Static Scheduling of Monoprocessor Real-Time Systems composed of Hard and Soft Tasks

Luis Alejandro Cortés, Petru Eles, and Zebo Peng
Embedded Systems Laboratory
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
{luico,petel,zebpe}@ida.liu.se

Technical Report
April 2003

Abstract

In this report we address the problem of static scheduling of real-time systems that include both hard and soft tasks. We consider systems in which both hard and soft tasks are periodic, and our analysis take into account the data dependencies among tasks. In order to capture the relative importance of soft tasks and how the quality of results is affected when missing a soft deadline, we use utility functions associated to soft tasks. Thus our objective is to find a schedule that maximizes the total utility and at the same time guarantees hard deadlines. We use the expected duration of tasks for evaluating utility functions whereas we use the maximum duration of tasks for ensuring that hard deadlines are always met. We show that the problem we study in this report is **NP**-complete and we present an algorithm that finds the optimal schedule as well as different heuristics that find near-optimal solutions at reasonable computational cost.

1 Introduction

There exist classes of real-time systems that require the execution of tasks which have distinct types of timing constraints. Such systems include activities whose completion before a given deadline is critical to the overall behavior of the system. Missing one such deadline has severe or catastrophic

consequences, hence these tasks are referred to as *hard*. At the same time, these real-time systems include activities that have looser timing constraints and a deadline miss can be tolerated though the quality of results might degrade. Such tasks are referred to as *soft*.

The problem of jointly scheduling hard and soft tasks has been studied, for example, in the frame of integrating multimedia applications into hard real-time systems [7], [1].

Most of the approaches consider that hard tasks are periodic whereas soft tasks are aperiodic. Buttazzo and Sensini [2] use the Earliest Deadline First (EDF) algorithm for scheduling hard tasks while soft tasks are assigned a priority (a fictitious deadline) when a request arrives, in such a way that aperiodic responsiveness is achieved while guaranteeing hard deadlines. This, as well as most of previous work, assumes that the sooner a soft task is served the better but makes no distinction among soft tasks, that is, there is no relative importance of soft tasks. Similarly, Ripoll et al. [10] propose an algorithm (also based on EDF) that guarantees that deadlines of hard periodic tasks are met and minimizes the response time of soft aperiodic tasks, provided they are served in FIFO order. Other approaches under dynamic priority assignment include [3], [6], and [11].

Approaches to the joint scheduling problem for fixed-priority systems have also been considered in the literature [4], [8], [12]. These make use of the Rate Monotonic (RM) algorithm for scheduling the hard periodic tasks and also try to minimize the response time of soft aperiodic tasks while guaranteeing hard deadlines.

In this report we address the problem of static scheduling (at design-time) of real time-systems made up of hard and soft tasks. We consider that both hard and soft tasks are periodic and, in order to capture the significance of soft tasks, we make use of utility functions (value or utility functions were first suggested by Locke [9] to represent importance and criticality of tasks). As opposed to the approaches cited above where tasks are assumed independent, we take into account the precedence relation among tasks.

Most of earlier work uses only the worst case execution time (WCET) for scheduling both hard and soft tasks (Abeni and Buttazzo's approach [1] does use WCET for guaranteeing hard deadlines and mean values for serving soft tasks though). We consider the fact that the actual execution time of a task is rarely its WCET. Thus we use the expected duration of tasks when evaluating the utility functions associated to soft tasks (we aim to find the schedule for which the total utility is maximum) and we use the maximum duration of tasks for ensuring that all hard deadlines are met in every possible scenario.

The rest of this report is organized as follows. In Section 2 we intro-

duce some definitions and notations used along the report. We use a simple example in Section 3 in order to motivate and illustrate our approach. In Section 4 we give a precise formulation of the problem and show that it is **NP**-complete. We present an exact algorithm that finds the optimal solution (Section 5) as well as a number of heuristics that find near-optimal solutions in reasonable time (Section 6). In Section 7 we present the results of the experimental evaluation of the proposed algorithms. Finally, some conclusions are drawn in Section 8.

2 Preliminaries

We consider that the system is represented by a directed acyclic graph $G = (T, E)$ where its nodes correspond to tasks (T) and their data dependencies are given by the graph edges. Throughout this report we assume that all the tasks of the system are mapped into a single processor.

We use ${}^\circ t$ to denote set of the predecessors of task t, that is, ${}^\circ t = \{t' \in T \mid \langle t', t\rangle \in E\}$. Similarly, $t^\circ = \{t' \in T \mid \langle t, t'\rangle \in E\}$ denotes the set of successors of task t.

The execution time of every task $t \in T$ (denoted $|t|$) lies in the interval bounded by the minimum duration $l(t)$ and the maximum duration $m(t)$ of the task, i.e. $l(t) \leq |t| \leq m(t)$. In our analysis we take into consideration the expected duration $e(t)$ of every task $t \in T$, which is the mean value of the possible execution times of the task. In the simple case that the execution time is uniformly distributed over the interval $[l(t), m(t)]$, we have $e(t) = (l(t) + m(t))/2$. For an arbitrary continuous probability distribution $f(\tau)$, the expected duration is $e(t) = \int_{l(t)}^{m(t)} \tau f(\tau) d\tau$.

We define a *schedule* as the execution order for the tasks in the system. We assume a single-rate semantics, that is, each task is executed exactly once for every activation of the system. Thus a schedule is a bijection $\sigma : T \rightarrow \{1, 2, \ldots, |T|\}$. We use $\sigma = t_1 t_2 \ldots t_n$ as shorthand for $\sigma(t_1) = 1, \sigma(t_2) = 2, \ldots, \sigma(t_n) = |T|$. In this context, a schedule does not provide the starting time for tasks, only their execution sequence. Thus, for the schedule $\sigma = t_1 t_2 \ldots t_n$, task t_1 will start when the system is activated and task t_{i+1} will start executing as soon as task t_i has finished. In the sequel, the times that we use are relative to the system activation instant. For example, for the schedule $\sigma = t_1 t_2 \ldots t_n$, t_1 starts executing at time 0. We assume that the system is activated periodically and its period is $\sum_{t \in T} m(t)$.

The tasks that make up a system can be classified as non-real-time, hard, or soft. Non-real-time tasks are neither hard nor soft, and have no timing constraints, though they may influence other hard or soft tasks through

precedence constraints as defined by the task graph $G = (T, E)$. Both hard and soft tasks have deadlines. A hard deadline $d(h)$ is the time by which a hard task $h \in T$ *must* be completed, otherwise the integrity of the system is jeopardized. A soft deadline $d(s)$ is the time by which a soft task $s \in T$ *should* be completed. Lateness of soft tasks is acceptable though it decreases the quality of results. In order to capture the relative importance among soft tasks and how the quality of results is affected when missing a soft deadline, we use a non-increasing utility function $u_i(\tau_i)$ for each soft task s_i (τ_i denotes the completion time of s_i). Typical utility functions are depicted in Figure 1.

