HAL
open science

A Family of scheduling algorithms for real-time systems using time value functions

Ken Chen, Paul Mühlethaler

To cite this version:

Ken Chen, Paul Mühlethaler. A Family of scheduling algorithms for real-time systems using time value functions. [Research Report] RR-1530, INRIA. 1991. inria-00075032

HAL Id: inria-00075032

https://inria.hal.science/inria-00075032

Submitted on 24 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.

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Rapports de Recherche

N° 1530

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

A FAMILY OF SCHEDULING ALGORITHMS FOR REAL-TIME SYSTEMS USING TIME VALUE FUNCTIONS

Ken CHEN
Paul MÜHLETHALER

Septembre 1991

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

RR_1530

A Family of Scheduling Algorithms for Real-Time Systems using Time Value Functions

Ken CHEN* and Paul MÜHLETHALER†

Abstract– Real-Time systems using Time-Value Functions (TVF) imply new scheduling problems. In such systems, each task provides at its completion time a contribution which is described by a temporal function. In this paper, we investigate the problem of scheduling a set of tasks under the criterion of maximizing the sum of each task's contribution evaluated at its actual completion time. For this NP-hard problem, our aim is to find efficient heuristics. First on a theoritical point of view, we define the optimal decomposition : the set of the tasks to be scheduled is divided into a ranked collection of subsets. To find an optimum decomposition, 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 respecting this optimal decomposition. On practical point of view, simulation results have shown that these algorithms yield optimal or nearly optimal sequences in many cases and thus are efficient heuristics. Furthermore these algorithms allow a polynomial search of the optimal decomposition. Therefore we can search exhaustively an optimum sequence, with a complexity likely to be far less than the initial problem's complexity depending on the optimal decomposition found.

Keywords: Real-time systems, Time Value Function, Single machine scheduling, Optimization, Decomposition.

Une Famille d'Algorithmes d'Ordonnancement Pour Des Systemes Temps Reel Utilisant Des Fonctions De Valeur

Ken CHEN et Paul MÜHLETHALER

Résumé– Les systèmes Temps-Réel à Fonction de Coût donnent naissance à de nouveaux problèmes d'ordonnancement. Dans de tels systèmes, chaque tâche fournit au moment de sa termination une contribution qui est décrite par une fonction de coût. Nous étudions dans ce rapport comment d'ordonnancer un ensemble de tâches pour maximiser la somme des contributions au temps d'accomplissement des tâches. Pour ce problème NP-difficile, notre but est de trouver des heuristiques efficaces. D'abord d'un point de vue théorique, nous définissons la décomposition optimale : l'ensemble des tâches à ordonnancer est divisé en une collection ordonnée de sous-ensembles. Pour obtenir un ordonnancement optimal, les tâches d'un sous-ensemble d'un indice inférieur doivent être dans l'ordonnancement avant ceux qui appartiennent aux sous-ensembles d'indice supérieur. Nous introduisons également des algorithmes d'ordonnancement qui fournissent des séquences compatibles avec cette décomposition optimale. D'un point de vue pratique, des résultats de simulation montrent que ces algorithmes fournissent une solution optimale pour de très nombreux cas et sont donc des heuristiques. De plus des algorithmes polynomiaux permettent de chercher la décomposition optimale. Donc nous pouvons chercher un optimum de façon exhaustive, avec une complexité qui peut être suivant la décomposition trouvé bien inférieure à celle du problème initial.

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

*Network Department, ENST PARIS; 46, rue Barrault; 75634 Paris cedex 13; France; E-mail: chen@res.enst.fr
This work was supported in part by a Post-doc scholarship of INRIA.
†Projet REFLECS, INRIA, B.P. 105; 78153 Le Chesnay Cedex; France; E-mail: pmu@reflecs.inria.fr

1 Introduction

Task scheduling under deadline constraint has raised up a great success among real-time system designers as well as in the scientific community. Indeed many real-time applications only require tasks to be completed prior to a given time. The deadline constraint, which offers a binary vision of the behavior of a task: (*alive, dead*), is perfectly suitable for these situations. Much results have been obtained for scheduling under deadline constraints (cf. [CMM67], [Cof76], and [Sah76]). Hereafter we give a sample list of results relative to scheduling under deadline constraint. Optimal policies have been found for minimizing the maximum lateness and tardiness or maximizing the minimum lateness and tardiness [Cof76]. Results on weighted deadline and weighted tardiness problems [Cof76, Pos88], or in a multiprocessor environment [ZRS87] has also been presented. The problem of scheduling periodic tasks (which have implicit deadline constraints) has focused a great attention. The *rate monotonic* policy [LiL73] gives an upper bound for the load of the submitted tasks which are assured to be executed in time ; numerous derivated policies for aperiodic tasks have also been developed (*e.g. Sporadic Server* [SSL89])

However, if deadline does reflect the limited lifetime of a task, it does also implicitly yield a binary vision of the task: (*alive, dead*). Things become more complicated when a release-time constraint is also introduced, *i.e.* a task is announced anticipately but is not yet available for processing. In this case, we have a triple-valued task: (*announced, available, dead*). But, one may also introduce the timeliness constraint, *i.e.* a task is preferably to be completed at a particular instant, namely the *optimal completion time*, although it can be completed elsewhere within a given time interval. Then, we have at least a quadruple-valued task: (*announced, available, optimal, available, dead*). But, this modeling may still be insufficient, since the *death* of a task may be abrupt (hard deadline), or continuous (soft deadline). In the later case, we have a multi-valued task.

Indeed, a general way to characterize a task's contribution is probably to describe the latter as a temporal function, the so called *Time Value Function* (TVF). This is precisely the trend in the design of recent real-time systems. For example, the *Alpha* system, a real-time distributed operating system developed at Carnegy Mellon, characterizes tasks by a combination of several simple form temporal functions (cf. [JLT85], [Nor88], [TWW87], [Wen88]). For such a system, a natural performance criterion consists in maximizing the sum of obtained values. Formally, for a set of n tasks, let f_i and p_i be respectively the TVF and processing time associated to the task i, 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 leads to an optimization problem. This problem is NP-hard, since a special case of the dual minimization problem in which each function is of the form $f_i(t) = |t - a_i|$ where a_i is the optimal starting time of the task i, is proved to be NP-hard in [GTW88]. After having made a survey of the problem, the most effective algorithm that we have found is the one proposed in [HK62], which has a complexity of $O(n2^n)$. In [LM69], a number of special instances of this problem have been solved in a polynomial manner.

