An efficient best response heuristic for a non-preemptive strictly periodic scheduling problem

Clément Pira, Christian Artigues

An efficient best response heuristic for a non-preemptive strictly periodic scheduling problem

Clément Pira, Christian Artigues

CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
{pira,artigues}@laas.fr

Keywords: Periodic scheduling, equilibrium, twodimensional optimization

Abstract. In this article, we study an original heuristic for a non-preemptive strictly periodic scheduling problem, based on the notion of equilibrium. This heuristic was first considered in [1]. Our contribution is to greatly improve its efficiency through a complete redesign of its main method, which is the best response procedure.

1 Problem and method

We consider a periodic scheduling problem introduced in [4, 3, 1, 2]. Such a problem arises in the avionic field, where a set of N periodic tasks (measure of a sensor, etc.) has to be scheduled on P processors distributed on the plane. In this problem, each task i has a fixed period T_i which cannot be modified. A solution is given by an assignment of the tasks to the processors and, for each task by the start time t_i of one of its occurrences. Each task has a processing time p_i and no two tasks assigned to the same processor can overlap during any time period.

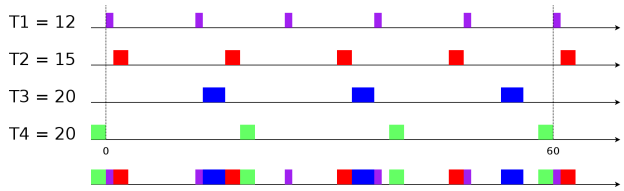

Fig. 1. $N = 4$ non-overlapping periodic tasks on $P = 1$ processor

In this paper, we adopt a more general model in which processing times p_i are generalized by positive latency delays $l_{i,j} \geq 0$ (or time lags). The former case is the particular case where $l_{i,j} = p_i$ for all other tasks j. The proposed heuristic does not suffer from this generalization even if some simplification occurs in the case of processing times (see section 2.5).

In this paper, we only consider the case where the offsets t_i are integers. Since the problem is periodic, all the periods T_i also need to be integers. We will see in section 1.2 why this hypothesis is important to prove convergence of the method. Note that the latency delays need not be integer *a priori*, especially in the case of the optimization problem presented in section 1.1.

1.1 Problem definition

Non-overlapping constraints We first focus on the monoprocessor problem. In the formulation of a periodic scheduling problem, we want to constrain all the occurrences of j with respect to all the one of i. More precisely, we want a latency delay $l_{i,j} \geq 0$ to be respected whenever an occurrence of j starts after an occurrence of i. Say differently, we want the smallest positive difference between an occurrence of j and an occurrence of i to be greater than $l_{i,j}$. The set of occurrences of i is $t_i + T_i\mathbb{Z}$, while the set of occurrences of j is $t_j + T_j\mathbb{Z}$. When doing the difference, and using Bézout identity, we obtain that the set of possible differences is $(t_j - t_i) + g_{i,j}\mathbb{Z}$ where $g_{i,j} = \gcd(T_i, T_j)$. The smallest positive representative of this set is $(t_j - t_i) \mod g_{i,j}$. Therefore, the constraint we want to pose is simply :

$$(t_j - t_i) \bmod g_{i,j} \geq l_{i,j}, \qquad \forall (i,j) \in \mathcal{G} \tag{1}$$

The modulo is defined here as the only representative of $(t_j - t_i) + g_{i,j}\mathbb{Z}$ which belongs to $[0, g_{i,j} - 1]$ (in particular, we consider a classic positive modulo, and not a signed modulo like in some programming languages). The graph $\mathcal{G}$ involved in constraint (1) contains the arcs (i,j) for which $l_{i,j} > 0$ (since otherwise the equation is trivially satisfied).

Objective to maximize In a context of robustness, it could be natural to maximize the feasibility of the system. More concretely, we want an execution of a task to be as far as possible from every other executions of another task which precedes or follows it. We could imagine that a task lasts longer than expected, due to failure of the processor. This increase in the duration is naturally proportional to the original processing time. Hence, we make all the delays $l_{i,j}$ proportional to a common factor $\alpha \geq 0$ that we try to optimize (see [1, 2]).

$$\begin{aligned}
\max \quad & \alpha & & (2)\\
s.t. \quad & (t_j - t_i) \bmod g_{i,j} \geq l_{i,j}\alpha & \forall (i,j) \in \mathcal{G} \quad & (3)\\
& t_i \in \mathbb{Z} & \forall i \quad & (4)\\
& \alpha \geq 0 & & (5)
\end{aligned}$$

We easily check that a schedule is feasible for the feasibility problem *iff* the optimization problem has a solution with $\alpha \geq 1$. Hence, this optimization problem can be interesting simply to find feasible solutions. Given the offsets (t_i) we can always compute the best compatible α :

$$\alpha = \min_{(i,j)\in\mathcal{G}} \frac{(t_j - t_i) \bmod g_{i,j}}{l_{i,j}} \tag{6}$$

This shows that an optimal solution is completely defined by the offsets. The following proposition allows to define an upper bound :

Proposition 1. *Let $(i,j) \in \mathcal{G}$. A necessary condition for this problem to be feasible is $\lceil l_{i,j}\alpha \rceil + \lceil l_{j,i}\alpha \rceil \leq g_{i,j}$. This is equivalent to $\alpha \leq \alpha^{i,j}_{\max}$ where :*

$$\alpha^{i,j}_{\max} = \begin{cases} g_{i,j}/l_{i,j} & \text{if } l_{j,i} = 0 \\ \max\left(\frac{1}{l_{i,j}}\left\lfloor \frac{g_{i,j}}{l_{i,j}+l_{j,i}} l_{i,j} \right\rfloor \ , \ \frac{1}{l_{j,i}}\left\lfloor \frac{g_{i,j}}{l_{i,j}+l_{j,i}} l_{j,i} \right\rfloor\right) & \text{if } l_{j,i} > 0 \end{cases}$$

We deduce that $\alpha_{\max} = \min_{(i,j)} \alpha^{i,j}_{\max}$ is an upper bound on the value of α.