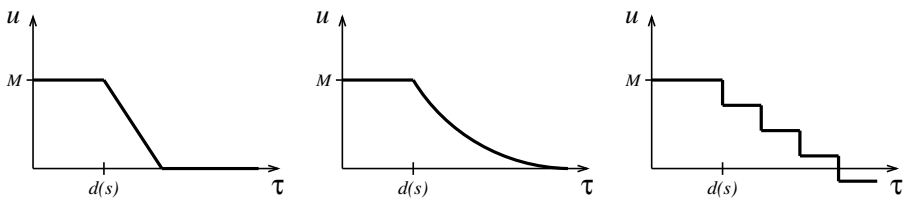

Figure 1: Typical utility functions for soft tasks

In this report we address the problem of finding a schedule that maximizes the sum of individual utilities of soft tasks when considering expected execution times, yet guaranteeing that hard deadlines are always met. Such a sum is called total utility and denoted U ($U = \sum_{s_i \in S} u_i(\tau_i)$, where S is the set of soft tasks).

3 Motivational Example

Let us consider a system that has five tasks t_1, t_2, t_3, t_4, and t_5, with data dependencies as shown in the graph of Figure 2. The expected and maximum duration of every task are given in Figure 2 in the form $e_i = e(t_i)$ and $m_i = m(t_i)$ respectively. The only hard task in the system is t_4 and its deadline is $d(t_4) = 30$. Tasks t_2 and t_3 are soft, their deadlines are $d(t_2) = 9$ and $d(t_3) = 21$, and their utility functions are given, respectively, by:

$$u_2(\tau_2) = \begin{cases} 3 & \text{if } \tau_2 \leq 9, \\ \dfrac{9}{2} - \dfrac{\tau_2}{6} & \text{if } 9 \leq \tau_2 \leq 27, \\ 0 & \text{if } \tau_2 \geq 27. \end{cases} \qquad u_3(\tau_3) = \begin{cases} 2 & \text{if } \tau_3 \leq 21, \\ 16 - \dfrac{2\tau_3}{3} & \text{if } 21 \leq \tau_3 \leq 24, \\ 0 & \text{if } \tau_3 \geq 24. \end{cases}$$

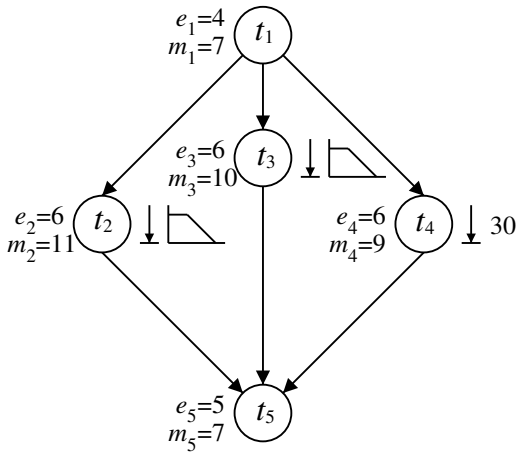

Figure 2: Motivational example

When we consider the expected duration for every task, that is $|t| = e(t)$ for each $t \in T$, the schedule $\sigma_a = t_1 t_2 t_3 t_4 t_5$ is the one that maximizes the total utility: the completion times for tasks t_2 and t_3 are $\tau_2 = 10$ and $\tau_3 = 16$, and the total utility is $U_a = u_2(10) + u_3(16) = 17/6 + 2 \approx 4.83$. However, σ_a does not guarantee the satisfaction of hard deadlines (take the possible scenario where $|t_1| = 7$, $|t_2| = |t_3| = 10$, $|t_4| = 8$: in such a case $\tau_4 = 35$ and therefore t_4 misses its deadline).

We aim to find the schedule that maximizes the total utility (sum of individual contributions by soft tasks), while guaranteeing that hard deadlines are met in all possible scenarios. When we consider only the upper bounds of execution time for every task, that is $|t| = m(t)$ for each $t \in T$, we obtain the schedule $\sigma_b = t_1 t_3 t_4 t_2 t_5$ which maximizes the total utility when every task takes its maximum duration and, at the same time, guarantees no hard deadline miss.

Although σ_b ensures that hard deadlines are always satisfied, it gives the maximum utility in the particular case of WCET for all tasks, a situation that, though possible, seldom occurs. It is better to find the schedule that yields the maximum utility in the more likely case of expected duration for all tasks, yet guaranteeing no hard deadline miss. Thus we must obtain the schedule that guarantees meeting all hard deadlines, when maximum duration is considered, *and* maximizes the total utility, when expected duration is considered. Such a schedule for the example of Figure 2 is $\sigma_c = t_1 t_2 t_4 t_3 t_5$. When every task lasts its expected duration, following σ_c, t_2 completes at $\tau_2 = 10$ and t_3 completes at $\tau_3 = 22$, and thus the total utility is $U_c = u_2(10) + u_3(22) = 17/6 + 4/3 \approx 4.17$. Note that in the same case (expected duration for all tasks) $\sigma_b = t_1 t_3 t_4 t_2 t_5$ yields a utility

$U_b = u_2(22) + u_3(10) = 5/6 + 2 \approx 2.83.$

4 Problem Formulation and NP-completeness

We have informally described the problem of scheduling real-time systems that have both soft and hard tasks. We want to find the schedule (an execution sequence for tasks) that, among all schedules that respect the hard constraints in the *worst-case*, maximizes the total utility when tasks last their *expected* duration. In this section we precisely formulate the problem and demonstrate that it is **NP**-complete.

SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY (SSHMU):
Given
· a set T of tasks,
· a directed acyclic graph $G = (T, E)$ defining precedence constraints for the tasks,
· a maximum duration $m(t) \in \mathbb{N}$ for each task $t \in T$,
· an expected duration $e(t) \in \mathbb{N}$ for each task $t \in T$ ($e(t) \leq m(t)$),
· a subset $H \subseteq T$ of hard tasks,
· a deadline $d(h) \in \mathbb{N}$ for each hard task $h \in H$,
· a subset $S \subseteq T$ of soft tasks ($S \cap H = \emptyset$),
· a non-increasing utility function $u_j(\tau_j)$ for each soft task $s_j \in S$ (τ_j is the completion time of s_j), and
· a positive integer K;
does there exist a one-processor schedule σ (a bijection $\sigma : T \to \{1, 2, \ldots, |T|\}$) such that
· $\sigma(t) < \sigma(t')$ for all $\langle t, t' \rangle \in E$,
· $\tau_i^m \leq d(h_i)$ for all $h_i \in H$, where τ_i^m is the completion time[1] of task h_i when every task $t \in T$ lasts its maximum duration $m(t)$, and
· $\sum_{s_j \in S} u_j(\tau_j^e) \geq K$, where τ_j^e is the completion time[2] of task s_j when every

[1] τ_i^m is given by

$$\tau_i^m = \begin{cases} m(t_i) & \text{if } \sigma(t_i) = 1, \\ \tau_k^m + m(t_i) & \text{if } \sigma(t_i) = \sigma(t_k) + 1. \end{cases}$$

[2] τ_j^e is given by

$$\tau_j^e = \begin{cases} e(t_j) & \text{if } \sigma(t_j) = 1, \\ \tau_k^e + e(t_j) & \text{if } \sigma(t_j) = \sigma(t_k) + 1. \end{cases}$$

task $t \in T$ lasts its expected duration $e(t)$?