In this paper we consider the general problem of scheduling with Time Value Functions. This paper is organized around the idea of *decomposition*. We give sufficient conditions to decompose a scheduling problem into several smaller size problems. Then we introduce scheduling algorithms which respect this decomposition and which are effective heuristics to solve our problem. The above decomposition and the scheduling algorithms generalize those introduced in the more restrictive case of tardiness penalty [SPL88] and timeliness constraint [Che91]. In section 2, the scheduling problem is formalized as the the maximization of the sum of n TVFs evaluated at the completion time of tasks. This problem is studied in section 3, where sufficient optimality conditions and conditions for an optimal decomposition are established. An optimal decomposition is a partition of the initial set of tasks into separate subsets for which an optimal scheduling ranking is established, *i.e.* tasks of a higher rank subset are to be scheduled prior to those of a lower rank subset to achieve an optimal sequencing. In section 4, we introduce our

scheduling algorithms and prove that they respect the decomposition sequencing order. We also present a complementary algorithm performing decomposition in section 5. We have proved that these algorithms are generally polynomial. Simulation results reported in section 6 suggest that our algorithms effectively yield optimal or near-optimal sequences for many scenarios.

The Appendix is devoted to technical details : a) a generalization of the optimal decomposition : the sub-optimal decompositions; and b) evaluation of upper bounds of the difference between the optimal criterion and the criterion for sequences respecting the optimal or the sub-optimal decompositions.

2 Problem formalization and notations

We consider a single processor scheduling problem. The processor is assumed to be regular, *i.e.* it continues to work until all tasks are completed, there is neither preemption nor idle time. A set of n independent tasks, numbered from 1 to n, is given. Time will be denoted by t. We assume that all the tasks are available at the beginning time, taken as reference and denoted by $t = 0$. Task i, $i = 1, \ldots, n$, is characterized by its Time Value Function $F_i()$ and its processing time $p_i > 0$.

Let $\mathcal{S}$ be the set of all $(n!)$ permutations of $(1, 2, \ldots, n)$. For $\sigma \in \mathcal{S}$, $\sigma(i)$ denotes the task occupying the ith place in the scheduling σ. We are interested on the following maximization problem:

$$\text{MAXIMIZE}_{\sigma \in \mathcal{S}} \; V(\sigma) \;\text{ with }\; V(\sigma) = \sum_{k=1}^{n} F_{\sigma(k)}(t_k) \;\text{ where }\; t_k = \sum_{i=1}^{k} p_{\sigma(i)}$$

In the past, efforts have been done on the study of penalty functions, most of them are focused on the deadline failure penalty modelization. Linear penalty functions have been investigated in [Wei81, RK76]. As the quadratic penalty sum minimization problem is concerned, a branch-and-bound algorithm has been proposed in [Tow78], which minimizes $\sum_{i=1}^{n} w_{\sigma(i)}(\sum_{j=1}^{i} p_{\sigma(j)})^2$. Improvements of [Tow78] have been proposed in [BK80, GS84]. Recently, the *Just-In-Time* (JIT) problem has also received increasing attentions (see, for example, the paper of Garey *et al.* [GTW88] and the survey paper of Baker *et al.* [BS90]). The JIT problem consists of not only taking into account the tardiness penalty, but also the *earliness* penalty due to the completion of a task prior to its optimal completion time.

The problem that we address in this paper can be viewed as a generalization of the previously mentioned penalty minimization problems, since general form temporal value functions are considered. This problem has received few attentions up to now, as our knowledge is concerned. Next, we will begin by define some definitions and notations which will be used in the rest of the paper.

The set of the n tasks in consideration will be denoted by $\mathcal{T}$. 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.

For the sake of simplicity, if we say that the scheduling order is $\mathcal{T}_1 \mathcal{T}_2 \ldots \mathcal{T}_m$ for a given partition $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$, it means that all the tasks of $\mathcal{T}_i$ are scheduled prior to those of $\mathcal{T}_j$, $j > i$.

For a given partition $\{\mathcal{T}_i\}_{i \in \{1,\ldots,m\}}$, we say that a sequence σ is of the form $\sigma = \sigma_1 \sigma_2 \ldots, \sigma_m$, where σ_i is a sequence of the subset $\mathcal{T}_i$, if

$$\forall l \in \{1, \ldots, m\}, \;\; \forall j \in \{1, \ldots, n_l\}, \quad \sigma_l(j) = \sigma(d + j) \text{ with } d = \sum_{k=1}^{l-1} n_k$$

A partial sequence σ_i relative to a subset $\mathcal{T}_i$ is said to be *locally optimal* if, with the scheduling of tasks of all the other subsets remaining unchanged, σ_i makes the maximal value of $V(\sigma)$ $(\sigma = \sigma_1 \ldots \sigma_i \ldots \sigma_m)$ among all the sequences of $\mathcal{T}_i$.

We will also occasionally write $\sum_{k \in \sigma_i} x_k$, where x_k is a generic task-number-indexed quantity, instead of the more rigorous expression $\sum_{k \in \mathcal{T}_i} x_k$. Always for the sake of simplicity, by convention, if $b < a$ then we have $\sum_{i=a}^{b} x_i = 0$ and $\cup_{i=a}^{b} \mathcal{T}_i = \emptyset$

For technical reasons we will assume that $p_i = l_i T, l_i \in \mathbb{N}, i = 1, \ldots, n$. T is the largest common disor of the set of $\{p_i\}_{i=1,\ldots,n}$. This hypothesis will be referred as "*assumption 1*" and will be used sometimes to ensure the polynomiality of our algorithms. With no loss of generality we can assume that $T = 1$ that we do in the following. Let us define K which satisfies $t_{tot} = \sum_{k=1}^{n} p_k = Kn$. We notice that K may not be an integer ; K depends on the ratio between the time durations of tasks and their greatest common divisor.

3 Analysis

3.1 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 complementary partial sequences of $\mathcal{T} - \{i, j\}$. Let $t = \sum_{k \in \Pi_1} p_k$, t is the instant at which tasks i and 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 $t \iff \Delta_{ij}(t) > 0$.

3.2 Sufficient optimality conditions

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

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

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

Proof:

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

Of course, this condition is fulfilled in very few cases. Nevertheless, some of them are interesting, since they lead to 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 on the time interval $[0, \sum_{j=1}^{n} p_j]$, then the optimal ordering is SPT (Shortest Processing Time first),
- and similarly, for monotonous increasing $F()$, the optimal ordering is LPT (Largest Processing Time first).