Proof. Since the offsets are integers, we have $\lceil l_{i,j}\alpha \rceil \leq (t_j - t_i) \bmod g_{i,j}$ and $\lceil l_{j,i}\alpha \rceil \leq (t_i - t_j) \bmod g_{i,j}$. Summing this two inequalities, we get $\lceil l_{i,j}\alpha \rceil + \lceil l_{j,i}\alpha \rceil \leq g_{i,j}$. Let $\alpha^{i,j}_{\max}$ be the largest α-value satisfying this condition : $\alpha^{i,j}_{\max} = \max\{\alpha \mid \lceil l_{i,j}\alpha \rceil + \lceil l_{j,i}\alpha \rceil \leq g_{i,j}\}$. If $l_{j,i} = 0$, we have $\alpha^{i,j}_{\max} \leq g_{i,j}/l_{i,j}$. Otherwise, we easily see that we have either $l_{i,j}\alpha^{i,j}_{\max} \in \mathbb{N}$ or $l_{j,i}\alpha^{i,j}_{\max} \in \mathbb{N}$. Suppose $l_{i,j}\alpha^{i,j}_{\max} = n \in \mathbb{Z}$. Then n is the largest integer such that $n + \lceil n l_{j,i}/l_{i,j} \rceil \leq g_{i,j}$, *i.e.* $n l_{j,i}/l_{i,j} \leq g_{i,j} - n$, *i.e.* $n \leq g_{i,j} l_{i,j}/(l_{i,j} + l_{j,i})$. Therefore $n = \lfloor g_{i,j} l_{i,j}/(l_{i,j} + l_{j,i}) \rfloor$. □

Remark 1. This necessary condition is stronger than $l_{i,j}\alpha + l_{j,i}\alpha \leq g_{i,j}$, hence the upper bound is tighter than $\min_{(i,j)} \frac{g_{i,j}}{l_{i,j}+l_{j,i}}$ which is the upper bound in the fractional case.

A word on zero delays Classically, processing times are strictly positive. However our model allows zero delays, which amounts to introduce an additional graph $\mathcal{G}$. Very often, the constraints associated with (i,j) and (j,i) work together and some complications are introduced when only one of the couples belongs to the graph *i.e.* only one of the delay is strictly positive (see for example proposition 1). In the following we will suppose that the graph is symetric. Since $\mathcal{G}$ is symetric, it can be seen as an undirected graph. We will write $\mathcal{G}(i)$ for the set of neighbors of i, *i.e.* the set of tasks with which i is constrained. This hypothesis is justified by the following proposition :

Proposition 2. *We can always suppose the graph $\mathcal{G}$ to be symetric,* i.e. *the delays $l_{i,j}$ and $l_{j,i}$ to be either both zero or both strictly positive, possibly by replacing a zero delay by $1/\alpha_{\max}$.*

Proof. Suppose $l_{i,j} > 0$, but $l_{j,i} = 0$. We consider a new program in which $l_{j,i}$ as been updated to be $1/\alpha_{\max}$. This new program is stronger hence a solution of this program is a solution of the original. Conversely, let $((t_i), \alpha)$ be a solution of the original. If $\alpha = 0$, then this is trivially a solution to the new program. Otherwise $\alpha > 0$. In this case, we have $(t_j - t_i) \bmod g_{i,j} \geq l_{i,j}\alpha > 0$. Therefore using proposition 3, we get $(t_i - t_j) \bmod g_{i,j} \geq 1 \geq \alpha/\alpha_{\max}$ and so we have a solution to the new program. □

A consequence of proposition 3 is that instead of considering equation (3) for each couple $(i,j) \in \mathcal{G}$, we can consider the following equation for each couple $(i,j) \in \mathcal{G}$ with $i < j$:

$$l_{i,j}\alpha \leq (t_j - t_i) \bmod g_{i,j} \leq g_{i,j} - l_{j,i}\alpha, \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{7}$$

1.2 An equilibrium-based heuristic

The best response method The main component of the algorithm is called the best response procedure. It takes its name from a game theory analogy. Each task is seen as an agent which tries to optimize its own offset, while the other offsets are fixed. Therefore, those other offsets are put in the RHS of the problem. Moreover, the agent only takes into account the constraints in which it is involved, *i.e.* only the constraints associated with its neighborhood $\mathcal{G}(i)$. Hence, the agent i solves the following program (BR_i) :

$$
\begin{aligned}
(BR_i) \qquad \max \quad & \alpha && && (8)\\
s.t. \quad & l_{j,i}\alpha \leq (t_i - t_j) \bmod g_{i,j} \leq g_{i,j} - l_{i,j}\alpha && \forall j \in \mathcal{G}(i) && (9)\\
& t_i \in \mathbb{Z} && && (10)\\
& \alpha \geq 0 && && (11)
\end{aligned}
$$

The concept of equilibrium and principle of the method to find one

Definition 1. *A solution* (t_i) *is an equilibrium* iff *for each task* i, t_i *is an optimal offset for the program* (BR_i).

In the following, we will say that a task i is stable if the current offset t_i is optimal for (BR_i). Therefore an equilibrium is exactly a solution (t_i) such that all the tasks are stable. The heuristic uses a counter N_{stab} to count the number of tasks known to be stable. It starts with an initial solution (for example randomly generated) and tries to improves this solution by a succession of unilateral optimizations. On each round, we choose cyclically a task i and try to optimize its schedule, *i.e.* we solve (BR_i). If no improvement was found, then one more task is stable, otherwise we update and reinitialize the counter of stable tasks. We continue until N tasks are stable. This is summarized in Algorithm 1. We refer to [1] for the proof of termination and correction.

Remark 2. The choice to search for integral offsets has some drawback, in particular when we are interested by a MILP resolution. However, the main reason for the use of integers is that it allows to guarantee the convergence of the heuristic. The termination proof relies on the fact that there is only a finite number of possible values for α. This is not the case with fractional offsets, and indeed, it is not hard to see that in this case, the heuristic has only a few chances to converge in a finite number of steps.