In the following we prove that the problem of scheduling with soft and hard tasks to maximize utility is **NP**-complete, first by showing that it is in **NP** (**NP** is the class of all languages that are decided by a polynomially bounded nondeterministic Turing machine), and second by showing that it is **NP**-hard (a problem Π is **NP**-hard iff any **NP** problem is polynomially reducible to Π).

Theorem 1. SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY is in **NP**.
Proof: We provide a polynomial-time verifier for SSHMU. Given an instance $\{T, G(T,E), m : T \to \mathbb{N}, e : T \to \mathbb{N}, H \subseteq T, d : H \to \mathbb{N}, S \subseteq T, \{u_1(\tau_1), u_2(\tau_2), \ldots, u_{|S|}(\tau_{|S|})\}, K\}$ of SSHMU and a schedule σ, we first check that the precedence constraints are satisfied, in other words, $\sigma(t) < \sigma(t')$ for each edge $\langle t, t' \rangle \in E$; this can clearly be accomplished in polynomial time.

Second, we check that all hard deadlines are met when all tasks last their maximum duration: for each task $t_k \in T$ we compute τ_k^m (as given above[1]), and then for each $h_i \in H$ we check that $\tau_i^m \leq d(h_i)$; computing τ_k^m as well as the check can be done in polynomial time.

Third, we check that the total utility (sum of utilities given by each function $u_j(\tau_j)$ of soft tasks s_j), when all tasks last their expected duration, is at least K: we compute τ_j^e (as given above[2]), calculate the individual values $u_j(\tau_j^e)$ for each $s_j \in S$, sum them, and check that the total utility is at least K; this can also be accomplished in polynomial time.

If any check fails then reject; else accept. ■

Before proving that SSHMU is **NP**-hard, for the sake of clarity, we first show that the following problem (called SCHEDULING TO MAXIMIZE UTILITY) is **NP**-complete.

SCHEDULING TO MAXIMIZE UTILITY (SMU): Given a set T of tasks, a directed acyclic graph $G = (T, E)$, a duration $e(t) \in \mathbb{N}$ for each task $t \in T$, a non-increasing utility function $u_j(\tau_j)$ for each task $t_j \in T$, and a positive integer K; does there exist a one-processor schedule σ (a bijection $\sigma : T \to \{1, 2, \ldots, |T|\}$) respecting the precedence constraints such that $\sum_{t_j \in T} u_j(\tau_j) \geq K$, where τ_j is the completion time[2] of task t_j?

In order to prove the **NP**-completeness of SMU, we transform a known **NP**-complete problem to an instance of SMU. We have selected SCHEDULING TO MINIMIZE WEIGHTED COMPLETION TIME (SMWCT) [5] for this purpose. The formulation of SMWCT is shown below.

SCHEDULING TO MINIMIZE WEIGHTED COMPLETION TIME (SMWCT): Given a set T of tasks, a partial order $\lessdot$ on T, a duration $e(t) \in \mathbb{N}$ and a weight $w(t) \in \mathbb{N}$ for each task $t \in T$, and a positive integer K; does there exist a one-processor schedule σ (a bijection $\sigma : T \to \{1, 2, \ldots, |T|\}$) respecting the precedence constraints imposed by $\lessdot$ such that $\sum_{t_j \in T} w(t_j)\tau_j \leq K$, where τ_j is the completion time[2] of task t_j?

Theorem 2. SCHEDULING TO MAXIMIZE UTILITY is **NP**-complete.
Proof: SMU is in **NP**: The proof is very similar to the one presented above showing that SSHMU is in **NP** and, therefore, omitted for brevity.

SMU is **NP**-hard: We transform SCHEDULING TO MINIMIZE WEIGHTED COMPLETION TIME (known to be **NP**-complete) to SCHEDULING TO MAXIMIZE UTILITY. Let $\Pi = \{T, \lessdot, e : T \to \mathbb{N}, w : T \to \mathbb{N}, K\}$ be an arbitrary instance of SMWCT. We construct an instance $\Pi' = \{T', G(T', E'), e' : T' \to \mathbb{N}, \{u'_1(\tau'_1), u'_2(\tau'_2), \ldots, u'_{|T'|}(\tau'_{|T'|})\}, K'\}$ of SMU as follows:
· $T' = T$,
· $\langle t_i, t_j \rangle \in E'$ iff $t_i \lessdot t_j$,
· $e'(t) = e(t)$ for each $t \in T$,
· the utility function $u'_j(\tau'_j)$ for each $t_j \in T$ is defined as $u'_j(\tau'_j) = w(t_j)(C - \tau'_j)$, where $C = \sum_{t \in T} e(t)$,
· $K' = (C \sum_{t \in T} w(t)) - K$.

To see that this transformation can be performed in polynomial time, it suffices to observe that T', $e' : T' \to \mathbb{N}$, and K' can be obtained in $O(|T|)$ time, $G(T', E')$ can be constructed in $O(|T| + |\lessdot|)$ time, and all the utility functions $u'_j(\tau'_j)$ can be obtained in $O(|T|)$ time. What remains to be shown in order to prove the **NP**-hardness of SMU is that Π has a schedule for which $\sum_{t_j \in T} w(t_j)\tau_j$ is K or less if and only if Π' has a schedule for which $\sum_{t'_j \in T'} u'_j(\tau'_j)$ is K' or greater.

We show that the schedule that minimizes $\sum_{t_j \in T} w(t_j)\tau_j$ for Π is exactly the one that maximizes $\sum_{t'_j \in T'} u'_j(\tau'_j)$ for Π'. Note that, due to the transformation we described above, the set of tasks is the same for Π and Π', and the precedence constraints for tasks is precisely the same in both cases. Assume that σ is a schedule respecting the precedence constraints in Π that minimizes $\sum_{t_j \in T} w(t_j)\tau_j$ and that K is such a minimum. Observe that σ also respects the precedence constraints in Π'. Moreover, since $e'(t) = e(t)$

for each $t \in T$, the completion time τ'_j of every task t_j, when we use σ as schedule in Π', is the very same as τ_j and thus:

$$\begin{aligned}\sum_{t'_j \in T'} u'_j(\tau'_j) &= \sum_{t_j \in T} u'_j(\tau_j) \\ &= \sum_{t_j \in T} w(t_j)(C - \tau_j) \\ &= C \sum_{t_j \in T} w(t_j) - \sum_{t_j \in T} w(t_j)\tau_j\end{aligned}$$

Since $C \sum_{t \in T} w(t)$ is a constant value that does not depend on σ and $\sum_{t_j \in T} w(t_j)\tau_j = K$ is the minimum for Π, we conclude that $(C \sum_{t \in T} w(t)) - K = K'$ is the maximum for Π', in other words, σ maximizes $\sum_{t'_j \in T'} u'_j(\tau'_j)$. Hence SMU is **NP**-hard.

Finally, SMU is **NP**-complete because it is in **NP** and it is **NP**-hard. ■

Once we have proved that SCHEDULING TO MAXIMIZE UTILITY is **NP**-complete, it is simple to show that SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY is **NP**-hard.