3.3 Precedence relations

The optimal condition we deduced has limited applications. Now, let us return to $\Delta_{ij}()$, in order to investigate the temporal behavior of the precedence relations. Let $t_{tot} := \sum_{j=1}^{n} p_j$, t_{tot} is the time required for the processing of all the n tasks. 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 $t_b(i, j)$ and $t_e(i, j)$, the interval $[t_b(i, j), t_e(i, j)]$ will be called the *scheduling scope* of the couple (i, j). Initially,

$t_b(i,j) = 0$ and $t_e(i,j) = t_{tot} - p_i - p_j$. The scheduling time considered subsequently for any couple (i,j) will be in the interval $[0, t_{tot} - p_i - p_j]$.

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

Otherwise, the scheduling scope is divided into sub-intervals, in which we have either $i \prec j$ or $j \prec i$, *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. An absolute precedence can be viewed as a strong precedence at $t = 0$.

The previous definition of strong precedence cannot have practical use, since it is impossible to check precedence at each point of a time interval. Fortunately, we can use a weaker definition of strong precedence, which consists of checking the precedences only at scheduling points. In fact, we only have to check the precedence at the scheduling points, *i.e.* to compute $\Delta_{ij}(t)$ where $0 \leq t \leq t_{tot}$ and t is an instant at which a task is completed according to at least one of the $n!$ possible sequences. With no assumption on the p_i we can see that we have at most $2^{n-1} - 1$ such points. So, we can't check strong precedence in polynomial time.

However, under *assumption 1*, the strong precedence checking becomes polynomial. Indeed, when p_i are integers, we only have to check points on the grid of scale 1. The function $\Delta_{ij}()$ is completely defined by the numbers :

$$\Delta_{ij}(l) \qquad \text{where } 1 \leq l \leq Kn$$

which can be computed once for all in $O(K\frac{n^3}{2})$ operations. To compute all the precedences between a couple of tasks, the complexity is less than $O(Kn)$.

Remark: With special TVF like weighted quadratic penalty, it is possible to define a *threshold time*, $t_{ij} \in [0, t_{tot} - p_i - p_j]$ such that $\Delta_{ij}(t_{ij}) = 0$ and $j \prec i\ \forall t > t_{ij}$, $i \prec j\ \forall t < t_{ij}$. That is to say that precedence between tasks can change only once in the scheduling scope [Che91].

3.4 Decomposition

The complexity of a combinatorial problem can be reduced if the latter can be decomposed into several smaller size problems. For this problem in particular, if we can establish the existence of a partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$, such that an optimal scheduling order is of the form $\mathcal{T}_1\,\mathcal{T}_2 \ldots \mathcal{T}_m$, then we achieve a complexity reduction, whose quality depends on the size of the largest set $\mathcal{T}_i, i \in \{1,\ldots,m\}$. Here, we present some lemmas concerning the decomposition technique:

Lemma 1 *Let $\mathcal{T}$ be partitioned into $\mathcal{T}_1$ and $\mathcal{T}_2$, with the scheduling order fixed to $\mathcal{T}_1\,\mathcal{T}_2$. If $\exists i \in \mathcal{T}_2$, such that $\forall j \in \mathcal{T}_2 - \{i\}$, $i\,! \prec j$ at $t = \sum_{i\in\mathcal{T}_1} p_i$, then an optimal sequence of $\mathcal{T}$ is of the form $\Pi_1\ i\ \Pi_2$, where Π_1 and Π_2 are permutations of $\mathcal{T}_1$ and $\mathcal{T}_2 - \{i\}$, respectively.*

Proof:

As tasks in $\mathcal{T}_1$ are to be scheduled first, we limit our consideration to the subset $\mathcal{T}_2$, which begins at t. Now, let σ be an optimal sequence of $\mathcal{T}_2$. Let us assume that $\sigma(j) = i$ with $j > 1$, then, since σ is optimal, we have $\sigma(j-1) \prec \sigma(j) = i$ at $t_1 = t + \sum_{k=1}^{j-2} p_{\sigma(k)}$. But, $t_1 \geq t$, and i has strong precedence at t over all the tasks in $\mathcal{T}_2$, so $i = \sigma(j)\,! \prec \sigma(j-1)$ will be true at t_1 in particular, which is contradictory. Consequently, $\sigma(j) = i$; $j > 1$ cannot be true, in other word $\sigma(1) = i$. ∎

We get also an extension of this lemma:

Lemma 2 *Let $\mathcal{T}$ be partitioned into $\mathcal{T}_1, \mathcal{T}_2$ and $\mathcal{T}_3$, with the scheduling order fixed to $\mathcal{T}_1\,\mathcal{T}_2\,\mathcal{T}_3$. If for all $(i,j) \in \mathcal{T}_2 \times \mathcal{T}_3$, we have $i\,!\prec j$ at $t = \sum_{i\in\mathcal{T}_1} p_i$, then an optimal sequence of $\mathcal{T}$ is of the form $\Pi_1\ \Pi_2\ \Pi_3$, where Π_1,Π_2 and Π_3 are permutations of $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ respectively.*

Proof:

We consider the reduced scheduling set $\mathcal{T}_2 \cup \mathcal{T}_3$. Suppose that the optimal ordering is not of the form $\Pi_2\ \Pi_3$. On the one hand, the optimal ordering is necessary of the form: $\sigma_1\ \sigma_2\ i\ \sigma_3$, where $i \in \mathcal{T}_2$, σ_1 is a permutation of the set $\mathcal{T}_2 - \{i\}$ (may be empty), and σ_2 and σ_3 are permutations of two partitions of $\mathcal{T}_3$, σ_2 must be non-empty. On the other hand, we have that i has strong precedence over all tasks in $\mathcal{T}_3$ at t, so necessary at $t_1 = \sum_{k\in\sigma_1} p_k + t$. From lemma 1, the sequence $\sigma_1\ i\ \sigma_4$, where σ_4 is some permutation of $\mathcal{T}_3$, is better than the sequencing $\sigma_1\ \sigma_2\ i\ \sigma_3$. This implies that σ_2 should be empty, which is contradictory. So, the optimal sequence is necessary of the form $\Pi_2\ \Pi_3$, where Π_2 and Π_3 are some permutations of $\mathcal{T}_2$ and $\mathcal{T}_3$. ∎

For the sake of simplicity, we will say that the subset $\mathcal{T}_i$ precedes the subsets $\mathcal{T}_j$ at t, and denote the fact by $\mathcal{T}_i \prec \mathcal{T}_j$, if $\forall (i,j) \in \mathcal{T}_i \times \mathcal{T}_j$, $i\,!\prec j$ at t.

