Line search method for solving a non-preemptive strictly periodic scheduling problem

Clément Pira, Christian Artigues

Line search method for solving 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

Abstract

We study a non-preemptive strictly periodic scheduling problem. This problem arises for example in the avionic field where a set of N periodic tasks (measure of a sensor, data presentation, etc.) has to be scheduled on P processors distributed on the plane. In this article, we consider an existing heuristic which is based on the notion of equilibrium. Following a game theory analogy, each task tries successively to optimize its own schedule and therefore to produce the best response, given the other schedules. This iterative process continues until an equilibrium is reached. We present a new method to compute the best response which significantly improves the previously proposed heuristic, and compares favorably with MILP solutions.

keywords Periodic scheduling equilibrium propagation mechanism

1 Introduction

The problem of scheduling a set of N periodic tasks on a set of P processors has a long history. The preemptive case has received more attention, starting with the work of Liu and Layland [12]. Under this hypothesis, good schedulability conditions can be derived, and the problem is efficiently solved by the EDF algorithm (Earliest Deadline First), at least in the uniprocessor context. In the non-preemptive case, schedulability conditions are generally much weaker and the problem becomes NP-complete in the strong sense, even with one processor, as shown by Jeffay *et al* [9]. The latter still considers a loose notion of periodicity in which the tasks have to execute in each time period, but not always at the same relative position. In order to enforce regularity of the schedule, some works have considered the problem of minimizing the jitter [5]. In the extreme case where the jitter is imposed to be zero, the problem is qualified as a strictly periodic problem. In the avionic field, a preemptive or loosely periodic schedule is problematic because of the need to certify solutions. In this article, we

consider a non-preemptive strictly periodic scheduling problem [10, 7, 8, 1]. In this problem, each task i has a fixed period T_i and a processing time p_i. They are subject to non-overlapping constraints : no two tasks assigned to the same processor can overlap during any time period (see Figure 1). A solution is given by an assignment of the tasks to the processors and, for each task by the start time t_i (offset) of one of its occurrences, the other starting at $t_i + kT_i$ by strict periodicity.

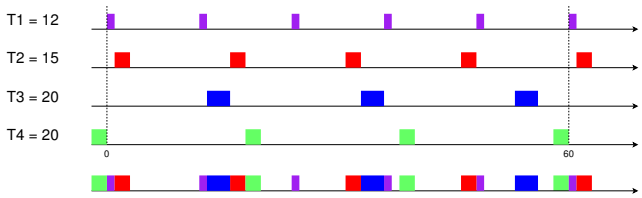

Figure 1: $N = 4$ non-overlapping periodic tasks on $P = 1$ processor

As shown by Baruah *et al* [4] and Korst *et al* [10, 11], this is a strongly NP-complete problem : with general processing times, it is NP-hard even with only two distincts periods, one dividing the other (harmonic case, reduction from 3-Partition). With general period, it remains NP-hard even with unitary processing times (reduction from k-Coloring [3]). It is however trivially solved in the monoperiodic case ($T_i = T$) : there is a solution *iff* $\sum_i p_i \leq T$.

In the following, unlike the papers mentioned above, 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. This last variant has been studied mostly in the monoperiodic case [6, 16]. In this case, the problem becomes NP-hard even in the monoperiodic case (reduction from Hamiltonian Circuit [16]). The proposed heuristic adapts easily to this generalization.

Previous works of the authors can be found in [14, 15]. The first article presents a simplified version of the heuristic (with a dual simplex method to solve the local best offset problem of section 4.1.3, and without the propagation mechanism of section 4.1.4). The second article presents the complete heuristic, but only in the case of integral offsets. In addition to considering the case of fractional offsets, the present paper gives much more details, especially regarding the satisfiability problem and its MILP formulation (see section 2.1), as well as about structural properties of the problem (see sections 3 and 5.2). Some experimental results have also been improved (see section 6).

The rest of the paper is organized as follows: in section 2, we present the periodic scheduling problem and its MILP formulation, as well as an additional objective function. In section 3, we study a decomposition of the problem which allows to deduce useful properties. Section 4 presents the best response problem and a method to solve it efficiently, while section 5 describes the heuristic. Results are presented in section 6.

