposition conserves the decreasing $UP(t,i)$ and $P(t,i)$ ordering on the subset level:

Proposition 8 *If the task problem is optimally decomposable in the sense of proposition 2 into* $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$, *with* $m > 1$. *We have, for every* $l \in \{1, \ldots, m-1\}$,

$$\forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\}, \ \forall t \geq t_1, \ P(t,i) > P(t,j) \ \text{ and } \ UP(t,i) > UP(t,j)$$

where $t_1 = \sum_{j=1}^{l-1} \sum_{k \in \mathcal{T}_j} p_k$,

Proof:

On the one hand, as task i has uniform precedence over all the tasks in $\cup_{r=l+1}^{m} \mathcal{T}_r$ at t_1, we have:

$$UP(t_1, i) \geq \sum_{r=l+1}^{m} Card(\mathcal{T}_r)$$

and for all $t > t_1$, $UP(t, i) \geq UP(t_1, i)$.
On the other hand, task j is uniformly preceded by all the tasks in $\{\cup_{r=1}^{l} \mathcal{T}_r\}$ at t_1, we have for all $t \geq t_1$:

$$P(t, j) \leq Card(\mathcal{T} - \{j\}) - \sum_{r=1}^{l} Card(\mathcal{T}_r) \leq \sum_{r=l+1}^{m} Card(\mathcal{T}_r) - 1$$

Consequently, we have $\forall t \geq t_1$, $UP(t, i) > P(t, j)$. Applying the relation $\forall t$ $P(t, i) \geq UP(t, i)$ at both side, we get the proof. ∎

5.1.3 Examples of scheduling algorithms

According to the previous results $G(t, i)$ can be selected among the following functions :

i $G_1(t, i) = P(t, i)$,

ii $G_2(t, i) = UP(t, i)$,

iii $G_3(t, i) = \sum_{j/t_j > t} P(t_j, i)$,

iv $G_4(t, i) = \sum_{j/t_j > t} UP(t_j, i)$.

where the t_j are k fixed points in the scheduling scope.

The function $R()$ used in the algorithm has to select among the tasks i having the maximum $G(t, i)$. For example, we can choose $R(.) = max(.)$ or $R(.) = min(.)$. Any selection function $R()$ can be chosen since it is not taken into account in the proof of proposition 6. We can mix the choice of function $R()$ and $G()$ to get a scheduling algorithm. Proposition 6 provides a set of algorithm. When we have defined a few of such algorithms, we can select the best sequence find by these algorithms. The obtained algorithm is still polynomial.

Remark: We can notice that with the previous fonctions, we obtain sequences compatible with the optimal decomposition in the sense of proposition 5 *i.e* if we do not distinguish between real tasks and intervals and if we consider strong precedences. It has not been shown in the paper but we indicate the steps to find this result. First we show that proposition 6 holds with a decomposition coming from proposition 5. Then we show that if we consider that $P(t, .)$ and $UP(t, .)$ not only for real tasks but also for intervals then the proposition 8 holds with a decomposition coming from proposition 5. The conclusion is then obvious. This result is a way to justify that the intervals are conviently disposed by our algorithms. Another is to invoke a similar treatment for the real tasks and for the intervals.

5.2 A second class algorithms

The algorithms we are going to introduce are close to the quick sort. If we consider a random sequence σ, this sequence may contain bad scheduled couples, *i.e.* for some i, $\sigma(i+1) \prec \sigma(i)$ at $t = \sum_{k=1}^{i-1} p_{\sigma(k)}$. In this first part we look for local optimality by considering indifferently tasks and intervals [MuCh92]. In a second part we only consider real tasks. For a given sequence, we denote by f_i^σ the rank of the ith task scheduled and k_i^σ the number of idle intervals in the scheduling σ between the ith and $i+1$th task scheduled. For two consecutive tasks in σ *i.e* in the scheduling they are only separated by intervals we are looking if we can exchange the two tasks while conserving the idle intervals between. Finally the algorithm can be written as follows :