Now, we present some decomposition condition, which, when fulfilled, will lead to a partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$, with an optimal ordering $\mathcal{T}_1\,\mathcal{T}_2\,\ldots\,\mathcal{T}_m$. It is in this sense that we refer it as *optimal decomposition*.

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

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

Then all the optimal sequences σ are of the type $\sigma_1\ \sigma_2 \ldots \sigma_m$, where σ_i is a locally optimal permutation of the subset $\mathcal{T}_i$. The decomposition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ is called an optimal decomposition of $\mathcal{T}$.

Proof:

We have, at $t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k$, $\mathcal{T}_l \prec \{\cup_{r=l+1}^{m} \mathcal{T}_r\}$. Using lemma 2 iteratively from $l = 1$, we establish that an optimally sequence is of the form $\sigma_1^{'}\ \sigma_2^{'} \ldots \sigma_m^{'}$, where $\sigma_i^{'}$ is a permutation of the subset $\mathcal{T}_i$. In addition, $\sigma_i^{'}$ must be locally optimal, otherwise the whole sequence would not be optimal. ∎

Proposition 3 *If an optimal decomposition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ is irreducible (no subset can be decomposed in two subsets which satisfy the strong precedence property of an optimal decomposition) then this optimal decomposition is unique.*

Proof:

Let us suppose that we have two different irreducible partitions $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ and $\{\mathcal{T}_i'\}_{i\in\{1,\ldots,m'\}}$. We note l the smallest index for which $\mathcal{T}_l \neq \mathcal{T}_l'$ and $t = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k$. We can find two tasks i,j such that $i \in \mathcal{T}_l - \mathcal{T}_l'$ and $j \in \mathcal{T}_l' - \mathcal{T}_l$. Thus we have $i\,!\prec j$ and $j\,!\prec i$ at time t. That is obviously a contradiction. ∎

4 Scheduling algorithms

4.1 Overview

The proposition 2 gives a sufficient condition for the existence of an optimal decomposition. If the latter does exist, then it suffices to work with each sub-set to find locally optimal sequences. But, the proposition does not help to find such a decomposition. In this section, we present a family of heuristic scheduling algorithms, each of them will yield a sequence, referenced subsequently as *algorithmic sequence*, which has the following property:

- the algorithmic sequence will respect the optimal decomposition, *i.e.* all the tasks of the subset of a lower rank will be executed prior to the tasks of a subset of a higher rank ;
- if the decomposition is not possible, the algorithmic sequence yields nevertheless values close to the optimal one in most cases, as suggest by our simulations,

Once the sequence is established, an algorithm allows to show off the optimal decomposition. The scheduling algorithms as well as the decomposition algorithm have a polynomial complexity.

4.2 A generic sequencing algorithm

In this section, we present the sequencing algorithm in its generic form. Let $R(.)$ denote a selection function returning a unique task number (*e.g.* , $R(.) = \max(.)$), $G(t,i)$ a real-value function mapping from $\mathbb{R} \times \{1,..,n\}$ into $\mathbb{R}$, and σ the algorithmic sequence. The following algorithm selects sequentially the n tasks. At each selection step, the chosen task, denoted by s, maximizes a function evaluated at t, which is the completion time of the lastly scheduled task. $\mathcal{T}_r$ denotes the set of the unscheduled tasks at time t.

<u>**Algorithm**</u>

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

4.3 Compatibility with the optimal decomposition

In the generic algorithm, the function $G(t,i)$ is not specified. Hereafter, we will prove that for a class of $G(t,i)$, this algorithm will yield a sequence respecting the optimal decomposition, if any.

Proposition 4 *If an optimal decomposition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$ exists, and if $G(t,i)$ is such that*

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

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

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

<u>*Proof:*</u>

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

On the one side, we have necessary $j > l_e$ since $l = \min \mathcal{E}$. The selection rule of our algorithm is such that

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

with $\mathcal{R} = \mathcal{T} - \{\sigma(h)\}_{h=1..i-1}$ 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 $\sigma(i) \in \mathcal{T}_r$ 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 $R()$.

4.4 Time complexity

Proposition 5 *If the computation of* $s := R(\{i \ / \ i \in \mathcal{T}_r \ \text{ and } G(t,i) = max\{G(t,j) \ / \ j \in \mathcal{T}_r\}\})$ *has time complexity upper bounded by* Ab^a, *where* $b = Card(\mathcal{T}_r)$ *and* A *is a parameter independent of* b, *then the algorithm has time complexity upper bounded by* $A\frac{n^{(a+1)}}{a+1}$.

Proof: In each step of the scheduling, the computation and the selection of s requires no more than Ab^a. As b is reduced by one at each step, the complexity is upper bounded by $A\sum_{r=1}^{n} r^a$ which is upper bounded by $\int_0^n A(x+1)^a dx \leq A(n+1)^{(a+1)}/(a+1)$, consequently, the complexity of the whole loop is upper bounded by $A\frac{n^{(a+1)}}{a+1}$. ∎

4.5 Sample scheduling algorithms

4.5.1 Precedence among tasks

Let us denote by $P(t,i)$ the number of tasks that the task i precedes at time t and $SP(t,i)$ the number of strong precedence, we have the obvious properties :

$$\begin{array}{ll} \forall t & P(t,i) \geq SP(t,i) \\ \forall t_1, t & t_1 \geq t \Longrightarrow \ SP(t_1,i) \geq SP(t,i) \end{array}$$

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 $SP(t,i)$ and $P(t,i)$ ordering on the subset level:

Proposition 6 *If the task set* $\mathcal{T}$ *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* $l \in \{1,\ldots,m-1\}$,

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

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

Proof:

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

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

and for all $t > t_1$, $SP(t,i) \geq SP(t_1,i)$.
On the other hand, j is strongly preceded by all 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 $\forall t \geq t_1$, $SP(t,i) > P(t,j)$. Applying the relation $\forall t$ $P(t,i) \geq SP(t,i)$ at both sides, we get the proof. ∎

4.5.2 Sample functions of $G(t,i)$

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

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

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

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

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

where the t_j are $O(n)$ fixed points in the scheduling scope. G_1 and G_2 yield sequences respecting the optimal decomposition in virtue of the proposition 6. It is the same with G_3 and G_4 which are respectively a linear combination of G_1 and G_2.

The selection function associated with G_1 require a computation of complexity $O(r^2)$ with r tasks, as G_3 require a computation of $O(nr^2)$ as there is at most $O(n)$ points $t_j > t$ in the scheduling scope. The scheduling algorithm using G_1 or G_3 is polynomial.

