Near-Optimal Fixed Priority Preemptive Scheduling of Offset Free Systems

Mathieu Grenier* Joël Goossens+ Nicolas Navet*

LORIA-INRIA*
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy- France
{grenier, nnavet}@loria.fr

Université Libre de Bruxelles+
Département d'Informatique, CP 212
1050 Bruxelles, Belgium
joel.goossens@ulb.ac.be

Abstract

In this paper, we study the problem of the fixed priority preemptive scheduling of hard real-time tasks. We consider independent tasks, which are characterized by a period, a hard deadline, a computation time, and an offset (the time at which the first request is issued) where the latter can be chosen by the scheduling algorithm.

Considering only the synchronous case is very pessimistic for offset free systems, since the synchronous case is the worst case in terms of schedulability. In this paper, we propose a new technique, based on the Audsley's priority assignment, that reduces significantly the search space of the combinatorial problem consisting in choosing the offsets. In addition, we propose new offset assignment heuristics and show the improvement of combining the new technique and the new heuristics.

1. Introduction

Problem definition. This study deals with the fixed priority preemptive scheduling of tasks in a real-time systems with *hard constraints, i.e.,* systems in which the respect of time constraints is mandatory. More specifically, we consider *offset free systems* where the offsets can be chosen by the scheduling algorithm. The activities of the system are modeled by independent *periodic tasks* τ_i as introduced in [8]. The model of the system is defined by a task set Δ of cardinality n, $\Delta = \{\tau_1, \tau_2, ..., \tau_n\}$. A periodic task τ_i is characterized by a quadruple (C_i, T_i, D_i, O_i) where each request of τ_i, called instance, has an execution time of C_i, a relative deadline D_i. T_i time units separate two consecutive instances of τ_i (hence T_i is the period of the task). The first instance of τ_i occurs at time O_i (the task offset in the following). The system is said schedulable if each instance finishes before its deadline.

Three different kinds of periodic task sets can be distinguished: *synchronous* sets, where all offsets are equal to 0, *asynchronous* sets, in which the constraints of the system determine the offsets, and finally *offset free* sets. In *offset free* systems, there is no constraint on offsets, hence they may be chosen beforehand by the scheduling algorithm. It may be noticed, that considering only the synchronous case is very pessimistic, since the synchronous case is the worst case, in the sense that, if the system is schedulable in the synchronous case it follows that this is also the case in all asynchronous situations (see [3], for instance). Our scheduling problem is the following, given the task characteristics T_i's, C_i's, and D_i's, determine a feasible offset (if any) and fixed priority assignment.

Related work. In [5], the concepts of *concrete* and *non-concrete* task sets are introduced. A *non-concrete* task set is a set for which the offsets are not determined, a *concrete* version of such a task set can be obtained by considering a particular offset configuration. Hence, a non-concrete task set *generates* a collection of concrete task sets. A non-concrete task set Δ is schedulable [5], if all the corresponding concrete sets are schedulable. While an offset free system is schedulable if at least one concrete task is schedulable.

Well-known results concern the optimality for asynchronous (and synchronous) task sets. But first a definition, a priority assignment rule is optimal for asynchronous (resp. synchronous) systems if, when a schedulable priority assignment exists for some asynchronous task set (resp. for the synchronous case), the priority assignment given by the rule is also schedulable. In [7], the non-optimality of the Deadline Monotonic (i.e., lower the deadline, higher the priority) is proven, and an optimal priority assignment rule is suggested by considering the $n!$ different priority assignments. In [1, 2] Audsley proposed an optimal priority assignment algorithm, which examines at most n^2 priority assignments, this algorithm is often referred as the Audsley's algorithm in the literature.

More recent results concern the optimality for offset free systems. In [4, 3], the authors show the interest of offset free systems and in [4] the non-optimality of rate-monotonic assignments when offset free systems are considered. Although there is an infinite number of asynchronous cases for a task set, the problem is restricted [3] by considering only non-equivalent offset assignments with an optimal offset assignment rule. Since, the number of combinations remains exponential, an efficient heuristic with a lower complexity is proposed, named *dissimilar*

offset assignment.

Well-known results concern the schedulability analysis for synchronous systems [6, 10]. For asynchronous systems as well, schedulability analysis has been studied. Due to space limitation, we shall not give details here. We know for instance (see [7]) that $[0, O_{\max} + 2P)$ where P is the LCM of the periods and $O_{\max} = \max_j(O_j)$, is a feasibility interval.