2 Periodic scheduling problem and its MILP formulation

2.1 Definition of the satisfiability problem

In this section, we present a mathematical formulation of the feasibility problem, based on congruence inequations, as well as a natural translation into a mixed integer linear program. We emphasize the difficulty linked with zero delays, and in doing so, we show how to handle strict inequalities. In section 2.1.6, we also briefly discuss a generalization to negative delays.

2.1.1 Non-overlapping constraints

In the formulation of a uniprocessor periodic scheduling problem, a latency delay $l_{i,j} \geq 0$ has to be respected whenever an occurrence of a task j starts after an occurrence of a task i. Said differently, the smallest positive difference between two such occurrences has to be greater than $l_{i,j}$. Using Bézout identity, the set $(t_j + T_j\mathbb{Z}) - (t_i + T_i\mathbb{Z})$ of all the possible differences is equal to $(t_j - t_i) + g_{i,j}\mathbb{Z}$ where $g_{i,j} = \gcd(T_i, T_j)$. The smallest positive representative of this set is $(t_j - t_i) \mod g_{i,j}$ (in particular, we consider a classic positive modulo, and not a signed modulo). Therefore, we simply consider the following constraint :

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

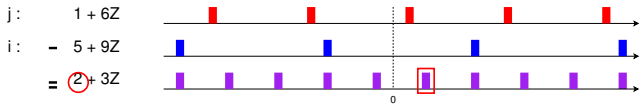

Figure 2: $(t_j - t_i) \bmod g_{i,j}$ is the smallest positive difference between an occurrence of task j and one of task i. This quantity should be greater than a prescribed delay $l_{i,j}$.

Since $(t_j - t_i) \bmod g_{i,j}$ is the only element of the form $t_j - t_i + g_{i,j}q_{i,j}$ in the interval $[0, g_{i,j})$, we get the following reformulation :

$$l_{i,j} \leq (t_j - t_i) + g_{i,j}q_{i,j} < g_{i,j}, \qquad \forall (i,j),\ i \neq j \tag{2}$$

Here, $q_{i,j}$ is an additional integer 'quotient variable'. Most of the time, we will define a solution only by its offsets, which are the only relevant variables. In fact, quotients will be handled implicitely in the heuristic. They are mostly artificial variables, which are completely determined by the offsets through equation :

$$q_{i,j} = \lceil (t_i - t_j)/g_{i,j} \rceil \tag{3}$$

Classically, processing times are strictly positive. At first sight, allowing zero delays seems obvious (see 2.1.6 for a discussion about negative delays). In fact, if $l_{i,j} = 0$, then the constraint associated with (i, j) is trivially satisfiable; just compute $q_{i,j}$ afterwards by expression (3). We could therefore only consider constraints associated with arcs in $\mathcal{R} = \{(i, j) \mid i \neq j,\ l_{i,j} > 0\}$. The problem is however that equation (2) involves a strict inequality which makes it unsuitable for MILP solving. As we will see, it can be removed easily only when $\mathcal{R}$ is symmetric, *i.e.* when $l_{i,j}$ and $l_{j,i}$ are either both strictly positive or none of them.

For the moment, we can at least remove an arc (i, j) if both $l_{i,j}$ and $l_{j,i}$ are zero. We therefore define a symmetric graph $\mathcal{G} = \{(i, j) \mid i \neq j,\ l_{i,j} + l_{j,i} > 0\}$ and consider constraints (2) for each one of those couples. The two contraints associated with arcs (i, j) and (j, i) work naturally together : by summing them, we get $(l_{i,j} + l_{j,i})/g_{i,j} \leq q_{i,j} + q_{j,i} < 2$. This first gives us a necessary schedulability condition : two tasks i and j can be scheduled on a processor only if $l_{i,j} + l_{j,i} \leq g_{i,j}$. Moreover, since $l_{i,j} + l_{j,i} > 0$, and since the quotients are integer, we have in fact $q_{i,j} + q_{j,i} = 1$. We obtain an equivalent formulation for (2), from which we removed redundancy :

$$(t_j - t_i) + g_{i,j}q_{i,j} \geq l_{i,j}, \qquad \forall (i, j) \in \mathcal{R} \tag{4}$$

$$(t_j - t_i) + g_{i,j}q_{i,j} > 0, \qquad \forall (i, j) \in \mathcal{G} \setminus \mathcal{R} \tag{5}$$