2 The best response procedure

Since the offsets are integer, and since the problem is periodic, we can always impose t_i to belong to $\{0, \cdots, T_i - 1\}$. Therefore we can trivially solve the best response program (BR_i) by computing the α-value for each $t_i \in \{0, \cdots, T_i - 1\}$, using expression (6), and select the best offset. This procedure runs in $O(T_i N)$, hence any method should at least be faster. In [1], the authors propose a method

Algorithm 1 The heuristic

```
1: procedure ImproveSolution((t_j)_{j∈I})
2:     N_stab ← 0                                  ▷ The number of stabilized tasks
3:     i ← 0                                       ▷ The task currently optimized
4:     while N_stab < N do                         ▷ We run until all the tasks are stable
5:         (newt_i, α_i) ← BestResponse(i, (t_j)_{j∈I})   ▷ We optimize the task i
6:         if newt_i = t_i then                    ▷ We do not have a strict improvement
7:             N_stab ← N_stab + 1                 ▷ One more task is stable
8:             α ← min(α_i, α)
9:         else                                    ▷ We have a strict improvement
10:            t_i ← newt_i
11:            N_stab ← 1                          ▷ We restart counting the stabilized tasks
12:            α ← α_i
13:        end if
14:        i ← (i + 1) mod N                       ▷ We consider the next task
15:    end while
16:    return (α, (t_j)_{j∈I})
17: end procedure
```

consisting in precomputing a set of intersection points (in the next section we will see that a fractional optimum is at the intersection of an increasing and a decreasing line). Then, they compute the α-value for each of these intersection points and select the best offset. In the following, we present a method which greatly improves (BR_i) solving.

2.1 Structure of the solution set of (BR_i)

Each task i is linked with all the tasks $j \in \mathcal{G}(i)$ through non-overlapping constraints. We want to describe the shape of the set of solutions (t_i, α) for the problem (BR_i), given some fixed offsets $(t_j)_{j\in\mathcal{G}(i)}$. If we try to draw the function $\alpha = (t_i - t_j) \bmod g_{i,j}/l_{j,i}$ representing the optimal value of α respecting constraint $(t_i - t_j) \bmod g_{i,j} \geq l_{j,i}\alpha$ when t_i takes rational values, we obtain a curve which is piecewise increasing and discontinuous since it becomes zero on $t_j + g_{i,j}\mathbb{Z}$. In the same way, if we draw $\alpha = (t_j - t_i) \bmod g_{i,j}/l_{i,j}$, we obtain a curve which is piecewise decreasing and becomes zero at the same points. It is therefore more natural to consider the two constraints jointly, *i.e.* (7), which gives a continuous curve (see Figure 2). Note that if $\mathcal{G}$ was not symetric, we would need to consider some degenerate cases were one of the slope (increasing or decreasing) would be infinite (since $l_{i,j} = 0$ or $l_{j,i} = 0$).

The set of solutions for all the constraints is the intersection of the curves decribed above for each $j \in \mathcal{G}(i)$ (see Figure 3). Hence, the solution set is composed of several adjacent polyhedra. We can give an upper bound on the number n_{poly} of such polyhedra. A polyhedron starts and ends at zero points. For a given constraint j, there is $T_i/g_{i,j}$ zero points in $[0, T_i - 1]$, hence n_{poly} is bounded by $T_i \sum_{j\neq i} \frac{1}{g_{i,j}}$. This upper bound can be reached when the offsets are fractional, since in this case, we can always choose the offsets t_j such that the sets of zero

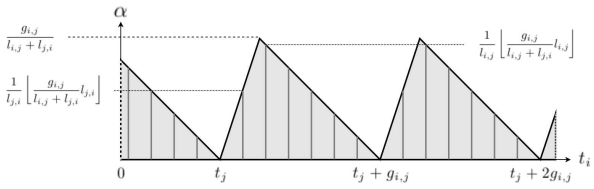

Fig. 2. Possible values for (t_i, α) when constrained by a single task j

points $(t_j + g_{i,j}\mathbb{Z})_{j\in\mathcal{G}(i)}$ are all disjoint. In the case of integer offsets, there is obvioulsy at most T_i zero points in the interval $[0, T_i - 1]$ and therefore, at most T_i polyhedra.

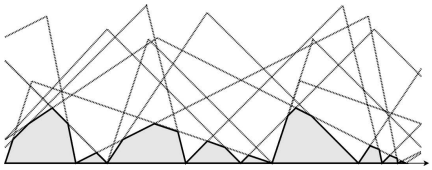

Fig. 3. Possible values for (t_i, α) constrained by all tasks $j \in \mathcal{G}(i)$

2.2 Local (two dimensional) polyhedron

We want to compute the local polyhedron which contains a reference offset t_i^*. Locally, the constraint $(t_i - t_j) \bmod g_{i,j} \geq l_{j,i}\alpha$ is linear, of the form $t_i - o_j \geq l_{j,i}\alpha$. Here, o_j is the largest $x \leq t_i^*$ such that $(x - t_j) \bmod g_{i,j} = 0$. In the same way, we can compute the decreasing constraint $o'_j - t_i \geq l_{i,j}\alpha$. In this case o'_j is the smallest $x \geq t_i^*$ such that $(o'_j - x) \bmod g_{i,j} = 0$. By proposition 4, we have :

$$o_j = t_i^* - (t_i^* - t_j) \bmod g_{i,j} \qquad \text{and} \qquad o'_j = t_i^* + (t_j - t_i^*) \bmod g_{i,j} \tag{12}$$