Contributions. In this paper, we show how to use the Audsley's algorithm to reduce the complexity of *offset assignment* by decreasing the number of tasks examined in the assignment. The optimal offset assignment cannot always be used due to its exponential complexity. Then, we propose new assignment heuristics that improve significantly upon the one presented in [3] as it will be shown in the experiments.

Organization. Section 2 recalls the results from [3] that are useful for the understanding of our contribution. Section 3 shows how the Audsley's algorithm can be used to decrease the complexity of the offset assignment algorithm. New heuristics are then proposed in Section 4, whose efficiency are assessed in Section 5.

2. Known offset assignments

In this section, we summarize known results on the scheduling of offset free systems. In particular, we summarize the approach developed in [3].

2.1. Scheduling of offset free systems

The topic of this study is the fixed (and preemptive) scheduling of offset free systems. In these systems, the offset of the tasks can be chosen by the scheduling algorithm. Consequently, we have to choose (off-line):

- the task priorities, and
- the task offsets.

2.2. Optimal offset assignment

Let us assume that the priorities of the tasks are already fixed, and we consider the specific priority assignment $\mathcal{P}$, which could be for instance the Deadline Monotonic. We consider *fixed* priority scheduler, hence at each time instant, the scheduling policy assigns the CPU to the instance of task with the highest priority (if any). Suppose that the system is not schedulable in the synchronous case with $\mathcal{P}$, we would like to find an asynchronous situation for which the system is schedulable. In the following, we shall distinguish between two kinds of optimality:

Definition 1 *A priority assignment rule $\mathcal{P}$ is optimal in the asynchronous case, if when a schedulable priority assignment exists, $\mathcal{P}$ provides a schedulable system in the very same asynchronous situation.*

Definition 2 *An assignment offset rule $\mathcal{O}$ is optimal under a priority allocation rule $\mathcal{P}$, if when a schedulable offset assignment exists with $\mathcal{P}$, $\mathcal{O}$ provides a schedulable asynchronous situation with the very same priority assignment $\mathcal{P}$.*

The optimal offset assignment considered in [3] is summarized in this section. The main idea is to test the schedulability of all the non-equivalent asynchronous situations of a task set.

All offset combinations may be found by restricting the offsets such as $O_1 = 0$ and $\forall i \in [2, n] \mid O_i \in [0, T_i)$. Consequently number of combinations is upper bounded by $\prod_{i=2}^{n} T_i = \mathcal{O}((\max_{2 \leq j \leq n} T_j)^{n-1})$.

To further reduce the number of offset assignments, it is possible to consider only offset assignments leading to non-equivalent asynchronous situations. Two asynchronous situations are defined to be *equivalent*, if they have the same periodic behavior. Indeed, the schedule becomes periodic with a period of $P = \text{lcm}\{T_1, .., T_n\}$). This periodic behavior only depends on the relative phasing of the task instances, i.e., on the tuple $(O_1 \pmod{T_1}, O_2 \pmod{T_2}, \ldots, O_n \pmod{T_n})$. This tuple characterizes the relative time shift between the instances of various tasks [4].

For two tasks τ_1 and τ_2, two choices ($O_2 = O_1 + v_1$ and $O_2 = O_1 + v_2$) are said equivalent if they define the same relative phasing:

$$\begin{gathered} \exists k_1, k_2 \in \mathbb{N} : (O_1 + v_1 + k_1 \cdot T_2) \bmod T_1 \\ = \\ (O_1 + v_2 + k_2 \cdot T_2) \bmod T_1 \,, \end{gathered} \tag{1}$$

which is equivalent to:

$$v_1 \equiv v_2 (\text{mod } \gcd\{T_1, T_2\}). \tag{2}$$

>From Equations 1 and 2 it follows that only the values $0, 1, ..., \gcd\{T_1, T_2\} - 1$ must be considered and are non-equivalent choices for O_1 and O_2.

The optimal offset assignment algorithm, in order to explore all possible non-equivalent asynchronous situations for the task set, constructs iteratively the situations. First, it sets the non-equivalent choices for O_2 (the offset O_1 is arbitrarily fixed to 0) by considering for O_2 all integer values in the $[0, \gcd\{T_1, T_2\})$ interval. Next, by assuming at each step that the offsets $O_1, O_2, ...O_{i-1}$ are set, consider for the offset O_i the interval $[0, \gcd\{T_i, \text{lcm}(T_1, ..., T_{i-1})\})$ (instances of task sub-set $\{\tau_1, ..., \tau_{i-1}\}$ having a period of $\text{lcm}(T_1, ..., T_{i-1})$).