Theorem 3. SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY is **NP**-hard.
Proof: We prove by restriction that SSHMU is **NP**-hard. A problem Π' is proved **NP**-hard by restriction by showing that Π' contains a known **NP**-complete problem Π as a special case [5]. Thus SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY ($\Pi' = \{T, G(T,E), m : T \to \mathbb{N}, e : T \to \mathbb{N}, H \subseteq T, d : H \to \mathbb{N}, S \subseteq T, \{u_1(\tau_1), u_2(\tau_2), \ldots, u_{|S|}(\tau_{|S|})\}, K\}$ is shown to be **NP**-hard by restricting its instances to cases in which $S = T$, $H = \emptyset$, and $m(t) = e(t)$ for each $t \in T$, thereby obtaining a problem identical to SCHEDULING TO MAXIMIZE UTILITY ($\Pi' = \{T, G(T,E), e : T \to \mathbb{N}, \{u_1(\tau_1), u_2(\tau_2), \ldots, u_{|T|}(\tau_{|T|})\}, K\}$). ■

Theorem 4. SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY is **NP**-complete.
Proof: SSHMU is in **NP** and is **NP**-hard, therefore it is **NP**-complete. ■

5 Exact Algorithm

We have proved that SCHEDULING WITH SOFT AND HARD TASKS TO MAXIMIZE UTILITY is an **NP**-complete problem. Therefore, unless **NP** = **P**,

there is no algorithm that solves every instance of the problem in polynomial time. This section presents an exact algorithm for SSHMU whose time complexity is $O(|T|^3|H||S|!)$.

The algorithm presented in Figure 3 computes the schedule that maximizes the total utility when tasks last their expected duration, while guaranteeing that all hard deadlines are met even when all tasks last their maximum duration. Initially we check, by using the algorithm IsSchedulable as presented in Figure 6 and explained later in this section, whether there exists at all a schedule that satisfies the hard time constraints. Note that if the system is not schedulable, the algorithm OptimalSchedule returns ϵ. For each one of the possible permutations S_k of soft tasks, the algorithm first checks whether S_k is valid (that is, the order given by S_k does not violate data dependencies) and, if it is so, the algorithm computes the best schedule σ_k (the one that yields the highest total utility) for the particular order for soft tasks as expressed by S_k. The schedule σ that, among all σ_k, provides the highest total utility when considering the expected duration for all tasks is the optimal one.

```
Algorithm OptimalSchedule()
output: The optimal schedule σ
begin
  σ := ε
  util := −∞
  if IsSchedulable(ε) then
    for k ← 1, 2, ... , |S|! do
      if IsValidPerm(S_k) then
        σ_k := BestSchedule(S_k)
        util_k := Σ_{s_j ∈ S} u_j(τ_j^e)
        if util_k > util then
          σ := σ_k
          util := util_k
        end if
      end if
    end for
  end if
end
```

Figure 3: Algorithm OptimalSchedule

In order to examine whether a permutation S of soft tasks defines a feasible schedule, as shown in the algorithm of Figure 4, we check if there exists

a path from the soft task $\mathsf{S}_{[j]}$ to the soft task $\mathsf{S}_{[i]}$, $j > i$: if so, S is not valid. In Figure 4, $\mathcal{P}$ denotes the path relation ($\mathcal{P} = \{(t, t') \in T \times T \mid$ there is a path leading from t to $t'\}$). $\mathcal{P}$ corresponds to the reflexive transitive closure of the relation E (set of edges in the task graph) and is computed only once for a given system.

Algorithm IsValidPerm(S)
input: A vector S containing a permutation of soft tasks
output: A boolean *valid* indicating whether it is possible to obtain a schedule where the soft tasks obey the order given by the permutation S

```
begin
  valid := true
  for i ← 1, 2, ... , |S| − 1 do
    for j ← i + 1, i + 2, ... , |S| do
      if (S[j], S[i]) ∈ P then
        valid := false
      end if
    end for
  end for
end
```

Figure 4: Algorithm IsValidPerm

The algorithm that computes the best schedule, for a given permutation of soft tasks S, is presented in Figure 5. The rationale is that the maximum total utility for the particular permutation S is obtained when the soft tasks are set in the schedule as early as possible respecting the order given by S. Let us consider again the example given in Figure 2. There are two permutations of soft tasks $\mathsf{S}_1 = [t_2, t_3]$ and $\mathsf{S}_2 = [t_3, t_2]$. The schedules that obey the order for soft tasks given by the permutation S_1 (and also the precedence constraints imposed by the task graph) are $\sigma_1 = t_1 t_2 t_4 t_3 t_5$, $\sigma_1' = t_1 t_4 t_2 t_3 t_5$, and $\sigma_1'' = t_1 t_2 t_3 t_4 t_5$. Note, first of all, that σ_1'' implies potential hard deadlines misses and therefore cannot be considered. Both σ_1 and σ_1' guarantee that hard deadlines are always met but σ_1 is better from the perspective of higher total utility. $\sigma_1 = t_1 t_2 t_4 t_3 t_5$ is the schedule that sets soft tasks as early as possible (guaranteeing hard deadlines) respecting the order given by $\mathsf{S}_1 = [t_2, t_3]$.

A simple proof of the fact that by setting soft tasks as early as possible according to the order given by S we get the maximum total utility for S is as follows: let σ be the schedule that respects the order of soft tasks given by S (that is, $1 \leq i < j \leq |\mathsf{S}| \Rightarrow \sigma(\mathsf{S}_{[i]}) < \sigma(\mathsf{S}_{[j]})$) and such that soft tasks are

set as early as possible (that is, for every schedule σ', different from σ, that obeys the order of soft tasks given by S and respects all hard deadlines in the worst-case, $\sigma'(\mathsf{S}_{[i]}) > \sigma(\mathsf{S}_{[i]})$ for some $1 \leq i \leq |\mathsf{S}|$). Take one such σ'. For at least one soft task $s_j \in S$ it holds $\sigma'(s_j) > \sigma(s_j)$, therefore $\tau'^e_j > \tau^e_j$ (τ'^e_j is the completion time of s_j when we use σ' as schedule while τ^e_j is the completion time of s_j when σ is used as schedule, considering in both cases expected duration for all tasks). Thus $u_j(\tau'^e_j) \leq u_j(\tau^e_j)$ because utility functions for soft tasks are non-increasing. Consequently $U' \leq U$, where U' and U are the total utility when using, respectively, σ' and σ as schedules. Hence we conclude that no schedule σ', which respects the order for soft tasks given by S, will yield a total utility greater than the one by σ.