For G_2 and G_4 we need *assumption 1* to obtain polynomial algorithms. Then, the selection function associated with G_2 requires a computation of complexity $O(Knr^2)$ with r tasks while the selection function associated with G_4 requires a computation of complexity $O(Knr^3)$ with r tasks.

We assume that function $R()$ is simple enough and has a complexity $O(r)$ if $R()$ selects among r tasks. For example, we can choose $R(.) = max(.)$ or $R(.) = min(.)$ Then the scheduling algorithm has a complexity of $O(n^3)$ with G_1, $O(\frac{K}{3}n^4)$ with G_2, $O(\frac{n^4}{3})$ with G_3 and $O(\frac{K}{5}n^4)$ with G_4.

The combination of function $R()$ and $G()$ yields a family of number of scheduling algorithms.

4.5.3 Reranking

The previous algorithms may produce bad scheduled couples, *i.e.* for some i, $\sigma(i+1) \prec \sigma(i)$ at $t = \sum_{k=1}^{i-1} p_k$. The selection rule based on precedence number is unable to predict the precedence between the currently selected task and the next one except when these two tasks belong to different subsets of decomposition.

We propose a local reranking, which checks and interchanges the bad couples, so leads to a better sequence.

1. *Initialization:* $t = 0$; $i = 1$;
2. **WHILE** $i \neq n$ **DO**
   BEGIN
   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

3. end of reranking. Output σ.

Remark: The goal of the reranking is to eliminate bad scheduled couples. However, whereas each iteration does decrease $V(\sigma)$, the algorithm does not prevent the creation of new bad couples. This would logically imply its repetitive application until the absence of bad scheduled couple. However, while each reranking requires only $O(n)$ time complexity, the number of iterations may not be polynomial. In our tests, we have set an upper bound on the number of iteration. Simulation experiences suggest that this number is in general small (less than n).

5 Decomposition algorithm

The following algorithm checks and finds out the explicit form of the decomposition of the algorithmic sequence σ, in the form $\sigma = \sigma_1\sigma_2 \ldots \sigma_m$, σ_i being a sequence of the subset $\mathcal{T}_i$ in the sense of proposition 2, under the constraint: $\forall(l,j) \in \{1,\ldots,m\} \times \{1,\ldots,n_l\}\ \sigma_l(j) = \sigma(d+j)$ with $d = \sum_{k=1}^{l-1} n_k$.

5.1 Algorithm

Notations: t is the current precedence check time; l the index of the current subset under check; $\mathcal{T}_l$ the current subset under check; $\mathcal{T}_r$ the remaining tasks set : $\mathcal{T}_r = \mathcal{T} - \cup_{i=1}^{l}\mathcal{T}_i$; and σ is the algorithmic sequence under check.

Algorithm

1. $l := 1$; $\mathcal{T}_l = \{\sigma(1)\}$; $\mathcal{T}_r := \mathcal{T} - \mathcal{T}_l$;
2. **WHILE** $\max\{i \,/\, \sigma(i) \in \mathcal{T}_l\} < n$ **DO**
   BEGIN
   { `Set precedence check time` }
   $t := \sum_{i\in\{\mathcal{T}-\mathcal{T}_l-\mathcal{T}_r\}} p_i$;
   IF $\mathcal{T}_l \prec \mathcal{T}_r$ at t **THEN**
   BEGIN { $\mathcal{T}_l$ `completion, initialzation of the next subset`}
   $l := l + 1$;
   $\mathcal{T}_l := \{\sigma(i) \,/\, i = \min\{j \,/\, \sigma(j) \in \mathcal{T}_r\}\}$;
   $\mathcal{T}_r := \mathcal{T}_r - \mathcal{T}_l$;
   END ELSE
   BEGIN { $\mathcal{T}_l$ `extension` }
   $b := \min\{i \,/\, \sigma(i) \in \mathcal{T}_r\}$;
   $e := \max\{i \,/\, \sigma(i) \in \mathcal{T}_r \text{ and } \exists j \in \mathcal{T}_l, \sigma(i) \prec j \text{ at } t\}$;
   $\mathcal{T}_l := \mathcal{T}_l \cup \{\sigma(i)\}_{i=b,\ldots,e}$;
   $\mathcal{T}_r := \mathcal{T}_r - \{\sigma(i)\}_{i=b,\ldots,e}$;
   END
   END WHILE
3. m:=l; Output $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$.

5.2 Time complexity

Under the *assumption 1*, we are able to prove that the decomposition can be done in polynomial time:

Proposition 7 *Under the assumption 1, the decomposition algorithm is polynomial with a complexity less than $O(Kn^3)$.*

Proof:

Suppose that $\mathcal{T}$ is partitioned into $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$. Recall that $n_i = Card(\mathcal{T}_i)$ and let $l = n - \sum_{k=1}^{i-1} n_k$. For the search of each subset, this algorithm uses an iterative approach. The size of each subset can increase at each step by at least one, the worst case is obtained when at each step only a new task is included. To simplify, we will bound by nK the number of computation for the check of strong precedence (at most nK instants are to be tested). The complexity to find $\mathcal{T}_i$, C_i, is thus

$$\begin{aligned} C_i &\leq \sum_{j=1}^{n_i}(l-j)nK \\ &= Kn(n_i l - \frac{n_i+1}{2}n_i) \\ &\leq Kn(n_i n - \frac{n_i+1}{2}n_i) \end{aligned}$$

since $l \leq n$. Thus, the overall complexity is

$$C = \sum_{i=1}^{m} C_i \leq \sum_{i=1}^{m} Kn(n_i n - \frac{n_i+1}{2}n_i) \leq Kn(n^2 - n/2 - \sum_{n_i} {n_i}^2/2) \leq Kn^3$$

Consequently, the algorithm's complexity is less than $O(Kn^3)$. ∎

6 Computational experience

6.1 Conditions and parameters

Algorithms are implemented and tested by simulation, in order to evaluate their performance. The TVFs associated to tasks are randomly generated. Only decreasing functions are selected for two reasons: a) these functions are realistic situations; b) the fact that functions are decreasing makes obsolete the insertion of "idle times", which could complicate the problem (cf. [BS90]). We have run tests with several kinds of TVFs. We have tried successively the following types:

1. The TVFs are linearly decreasing. (cf. Figure 1).
2. The TVFs are constant on a first time interval and linearly decreasing on a second time interval (cf. Figure 2).
3. The TVFs are constant on a first interval and exponentially decreasing on a second time interval (cf. Figure 3).
4. The TVFs are constant on a first time interval and constantly null on a second time interval (cf. Figure 4).
5. The TVFs can be written as $F(t) = a - b(t-c)^2$ with a, b, c randomly selected. These TVFs can be associated with a JIT problem.
6. The TVFs are mixed and chosen among the previous types.