2.3. Dissimilar offset assignment

The method of [3], presented in Section 2.2, reduces the non-equivalent offset assignment from $\prod_{i=2}^{n} T_i$ to $\frac{\prod_{i=2}^{n} T_i}{P}$. Despite this significant reduction, the number of offsets considered by the optimal algorithm remains exponential. In [3], the author defines then a heuristic that provides a single offset assignment for a task set.

The basic idea of the heuristic is to shift away, as far as possible, the offsets of the tasks for which some instances would be most probably in conflict for the use of the CPU. Precisely, the offset of tasks having instances released in small periods of time, and thus being close to the "synchronous" case, will be shift away as far as possible. Hence, a measure is introduced to estimate the proximity of an offset assignment with the synchronous case. The dissimilar offset assignment algorithm allocates the offsets of the periodic tasks to maximize this measure, which is defined as the length of the shortest interval that contains at least one instance of each task.

The technique considers first the (minimal) distance between two instances of tasks τ_i and τ_j in the periodic part of the schedule. The computation of this distance is performed according to Theorem 1.

Theorem 1 ([3]) *Let $r \in [0, \gcd\{T_i, T_j\})$. If $O_i = O_j + r$ (or $O_j = O_i + r$), the minimum distance between an instance of τ_i and an instance of τ_j is* $\min\{r, \gcd\{T_i, T_j\} - r\}$.

It follows from Theorem 1 that the minimum distance between an instance of τ_i and τ_j is upper bounded by $\left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ and corresponds to the offset assignment $O_i = O_j + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ (or $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$). In this case, r is equal to $\left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ or $\left\lceil \frac{\gcd\{T_i,T_j\}}{2} \right\rceil$.

The dissimilar offset assignment algorithm fixes the offsets of the periodic tasks. The algorithm sorts the couples of tasks (τ_i, τ_j) in decreasing value of $\gcd\{T_i, T_j\}$, in order to maximize the measure defined above. Next, it sets iteratively the offset O_i and O_j of the sorted couples of tasks (T_i, T_j) to obtain the highest minimum distance (i.e., $r = \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$). During this assignment, three cases may occur:

1. when O_i and O_j are not yet set, a random offset is chosen for O_i and $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$,
2. when O_i (resp. O_j) is fixed and O_j (resp. O_i) is not, $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ (resp. $O_i = O_j + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$),
3. when O_j and O_i are already chosen, there is nothing to do.

The maximal time complexity of this algorithm for assigning the offsets is $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2)$ where $T^{\max} \stackrel{\text{def}}{=} \max_{1 \leq k \leq n}(T_k)$.

3. Complexity reduction

In this section we propose a technique, based on the Audsley's priority assignment, to reduce significantly the search space. But first, we shall present the Audsley's algorithm [1] itself.

3.1. Audsley's algorithm

The Audsley's algorithm [1] performs an optimal static priority assignment for asynchronous systems (according to Definition 1).

A priority assignment is defined by:

$$\gamma : \{1, 2, ...n\} \rightarrow \{\tau_1, \tau_2 ... \tau_n\},$$

where the assignment function $\gamma(i)$ gives the task τ_k assigned to the priority level i using the convention: lower the priority level, higher the priority.

The Audsley's algorithm considers at most $\mathcal{O}(n^2)$ distinct priority assignments. First, it attempts to find a *lowest priority viable* task τ_i in Δ, i.e., tries to assign the priority level n.

Definition 3 *Task τ_i is lowest priority viable when τ_i is assigned the lowest priority of any task in Δ and:*

- *The remaining tasks in Δ are assigned priorities in any arbitrary order, the sole restriction being that all these priorities be higher than the priority assigned to τ_i.*
- *During run-time scheduling, the semantics is weakened as follows: instances generated by tasks other than τ_i may miss their deadlines (if they do so, they continue execution until completion); however, instances generated by τ_i may not miss any deadlines.*

Next, the algorithm recursively determines a lowest priority viable task in the sub-set $\Delta \backslash \{\tau_i\}$ of $n-1$ tasks (i.e., assigning priority level $n-1$). The Audsley's pseudo-algorithm is given in Algorithm 1.

Input: task set $\Delta = \{\tau_1, \tau_2 ..., \tau_n\}$
Result: task set with no assigned priority

procedure `audsley(`Δ`);`
if $\Delta = \emptyset$ **then**
  `priority assignment succeed:`
  return Δ;
end
if *no task is lowest priority viable* **then**
  `priority assignment failed:`
  return Δ;
else
  let τ_i a lowest priority viable task;
  `assign lowest priority to` τ_i`:`
  $\gamma(|\Delta|) = \tau_i$;
  return `audsley(`$\Delta \backslash \{\tau_i\}$`);`
end

Algorithm 1: Audsley's algorithm.