$$q_{i,j} + q_{j,i} = 1, \qquad \forall (i, j) \in \mathcal{G},\ i < j \tag{6}$$

The non-strict inequalities (4) are set only for $(i, j) \in \mathcal{R}$, *i.e.* when $l_{i,j} > 0$. The strict inequalities (5) only occur when $(i, j) \in \mathcal{G} \setminus \mathcal{R}$, *i.e.* when $l_{i,j}$ is zero, but $l_{j,i}$ is not[1]. In section 3.1.2, we show how to discard these pathological cases by replacing constraints (5) by constraints (7) below (where $\gamma = \gcd((T_i)_i, (l_{i,j})_{i,j})$ is the gcd of all the periods and delays) :

$$(t_j - t_i) + g_{i,j}q_{i,j} \geq \frac{\gamma}{N}, \qquad \forall (i, j) \in \mathcal{G} \setminus \mathcal{R} \tag{7}$$

In the following, $\mathcal{G}$ will only contain strictly positive delays ($\mathcal{G} = \mathcal{R}$) and we will only deal with non-strict inequalities.

2.1.2 Integral *vs* fractional offsets

It is common to additionally suppose that the offsets are integer. In this case, we will also suppose that all the periods T_i are integer : this is not necessary for the MILP to work, but it is mandatory if we want to maintain periodicity : if t_i is an integer solution then $t_i + T_i$ should also be an integer solution, hence their difference T_i should be integer. In particular, the cœfficients $g_{i,j}$ will be integer.

[1]Note that if $l_{j,i} > 0$, the tasks j and i are constrained not to start at the same time. Therefore, it is natural to have at least $(t_j - t_i) \bmod g_{i,j} > 0$ even when $l_{i,j} = 0$.

Strict inequalities are more easily tackled when we make this assumption. In this case, a general strict inequality $t_j - t_i + g_{i,j}q_{i,j} > l_{i,j}$ can always be converted into $t_j - t_i + g_{i,j}q_{i,j} \geq \lfloor l_{i,j} \rfloor + 1$, while a non-strict inequality (4) can always be converted into $t_j - t_i + g_{i,j}q_{i,j} \geq \lceil l_{i,j} \rceil$. In particular, the original constraints (4) and (5) can be captured together by :

$$(t_j - t_i) + g_{i,j}q_{i,j} \geq \max(1, \lceil l_{i,j} \rceil), \qquad \forall (i,j) \in \mathcal{G} \tag{8}$$

This transformation does not change the set of feasible offsets, which is stronger than what we get in the fractional case[2]. In the integral case, we can therefore suppose that each delay on $\mathcal{G}$ is integer and greater than 1.

2.1.3 Multiprocessor problem

In order to reduce the number of variables, and thus obtain a more efficient model, we remove variables $q_{i,j}$ for each $i > j$, and we use relation (6) to replace contraint (4) by the following interval constraint[3] :

$$l_{i,j} \leq (t_j - t_i) + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}, \quad \forall (i,j) \in \mathcal{G},\ i < j \tag{9}$$

In the multiprocessor context, this equation only has to hold between tasks i and j on the same processor. If $x_{i,j}$ is a binary variable representing this fact, then we get constraints (12) and (13). When i and j are on the same processor ($x_{i,j} = 1$), we get the original constraint, otherwise we get $0 \leq (t_j - t_i) + g_{i,j}q_{i,j} \leq g_{i,j}$, which is always satisfiable. In order to represent the assignment, we introduce binary variables ($a_{i,k}$) which indicate if a task i is assigned to a processor k. These variables must satisfy (10) and are related to ($x_{i,j}$) through equations (11). Note that variables ($x_{i,j}$) can be relaxed to be continuous variables (even unbounded), instead of binary ones[4]. We obtain the following program [8] :

$$\sum_k a_{i,k} = 1, \qquad \forall i \tag{10}$$

$$x_{i,j} \geq a_{i,k} + a_{j,k} - 1, \qquad \forall k, \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{11}$$

$$t_j - t_i + g_{i,j}q_{i,j} - l_{i,j}x_{i,j} \geq 0, \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{12}$$

$$t_j - t_i + g_{i,j}q_{i,j} + l_{j,i}x_{i,j} \leq g_{i,j}, \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{13}$$

$$t_i \in \mathbb{Q} \quad \text{or } t_i \in \mathbb{Z}, \qquad \forall i \tag{14}$$

$$q_{i,j} \in \mathbb{Z}, \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{15}$$

$$x_{i,j} \in [0,1], \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{16}$$

$$a_{i,k} \in \{0,1\}, \qquad \forall k, \forall i \tag{17}$$

[2] In the fractional case, it is obviously impossible to replace a (usefull) strict inequality by a non-strict one without changing the set of feasible offsets. The best we have is that the set of feasible quotients (in particular the feasibility of the system) is unchanged.