The first three types can be interpreted as functions associated with soft dead-lines. The 4th type can clearly be associated with the modelization of a dead-line.

The execution durations of the tasks are random integers. We have run a family of algorithms associated to respectively the first three functions $G_k(t,i)$ defined in 4.5.2. We subsequently refer as algorithm k the algorithm using function $G_k(t,i)$ with $k \in \{1,2,3\}$.

To evaluate our algorithms, we have programmed the dynamic programming approach [HK62] which yields the optimal value and one optimal sequence. This algorithm requires much computation space and time, and so we have to limit our investigation to 18 tasks.

6.2 Results of simulations

For the first type of TVF, our three algorithms give an exact optimum every time. That is not astonishing since we can easily see that the problem can be completely decomposed (each subset contains exactly one task or tasks having the same TVF). So the three algorithms give perfect results under these conditions.

For the second type of TVF, our three algorithms give different results. Algorithms 1 and 3 find an optimum in approximately 80% of the cases and algorithm 2 finds an optimum in 40% when $n \leq 10$. For $n = 10$ the scheduling problem can be efficiently decomposed in 50% of the cases. For $n = 18$ the algorithm 1 gives around 90% of success as the two others find an optimum only 10% of the cases.

For the third type of TVF, our three algorithms give different results. Algorithm 1 find an optimum in approximately 90% of the cases as algorithm 2 and 3 find an optimum in 50% when $n \leq 10$. For $n = 10$ the scheduling problem can be efficiently decomposed in 60% of the cases. Algorithm 1 find an optimum in approximately 70% of the cases as algorithms 2 and 3 find an optimum in 10% when $n = 18$.

For the fourth type of TVF, our three algorithms give poor results. It is always algorithm 1 which provides the best results finding an optimal sequence in about 10% of the cases. This algorithm is anyhow better than the simplest one which consists in selecting among $O(n^2)$ sequences the one giving the best $V(\sigma)$ (algorithm which has the same complexity than the algorithm 1).

For the fifth type of TVFs our three algorithms good results. It is always algorithm 1 which provides the best results finding an optimal sequence in about 80% of the cases. We find again the results presented in [Che91].

When we mixed the previous types of TVFs we still find good results. We can see that the results are getting worst as the number of TVFs of the fourth type is increasing.

We have noticed that the reranking algorithm does give improvements, especially for fourth and fifth type functions.

The previous results must be analyzed with precaution. In fact, it is not easy to define a set of general bench mark TVF. Anyhow the previous results show that algorithm 1 behave better than the two other besides its complexity is lower. We have noticed that wide decompositions often occur for small size problems is ($n \leq 10$), and occasionally with n around 15. In case of a wide decomposition our three algorithms work perfectly and of course we can verify that they do respect the decomposition. It is interesting to see that the algorithm 1 still provides good results when the decomposition of the problem is poor or inexistant if the TVFs are regular enough.

Our computational experimentations suggest that the optimal decomposition conditions are sometimes too restrictive to obtain small size decompositions. For these reasons, we have also developed sub-optimal decomposition policies, using similar technique as the optimal decomposition. These policies, which are less restrictive than the optimal decomposition are described in the Appendix, where we also give the upper-bound error for the difference between $V(\sigma)$ for an optimal sequence and $V(\sigma)$ for a given sequence. Such results could be used in a scheduling software.

7 Conclusion

This study is focused on the real-time systems. Time Value Functions are a new subject which has received less attention than the problems of scheduling under deadline constraints.

We have formalized this scheduling problem as an optimization problem, in occurrence the maximization of a sum of time value functions evaluated at the completion time of the task.

For the maximization of the sum time value functions related to tasks running of a same machine, we have developed sufficient optimality conditions and a set of optimal decomposition conditions. Based on these results, sub-optimal polynomial algorithms are proposed which perform the scheduling and the decomposition. Computational experience suggests that these algorithms are rather efficient for various scenarii. Our algorithms can be used with more general hypothesis, for example we can handle the problem of tasks with released-times.

Acknowledgements

The authors are indebted to Dr. G. Le Lann, who introduced them to the subject, for his numerous helps and suggestions during this work.

Different kinds of TVF experienced

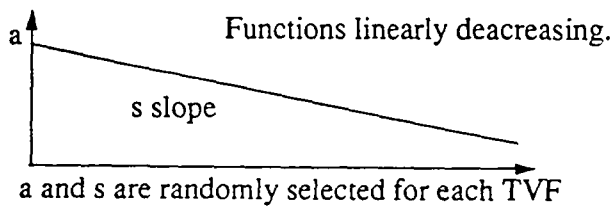

Figure 1

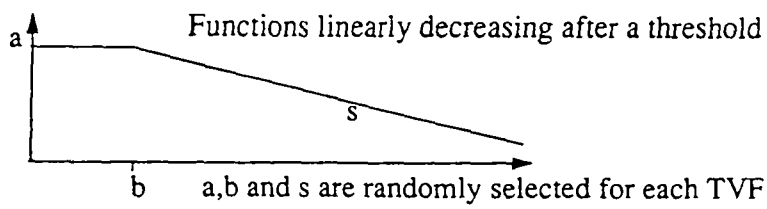

Figure 2

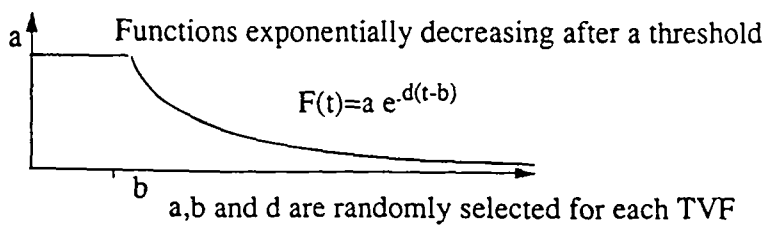

Figure 3

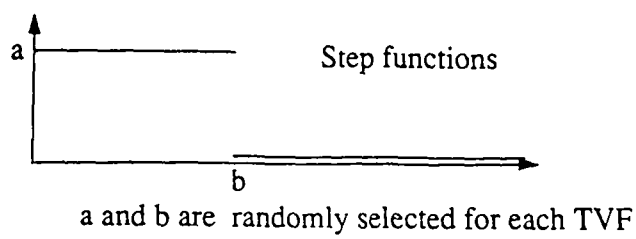

Figure 4

8 APPENDIX

In this appendix, we give in a first part the description of a sub-optimal decomposition, and in a second part we give an upper bound for the difference between $V(\sigma)$ for an optimal sequence and $V(\sigma)$ for any given sequence.