After executing the Audsley's algorithm, two cases may occur:

1. The priority assignment of the Audsley's algorithm leads to a schedulable system (i.e., priority assignment succeed): the set of task Δ is schedulable with the priority assignment given by function γ.

2. Otherwise, the Audsley's algorithm fails to assign the priority of level i where $i \in [1, n]$ (i.e., priority assignment failed). However, instances of the set of tasks $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ meet their deadline. Indeed, the schedulability of a task at a priority level, with a fixed scheduling preemptive policy, depends only on the set of higher priority tasks, whatever the assignment of priority among this set [1, 2].

The non-optimality of the Audsley's priority assignment for offset free systems

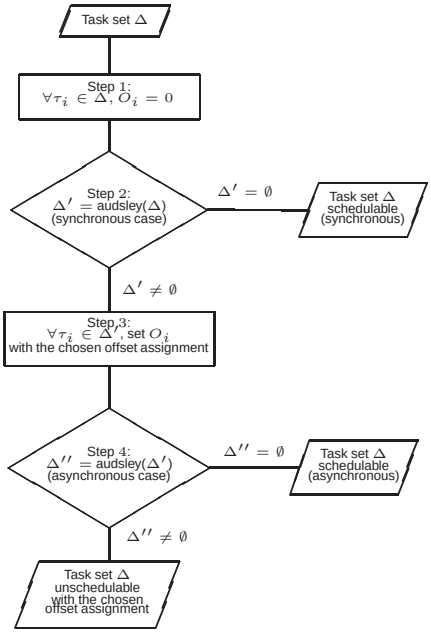
We shall see that while the Audsley's priority assignment is optimal for asynchronous systems it is not the case for offset free systems. But first a definition.

Definition 4 ([4]) *A priority assignment rule is optimal for offset free systems if when a schedulable priority assignment ($\mathcal{P}$) and offset assignment ($\mathcal{O}$) exist for some offset free task set, there is a schedulable offset assignment ($\mathcal{O}'$) for the priority assignment given by the rule.*

The priority assignment of the Audsley's algorithm depends on the offset assignment $\mathcal{O}$, actually Audsley consider *asynchronous systems* (the offsets must be already fixed). Thus, Definition 4 is not applicable in the case of the Audsley's priority assignment and consequently the Audsley's priority assignment is not optimal for offset free systems.

3.2. Reducing the search space using the Audsley's algorithm

In this section, we shall explain how to assign the priorities *and* the offsets together. Figure 2 presents the flow of our approach in a pseudo-algorithmic form. First, we initialize the offsets to consider the *synchronous* situation. Then, the Audsley's algorithm is used to assign priorities (in the synchronous case), more precisely the (recursive) function `audsley` (Algorithm 1) is used. If it successfully assigns priorities (case 1, Section 3.1), the system is schedulable in the synchronous case. Otherwise, the Audsley's algorithm fails in the synchronous case (case 2, Section 3.1), a schedulable asynchronous situation should be looked for. Consequently we first use at this step a rule to choose the offsets—for the subset of tasks returned by `audsley`: $\Delta' \stackrel{\text{def}}{=} \Delta \backslash \{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$—and *then* the priorities using the Audsley's algorithm for the second times *but* on the subset Δ' (not on the original task set). Indeed, the sub-set of tasks $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ respects their timing constraints in the synchronous situation without considering the offsets and the priorities among the set of higher priority tasks. Thus, the tasks in $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ are lowest priority viable in the synchronous case. Since the synchronous case is the worst case, these tasks remain lowest priority viable in an asynchronous situation. That is why, in the following, the offset assignment scheme can safely take into account only the tasks in the set Δ'.



Algorithm 2: Offset and priority allocation algorithm.

With this method the number of tasks to consider for the offset assignment is much lower as it will shown in the experiments of Section 5.2. Since the time complexity of the offset assignment depends on the number of tasks and their periods, the time complexity is, thus, reduced.

4. Near-optimal offset assignment heuristics

In this section, we propose several assignment heuristics, which provide alternative offset allocations when the dissimilar offset assignment fails to produce a schedulable asynchronous situation.