Initialization: $t = 0$; $i = 1$;
While $i \neq n + v$ Do
IF $\sigma(i+1) \prec \sigma(i)$ at t **THEN** exchange $\sigma(i+1)$ and $\sigma(i)$
$t := t + p_{\sigma(i)}$; $i := i + 1$;
End While

$t = 0$; $i = 1$;
While $i \neq n$ Do
IF $\sigma(f_i^\sigma) \prec^{k_i^\sigma} \sigma(f_{i+1}^\sigma)$ at t **THEN** exchange $\sigma(f_{i+1}^\sigma)$ and $\sigma(f_i^\sigma)$
$t := t + p_{\sigma(i)} + k_i^\sigma$; $i := i + 1$;
End While.

This operation that we will note in the following by *Rer* (*Rer* transforms σ in $Rer(\sigma)$) is very simple and has also very nice properties that we will now describe.

5.2.1 Properties of Rer()

Proposition 9 *The sequence $Rer(\sigma)$ obtained by the reranking of σ is such that $V(Rer(\sigma)) \geq V(\sigma)$.*

Proof: It is obvious since at each step of the reranking the criterion is increased. ■

Proposition 10 *If we apply Rer() recursively, a change is produced in a given sequence σ only a finite number of times.*

Proof: That is the consequence of two facts : a reranking procedure which changes the sequence strictly increases the criterion and there is only a finite number of sequences. Therefore an infinite loop is impossible. ■

Proposition 11 *Let us suppose that we apply recursively the reranking procedure starting with a given sequence σ until the reranking procedure leaves the sequence unchanged. Then the obtained sequence is compatible with the optimal decomposition.*

Proof: Let us assume that the optimal sequences σ are of the type $\tilde{\sigma}_1\ \tilde{\sigma}_2 \ldots \tilde{\sigma}_m$, where σ_i is a local optimal permutation of the subset $\tilde{\mathcal{T}}_i$. The partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$ is such that

$$\forall l \in \{1,\ldots,m-1\}, \forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\},\ \ i \preceq j \ \text{ at } \ t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k.$$

If the obtained sequence by successive reranking procedures σ' is not compatible with the optimal decomposition then we can find in the scheduling two consecutive tasks a and b (only separated by idle intervals) such that

$$(a,b) \in (\mathcal{T}_i \times \mathcal{T}_j) \quad \text{a starts in } \sigma' \text{ after time } \ t = \sum_{r=1}^{j-1} \sum_{k\in\mathcal{T}_r} p_k \text{ and } \ i < j$$

In such a case we know that at the time when task a begins, task b precedes task a since $b \preceq a$ at t. Therefore we find a contradiction with our hypothesis which states that the reranking procedure leaves the sequence unchanged. ■

The following results we will precise the number of iterations necessary to obtain a sequence compatible with the optimal decomposition.

Proposition 12 *If we apply recursively the reranking procedure $\frac{n^2}{2}$ times starting with a given sequence, the obtained sequence is compatible with the optimal decomposition.*

Proof: Let us assume that the optimal sequences σ are of the type $\tilde{\sigma}_1\ \tilde{\sigma}_2 \ldots \tilde{\sigma}_m$, where $\tilde{\sigma}_i$ is a local optimal permutation of the subset $\tilde{\mathcal{T}}_i$. The partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$ is such that

$$\forall l \in \{1,\ldots,m-1\}, \forall (i,j) \in \mathcal{T}_l \times \{\cup_{r=l+1}^{m} \mathcal{T}_r\},\quad i \,!\prec j \ \text{ at } \ t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k.$$