Algorithm BestSchedule(S)
input: A vector S containing a permutation of soft tasks
output: The best schedule σ for which soft tasks obey the order given by the permutation S

---

begin
  $Ready := \{t \in T \mid {}^\circ t = \emptyset\}$
  $\sigma := \epsilon$
  $cnt := 1$
  while $Ready \neq \emptyset$ **do**
    $A := \{t \in Ready \mid \text{IsSchedulable}(\sigma t)\}$
    $B := \{t \in Ready \mid (t, \mathsf{S}_{[cnt]}) \in \mathcal{P}\}$
    if $A \cap B = \emptyset$ **then**
      select $\bar{t} \in A$
    else
      select $\bar{t} \in A \cap B$
    end if
    if $\bar{t} = \mathsf{S}_{[cnt]}$ **then**
      $cnt := cnt + 1$
    end if
    $\sigma := \sigma\bar{t}$
    $Ready := Ready \setminus \{\bar{t}\} \cup \{t \in \bar{t}^\circ \mid \text{all } q \in {}^\circ t \text{ are in } \sigma\}$
  end while
end

Figure 5: Algorithm BestSchedule

The algorithm BestSchedule(S) first tries to schedule the soft task $\mathsf{S}_{[1]}$ as early as possible. In order to do so, it will set in first place all tasks from which there exists a path leading to $\mathsf{S}_{[1]}$, taking care of not incurring potential

deadlines misses by the hard tasks. Then, a similar procedure is followed for $\mathsf{S}_{[2]}, \mathsf{S}_{[3]}, \ldots, \mathsf{S}_{[|S|]}$.

The algorithm BESTSCHEDULE(S) keeps a list *Ready* of tasks that are available at every step and constructs the schedule by progressively concatenating tasks to the string σ (initially $\sigma = \epsilon$). In Figure 5, A is the set of available tasks that, at that step, can be added to σ without posing the risk of hard deadline misses. In other words, if we added a task $t \in Ready \setminus A$ to σ we could no longer guarantee that all hard constraints are met. B is the set of available tasks that have a path to the next soft task $\mathsf{S}_{[cnt]}$ to be scheduled. Once an available task $\bar{t}$ is selected, it is concatenated to σ ($\sigma := \sigma\bar{t}$), $\bar{t}$ is removed from *Ready*, and all its successors that become available are added to *Ready*.

At every iteration of the **while** loop of the algorithm given in Figure 5, we must construct the set A by checking, for every $t \in Ready$, whether concatenating t to the schedule prefix σ would imply a possible violation of a hard deadline. For this purpose we use the algorithm ISSCHEDULABLE(ς) shown in Figure 6. This algorithm, in turn, makes use of SLACK(ς) (Figure 7). The algorithm SLACK(ς) returns the slack time (deadline minus completion time) of hard tasks, when using a schedule that agrees with the prefix ς and for which hard tasks are set as early as possible, and considering maximum duration for all tasks. This means that if some of the $|H|$ elements of the vector HS = SLACK(ς) is negative, there is no schedule having ς as prefix that guarantees hard deadline satisfaction. For the algorithm of Figure 7, we have assumed that the hard tasks in H are ordered according to their deadline, that is, $d(h_i) \leq d(h_j)$ for $1 \leq i < j \leq |H|$.

Algorithm ISSCHEDULABLE(ς)
input: A schedule prefix ς
output: A boolean *schedulable* indicating whether there exists a a schedule that agrees with the prefix ς and such that all hard deadlines are met

---

```
begin
  schedulable := true
  HS := SLACK(ς)
  for i ← 1, 2, ..., |H| do
    if HS[i] < 0 then
      schedulable := false
    end if
  end for
end
```

Figure 6: Algorithm ISSCHEDULABLE

Algorithm SLACK(ς)
input: A schedule prefix ς
output: A vector HS with the slack time for every hard task

```
begin
  A := {t ∈ T | t is in the schedule prefix ς}
  B := ∅
  for i ← 1, 2, ..., |H| do
    if h_i ∈ A then
      HS_[i] := d(h_i) − τ_i^m
    else
      for j ← 1, 2, ..., |T| do
        if (t_j, h_i) ∈ P and t_j ∉ A then
          B := B ∪ {t_j}
        end if
      end for
      HS_[i] := d(h_i) − Σ_{t∈A∪B} m(t)
    end if
  end for
end
```

Figure 7: Algorithm SLACK

6 Heuristics

In this section we present several heuristic procedures for finding a near-optimal solution to the problem of scheduling with soft and hard tasks to maximize utility as formulated in Section 4.

The algorithms progressively construct the schedule σ by concatenating tasks to the string σ that at the end will contain the final schedule. All the heuristics that we propose in this section make use of the list *Ready* of available tasks at every step. The heuristics differ in how the next task, among those in *Ready*, is selected as the one to be concatenated to σ. Note that the algorithms presented in this section are applicable only if the system is schedulable in first place (there exists a schedule that satisfies the hard time constraints).

The algorithms make use of a list scheduling heuristic. The basic algorithm is shown in Figure 8. Initially, $\sigma = \epsilon$ (the empty string) and the list *Ready* contains those tasks that have no predecessor. The set A contains the tasks that are in σ (initially $A := \emptyset$). The **while** loop is executed exactly $|T|$ times. At every iteration we compute the set B of ready tasks that do

not pose risk of hard deadline misses by being concatenated to the schedule prefix σ. If all soft tasks have already been set in σ we select any $\bar{t} \in B$, else we compute a priority for soft tasks ($\mathsf{SP} := \textsc{Priority}(\sigma)$). The way such priorities are calculated is what differentiates the heuristics proposed in this report. Among those soft tasks that are not in σ, we select s_k as the one with the highest priority. Then, we compute the set C of tasks that cause no hard deadline miss and that have a path leading to s_k. We select any $\bar{t} \in C$ if $C \neq \emptyset$, else we choose any $\bar{t} \in B$. Once an available task $\bar{t}$ is selected as described above, it is concatenated to σ, $\bar{t}$ is added to A, $\bar{t}$ is removed from the list *Ready*, and those successors of $\bar{t}$ that become available are added to *Ready*.

Algorithm BasicHeuristic()
output: A near-optimal schedule σ

---

begin
  $Ready := \{t \in T \mid {}^\circ t = \emptyset\}$
  $\sigma := \epsilon$
  $A := \emptyset$
  while $Ready \neq \emptyset$ **do**
    $B := \{t \in Ready \mid \textsc{IsSchedulable}(\sigma t)\}$
    if $S \backslash A = \emptyset$ **then**
      select $\bar{t} \in B$
    else
      $\mathsf{SP} := \textsc{Priority}(\sigma)$
      select $s_k \in S \backslash A$ such that $\mathsf{SP}_{[k]} \geq \mathsf{SP}_{[i]}$ for all $s_i \in S \backslash A$
      $C := \{t \in B \mid (t, s_k) \in \mathcal{P}\}$
      if $C \neq \emptyset$ **then**
        select $\bar{t} \in C$
      else
        select $\bar{t} \in B$
      end if
    end if
    $\sigma := \sigma \bar{t}$
    $A := A \cup \{\bar{t}\}$
    $Ready := Ready \setminus \{\bar{t}\} \cup \{t \in \bar{t}^\circ \mid \text{all } q \in {}^\circ t \text{ are in } \sigma\}$
  end while
end

Figure 8: Basic heuristic

The first of the proposed heuristics makes use of the basic algorithm presented in Figure 8 and the algorithm given in Figure 9 for computing the

priorities of soft tasks. The procedure PRIORITYMAXUTILITY(ς) assigns a priority to soft tasks, for a given schedule prefix ς, as follows: if s_i is in ς, its priority is $\mathsf{SP}_{[i]} := -\infty$; if s_i is not in ς, we compute the earliest completion time $\tau_i^{e'}$ when considering expected duration for all tasks. Then we make use of the maximum utility M_i (see Figure 1) for s_i in order to calculate $\mathsf{SP}_{[i]} := M_i/\tau_i^{e'}$.