8.1 Sub-optimal decompositions

In section 3, we have seen the optimal decomposition. However, as the required conditions are rather restrictives, the reduction (*i.e.* $m > 1$) is not always possible. Hereafter, we give a sub-optimal decomposition conditions. Compared to the optimal one, the sub-optimal conditions are less restrictive, so leads more often to smaller size decompositions; inversely, it does not assure an optimal subset ordering, instead, we have obtained the error upper bound of the resulted sub-optimal sequence relative to the optimal one.

Proposition 8 *If there exists a partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$, such that*

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

where $t_e = \sum_{r=1}^{l} \sum_{k\in\mathcal{T}_r} p_k$. Then $\Pi_s = \sigma_1\, \sigma_2 \ldots \sigma_m$, where σ_i is a local optimal permutation of the subset $\mathcal{T}_i$, yields a value whose difference with the optimal one's is bounded by

$$D = -\sum_{l=1}^{m-1} \sum_{(i,j)\in\mathcal{T}_l\times\{\cup_{r=l+1}^{m}\mathcal{T}_r\}} \min_{t\in[t_b,t_e]} (\Delta_{ji}(t), 0)$$

where $t_b = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k$.

Proof:

First, remark that t_b and t_e are respectively the time at which the subset $\mathcal{T}_l$ is to be scheduled, and the time at which the subset $\mathcal{T}_l$ is finished, with respect to the ordering $\mathcal{T}_1\mathcal{T}_2\ldots\mathcal{T}_m$

Now, let Π_o be an optimal sequence. Let Π_1 be a sequence in the form $\sigma_1^{'}\, \sigma_2^{'} \ldots \sigma_m^{'}$, where $\sigma_i^{'}$ is a permutation of the subset $\mathcal{T}_i$. $\sigma_i^{'}$ is an intermediary sequence, not necessary local optimal, but in which tasks conserve the same relative position as in Π_o, *i.e.* for any $l \in \{1..m\}$ and for any $(i,j) \in \{1..n_l\}^2$, with, say, $i < j$, we have $\sigma_l^{'}(i) = \Pi_o(a)$ and $\sigma_l^{'}(j) = \Pi_o(b)$ with $(a,b) \in \{1..n\}^2$ and $a < b$.

Now, we transform Π_o to Π_1 by moving tasks from their place in Π_o into the one in $\Pi_1$via task interchange, this is done in an iterative manner from $\Pi_1(1)$. Each task may cost interchange penalty. As the subset ordering is preserved in Π_1, the penalty may occur only when interchanges takes place between $]t_b, t_e[$, since the task is scheduled after t_b, and at t_e we have $\mathcal{T}_i \prec \mathcal{T}_j$, so no penalty. More precisely, it may occur between $]t_b, t_e[$ with $j < i$, as we have proved in the first part. Let $t \in]t_b, t_e[$ be such an instant, we have

$$\Delta_{ji}(t) \geq \min_{t\in[t_b,t_e]} (\Delta_{ji}(t), 0)$$

Apply this reasoning to all the subsets and we get the upper bound D, with

$$V(\Pi_1) \geq V(\Pi_o) - D$$

Now, consider the sequence $\Pi_s = \sigma_1\, \sigma_2 \ldots \sigma_m$ where σ_i are local optimal. We have obviously

$$V(\Pi_s) \geq V(\Pi_1) \geq V(\Pi_o) - D$$

and we get the proof. ∎

Remark: In $\min_{t\in[t_b,t_e]}(\Delta_{ji}(t),0)$, we only consider values on scheduling points.

With this sub-optimal decomposition, as the strong precedence is true only from the starting time of the next subset, it is possible that the couple formed by the last task of some subset and the first task of the next subset does not respect the good precedence relation. The following rule will eliminate this case, but the decomposition conditions become more restrictive.

Proposition 9 *If there exists a partition $\{\mathcal{T}_i\}_{i\in\{1,\ldots,m\}}$ of $\mathcal{T}$, such that*

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

where $t_e = \sum_{r=1}^{l} \sum_{k\in\mathcal{T}_r} p_k - \max\{p_k \ / \ k \in \mathcal{T}_l\}$. Then

1. *the sequence $\Pi_s = \sigma_1\ \sigma_2 \ldots \sigma_m$, where σ_i is an local optimal permutation of the subset $\mathcal{T}_i$, yields a value whose difference with the optimal one's is bounded by*

$$D = -\sum_{l=1}^{m-1} \sum_{(i,j)\in\mathcal{T}_l\times\{\cup_{r=l+1}^{m}\mathcal{T}_r\}} \min_{t\in[t_b,t_e]}(\Delta_{ji}(t),0)$$

   where $t_b = \sum_{r=1}^{l-1} \sum_{k\in\mathcal{T}_r} p_k$. and

2. $\forall l < m$, $\sigma_l(n_l)! \prec \sigma_{l+1}(1)$ *at* $t = \sum_{i\in\{\cup_{j=1}^{l}\mathcal{T}_j\}} p_i - p_{\sigma_l(n_l)}$

Proof:

The first point can be proved in the same way as the one for the precedent proposition. So, it is sufficient to prove the second point:

In fact: as $p_{\sigma_l(n_i)} \leq \max\{p_k \ / \ k \in \mathcal{T}_l\}$, we have $t \geq t_e$; as $\mathcal{T}_l \prec \mathcal{T}_{(l+1)}$ at t_e, so $\mathcal{T}_l \prec \mathcal{T}_{(l+1)}$ at t, and consequently $\sigma_l(n_l)! \prec \sigma_{l+1}(1)$ at t. ∎

Subsequently, the decomposition in the sense of proposition 8 will be called *wide sub-optimal decomposition*, and the one in the sense of proposition 9 will be called *strict sub-optimal decomposition*.

8.2 Upper bound of a given sequence

The upper bound of deviation of any given sequence can be computed using the same principle, *i.e.* by enumerating all the potential penalties.