The functioning scheme of these new heuristics is very similar to the one of the dissimilar offset assignment: couples of tasks are ordered according to a criteria, then the task offsets are chosen from the top of the resulting ordered list to its bottom. The new heuristics provide different offset allocations than the dissimilar offset strategy since they do not only consider the minimal distance between tasks. For instance, some try to "separate" tasks with the highest utilization rate (i.e. $\frac{C_k}{T_k}$). We propose 4 new offset assignment heuristics that take into account other characteristics of the task set than the minimal distance between tasks. Our 4 heuristics consider the couples (τ_k, τ_i) by decreasing values of:

1. $\left(\frac{C_k}{T_k} + \frac{C_i}{T_i}\right) \cdot \text{gcd}(T_k, T_i)$
2. $\max\left(\frac{C_k}{T_k}, \frac{C_i}{T_i}\right) \cdot \text{gcd}(T_k, T_i)$
3. $\frac{C_k}{T_k} + \frac{C_i}{T_i}$
4. $-\text{gcd}(T_k, T_i)$

The heuristics 1,2 and 3 sort the couples of tasks by considering their utilization rate. Different ways of introducing the utilization rate in the ordering provide several asynchronous situations, which may lead to a schedulable asynchronous situation. In heuristic 1 (resp. 2), the utilization rate of the couples of tasks (resp. the maximal utilization rate) is taken into account balanced by their gcd. Rule 3 arranges the (τ_k, τ_i) according to decreasing utilization rate.

Heuristic 4 first focuses on the couples of tasks (τ_k, τ_i) for which the minimal length between instances is small. The (τ_k, τ_i) are thus ordered according to decreasing value of $-\text{gcd}(T_k, T_i)$ to set the offset of the couples with the less choices in the offset assignment.

These new assignment heuristics are considered together. The combined used of these heuristics, in our experiments (Section 5.5), provides a "near-optimal" offset assignment. The complexity of these new heuristics is identical as the one of the dissimilar offset assignment (i.e., $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2))$, because the algorithm that performs the assignment is the same, except for the ordering of the couples of tasks.

5. Experimental results

In this section, we present our experimental results. We make use of the Algorithm 2 defined in Section 3.2.

5.1. Experimental setup

In the experiments, the global load U is chosen for each set Δ of n tasks. Since the sets Δ have to be unschedulable in the synchronous case, the load U has to be sufficiently high. The utilization rate ($\frac{C_k}{T_k}$) of each task τ_k is uniformally distributed in the $\left[\frac{U}{n} \cdot 0.9\ ,\ \frac{U}{n} \cdot 1.1\right]$ interval. The computation time C_k of each task τ_k is randomly chosen with an uniform law in the $[c_{\min}, c_{\max}]$ interval, the relative deadline D_k is uniformally chosen in the $[d_{\min}, d_{\max}]$ interval , and the period T_k is upper bounded by $t_{\max}$.

In the following, we make use of the tuple $(n, U, c_{\min}, c_{\max}, d_{\min}, d_{\max}, t_{\max})$ to denote the actual parameters used in our task sets random generation.

5.2. Complexity reduction using the Audsley's algorithm

In this section, the actual reduction of the search space using the Audsley's algorithm is studied. The improvement is evaluated with task sets randomly generated according to the tuple $(n, 0.8, 2, 30, T_k - 0.9 \times (T_k - C_k), T_k + 0.9 \times (T_k - C_k), 200)$ with n being the number of tasks in the $[5, 17]$ interval. We made approximately 13000 simulations for each graph (13 points per graph).

In Figure 1, the curve in plain style presents the percentage of task sets unschedulable in the synchronous case which have at least one lowest priority viable task in the synchronous situation. One can observe that at least 38 % of the task sets include a lowest priority viable task. For these task sets, the Audsley's algorithm (step 2, Algorithm 2) allows to reduce the number of tasks in the offset assignment. One can also note from Figure 1 that the percentage decreases with the number of tasks. This phenomenon is probably related to our task generation algorithm. Indeed, in order to keep the lcm of the tasks within bounds that still allow to assess the feasibility by simulation, restrictions are imposed on the task set characteristics. When the number of tasks becomes large, the tasks tend to have the same characteristics and they tend thus to behave in a rather similar manner. Hence, when a task is not lowest priority viable, the probability to find another lowest priority viable task is rather low.

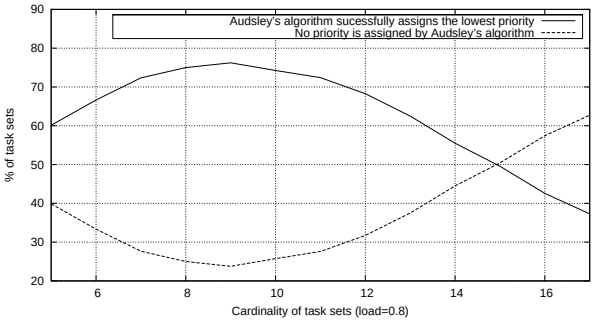


Figure 1. Percentage of unschedulable task sets in the synchronous case, which includes at least one lowest priority viable task in the synchronous case.