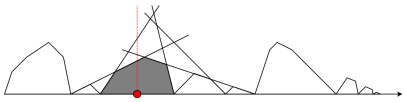

Fig. 4. Selection of the polyhedron containing a reference offset t_i^*

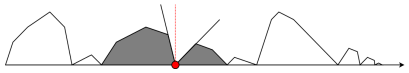

Fig. 5. Two possibilities in the degenerate case $\alpha = 0$

Therefore, we obtain a local polyhedron (see figure 4). Note that when $(t_i^* - t_j) \bmod g_{i,j} > 0$, we simply have $o'_j = o_j + g_{i,j}$ by proposition 3. However, when $(t_i^* - t_j) \bmod g_{i,j} = 0$, we have $o_j = o'_j = t_i^*$. In this case, the polyhedron is degenerate since it contains only $\{t_i^*\}$, and the α-value at t_i^* is zero. Instead of computing this polyhedron, we prefer to choose either the polyhedron on the right, or on the left (see figure 5). If we choose the one on the right, this

amounts to defining o'_j to be the smallest $x > t^*_i$ which annulates the constraint. By proposition 4, we have $o'_{i,j} = t^*_i + g_{i,j} - (t^*_i - t_j) \bmod g_{i,j}$. Therefore, this simply amounts to enforcing $o'_j = o_j + g_{i,j}$. Choosing the polyhedron on the left amounts to defining o'_j using (12) and to enforcing $o_j = o'_j - g_{i,j}$. Once the local polyhedron has been defined, the problem is now to solve the following MILP :

$$
\begin{aligned}
(Loc{-}BR_i) \qquad \max \quad & \alpha & & (13)\\
s.t. \quad & t_i - l_{j,i}\alpha \geq o_j & \forall j \in \mathcal{G}(i) \quad & (14)\\
& t_i + l_{i,j}\alpha \leq o'_j & \forall j \in \mathcal{G}(i) \quad & (15)\\
& t_i \in \mathbb{Z} & & (16)
\end{aligned}
$$

2.3 Solving the local best response problem

We now need a method to solve efficiently the program $(Loc{-}BR_i)$. Even if we are interested in integer offsets, we can first search for a fractional solution, and for this we can use any available method of linear programming. However, since the problem is a particular two dimensional program, we can give special implementations of these methods. Most of them (primal or dual simplex) run in $O(N^2)$, but Megiddo algorithm [5] allows to find such a solution in $O(N)$.

Once a fractional solution has been found, we can round it to the closest smaller and larger integers, compute the α-value associated with these two offsets (using expression (6)), and select the best one. Since the additional computation of the two α-values runs in $O(N)$, this gives immediately a method to compute an integer solution in $O(N)$. In practice however, Megiddo algorithm is outperformed by the dual simplex method presented below, at least for the sizes of instances we considered. In fact, the integrality assumption allows to improve the dual simplex algorithm and obtain a complexity in $O(NW)$ where W is the width of the polyhedron. This is enough to give an acceptable complexity in $O(T_iN)$ for the global best response problem (described in section 2.4). Hence, the $O(N^2)$ complexity of the dual simplex approach is not penalizing.

A subroutine to determine the constraint active at a given point Given an offset x_{ref}, expression (6) allows to compute the best compatible α-value, say α_{ref}. Once the local polyhedron (on the right) has been computed, this value can be defined by $\alpha_{\text{ref}} = \min(\alpha_{\text{inc}}, \alpha_{\text{dec}})$ with :

$$
\alpha_{\text{inc}} = \min_{j \in \mathcal{G}(i)} \frac{x_{\text{ref}} - o_j}{l_{j,i}} \qquad \text{and} \qquad \alpha_{\text{dec}} = \min_{j \in \mathcal{G}(i)} \frac{o'_j - x_{\text{ref}}}{l_{i,j}} \tag{17}
$$

Here, we distinguish the use of increasing and decreasing constraints. At the point $(x_{\text{ref}}, \alpha_{\text{inc}})$, at least one increasing line with equation $x - o_j = \alpha l_{j,i}$ is active (*i.e.* pass through $(x_{\text{ref}}, \alpha_{\text{inc}})$). Hence, the slope of this line is given by $l^{-1}_{j,i}$. Among all the active constraints of this form, we are interested by the one with the smallest slope *i.e.* the largest delay $l_{j,i}$. In the same way, at the point $(x_{\text{ref}}, \alpha_{\text{dec}})$, at least one decreasing constraint is active. We can compute the

decreasing line with equation $o'_k - x = \alpha l_{i,k}$ which pass through $(x_{\text{ref}}, \alpha_{\text{dec}})$ and which has the largest slope (since this slope is negative, this is the one which is the most close to 0). As this slope is given by $-1/l_{i,k}$, we select the line with the largest $l_{i,k}$. Since $\alpha_{\text{ref}} = \min(\alpha_{\text{inc}}, \alpha_{\text{dec}})$, there are three possibilities, as shown in Figure 6 :

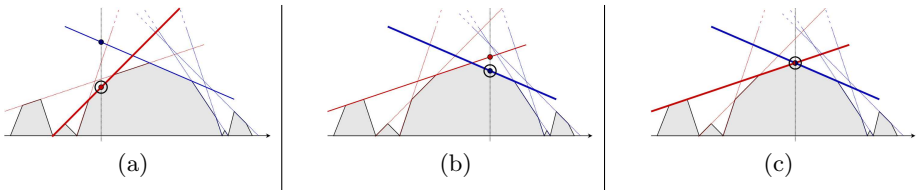

Fig. 6. Active lines computed at different offsets

(a) If $\alpha_{\text{inc}} < \alpha_{\text{dec}}$, we have $\alpha_{\text{ref}} = \alpha_{\text{inc}}$. In this case, the only active constraints are increasing. If expression (6) is seen as a function $x \mapsto \alpha(x)$, the value $1/l_{j,i}$ represents the derivative on the right of this function at x_{ref}. In particular, the optimum is strictly on the right of x_{ref}.
(b) If $\alpha_{\text{inc}} > \alpha_{\text{dec}}$, we have $\alpha_{\text{ref}} = \alpha_{\text{dec}}$. In this case, the only active constraints are decreasing and the value $-1/l_{i,k}$ represent the derivative on the left of the function at x_{ref}. The optimum is then strictly on the left of x_{ref}.
(c) If $\alpha_{\text{inc}} = \alpha_{\text{dec}}$, we are both on an increasing and a decreasing line. The derivative on the left is given by $1/l_{j,i}$ while the derivative on the right is given by $-1/l_{k,i}$. In this case, x_{ref} is the optimum.