In the following proof we are only considering real tasks, the reasoning simply ignores the idle intervals. Moreover in this proof we are only considering the effect of the second part of the Rer algorithm *ie* the part of the algorithm which only exchanges real tasks. Since all the tasks of the subset $\mathcal{T}_1$ uniformly precede tasks of the other subsets at time 0, we know that if we look the places of the tasks of subset $\mathcal{T}_1$, at each reranking these tasks are moving of one step towards the beginning of the scheduling if these tasks are not correctly scheduled *ie* in the first n_1 places of the sequence (Remind that in this ranking we are only taking into account real tasks). It is easy to see it for a task of $\mathcal{T}_1$ uniformly preceded by a task of $\mathcal{T} - \mathcal{T}_1$ in the initial sequence. If more than one task of $\mathcal{T}_1$ follow each other it is true for the first task of the group in the scheduling. When the reranking will reach this task, this procedure permutes it with a task of $\mathcal{T} - \mathcal{T}_1$. Thus the second task of our subset is now preceded by a task of $\mathcal{T} - \mathcal{T}_1$ as the procedure has not yet reach this task. We can continue this reasoning. Therefore after $n - n_1$ reranking all the tasks of subset $\mathcal{T}_1$ are scheduled accordingly to the optimal decomposition. We can do the same reasoning with the subset $\mathcal{T}_2$, we will see that to schedule it accordingly to the optimal decomposition we need $n - n_1 - n_2$ extra reranking procedures. Therefore after at most $\frac{n^2}{2}$ reranking procedures we are sure that the obtained sequence is compatible with the optimal decomposition. Of course the first part of the Rer algorithm does not exchange tasks scheduled according to the optimal decomposition. ∎

Since in the conditions of the optimal decomposition precedence between tasks are due at a given time, it is not obvious that we have a parallel ordering for the tasks of all the subsets. We precise this point in the following proposition.

Proposition 13 *If we assume that all the tasks have the same duration then we need only to apply recursively the reranking procedure n times starting from a random sequence to obtain a sequence compatible with the optimal decomposition.*

Proof: Again we only consider the second part of the algorithm Rer and the scheduling of the real tasks. As in the previous demonstration we can show that after $n - n_1$ reranking procedures all the tasks of the subset $\mathcal{T}_1$ are at the right place according to the optimal decomposition. But in the same time the tasks of the subset $\mathcal{T}_2$ which are not in the first $n_1 + n_2$ places are moving of one step towards the beginning of the scheduling. This result is true because all the tasks have the same duration; thus a task of $\mathcal{T}_2$ not in the n_1 first positions of the scheduling uniformly precedes a task of $\mathcal{T} - \mathcal{T}_1 - \mathcal{T}_2$ whatever are the first n_1 first tasks in the sequence (their durations are the same).

5.2.2 Examples of algorithms of the second class

After the previous results, we can easily imagine algorithms of the second class. A first algorithm can be :

1. While($i < G$) Do
2. *Initialization:* select at random a sequence S, $l = 1$, $\sigma = S$;
   WHILE($l < \frac{n^2}{2}$ *and* $S \neq \sigma$) Do
   $S = Rer(\sigma)$, $l = l + 1$;
   End WHILE
3. If($V(S) > A$) THEN ($A = V(\sigma)$ and $\sigma = S$);
4. $i = i + 1$;
   End WHILE
5. Output σ as the algorithmic sequence

We can see that this algorithm chooses among G random sequences the one which after being reranked provides the largest $V(\sigma)$. We have selected a number of reranking procedures which assumes that our found sequence respects the optimal decomposition. Obviously we can introduce slight differences in the previous algorithm. For example we can select a random sequence and apply to it the reranking procedure enough times so that our new sequence is compatible with the optimal decomposition. Then we can try slight modification of this sequence followed by reranking procedures (to be compatible with the optimal decomposition) to increase the award $V()$.
We have the following proposition about the time complexity of this algorithm :

Proposition 14 *The algorithm has a complexity less than* $O((n + v) * n^2)$.