Algorithm PRIORITYMAXUTILITY(ς)
input: A schedule prefix ς
output: A vector SP containing the priority for soft tasks

---

begin
  $A := \{t \in T \mid t \text{ is in } \varsigma\}$
  for $i \leftarrow 1, 2, \ldots, |S|$ **do**
    if $s_i \in A$ **then**
      $\mathsf{SP}_{[i]} := -\infty$
    else
      $B := \{t \in T \backslash A \mid (t, s_i) \in \mathcal{P}\}$
      $\tau_i^{e'} := \sum_{t \in A \cup B} e(t)$
      $\mathsf{SP}_{[i]} := M_i/\tau_i^{e'}$
    end if
  end for
end

Figure 9: Algorithm PRIORITYMAXUTILITY

The algorithm PRIORITYMAXUTILITY makes use of $M_i = u_i(0)$ for every soft task $s_i \in S$ but does not exploit the transition functions $u_i(\tau_i)$ themselves. Our second heuristic procedure relies on the algorithm PRIORITYSINGLEUTILITY(ς) (Figure 10) for computing the priorities of soft tasks. If s_i is in ς, $\mathsf{SP}_{[i]} := -\infty$, else we compute $\tau_i^{e'}$ (it corresponds to the completion time, considering expected durations, of s_i in a schedule that agrees with the prefix ς and for which s_i is set the earliest). Then we assign the priority $\mathsf{SP}_{[i]}$ as the single utility of s_i evaluated at $\tau_i^{e'}$.

The algorithm PRIORITYTOTALUTILITY(ς) shown in Figure 11 also exploits the information of utility functions but, as opposed to PRIORITYSINGLEUTILITY, it considers the utility contributions of other soft tasks when computing the priority $\mathsf{SP}_{[i]}$ of the soft task s_i. If the soft task s_i is not in ς its priority is computed as follows. First, we obtain the completion time $\tau_i^{e'}$ when s_i is earliest set in a schedule that agrees with the prefix ς, using expected durations. Second, for each soft task s_j different from s_i that is not in ς, we compute $\tau_j^{e'}$ and $\tau_j^{e''}$. The former corresponds to the completion time

Algorithm PRIORITYSINGLEUTILITY(ς)
input: A schedule prefix ς
output: A vector SP containing the priority for soft tasks

begin
  $A := \{t \in T \mid t \text{ is in } \varsigma\}$
  for $i \leftarrow 1, 2, \ldots, |S|$ **do**
    if $s_i \in A$ **then**
      $\mathsf{SP}_{[i]} := -\infty$
    else
      $B := \{t \in T \backslash A \mid (t, s_i) \in \mathcal{P}\}$
      $\tau_i^{e'} := \sum_{t \in A \cup B} e(t)$
      $\mathsf{SP}_{[i]} := u_i(\tau_i^{e'})$
    end if
  end for
end

Figure 10: Algorithm PRIORITYSINGLEUTILITY

when s_i is earliest set in a schedule that agrees with the prefix ς. The latter corresponds to the completion time when s_i is latest set in a schedule that agrees with ς. In both cases, expected duration of tasks are considered. The average of $\tau_j^{e'}$ and $\tau_j^{e''}$ is used as argument for the utility function u_j. Thus the priority of s_i is given by $\mathsf{SP}_{[i]} := u_i(\tau_i^{e'}) + \sum_{s_j \in S \backslash (A \cup \{s_i\})} u_j((\tau_j^{e'} + \tau_j^{e''})/2)$.

To sum up this section, we have presented three heuristics aimed to find near-optimal solutions to the problem of scheduling with soft and hard tasks to maximize utility. Such heuristics are based on the algorithm of Figure 8 and their difference lies in how the priorities for soft tasks are calculated. The first heuristic uses PRIORITYMAXUTILITY (Figure 9), the second uses PRIORITYSINGLEUTILITY (Figure 10), and the third one uses PRIORITY-TOTALUTILITY (Figure 11).

We have named the heuristics after the algorithms they use for computing priorities: MAXUTILITY (MU), SINGLEUTILITY (SU), and TOTALUTILITY (TU), respectively. The first two have a time complexity $O(|T|^3(|H| + |S|))$ whereas the third one has a time complexity $O(|T|^3(|H| + |S|^2))$.

7 Experimental Results

In this section we experimentally evaluate the heuristics proposed in Section 6. We are initially interested in the quality of the schedules obtained by the heuristics MAXUTILITY (MU), SINGLEUTILITY (SU), and TOTALUTILITY

Algorithm PRIORITYTOTALUTILITY(ς)
input: A schedule prefix ς
output: A vector SP containing the priority for soft tasks

```
begin
  A := {t ∈ T | t is in ς}
  for i ← 1, 2, ... , |S| do
    if s_i ∈ A then
      SP_[i] := −∞
    else
      B := {t ∈ T\A | (t, s_i) ∈ P}
      τ_i^e' := Σ_{t∈A∪B} e(t)
      total := u_i(τ_i^e')
      for j ← 1, 2, ... , |S| do
        if s_j ≠ s_i and s_j ∉ A then
          C := {t ∈ T\A | (t, s_j) ∈ P}
          D := {t ∈ T\{s_j} | (s_j, t) ∈ P}
          τ_j^e' := Σ_{t∈A∪C} e(t)
          τ_j^e'' := Σ_{t∈T\D} e(t)
          total := total + u_j((τ_j^e' + τ_j^e'')/2)
        end if
      end for
      SP_[i] := total
    end if
  end for
end
```

Figure 11: Algorithm PRIORITYTOTALUTILITY

(TU) with respect to the optimal schedule as given by the exact algorithm OPTIMALSCHEDULE. We use as criterion the deviation dev given by:

$$dev = \frac{U_{opt} - U_{heur}}{U_{opt}}$$

where U_{opt} is the total utility corresponding to the optimal schedule and U_{heur} is the total utility corresponding to the schedule obtained with a heuristic.

We have randomly generated a large number of tasks graphs in our experiments. We initially considered graphs with 100, 200, 300, 400, 500, and 600 tasks. For these, we considered systems with 2, 3, 4, 5, 6, 7, and 8 soft tasks. For the case $|T|$=200 tasks, we considered systems with 25, 50, 75, 100, and 125 hard tasks. We generated 500 graphs for each graph dimension. Expected and maximum durations of tasks were also assigned randomly. For

every task graph, hard tasks and their deadlines were selected randomly as well as soft tasks and their utility functions. All the experiments were run on a Sun Ultra 10 workstation.

We have plotted the average deviation as a function of the number of tasks in Figures 12, 13, and 14. These correspond to systems with 3, 5, and 8 soft tasks respectively. All the systems considered in Figures 12, 13, and 14 have 50 hard tasks. These plots consistently show that heuristic TOTALUTILITY (TU) gives the best results for the considered cases.