In Figure 2, we consider only task sets which have at least one lowest priority viable task. The curve in plain style shows the percentage of tasks being lowest priority viable after step 2, Algorithm 2 (i.e., tasks in the set $\Delta \setminus \Delta'$). The dotted curve represents the percentage of tasks τ_j, which are not (i.e., tasks in the set Δ').

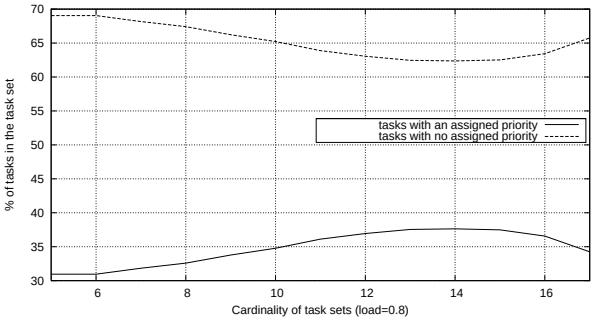


Figure 2. Proportion of lowest priority viable tasks.

As can be seen from the plot of Figure 2, at least 30 % of tasks are lowest priority viable (in the synchronous case). Thus, less than 70 % of the tasks have actually to

be considered for the offset assignment.

In order to accurately evaluate the complexity reduction obtained with the Audsley's algorithm, we study the actual reduction of the search space brought by the use of the Audsley's algorithm. In Figure 3, we consider again only task sets with at least one lowest priority viable task. The curve shows the percentage of search space reduction.

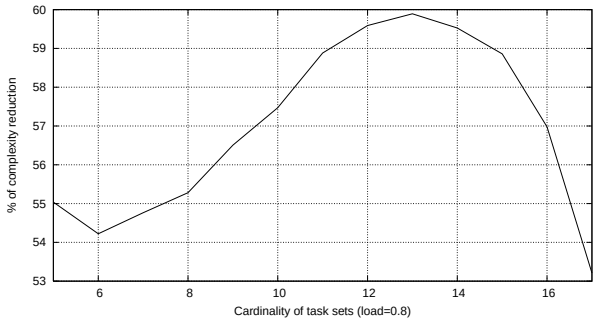


Figure 3. Search space reduction using the Audsley's algorithm.

From the simulation results, presented in Figure 3, the search space reduction is always greater than 53 %.

The conclusion that can be drawn from these experiments is that for a very significant number of systems (more than 38 % in our experiments), at least 30 % of the tasks can be allocated a priority by the Audsley's algorithm (i.e., are lowest priority viable). This allows to reduce the search space of the offset assignment scheme by at least 53 %.

5.3. Offset free for increasing feasibility

This subsection aims to show the interest of offset free systems for schedulability, by using the optimal offset assignment.

Task sets are randomly generated according to the tuple $(5, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ with U chosen in the $[0.73, 0.95]$ interval. We made approximately 6000 simulations for each graph (6 points per graph). It should be noticed that the time complexity of the optimal assignment rule, that is used in these experiments, is high, and checking if a system is schedulable or not may require a very long computation time (since we have to consider—in the worst case—all non-equivalent offset assignments). For this reason, we have strongly limited the number of tasks n and the maximum value of the periods in our simulations to reduce the number of non-equivalent offset assignment and thus diminish the complexity of the schedulability.

We now evaluate the percentage of systems unschedulable in the synchronous case which becomes schedulable in an asynchronous case (i.e., we use the optimal offset assignment). Once again, we use Algorithm 2 to determine these percentages. Figure 4 represents the percentage of systems unschedulable in the synchronous case

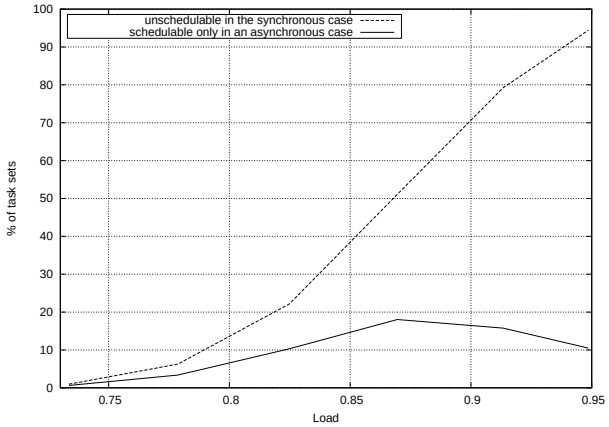


Figure 4. Percentage of systems unschedulable in the synchronous case (dotted curve) and systems only schedulable with an asynchronous configuration (plain curve). The cpu load ranges from 0.7 to 0.95 .