Proof:

Each reranking procedure needs $2n + v$ computations, and to reach a sequence compatible with the optimal decomposition we need at most $\frac{n^2}{2}$ successive reranking. Therefore the complete reranking requires at most $O((2n + v)\frac{n^2}{2})$ operations. The reranking procedure is executed G times, thus the algorithm has a complexity less than $O(G(2n + v)\frac{n^2}{2})$. With $G = 1$ we have an algorithm of complexity $O((2n + v)\frac{n^2}{2})$ which produces sequences compatible with the optimal decomposition ∎

Remark: We can notice that with the previous algorithms, we obtain sequences compatible with the optimal decomposition in the sense of proposition 5 *i.e* if we do not distinguish between real tasks and intervals and if we consider strong precedences. This property is due to the first part of the reranking algorithm which involves the real tasks and the intervals. A demonstration of this fact can be found in [MuCh92]. This result is a way to justify that the intervals are conviently disposed by our algorithms. Another is to invoke a similar treatment for the real tasks and for the intervals.

6 Computational experience

6.1 Conditions and parameters

Algorithms are implemented and tested by simulation, in order evaluate their performance. The TVFs associated to tasks are randomly generated.
We have run tests with several kinds of TVFs. We have tried successively the following types:

1. The TVFs are non increasing. (cf. Figure 1).
2. The TVFs are linearly increasing on a first interval and linearly decreasing on a second interval. (cf. Figure 2).

3. The TVFs are quadratic [Che91].

4. The TVFs can be written $F(t) = te^{-at}$ with a randomly selected. (cf. Figure 3)

5. The TVFs are mixed and chosen among the previous types

The first type can be interpreted as functions associated to a soft dead-line. The second, third and fourth type can be associated with the modelization of both a timeliness and a soft dead-line constraint.

To evaluate our algorithms we have programmed the dynamic programming approach which is described in the part 3.2. This algorithm requires much computation space and time, we have to limit our investigation to 15 real tasks and 30 intervals. We will compare the previous algorithms with the exact solution and we will give the obtained percentages of the maximum criterion for each heuristics. Moreover we give this percentage for the random algorithm (RA). This algorithm selects the best sequence among random sequences. The number of sequences investigated is such that the complexity of this algorithm is the same as algorithm 1. In fact since we can add or substract a constant to each TVF without changing the problem and in order to see clearly the difference between the algorithms we have decided to substract from the criterion its mean value for random sequences.

6.2 Results of simulations

For the first type of TVFs our algorithms give good results which have already been presented in [ChMu91] and [MuCh92]. As a matter of fact since the TVFs are non increasing, intervals ranked before the real tasks will yield a value loss. Therefore the idle intervals will all be scheduled after the real tasks and the real tasks are scheduled according to the algorithms described in [ChMu91] and [MuCh92].

For the second type of TVFs and with 10 tasks and 15 intervals, we find the following percentages : 98%;89%;95%;76% respectively for algorithm 1, algorithm 2, the reranking algorithm, the RA, in following all the given % will be given in this order. With 12 tasks and 15 intervals, we find the percentages : 96%;94%;4%;3%. It seems that in that case the ranking of the heuristics should be first algorithm 1, second algorithm 2, third reranking, fourth RA.

For the third type of TVFs the ranking of the heuristics is : first reranking, second algorithm 1, third algorithm 2, fourth RA. For example we find the following percentages : 97%;95%;99%;91% with 12 tasks and 15 intervals. With 12 tasks and 15 intervals, we find the percentages : 95%;94%;99%;92%. The ranking of the heuristics is first reranking, second algorithm 1, third algorithm 2, fourth RA.

For the fourth type of TVFs, we find the following percentages :96%;43%;99%;76% with 10 tasks and 15 intervals. With 12 tasks and 15 intervals we find the percentages :95%;54%;99%;73%. The ranking of the heuristics is here first reranking, second algorithm 1, third RA, fourth algorithm 2.

When we mixed the previous types of TVFs, the ranking of the heuristics is : first reranking, second algorithm 1, third RA, fourth algorithm 2. For example we find the following percentages : 90%;90%;10%;10% with 10 tasks and 15 intervals. With 12 tasks and 15 intervals we find the percentages : 89%;88%;2%;3%.. Here The ranking of the heuristics is here first algorithm 1, second algorithm 2, third RA, fourth reranking.