In the following, we synthetize the previous computation into a function which returns a tuple $(d, \alpha, j_{\text{inc}}, j_{\text{dec}})$, where α is the value at x_{ref}, and j_{inc} and j_{dec} are respectively the indices of the selected increasing and decreasing lines. We call this function ValueAndDerivative (see Algorithm 2) since it returns both the value and informations on the slope of the active line. The field d can take value in $\{-1, 0, 1\}$ and indicate where should be searched the optimum. If $d = 1$ is returned, we are on the increasing line having equation $x - o_{j_{\text{inc}}} = \alpha l_{j_{\text{inc}},i}$ and the optimum should be searched on the right. If $d = -1$ is returned, we are on the decreasing line having equation $o'_{j_{\text{dec}}} - x = \alpha l_{i,j_{\text{dec}}}$ and the optimum should be searched on the left. Finally, if 0 is returned, we are at the intersection of the two lines which is the optimum. This procedure trivially runs in $O(N)$.

Remark 3. Since the procedure tells us where to search for the optimum, we easily deduce a dichotomous procedure which solves problem $(Loc{-}BR_i)$ and which runs in $O(N \log(W))$ where W is the width of the polyhedron[1].

A dual simplex approach We present a dual simplex approach. As we will see, this seems to be a primal approach since we traverse the vertices of the

[1] Note however that contrary to the simplex based approach, we cannot garantee a complexity in $O(N^2)$.

Algorithm 2 A routine to compute the α-value and the active line

procedure ValueAndDerivative(x_{ref})
  $\alpha_{\text{inc}} \leftarrow \infty$; $\alpha_{\text{dec}} \leftarrow \infty$
  for all $j \in \mathcal{G}(i)$ **do**
    $\alpha \leftarrow (x_{\text{ref}} - o_j)/l_{j,i}$ ▷ Or directly $\alpha \leftarrow (x_{\text{ref}} - t_j) \bmod g_{i,j}/l_{j,i}$
    if $\alpha < \alpha_{\text{inc}}$ **or** ($\alpha = \alpha_{\text{inc}}$ **and** $l_{j,i} > l_{j_{\text{inc}},i}$) **then** $\alpha_{\text{inc}} \leftarrow \alpha$; $j_{\text{inc}} \leftarrow j$
    $\alpha \leftarrow (o'_j - x_{\text{ref}})/l_{i,j}$ ▷ Or directly $\alpha \leftarrow (g_{i,j} - (x_{\text{ref}} - t_j) \bmod g_{i,j})/l_{i,j}$
    if $\alpha < \alpha_{\text{dec}}$ **or** ($\alpha = \alpha_{\text{dec}}$ **and** $l_{i,j} > l_{i,j_{\text{dec}}}$) **then** $\alpha_{\text{dec}} \leftarrow \alpha$; $j_{\text{dec}} \leftarrow j$
  end for
  if $\alpha_{\text{inc}} > \alpha_{\text{dec}}$ **then return** $(-1, \alpha_{\text{dec}}, j_{\text{inc}}, j_{\text{dec}})$
  if $\alpha_{\text{inc}} < \alpha_{\text{dec}}$ **then return** $(1, \alpha_{\text{inc}}, j_{\text{inc}}, j_{\text{dec}})$
  return $(0, \alpha_{\text{inc}}, j_{\text{inc}}, j_{\text{dec}})$
end procedure

polyhedron (see Figure 8). However we apply the dual simplex not on problem (BR_i) but on its dual which is in standard form. We find unnecessary to present the primal simplex approach which is illustrated on Figure 7.

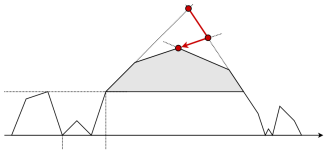

Fig. 7. Finding the fractional optimum with a primal simplex approach

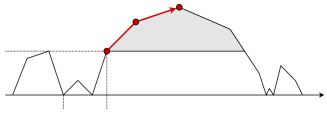

Fig. 8. Finding the fractional optimum with a dual simplex approach

We first describe an algorithm to find a fractional solution. We will present some modifications afterward in order to return an integral solution. Let x_{frac} be the variable intended to contain the offset of the fractional solution. In order to initialize the algorithm we start with $x_{\text{frac}} = t_i$, where t_i is the current offset of the task i. We will try to find the best value in the local polyhedron containing x_{frac}. Using Algorithm 2, we can compute the associated α-value $\alpha_{\min}$, as well as the 'derivative', *i.e.* the slope of an active line (see Figure 6). If we are both on an increasing and a decreasing line, we know that we are at the optimum, we return the solution $(t_i, \alpha_{\min})$. Otherwise the active line is either increasing or decreasing. In the following, we will suppose that we are on an increasing line, say with equation $x - o = \alpha l$. Hence, in order to reach the optimum, we will traverse the vertices of the polyhedron in the right direction (see Figure 8). In order to determine the next vertex, we first compute intersections of the active line with decreasing lines having equation $o'_{i,k} - x = \alpha l_{i,k}$. Such an intersection point has the following coordinates :

$$\alpha = \frac{o'_k - o}{l + l_{i,k}} \qquad \text{and} \qquad x = \frac{l o'_k + l_{i,k} o}{l + l_{i,k}} \tag{18}$$

We keep the one with the smallest α-value (see figure 9). In the same way, we also search for intersections of the active line with other increasing lines having equation $x - o_j = \alpha l_{j,i}$. The coordinates are given by :

$$\alpha = \frac{o - o_j}{l_{j,i} - l} \qquad \text{and} \qquad x = \frac{o l_{j,i} - o_j l}{l_{j,i} - l} \tag{19}$$

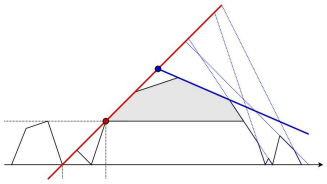

Fig. 9. The lowest intersection point with decreasing lines

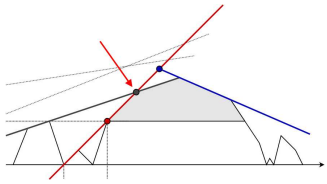

Fig. 10. The lowest intersection point with increasing lines

Among the increasing lines, we are only interested with lines having a strictly smaller slope, *i.e.* a strictly greater delay $l_{j,i} > l$ [2]. At the end, we have an intersection point (see figure 10) which becomes our new fractional solution $(x_{\text{frac}}, \alpha_{\text{frac}})$. Note that in the above computation, only the α-coordinate is needed since the x-coordinate of the selected intersection point can be computed afterward by $x = o + \alpha l$. We now proceed following one of the two alternatives :

A1 If we fail to find an intersection point of the second kind (with an increasing line) with a better value, this means the selected point $(x_{\text{frac}}, \alpha_{\text{frac}})$ correspond to the intersection of an increasing and a decreasing line. Therefore, this is the fractional optimum (see Figure 12), hence the procedure ends.