Proposition 10 *Let σ be a given sequence and σ_o the (unknown) optimal one. The upper bound D of the deviation of $V(\sigma)$ from $V(\sigma_o)$, i.e. $V(\sigma) \geq V(\sigma_o) - D$, is given by:*

$$D := -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \min_{r\in[k_b,k_e]}(\Delta_{ji}(r),0)$$

with $t_b = \sum_{l=1}^{i-1} p_{\sigma(l)}$ and $t_e = t_{tot} - p_{\sigma(i)} - p_{\sigma(j)}$

Proof:

The transformation from σ_o to σ can be achieved step by step in the following way: $\sigma_o \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots \sigma_{n-1} \rightarrow \sigma$.

σ_1 is constructed in the following manner: Suppose that $\sigma(1) = \sigma_o(k)$, for some $k \geq 1$. We transform σ_o into σ_1, with:

- $\sigma_1(1) := \sigma_o(k) = \sigma(1)$;

- $\sigma_1(i) := \sigma_o(i+1)$, for $i < k$
- $\sigma_1(i) := \sigma_o(i)$, for $i > k$

Then we begin with $\sigma(2) = \sigma_1(k)$, for some $k \geq 2$, and transform σ_1 into σ_2, with:

- $\sigma_2(1) = \sigma_1(1) = \sigma(1)$;
- $\sigma_2(2) := \sigma_1(k) = \sigma(2)$;
- $\sigma_2(i) := \sigma_1(i+1)$, for $i < k$
- $\sigma_2(i) := \sigma_1(i)$, for $i > k$

And so on. We can evaluate the local error upper bound for each step of the transformation, the overall error, D, is obviously less than the sum of the error upper bound of each step,

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \min_{t \in [t_b, t_e]} (\Delta_{ji}(t), 0)$$

Consider the first step $\sigma_o \rightarrow \sigma_1$: the worst case is that $\sigma(1) = \sigma_o(n)$, *i.e.* $\sigma(1)$ must be interchanged with every other task. Now, denote $\sigma(1)$ by i, and take a task j.To bound the penalty we have to consider the maximum value of $\Delta_{ij}(t)$ within the scheduling scope which is $[t_b, t_e]$.

Apply this reasoning to all the steps, by remarking that at each step we have one less task to be compared, the values of d_{ij} are straightforward. ∎

This upper bound computing method can of course be applied at the subset level. Consequently, if a decomposition is possible, the maximal error of the sequence, D, is given by $D = \max(D_1, D_2)$, where

- D_1 is the error upper bound of the sequence taken as a whole one
- D_2 is the sum of the decomposition error upper bound and all subsets' local sequence error upper bound.

We may have $D_1 < D_2$, since D_2 is a sum of upper bounds.

If $D = 0$, the sequence is obviously optimal. Otherwise, if $D < V(\sigma)$, we can assess the upper bound of the *closeness* of σ to σ_o. The *closeness* of σ to $\sigma_o$being defined by $\frac{V(\sigma)}{V(\sigma_o)}$, we have effectively $\frac{V(\sigma)}{V(\sigma_o)} \leq \frac{V(\sigma)}{V(\sigma)-D}$.

An upper bound error, for the decomposition as well as for the sequence, is obtained by simply applying the formulas found in the above sections. As the complexity is concerned, it is just $O(n^2)$, since eack task is compared with each other for assessing the possible penalty just once.

8.3 Decomposition algorithms

The algorithms for sub-optimal decomposition are nearly the same as the one given in sub-section 5.1. The only differences are the setting of precedence check time on the one hand, and the determination of value of e when performing T_l extension. We have simply $e = b$ here. Similarly we have

Proposition 11 *If assumption 1 holds, these algorithms are polynomial with a complexity less than* $O(Kn^3)$.

The proof of proposition 7 for optimal decomposition can be applied here.

References

[BK80] P. C. Bagga, K. R. Kalra, "A Node Elimination Procedure for Towsend's Algorithm for Solving the Single Machine Quadratic Penality Function Scheduling Problem," Manag. Science, V-26, No.6, Jun. 1980, pp. 633–636.

[BS90] K. R. Baker, G. D. Scudder, "Sequencing with earliness and tardiness penalities: a review," Oper. Res., V.38, No.1, Jan.-Feb. 1990, pp. 22–36.

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

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

[Cof76] E.G. Coffman Jr., "Computer and Job-Shop Scheduling Theory," John Wiley & Sons, 1976.

[GJ79] M. R. Garey, D. S. Johnson, "Computera and Intractability: A Guide to the Theory of NP-Completeness," Freeman & Co., NewYork, 1979.

[GS84] S. K. Gupta, T. Sen, "On The Single Machine Scheduling Problem with Quadratic Penality Function of Completion Times: An Improved Branching procedure," Manag. Science, V-30, No.5, May. 1984, pp. 644–647.

[GTW88] M. R. Garey, R. E. Tarjan, G. T. Wilfong, "One-processor scheduling with symmetri earliness and tardiness penalities," Oper. Res., V.13, No.2, May. 1988, pp. 330–348.

[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.

[LiL73] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a hard-real-Time Environment," J.ACM, V.20, No.1, Jan. 1973, pp. 46–61.

[LM69] E. L. Lawler, J. M. Moore, "A Functional Equation and its Application to Resource Allocation and Sequencing Problems," Manag. Science, V.16, No.1, Sept. 1969, pp. 77–84.

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

[Pos88] M.E. Posner, "the deadline constrainted completion time problem: analysis of a heuristic," Oper. Res,, V.36, No.5,Sept-Oct. 1988, pp. 742–746.

[RK76] A. H. G. Rinooy Kan, "Machine Scheduling Problems," Martinus Nijhoff, the Hague, 1976.

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

[SC79] S. Sahni, Y-K Cho, "Nearly on line scheduling for a uniform processor system with release times," SIAM J. Comp, V-8, May 1979, pp. 275–285.

[SPL88] W. Szwarc, M. E. Posner, J. Liu, "The Single Machine Problem with a Quadratic Cost Function of Completion Time," Manag. Science, V-34, No.12, Dec. 1989, pp. 1480-1488.

[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.

[Tow78] W. Towsend, "The Single Machine Problem with Quadratic Penality Function of Completion Times: A Branch-and-Bound Solution," Manag. Science, V-24, No.5, Jan. 1978, pp. 530-534.

[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.

[Wei81] H. J. Weiss, "A Greedy Heuristic for Single Machine Sequencing with Precedence Constraints," Manag. Science, V-27, No.10, Oct. 1981, pp. 1209-1216.

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

[ZRS87] W. Zhao, K. Ramamritham and J. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," IEEE Trans. Computer, Vol. 36, No. 8, Aug. 1987, pp. 949-960.

ISSN 0249-6399