Concerning the optimal decomposition, we have observed that a decomposition rarely occurs with the second type of TVF and we have always poor decomposition (one large subset and one subset reduced to one or two tasks). On the contrary we often see (in more than 50% of the cases) wide decomposition (many small subsets) with the third, the fourth and even with the fifth type of TVF The previous results must be analyzed very carefully. As a matter of fact it is very difficult to define a random TVF to build a bench mark. Anyhow the previous results show that algorithm 1 behaves better than algorithm 2 besides its complexity is lower. The reranking algorithm has similar performance as algorithm 1, depending on situations the reranking is slightly better or slightly worst than algorithm 1. It is interesting to see that the algorithm 1 and the reranking algorithm provide good results even when the decomposition of the problem is poor or inexistent.

6.3 Generalization

In fact the previous study shows that our algorithms based on a decomposition of the problem and operating on a search of local optimality is not (of course) universal but provides in a lot of cases very nice results especially for scheduling tasks on a single processor. It could be interesting to see if our algorithms can be applied in the context of distributed systems: many processors, tasks share resources with potential conflicts,etc...

7 Conclusion

This study is concerned with with real-time system uniprocessors. Time Value oriented scheduling has received until now less attention than the problem of deadline oriented scheduling. We have formalized the former as an optimization problem, namely the maximization of a sum of time value functions evaluated at the task completion times running of a same machine. Contrary to previous studies, successive tasks on the processor can be separated by idle intervals. For this maximization problem we have developed a set of conditions for an optimal decomposition. Based on these results, sub-optimal polynomial algorithms are proposed, which generate quasi optimal sequences. Computational experience suggests that these algorithms are rather efficient for various scenarios.

References

[CMM67] R.W. Conway, W.L. Maxwell and L.W. Miller, "Theory of scheduling," Addison-welsey, 1967.

[Cof76] E.G. Coffman Jr., "Theory of scheduling," John Wiley, 1976.

[Che91] K. Chen, "A Study on the Timeliness Property in Real-Time Systems," J. Real-Time Systems, No.3, 1991.

[ChMu91] Single Machine Scheduling With Time Value Functions In Real-Time Systems, K. Chen, P Muhlethaler, 10th IFAC, 9-11 September 1991, Sommering, Austria.

[HK62] M. Held and R.M. Karp, "A Dynamic Programming Approach to Sequencing Problems," J. SIAM, V.10, No. 1, Mar. 1962, pp. 196–210.

[JLT85] E.D. Jensen, C.D. Locke and H.Tokuda, "A Time-driven scheduling Model for Real-Time Operating System," IEEE Real-Time Symposium, Dec. 1985, pp. 112–122.

[Knu69] D.E. Knuth, "The Art of Computer Programming, Volume One: Fundamental Algorithms," Addison-Welsey, 1969.

[LeL83] G. Le Lann, "On Real-Time Distributed Computing,", Invited paper, IFIP Congress 83, North Holland Ed., Sept. 1983, pp. 741–753.

[MuCh92] P. Muhlethaler, K. Chen , "Two Classes Of Effective Heuristics For Time Value Functions Based Scheduling," Third International Conference on Future Trends of Computer Communications. Tapei April 1992.

[Nor88] J. D. Northcutt, "The Alpha Operating System: Requirements and Rationale," Archons Project Tech. Rep. No. 88011, 1988.

[Sah76] S. Sahni, "Algorithm for Scheduling Independent Tasks," J. ACM, V.23, No.1, Jan. 1976, pp. 116-127.

[SSL89] B. Sprunt, L. Sha and J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," J. Real-time syst., V-1, 1989, pp. 27–60.

[Sta88] J. A. Stankovic, "Misconceptions about Real-Time Computing, A Serious Problem for Next-generation Systems," IEEE Computer, Oct. 1988, pp. 10–19.

[TWW87] H. Tokuda, J.W. Wendorf and H-Y Wang, "Implementation of a Time-driven Scheduler for real-Time operating systems," IEEE Real-Time Symposium, Dec. 1987, pp. 271–280.