(dotted curve), and the percentage of systems schedulable only in an asynchronous case (plotted style curve). From Figure 4, one can observe that the percentage of task sets unschedulable in the synchronous case increases with the load, which confirm the intuition that it is harder to find a schedulable system when the load is high. Moreover, the percentage of task sets schedulable in an asynchronous situation increases with the load (up to 18 %) until the load reaches 0.87, then it starts to decrease. Intuitively, it is clear that task sets tend to be unschedulable, whatever the offset allocations, when the load becomes too high.

5.4 Combined use of the heuristics: efficiency compared to the optimal allocation

Figure 5 shows the percentage of task sets schedulable in a particular asynchronous situation (non-equivalent to the synchronous situation) which remains schedulable with the dissimilar offset assignment rule (dashed curve) and with at least one of our new heuristics (curve in plain style).

As can be seen on Figure 5, the assignment heuristics find a schedulable asynchronous situation for at least 51 % and up to 95 % of the task sets in which such a situation exists. The chance of finding a schedulable assignment logically decreases with the load.

The combined used of the heuristics enables us to find an important percentage of the schedulable asynchronous situations. From Figure 5, it is obvious that the combination of our new heuristics outperforms the dissimilar offset assignment.

5.5. Relative performances of the heuristics

In this section, the improvement brought by the new heuristics is discussed more precisely. Task sets are ran-

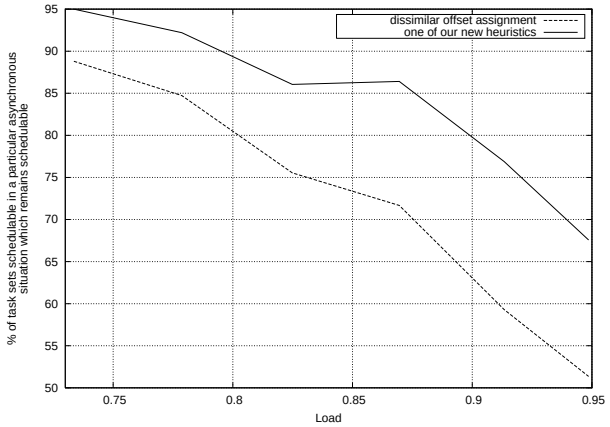


Figure 5. Dissimilar offset assignment vs. our new heuristics.

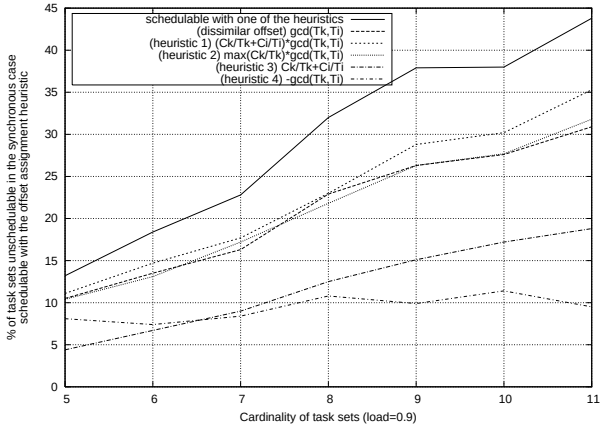


Figure 7. Percentage of the task sets unschedulable in the synchronous case that becomes schedulable with the different offset assignment heuristics (90 % CPU load).

domly generated according to the tuple $(n, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ with U chosen in the $\{0.8, 0.9\}$ set and n in the $[5, 11]$ interval. We made approximately 7000 simulations for each graph (7 points per graph).

The offsets and priorities assignment are performed according to Algorithm 2 of Section 3.2. At step 4, the asynchronous situations correspond to the offset assignments produced by the dissimilar offset assignment and by the new heuristics.

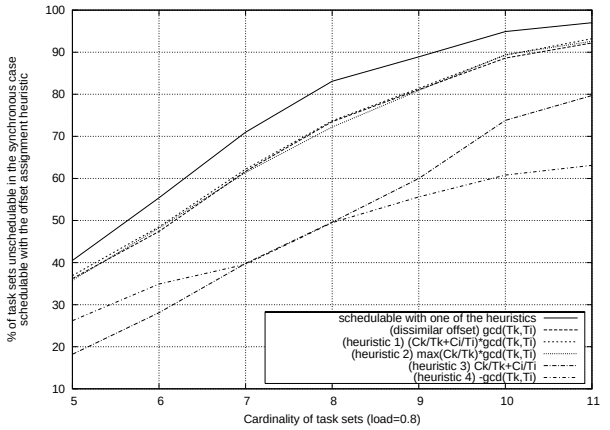


Figure 6. Percentage of the task sets unschedulable in the synchronous case that becomes schedulable with the different offset assignment heuristics (80 % CPU load).