[3] Note that if $\mathcal{G}$ contained zero delays, we should add the following open interval constraints $0 < (t_j - t_i) + g_{i,j}q_{i,j} < g_{i,j}$.

[4] If there is a solution with continuous ($x_{i,j}$), we can define an equivalent solution from the assignment ($a_{i,k}$) by $x_{i,j} = \max_k a_{i,k} + a_{j,k} - 1$, which is binary.

2.1.4 Bounds on the offsets

Since $\mathcal{G}$ is symmetric, it can be seen as an undirected graph. In the following, we will write $\mathcal{G}(i)$ for the set of neighbors of i, *i.e.* the set of tasks linked to i through non-overlapping constraints.

The set of feasible offsets is periodic : if (t_i) is feasible and if (n_i) is a vector of integers, then the solution (t'_i) defined by $t'_i = t_i + n_i T_i$ is also feasible, provided we define new quotients $q'_{i,j} = q_{i,j} + n_i \frac{T_i}{g_{i,j}} - n_j \frac{T_j}{g_{i,j}}$. Here, the only important fact is that $q'_{i,j}$ is again integer since $g_{i,j}$ divides T_i and T_j. If (θ_i) were different periods, the reasoning would remain true as soon as $g_{i,j}$ would divide θ_i for each $(i,j) \in \mathcal{G}$. Hence, the smallest periods for which the set remains periodic are given by :

$$T_i^* = \mathrm{lcm}((g_{i,j})_{j \in \mathcal{G}(i)}) \tag{18}$$

Since $T_i^* = \gcd(T_i, \mathrm{lcm}((T_j)_{j \in \mathcal{G}(i)}))$, this updated period T_i^* always divides T_i, and can be strictly smaller than T_i, for example if a prime factor occurs with a given multiplicity in T_i but with strictly smaller multiplicities in any other periods of a task $j \in \mathcal{G}(i)$. Indeed, even if the (T_i) are the initial parameters, only the $(g_{i,j})$ appear in the constraints. Therefore, if a prime factor occurs in only one period, then it completely disappears in the $(g_{i,j})$. By replacing the initial periods (T_i) by the updated ones (T_i^*), we simply remove some irrelevant factors. In the following, we will suppose that the periods have been updated to have no proper factors, *i.e.* $T_i = T_i^*$.

Since $t_i \bmod T_i = t_i + n_i T_i$ for some $n_i \in \mathbb{Z}$, we deduce that $(t_i \bmod T_i)$ is also a solution. We can therefore impose t_i to belong to $[0, T_i)$.

2.1.5 Bounds on the quotients

Since we have $0 \leq t_j - t_i + g_{i,j} q_{i,j} \leq g_{i,j}$, these bounds on the offsets immediately induce associated bounds on the quotients :

$$1 - \frac{T_j}{g_{i,j}} \leq q_{i,j} \leq \frac{T_i}{g_{i,j}} \tag{19}$$

In the harmonic case, either T_j divides T_i, or T_i divides T_j. Hence, the quotient is either positive ($q_{i,j} \in [\![0, T_i/T_j]\!]$) or less than one ($q_{i,j} \in [\![1 - T_j/T_i, 1]\!]$). If $T_i = T_j$, then $q_{i,j} \in \{0, 1\}$ which shows that the monoperiodic case correspond exactly to the case of binary variables.

2.1.6 Allowing negative delays

Even if we generalized processing times by (possibly zero) latency delays, we still considered positive delays. Indeed, equation (1) is trivially satisfied when $l_{i,j} \leq 0$, since by definition the modulo belongs to the reference interval $[0, g_{i,j})$.

In order to generalize, we can however bypass the modulo formulation and directly consider a set of interval constraints of the form :

$$l_{i,j} \leq t_j - t_i + g_{i,j} q_{i,j} \leq u_{i,j}, \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{20}$$

Here, $l_{i,j}$ and $u_{i,j}$ can be any values. In order to be feasible, we need at least to have $l_{i,j} \leq u_{i,j}$. Moreover, the constraint is trivially satisfiable and can be removed, unless $u_{i,j} - l_{i,j} < g_{i,j}$.