A2 Otherwise, we are not at the fractional optimum. The line which is involved in the intersection is increasing, with equation $x - o_j = \alpha l_{j,i}$. We take this line as the new active line, *i.e.* we set $o = o_j$ and $l = l_{j,i}$, and we restart at the point where we search for intersection points (see Figure 11). If the minimum is reached with several increasing lines, we can take the line with the smallest slope, *i.e.* the largest $l_{j,i}$.

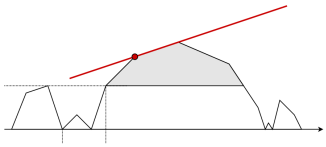

Fig. 11. We restart from the new vertex

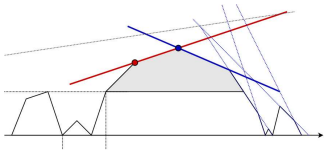

Fig. 12. Until we reach the optimum

A phase involves the computation of up to $2N$ intersection points. At each phase, the slope of the active increasing line strictly decreases (*i.e.* the delay l

[2] Indeed, since l has been choosen to be minimal, we easily check that the α-value of the intersection point is strictly better than α_{frac} *iff* $l_{j,i} > l$.

increases). Therefore, the number of phases is bounded by the number of distinct delays (itself bounded by N), which gives a complexity in $O(N^2)$. This gives us a method to compute an integral solution in $O(N^2)$ since once we have a fractional solution, we can deduce an integral solution with an additional computation in $O(N)$. Indeed, if x is integer, then (x, α) is the desired solution. Otherwise we can compute the α-values α_- and α_+ associated with $\lfloor x \rfloor$ and $\lceil x \rceil$ and take the largest one. Note that since $\lfloor x \rfloor$ (*resp.* $\lceil x \rceil$) is on the increasing phase (*resp.* decreasing phase), only the corresponding constraints are needed to compute α_- (*resp.* α_+) :

$$\alpha_- = \min_{k \in \mathcal{G}(i)} (\lfloor x \rfloor - o_k)/l_{k,i} \qquad \text{and} \qquad \alpha_+ = \min_{k \in \mathcal{G}(i)} (o'_k - \lceil x \rceil)/l_{i,k} \tag{20}$$

Improvement due to integrality assumption Instead of computing the value α_- at the end, we can compute this value during the algorithm. For this, it is possible to maintain a variable x_{int} representing the current best integral solution, and α_{int} representing its value. Another problem with the dual simplex procedure is that between two integer offsets, there is possibly several intermediary intersection points. In order to skip this unnecessary points, we modify alternative $A2$ as follow :

A2' Since $A1$ didn't occur, we know that we are not at the fractional optimum. However it is still possible that the fractional optimum belongs to $(x_{\text{int}}, x_{\text{int}} + 1]$. In order to test this, we use procedure ValueAndDerivative to compute the value α' associated with $x_{\text{int}} + 1$, and the active line at this point. If a decreasing line is active, this means we passed the fractional optimum. In this case, we also return the optimal solution which is the best solution among $(x_{\text{int}}, \alpha_{\text{int}})$ and $(x_{\text{int}} + 1, \alpha')$. Otherwise, only an increasing line is active at $x_{\text{int}} + 1$. We redefine $x - o = \alpha l$ to the the equation of this line. We update $(x_{\text{frac}}, \alpha_{\text{frac}}) = (x_{\text{int}} + 1, \alpha')$ and we restart at the point were we search for intersection points.

With this modification, x_{int} is garanteed to increase by one on each round. This allows to obtain a complexity in $O(NW)$ where W is the width of the polyhedron.

2.4 Solving the best response problem

We now have all the elements to describe the procedure which solves (BR_i). We saw that t_i can be supposed to belong to $\{0, \cdots, T_i - 1\}$. More generally we can start at any initial offset x and run on the right until we reach the offset $x + T_i - 1$, hence we obtain a solution $t_i \in \{x, \cdots, x + T_i - 1\}$. If needed, we can consider $t_i \bmod T_i$ which is an equivalent solution in $\{0, \cdots, T_i - 1\}$. Following the idea described in section 2.2, we can compute the local polyhedron (on the right) which contains the current reference offset. Using the dual simplex method presented in 2.3, we solve the associated problem $(Loc{-}BR_i)$. If the local optimum is better than the current solution, we update. We now want to go to the next polyhedron. At the local optimum, there are two active lines, an increasing

one with equation $x - o_j = l_{j,i}\alpha$ and a decreasing one with equation $o'_k - x = l_{i,k}\alpha$. Since the procedure runs from left to right, we follow the decreasing line until we reach the x-axis, *i.e.* the offset o'_k. We can use this point as the new reference point. We continue until the period has been traversed. This method is illustrated on Figure 13.

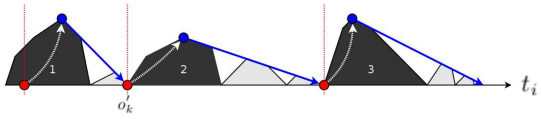

Fig. 13. Principle of the best response procedure

If we use Megiddo algorithm to solve the local problems, this best response algorithm runs in $O(n_{\text{poly}}N)$ and we saw that in the integral case, n_{poly} is bounded by T_i. If we use the dual simplex adapted for the integral case, we have a complexity in $O(T_iN)$ since the complexity on each polyhedron is proportional to its width. Note that in practice, a lot of polyhedra are skipped (see grey polyhedra in Figure 13).