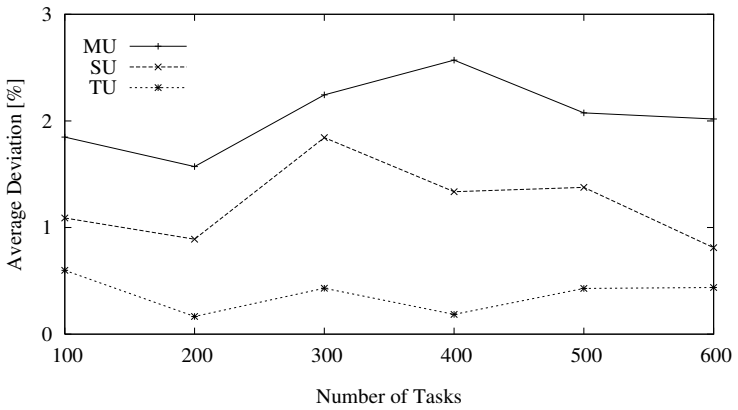

Figure 12: Evaluation of the heuristics (50 hard tasks, 3 soft tasks)

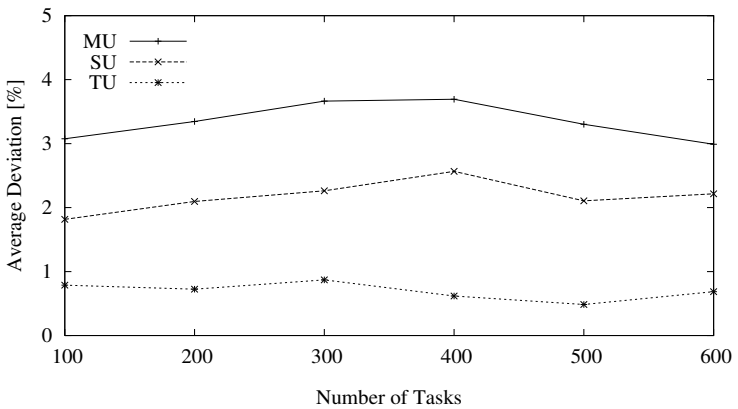

Figure 13: Evaluation of the heuristics (50 hard tasks, 5 soft tasks)

The plot in Figure 15 depicts the average deviation as a function of the number of hard tasks. In this case, we have considered systems with 200 tasks, out of which 5 are soft. In this graph we observe that the number

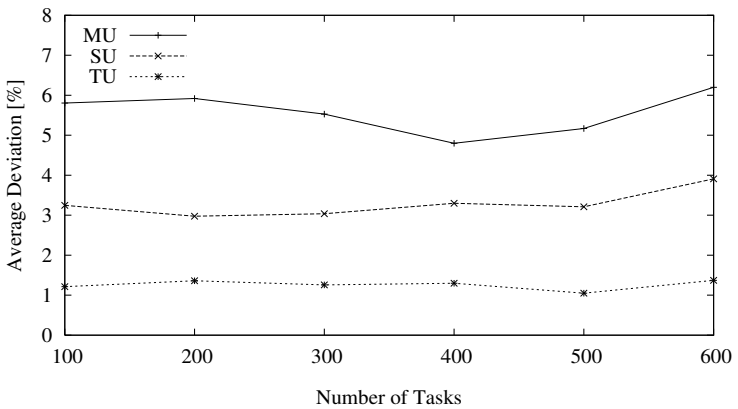

Figure 14: Evaluation of the heuristics (50 hard tasks, 8 soft tasks)

of hard tasks does not affect significantly the quality the schedules obtained with the proposed heuristics.

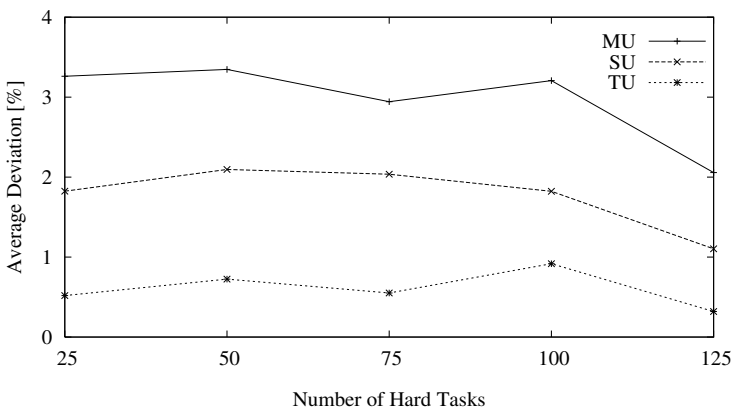

Figure 15: Evaluation of the heuristics (200 tasks, 5 soft tasks)

We have also studied the average deviation as a function of the number of soft tasks and the results are plotted in Figure 16. The considered systems have 100 tasks, 50 of them being hard. We again see that the heuristic TU consistently provides the best results. We can also note that there is a trend showing an increasing average deviation as the number of soft tasks grows, especially for the heuristics MU and SU.

In Section 6 we pointed out that the worst-case time complexity of the algorithms MaxUtility and SingleUtility is $O(|T|^3(|H|+|S|))$ and that one of TotalUtility is $O(|T|^3(|H|+|S|^2))$. In order to give a quantitative idea of the execution times of these heuristics and the exact algorithm we

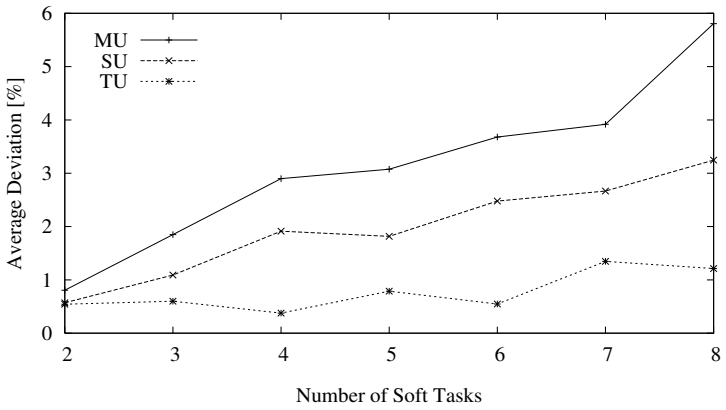

Figure 16: Evaluation of the heuristics (100 tasks, 50 hard tasks)

used throughout this section, in Table 1 we present the average running time for these algorithms in the case of systems containing 100 tasks out of which 50 are hard.

Num. Soft Tasks	Average Execution Time [s]			
	Exact	MU	SU	TU
2	0.085	0.051	0.051	0.052
3	0.237	0.053	0.052	0.053
4	0.879	0.053	0.053	0.055
5	3.623	0.055	0.055	0.058
6	20.78	0.056	0.056	0.059
7	115.36	0.057	0.058	0.061
8	896.36	0.059	0.059	0.063

Table 1: Average execution times (100 tasks, 50 hard tasks)

Note that, in the experiments we have presented so far, the number of soft tasks is small. Recall that the time complexity of the exact algorithm is $O(|T|^3|H||S|!)$ and therefore any comparison that requires computing the optimal schedule is infeasible for a large number of soft tasks.