Note however, that we can always write $u_{i,j}$ in the form $g_{i,j} - l_{j,i}$. Hence, we simply obtain constraint (9), but this time with arbitrary delays. Necessary condition $l_{i,j} \leq u_{i,j}$ gives $l_{i,j} + l_{j,i} \leq g_{i,j}$, while non-triviality condition $u_{i,j} - l_{i,j} < g_{i,j}$ becomes $l_{i,j} + l_{j,i} > 0$. Therefore we can still define graph $\mathcal{G}$ and consider only the constraints associated with these arcs. On $\mathcal{G}$, we still have $q_{i,j} + q_{j,i} = 1$. We can still replace strict inequalities if there are some (see section 3.1.2). In the multiprocessor context, we can still multiply the delays by $x_{i,j}$ in order to make the contraints conditional. Offsets can still be choosen in $[0, T_i)$. Hence, most of the MILP model still works with arbitrary delays. However, since we dropped the reference interval $[0, g_{i,j})$, quotients cannot be retrieved from the offsets by (3), and are not bounded anymore by (19). Instead, we now have[5] $q_{i,j} = \left\lceil \frac{t_i - t_j + l_{i,j}}{g_{i,j}} \right\rceil$ and :

$$\left\lfloor \frac{l_{i,j}}{g_{i,j}} \right\rfloor + 1 - \frac{T_j}{g_{i,j}} \leq q_{i,j} \leq \frac{T_i}{g_{i,j}} - \left\lfloor \frac{l_{j,i}}{g_{i,j}} \right\rfloor \tag{21}$$

2.2 Definition of an optimization problem

In this section, we investigate an objective function. The α-criterion given below amounts to maximizing space between tasks, and the heuristic presented in this paper is based on this idea. In the following, we will suppose that all the delays are strictly positive on $\mathcal{G}$.

2.2.1 Objective to maximize

We opt for an objective useful in the context of robustness. The latter is concerned with the problem of computing solutions which can withstand uncertainties on the parameters. In the context of hard real-time systems, such as those encountered in the avionics domain, the uncertainties on the delays (processing times, latencies) are particularly critical. A task may last longer than expected,

[5] From equation (9), we obtain $(t_i - t_j + l_{i,j})/g_{i,j} \leq q_{i,j} \leq 1 + (t_i - t_j - l_{j,i})/g_{i,j}$. After rounding, we deduce the smallest integer solution $q_{i,j} = \lceil \frac{t_i - t_j + l_{i,j}}{g_{i,j}} \rceil$ (and the largest $1 - \lceil \frac{t_j - t_i + l_{j,i}}{g_{i,j}} \rceil$). However, when $l_{i,j} + l_{j,i} > 0$, there is in fact at most one solution. Applying the bounds $t_i \in [0, T_i)$, we further get $(l_{i,j} - T_j)/g_{i,j} < q_{i,j} < 1 + (T_i - l_{j,i})/g_{i,j}$. After rounding, we get $\lfloor \frac{l_{i,j} - T_j}{g_{i,j}} \rfloor + 1 \leq q_{i,j} \leq \lfloor \frac{T_i - l_{j,i}}{g_{i,j}} \rfloor$, hence the desired expression, since $g_{i,j}$ divides T_i and T_j.

and therefore these parameters are generally overestimated *a priori*. Instead of this, we define here a measure of the robustness of a schedule which captures its capacity to handle increases of the durations. For this, we suppose that the possible variations of the delays are proportional to their original values : tasks with large processing times are more likely to be subject to large overrun. Hence, all the delays $l_{i,j}$ are made proportional to a common factor $\alpha \geq 0$ that we try to optimize. In a uniprocessor context, this gives [1] :

$$\begin{aligned}
\max \quad & \alpha && (22)\\
\text{s.t.} \quad & l_{i,j}\alpha \leq t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha \\
& \qquad\qquad \forall (i,j) \in \mathcal{G},\ i<j && (23)\\
& t_i \in \mathbb{Q} \quad \text{or } t_i \in \mathbb{Z} \qquad \forall i && (24)\\
& q_{i,j} \in \mathbb{Z} \qquad \forall (i,j) \in \mathcal{G},\ i<j && (25)\\
& \alpha \geq 0 && (26)
\end{aligned}$$

The resulting model always admits a trivial feasible solution (with $\alpha = 0$). Whenever a solution with $\alpha \geq 1$ is found, we have a solution for the feasibility problem. If a factor larger than one is found, the schedule can handle critical situations where all the delays have been multiplied by this factor.