2.5 Some possible improvements

The case of processing times In the case of processing times, no such refinement as Megiddo algorithm is needed in order to obtain a complexity in $O(N)$. Indeed, in this case, we have $l_{i,j} = p_i$. Since the delays $l_{i,j}$ gives the slope of the decreasing lines, this implies that all of them have the same slope $-1/p_i$. Thus, even if initially there are N decreasing lines with equations $o'_j - x = \alpha p_i$, we can remove all of them except one : the one with the smallest o'_j. Once this decreasing line is selected, we compute the intersection points with the N increasing lines (having equation $x - o_j = \alpha p_j$), and we select the one with the smallest α-value. In order to jump over more polyhedra, it is also more interesting to run from right to left (see Figure 14).

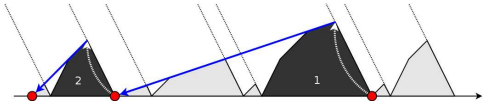

Fig. 14. The case of processing times

Finding the next polyhedron Suppose that after a local optimization, the current best solution t_i has value $\alpha_{\min}$. This becomes a lower bound for the optimal value and we are only interested by polyhedra which could improve it. However if $\alpha_{\min}$ is a good value, most of the polyhedra will lie completely below this level. Therefore, once a local optimum has been reached, it is possible to add a propagation mechanism which allows to skip more polyhedra by finding the smallest offset strictly greater than t_i with an α-value strictly greater than $\alpha_{\min}$. Instead of starting again at o'_k (which has a zero α-value), as described in section 2.4, we start at this new offset. Then, we are sure to improve the α-value.

3 The multiprocessor problem

3.1 Definition and method

MILP formulation of the multiprocessor problem In the multiprocessor case, the scheduling problem is coupled with an assignment problem. In order to define a MILP, we introduce binary variables $a_{i,k}$ which indicate if a task i is assigned to a processor k, and variables $x_{i,j}$ which indicate if i and j are on different processors. These variables must satisfy (22) and (23). For a given couple $(i,j) \in \mathcal{G}$, with $i < j$, the non-overlapping constraint (7) can be rewritten, using an additional 'quotient variable' $q_{i,j}$, as $l_{i,j}\alpha \leq t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha$. Since it has to be satisfied whenever the two tasks are on the same processor, this yields constraints (24) and (25).

$$\max \quad \alpha \tag{21}$$

$$\sum_k a_{i,k} = 1 \qquad \forall i \tag{22}$$

$$x_{i,j} \leq 2 - a_{i,k} - a_{j,k} \qquad \forall k, \quad \forall (i,j) \in \mathcal{G}, i < j \tag{23}$$

$$t_j - t_i + g_{i,j}q_{i,j} \geq l_{i,j}\alpha - \alpha_{\max} l_{i,j} x_{i,j} \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{24}$$

$$t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha + \alpha_{\max} l_{j,i} x_{i,j} \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{25}$$

$$t_i \in \mathbb{Z} \cap [0, T_i - 1] \qquad \forall i \tag{26}$$

$$q_{i,j} \in \mathbb{Z} \cap [1 - T_j/g_{i,j}, T_i/g_{i,j}] \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{27}$$

$$x_{i,j} \in [0,1] \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{28}$$

$$a_{i,k} \in \{0,1\} \qquad \forall k, \forall i \tag{29}$$

In the multiprocessor case, $\alpha_{\max}$ cannot be defined by $\min_{i,j} \alpha^{i,j}_{\max}$ as in proposition 1, since it corresponds to the worst case where all the tasks are on the same processor. Instead, we can use $\alpha_{\max} = \max_{i,j} \alpha^{i,j}_{\max}$. However, since $\alpha_{\max}$ is used as a 'big-M', we try to find its lowest possible value in order to improve the efficiency of the model. For this, we solve a preliminary model in which constraints (24-27) are replaced by the following weaker constraint :

$$\alpha \leq \alpha^{i,j}_{\max} + \alpha_{\max} x_{i,j}, \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{30}$$

The resolution of this model is much faster than for the original one (less that $1s$ for the instances considered in the results). It gives us a new value $\alpha_{\max}$ which can be used in the original model.

Multiprocessor best-response The adaptation of the heuristic to the multiprocessor case is quite simple. On each phase, a task is optimized and its offset and assignment are changed accordingly. In order to solve the multiprocessor best response, we solve the best response on each processor and move the task on the processor which gives the best result.

3.2 Results

We test the method on non-harmonic instances, the same which were used in [1, 2]. These instances were generated using the procedure described in [3] : the periods were choosen in the set $\{2^x 3^y 50 \mid x \in [0,4], y \in [0,3]\}$ and the processing times were generated following an exponential distribution and averaging at about 20% of the period of the task. The results on 25 instances with $N = 20$ tasks and $P = 4$ processors are presented in Table 1.