In a second set of experiments, we have compared the heuristics among themselves considering systems with larger numbers of soft and hard tasks. We normalize the utility obtained the heuristics with respect to the utility given by the algorithm TotalUtility (TU):

$$\|U_{heur}\| = \frac{U_{heur}}{U_{TU}}$$

We generated, for these experiments, graphs with 500 tasks and consid-

ered cases with 50, 100, 150, 200, and 250 hard tasks and 50, 100, 150, 200, and 250 soft tasks. The results are shown in Figures 17 and 18. We can note that for systems with many soft tasks, the algorithm SINGLEUTILITY gives results very close to those of the algorithm TOTALUTILITY. In the particular case T=500, H=150, S=200, SU slightly outperforms TU ($\|U_{SU}\| = 1.0014$, $\|U_{TU}\| = 1$).

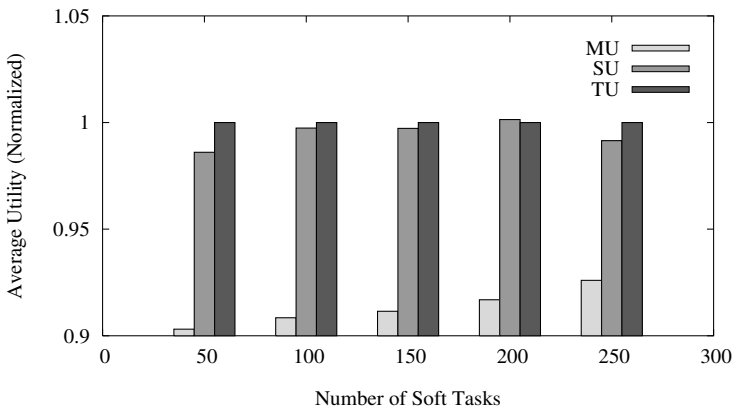

Figure 17: Comparison among the heuristics (500 tasks, 150 hard tasks)

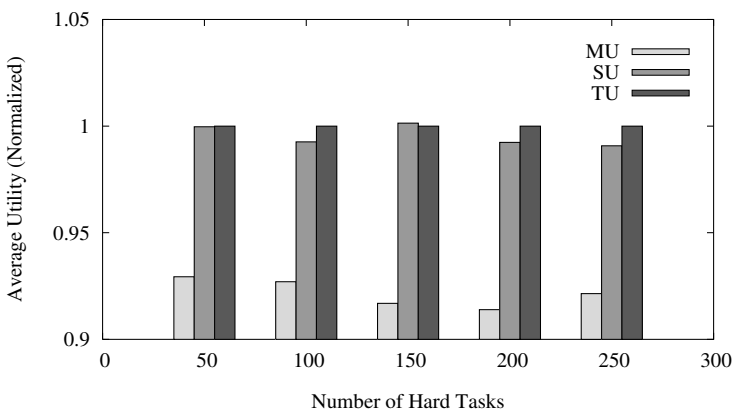

Figure 18: Comparison among the heuristics (500 tasks, 200 soft tasks)

From the extensive set of experiments that we have performed and its results (Figures 12 through 18) we conclude, that if one of the proposed heuristics is to be chosen, TOTALUTILITY is the procedure that should be used for solving the problem of scheduling with soft and hard tasks to maximize utility as formulated in Section 4. This does not mean that TOTALUTILITY gives the best results in every case, but in average it performs better. Since

the proposed heuristics are computationally cheap, we could run all three and choose, among their results, the schedule that yields the highest total utility.

8 Conclusions

We have presented an approach to the problem of static scheduling of real-time systems that have hard and soft tasks. Our approach considers that hard, soft, and non-real-time tasks are periodic and they all are mapped into a single processor.

We made use of non-increasing utility functions to represent the relevance of soft tasks and how the quality of results is diminished when missing a soft deadline. The problem we have addressed is thus that one of finding the execution order of tasks in such a way that the sum of individual utilities of soft tasks is maximum and, at the same time, there is guarantee that no hard deadline will be missed. We used maximum duration of tasks for guaranteeing hard deadlines and expected duration of tasks for calculating the total utility.

We have showed that the problem of scheduling with soft and hard tasks to maximize utility is **NP**-complete. We proposed an exact algorithm that gives the optimal schedule in $O(|T|^3|H||S|!)$ time. We also presented three heuristic procedures that find near-optimal solutions in short time.

We have randomly generated 17500 task graphs for experimental evaluation. The experiments showed that the heuristic TOTALUTILITY is the one that gives the best results in average. Its time complexity ($O(|T|^3(|H|+|S|^2))$) is however larger than the one of the other two heuristics ($O(|T|^3(|H|+|S|))$). In the cases where it was feasible to compute the optimal schedule (up to 8 soft tasks), we obtained an average deviation smaller than 2% when using the heuristic TOTALUTILITY.

As part of our future work, we are currently studying the issue of quasi-static scheduling of real-time systems with soft and hard tasks. The idea is to compute off-line a number of schedules and schedule-switching time points, considering the duration intervals for tasks in order to exploit information about the actual duration of tasks already executed and thus adapt the schedule for the remaining tasks. Then one of the precomputed schedules is selected on-line based on the actual execution times and given switching points. Thus the only overhead at run-time is the selection of schedule, which is computationally cheap because it requires a simple comparison between a given time point and the actual completion time.

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. Real-Time Systems Symposium*, pages 4–13, 1998.

[2] G. Buttazzo and F. Sensini. Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments. *IEEE. Trans. on Computers*, 48(10):1035–1052, Oct. 1999.

[3] H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE. Trans. on Software Engineering*, 15(10):1261–1269, Oct. 1989.

[4] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling Slack Time in Fixed Priority Pre-emptive Systems. In *Proc. Real-Time Systems Symposium*, pages 222–231, 1993.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, CA, 1979.

[6] N. Homayoun and P. Ramanathan. Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems. *Real-Time Systems*, 6(2):207–232, Mar. 1994.

[7] H. Kaneko, J. A. Stankovic, S. Sen, and K. Ramamritham. Integrated Scheduling of Multimedia and Hard Real-Time Tasks. In *Proc. Real-Time Systems Symposium*, pages 206–217, 1996.

[8] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. In *Proc. Real-Time Systems Symposium*, pages 110–123, 1992.

[9] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling.* PhD thesis, Department of Computer Science, Carnegie-Mellon University, May 1986.

[10] I. Ripoll, A. Crespo, and A. García-Fornes. An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Dynamic-Priority Preemptive Systems. *IEEE. Trans. on Software Engineering*, 23(6):388–400, Oct. 1997.

[11] M. Spuri, G. Buttazzo, and F. Sensini. Robust Aperiodic Scheduling under Dynamic Priority Systems. In *Proc. Real-Time Systems Symposium*, pages 210–219, 1995.

[12] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE. Trans. on Computers*, 44(1):73–91, Jan. 1995.