2.2.2 Upper bound on the α-value

Let i and j be two tasks on the same processor. Then, constraint (23) implies the following upper bound :

$$\alpha \leq \alpha^{i,j}_{\max} = \frac{g_{i,j}}{l_{i,j} + l_{j,i}} \tag{27}$$

In the integral case, we can round the left and right terms of (23). Hence, we have in fact $\lceil l_{i,j}\alpha \rceil + \lceil l_{j,i}\alpha \rceil \leq g_{i,j}$. This gives an even tighter upper bound[6] :

$$\alpha^{i,j}_{\max} = \max\left(\frac{1}{l_{i,j}} \left\lfloor \frac{g_{i,j}l_{i,j}}{l_{i,j}+l_{j,i}} \right\rfloor, \frac{1}{l_{j,i}} \left\lfloor \frac{g_{i,j}l_{j,i}}{l_{i,j}+l_{j,i}} \right\rfloor \right) \tag{28}$$

These bounds are illustrated on Figure 3. In the uniprocessor case, we deduce that $\alpha_{\max} = \min_{i,j} \alpha^{i,j}_{\max}$ is an upper bound on the value of α. In the multiprocessor case, we have at least the trivial upper bound $\alpha_{\max} = \max_{i,j} \alpha^{i,j}_{\max}$ (we will see how to improve this value in the next section).

[6] Indeed, let $\alpha^{i,j}_{\max} = \max\{\alpha \mid \lceil l_{i,j}\alpha \rceil + \lceil l_{j,i}\alpha \rceil \leq g_{i,j}\}$ be the largest value satisfying this condition. We easily see that we have either $l_{i,j}\alpha^{i,j}_{\max} \in \mathbb{Z}$, or $l_{j,i}\alpha^{i,j}_{\max} \in \mathbb{Z}$. Suppose w.l.o.g. that $l_{i,j}\alpha^{i,j}_{\max} = n \in \mathbb{Z}$. Then n is the largest integer such that $n + \lceil nl_{j,i}/l_{i,j} \rceil \leq g_{i,j}$, *i.e.* $nl_{j,i}/l_{i,j} \leq g_{i,j} - n$ since the right term is integer, *i.e.* $n \leq \frac{g_{i,j}l_{i,j}}{l_{i,j}+l_{j,i}}$. This implies $n = \lfloor \frac{g_{i,j}l_{i,j}}{l_{i,j}+l_{j,i}} \rfloor$, which completes the proof.

2.2.3 Multiprocessor MILP formulation

With this additional objective, it is now more convenient to define a variable $x_{i,j}$ representing the fact that tasks i and j are on different processors. In this case, these variables are related to the $(a_{i,k})$ through (31) instead of (11). Since the non-overlapping constraints have to hold whenever two tasks are on the same processor, we replace the term $l_{i,j}\alpha$ by $l_{i,j}\alpha - l_{i,j}\alpha_{\max}x_{i,j}$, which gives constraints (32) and (33). When i and j are on the same processor ($x_{i,j} = 0$), we get back the original term $l_{i,j}\alpha$. Otherwise, we obtain a negative term $l_{i,j}\alpha - l_{i,j}\alpha_{\max}$ which makes the constraint trivially satisfiable.

$$\max \quad \alpha \tag{29}$$

$$\text{s.t.} \quad \sum_k a_{i,k} = 1, \qquad \forall i \tag{30}$$

$$x_{i,j} \leq 2 - a_{i,k} - a_{j,k}, \quad \forall k, \forall (i,j) \in \mathcal{G}, i < j \tag{31}$$

$$t_j - t_i + g_{i,j}q_{i,j} \geq l_{i,j}\alpha - l_{i,j}\alpha_{\max}x_{i,j}, \quad \forall (i,j) \in \mathcal{G}, i < j \tag{32}$$

$$t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha + l_{j,i}\alpha_{\max}x_{i,j}, \quad \forall (i,j) \in \mathcal{G}, i < j \tag{33}$$