	MILP (200s)		Original heuristic [2] (bayesian test)				New heuristic (2s)			
id	α_{MILP}	$\mathbf{time}_{\text{sol}}$	$\alpha_{\text{heuristic}}$	$\mathbf{time}_{\text{single}}$	$\mathbf{time}_{\text{stop}}$	$\mathbf{starts}_{2s}$	$\alpha_{\text{heuristic}}$	$\mathbf{starts}_{\text{sol}}$	$\mathbf{starts}_{2s}$	$\mathbf{time}_{\text{sol}}$
0	2.5	159	2.3	1.43	101.33	1.4	2.5	35	3624	0.01932
1	2	18	2.01091	3.27	5064.67	0.61	2.01091	28	4477	0.01251
2	1.6	6	1.40455	1.52	869.45	1.32	1.6	2	2462	0.00162
3	1.6	4	1.6	4.34	8704.45	0.46	1.64324	45	2910	0.03093
4	2	5	1.92	3.48	1115.51	0.57	2	1	2489	0.00080
5*	3	7	1.43413	1.63	1498.21	1.23	3	1	2107	0.00095
6	2.5	54	2.3	1.44	101.25	1.39	2.5	35	3805	0.01840
7	2	19	2	0.23	302.27	8.7	2	3	4431	0.00135
8	2.12222	8	1.75794	1.03	871.8	1.94	2.12222	3	2513	0.00239
9*	2	11	2	2.42	3541.79	0.83	2	3	3575	0.00168
10	1.12	6	0.87	0.72	368.44	2.78	1.12	4	3466	0.00231
11	2.81098	20	0.847368	3.78	478.63	0.53	2.81098	1	3421	0.00058
12	1.5	7	1.5	0.27	313.74	7.4	1.5	4	3645	0.00219
13	1.56833	49	1.5	1.77	3293.33	1.13	1.56833	1	3863	0.00052
14	2	8	2	1.85	3873	1.08	2	2	2331	0.00172
15	1.28	7	1.28	7.89	5468.86	0.25	1.28	1	1782	0.00112
16	1.8	24	1.55	0.38	314.87	5.26	1.8	39	3802	0.02052
17	1	5	0.58	2.43	815.08	0.82	1	1	2507	0.00080
18	2.25	7	2	1.99	3059.82	1.01	2.25	2	1522	0.00263
19	2.3	6	2.3	0.99	710.03	2.02	2.3	1	2537	0.00079
20	1.65506	86	1.6	0.97	817.33	2.06	1.65506	12	2014	0.01192
21	1.90312	13	0.97	1.31	205.95	1.53	1.90312	195	3123	0.12488
22	2.07636	11	1.66364	2	1093.37	1	2.07636	40	5574	0.01435
23	3.7	4	1.36	4.23	2034.59	0.47	3.7	2	4540	0.00088
24	1.73	36	1.41053	1.06	481.4	1.89	1.73	19	2753	0.01380

Table 1. Results of the MILP, the heuristic of [1], and the new version of the heuristic

Columns 2 and 3 give the results of the MILP formulation. One advantage of the MILP over the heuristic could be that it gives a proof of optimality. However, it is rarely the case even with $1h$ of CPU time (optimality could only be proved for instances 5 and 9 in $7s$ and $20s$, respectively). Therefore, we fix a timeout of $200s$. α_{MILP} represents the best solution found during this time, and $\mathbf{time}_{\text{sol}}$ the time needed to find it. Table 1 also includes results about the original heuristic presented in [1, 2] (columns 4-7). Here, $\mathbf{time}_{\text{single}}$ is the time needed for a single run of the heuristic. In the results presented in [1, 2], the heuristic was started several times with different initial solutions until a bayesian test was satisfied, which gives a value $\mathbf{time}_{\text{stop}}$. In order to compare with our version of the heuristic, we add a column $\mathbf{starts}_{2s} = 2/\mathbf{time}_{\text{single}}$ which measures the average number of starts performed by the original heuristic in $2s$. Table 1 also includes the results for our version of the heuristic, called the new heuristic (columns 8-11). The results of the original heuristic show that on a lot of instances, it stops with a solution which is far from the best solution found

by the MILP. Therefore, for the new heuristic, we abandoned the bayesian test and we simply run the heuristic during $2s$. The value $\mathbf{start}_{2s}$ is the number of times the heuristic was started during this time. We immediately see that this number is much greater that the equivalent number for the original heuristic. Therefore our heuristic is much faster (about 1660 times on these instances). Moreover, $\mathbf{start}_{sol}$ is the number of starts needed in order to find the best solution, and $\mathbf{time}_{sol} = 2\mathbf{start}_{sol}/\mathbf{start}_{2s}$ represents approximately the time needed to find the best solution. This is to compare with the column $\mathbf{time}_{sol}$ of the MILP formulation, which shows that our version of the heuristic is very competitive compared to the MILP. Finally, we performed additional tests on big instances (48 processors, 1000 tasks), which show that our heuristic can give feasible solutions ($\alpha \geq 1$) in about $1min$, while these instances cannot even be loaded by the MILP solver.

4 Conclusion

We have proposed an enhanced version of the original heuristic proposed in [1, 2] to solve a NP-hard strictly periodic scheduling problem. Inspired by game theory, the heuristic reaches an equilibrium by iteratively solving best response problems. We propose acceleration techniques taking advantage of the two-dimentionality of the best response problem. The results show that the new heuristic greatly improves the original one and compares favorably with MILP solutions.

References

1. A. Al Sheikh. Resource allocation in hard real-time avionic systems - Scheduling and routing problems. PhD thesis, LAAS, Toulouse, France, 2011.
2. A. Al Sheikh, O. Brun, P.E. Hladik, B. Prabhu. Strictly periodic scheduling in IMA-based architectures. Real Time Systems, Vol 48, N°4, pp.359-386, 2012.
3. F. Eisenbrand, K. Kesavan, R.S. Mattikalli, M. Niemeier, A.W. Nordsieck, M. Skutella, J. Verschae, A. Wiese. Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods. ESA 2010, pp.11-22, 2010.
4. J. Korst. Periodic multiprocessors scheduling. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.
5. N. Megiddo. Linear-Time Algorithms for Linear Programming in R^3 and Related Problems. SIAM J. Comput., Vol 12, N°4, pp.759-776, 1983.

A Appendix

Proposition 3. *If* $x \bmod a = 0$*, then* $(-x) \bmod a = 0$*, otherwise* $(-x) \bmod a = a - x \bmod a$*. In particular* $x \bmod a > 0 \Leftrightarrow (-x) \bmod a > 0$.

Proposition 4. *(1) The smallest* $y \geq a$ *such that* $(y-b) \bmod c = 0$ *is given by* $y = a + (b-a) \bmod c$*. (2) The smallest* $y > a$ *such that* $(y-b) \bmod c = 0$ *is given by* $y = a + c - (a-b) \bmod c$*. (3) The largest* $y \leq a$ *such that* $(y-b) \bmod c = 0$ *is given by* $y = a - (a-b) \bmod c$*. (4) The largest* $y < a$ *such that* $(y-b) \bmod c = 0$ *is given by* $y = a - c + (b-a) \bmod c$.