Figure 6 and 7 display the percentage of tasks sets unschedulable in the synchronous situation which become schedulable in the asynchronous situation produced by each of the heuristics. The experiments are done with a global load of 0.8 in Figure 6 and of 0.9 in Figure 7. From these Figures, one sees that the offset assignment heuristics significantly increase the schedulability compared the synchronous case. For instance, in Figure 6, the percentage of task sets schedulable with an asynchronous situation produced by the heuristics is at least 40.5 % and up to 97 %. The improvement steadily increases with the number of tasks: for instance, in Figure 6, the percentage of schedulable task sets in an asynchronous situation is equal to 71 % for 7 tasks, while it is 88.9 % for 9 tasks. This can be intuitively explained by the fact that the higher the number of tasks, the higher the freedom degree to set the offsets, and thus, the farther from the synchronous case the system can be.

One also observes that, very logically, the percentage of systems schedulable in an asynchronous situation strongly decreases when the load is high. For instance, the percentage of schedulable systems for sets of 8 tasks is 83.1 % for a load of 0.8 of 32 % for a load of 0.9 (Figure 7).

The different heuristics can be compared using Figure 6 and 7. We observe that the dissimilar offset assignment performs very well, usually better than the new heuristics. However, using all heuristics together (i.e., try the offset assignment returned by each of the heuristics) allows to clearly outperform the dissimilar offset assignment alone. The heuristics (including the dissimilar offset assignment) are in some way very complementary. For instance, 37.9 % of the task sets are schedulable with at least one of our heuristics for 9 tasks (Figure 7) while only 26.3 % are schedulable with the dissimilar offset assignment. It is worth noting that the complexity of each of the new heuristics is the same as the dissimilar offset assignment and, in practice, the computing time does not raise problem whatever the cardinality of the task set.

In conclusion, our experiments show that the combined used of all the heuristics lead to a near near-optimal offset assignment, which allows to increase considerably the

percentage of systems schedulable compared to the sole asynchronous situation.

6. Conclusion

In this paper, we have studied the problem of the static preemptive scheduling of offset free systems. First, we have shown that the search space for assigning the offset may be reduce of up to 50 % with an appropriate use of the Audsley's algorithm. Then, new heuristics are proposed to improve upon the result of the dissimilar offset assignment scheme introduced in [3]. These heuristics provide alternative asynchronous cases, which allow to increase very significantly the number of schedulable systems with regards to pessimistic synchronous case. The combined use of all these heuristics provides a near-optimal offset assignment. Indeed, according to our experiments conducted with for a global load of 0.8 for task sets having a cardinality in $[5, 11]$, the set of heuristics enables to schedule at least 40.5% and up to 97% of task sets, which are unschedulable in the synchronous case.

A similar study remains to be conducted for the non-preemptive case, which is of interest for scheduling frames on networks but also for many small embedded systems without preemptive capabilities. In a first step, it has to investigated whether a similar complexity reduction procedure based on the Audsley's algorithm can be devised for the non-preemptive case (see [9] for some results on the use of the Audsley's algorithm in the non-preemptive case). Then, offset assignment heuristics dedicated to the non-preemptive case have to be proposed and their efficiency evaluated.

In the future, we also intend to evaluate if integer linear programming can be used to determine offsets in an efficient manner; the main problem will be here to define the cost functions that lead to schedulable systems.

References

[1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Report YO1 5DD, Dept. of Computer Science, University of York, England, 1991.

[2] N. Audsley. On priority assignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, 2001.

[3] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, March 2003.

[4] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignements for hard real-time systems. *Real-Time Systems*, 13(2):107–126, sep 1997.

[5] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proc. of the 12th IEEE Real-time Systems Symposium (RTSS 1991)*, 1991.

[6] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. of the 11th IEEE Real-Time Systems Symposium*, pages 201–213, Florida, USA, 1990.

[7] J. Leung and J. Whitehead. On the complexity of fixed priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard-real time environnement. *Journal of the ACM*, 20(1):40–61, 1973.

[9] R. Saket and N. Navet. Frame packing algorithms for automotive applications. Available as research report INRIA RR-4998, to appear in Journal of Embedded Computing, 2006.

[10] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.