$$t_i \in [0, T_i] \qquad \text{or } t_i \in \mathbb{Z} \cap [0, T_i - 1], \qquad \forall i \tag{34}$$

$$q_{i,j} \in \left[\!\left[1 - \frac{T_j}{g_{i,j}}, \frac{T_i}{g_{i,j}}\right]\!\right], \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{35}$$

$$x_{i,j} \in [0,1], \qquad \forall (i,j) \in \mathcal{G}, i < j \tag{36}$$

$$a_{i,k} \in \{0,1\}, \qquad \forall k, \forall i \tag{37}$$

$$\alpha \in [0, \alpha_{\max}] \tag{38}$$

Since $\alpha_{\max}$ is used as a 'big-M' in this MILP formulation, we would like the lowest possible value in order to improve the efficiency of the model. For this, we can solve a preliminary model, dealing only with the assignment (hence without variables (t_i) and $(q_{i,j})$), in which non-overlapping constraints (32,33) are replaced by the following weaker constraint :

$$\alpha \leq \alpha^{i,j}_{\max} + (\alpha_{\max} - \alpha^{i,j}_{\max})x_{i,j}, \quad \forall (i,j) \in \mathcal{G},\ i < j \tag{39}$$

Intuitively, this constraint indicates that α should be less than $\alpha^{i,j}_{\max}$ if i and j are assigned to the same processor ($x_{i,j} = 0$), otherwise it is bounded by $\alpha_{\max}$ which is the trivial upper bound. Solving this model is much faster than for the original one (less that $1s$ for instances with 4 processors and 20 tasks). It gives us a new value $\alpha_{\max}$ which can be used in the original model.

3 Decomposition and structural properties

Even if the formulation of the periodic scheduling problem involves two distinct sets of variables, the offsets (t_i) and the quotients $(q_{i,j})$, the two are related. We

already saw how quotients are determined by offsets through equation (3). In the following we will emphasize conversely that offsets can be efficiently recovered from quotients. This subsequently allows to deduce several useful properties.

3.1 Decomposition of the satisfiability problem

3.1.1 Recovering the offsets from the quotients

In this section, we suppose generally that we want to solve a system composed of strict inequalities (indexed by $(i,j) \in \mathcal{S}$), and non-strict inequalities (indexed by $(i,j) \in \mathcal{L}$), with $\mathcal{S} \cap \mathcal{L} = \emptyset$:

$$(t_j - t_i) + g_{i,j} q_{i,j} \geq l_{i,j}, \qquad \forall (i,j) \in \mathcal{L} \tag{40}$$

$$(t_j - t_i) + g_{i,j} q_{i,j} > l_{i,j}, \qquad \forall (i,j) \in \mathcal{S} \tag{41}$$

Note first that once the quotients are fixed, precedence constraints naturally occur as a subproblem :

$$t_j - t_i \geq l_{i,j} - g_{i,j} q_{i,j}, \qquad \forall (i,j) \in \mathcal{L} \tag{42}$$

$$t_j - t_i > l_{i,j} - g_{i,j} q_{i,j}, \qquad \forall (i,j) \in \mathcal{S} \tag{43}$$

As long as there are no strict inequalities ($\mathcal{S} = \emptyset$), this subproblem is simply the dual of a longest-path problem. Hence, compatible offsets can be recovered from the quotients by the Bellman-Ford algorithm. In addition, not every choice for $(q_{i,j})$ produces a solution for (t_i). It is well-know, using Farkas lemma, that system (42) has a solution *iff* there is no elementary circuit with a strictly positive cost, *i.e.* (44). If there are strict inequalities, we can use Motzkin transposition theorem instead to define a set of constraints describing the feasible quotients, *i.e.* the quotients associated with a feasible solution. Those are defined by :

$$\sum_{(i,j) \in C} g_{i,j} q_{i,j} \geq \sum_{(i,j) \in C} l_{i,j}, \qquad \forall \text{ el}^{\text{t}}\text{circuit } C \subseteq \mathcal{L} \tag{44}$$

$$\sum_{(i,j) \in C} g_{i,j} q_{i,j} > \sum_{(i,j) \in C} l_{i,j}, \qquad \forall \text{ el}^{\text{t}}\text{circuit } C \not\subseteq \mathcal{L} \tag{45}$$

In the previous constraints the elementary circuits $C \subseteq \mathcal{L}$ only involve non-strict inequalities, while the circuits $C \not\subseteq \mathcal{L}$ contain strict inequalities ($C \cap \mathcal{S} \neq \emptyset$).