[Wen88] J. W. Wendorf, "Implementation and Evaluation of a Time-driven Scheduling Processor," IEEE Real-Time Symposium, Dec. 1988, pp. 172–180.

Different kinds of TVF experienced

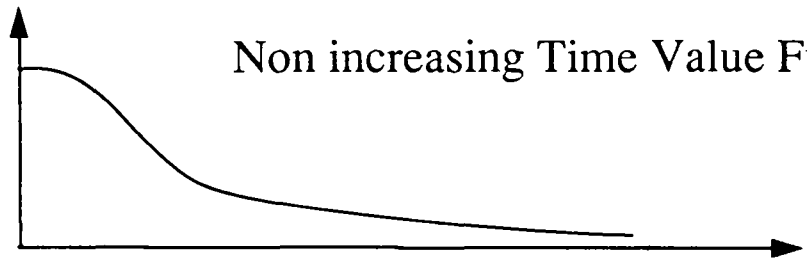
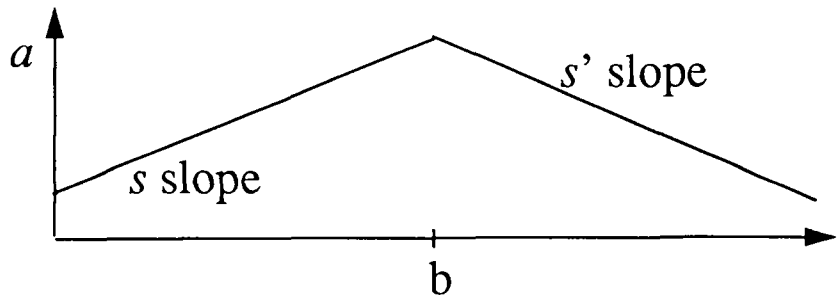


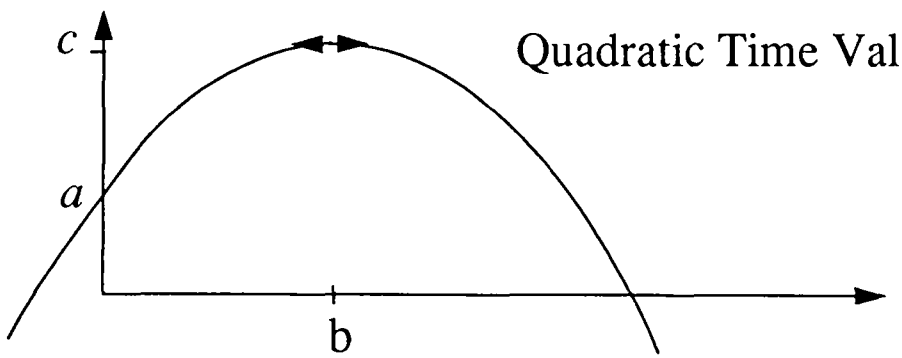
Figure 1

Functions linearly increasing then decreasing after a threshold



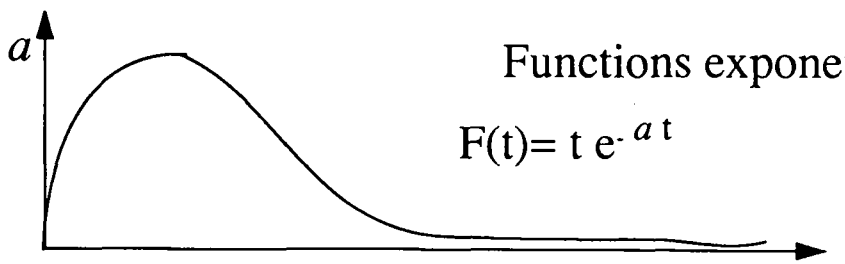
a,*b* and *s* are randomly selected for each TVF

Figure 2



a, b,c are randomly selected for each TVF

Figure 3



a is randomly selected for each TVF

Figure 4

ISSN 0249-6399