Remark 1. *The projection on the quotients space is naturally associated with a Benders decomposition scheme*[7]*. However, until now, we did not get competitive*

[7]There is an exponential number of circuit constraints (44). Therefore, in a Benders approach, a master problem tries to find the hard quotients variables, initially considering only a subset of these constraints. For example, it can start with constraints (6) (which implies constraints associated with circuits of length 2). Given a first guess of the master, the subproblem tries to find compatible offsets solving (42). If it succeeds, we obtain a solution to the original problem. Otherwise, the subproblem ends with a circuit having a strictly positive cost. The associated constraint is added in the master, which is then solved again. The loop continues until the subproblem has a solution or the master has none.

results with such an approach.

3.1.2 Removing the strict inequalities

Let $\gamma = \gcd((T_i)_i, (l_{i,j})_{i,j})$ be the gcd of all the periods and delays. If we divide constraint (45) by γ, we obtain a strict inequality with integer cœfficients on integer variables. It can therefore be rounded, which gives an equivalent non-strict inequality :

$$\sum_{(i,j)\in C} g_{i,j}q_{i,j} \geq \sum_{(i,j)\in C} l_{i,j} + \gamma, \ \forall \text{ el}^{\text{t}}\text{circuit } C \not\subseteq \mathcal{L} \tag{46}$$

Proposition 1. *The strict inequalities* (41) *can be replaced by the following non-strict inequalities* (47)*, while preserving the set of feasible quotients :*

$$(t_j - t_i) + g_{i,j}q_{i,j} \geq l_{i,j} + \frac{\gamma}{N}, \qquad \forall (i,j) \in \mathcal{S} \tag{47}$$

Proof. On one side, the set of quotients for system (40,41) is caracterized (44,45), but also by (44,46). On the other side, system (40,47) only involves non-strict inequalities, hence the feasible quotients for this system are caracterized by :

$$\sum_{(i,j)\in C} g_{i,j}q_{i,j} \geq \sum_{(i,j)\in C} l_{i,j} + \gamma\frac{|C \cap \mathcal{S}|}{N}, \ \forall \text{ el}^{\text{t}}\text{circuit } C$$

If $C \subseteq \mathcal{L}$, the previous constraint simply reduces to (44) since $|C \cap \mathcal{S}| = 0$. If $C \not\subseteq \mathcal{L}$, it is stronger than (45) since $|C \cap \mathcal{S}| > 0$, but it is weaker than (46) because C is elementary, hence we have $|C \cap \mathcal{S}|/N \leq 1$. Therefore, the previous constraints define the same set of quotients as (44,45) and (44,46). □ □

3.1.3 Equivalence between the integral and the fractional variants

In previous works [14, 15], we only considered the integral case. This hypothesis was important to prove convergence of the heuristic. In this article, we have also considered the fractional case. Note however that, if there are only non-strict inequalities, we easily move from one variant to the other, thanks to the fact that subproblem (42) is a longest-path problem :

- If we can solve instances with fractional offsets, then we can solve instances with integral offsets. Indeed, in the latter case, we saw in section 2.1.2 that all the data (periods and delays) can be supposed or made integer. If we have a fractional solution, we can retain only the quotients and use the subproblem to compute new offsets. Since the right-hand side of this subproblem is integral, the Bellman-Ford algorithm will give an integral solution.

- Conversely, if we can solve instances with integral offsets, then we can solve instances with fractional offsets. Indeed, (t_i) is a fractional solution for

instance $(T_i, l_{i,j})$ *iff* (t_i/γ) is one for instance $(T_i/\gamma, l_{i,j}/\gamma)$. But now, all the parameters T_i/γ and $l_{i,j}/\gamma$ are integer. Hence the instance can be solved with integral offsets.

This equivalence does not hold for the optimization problem (see section 3.2).

3.1.4 Schedulability conditions

A well-known necessary condition for the schedulability of two tasks is $l_{i,j}+l_{j,i} \leq g_{i,j}$ (see section 2.1.1). If zero delays are allowed, we must add the additional conditions $l_{i,j} < g_{i,j}$ and $l_{j,i} < g_{i,j}$. If there are only two tasks, these conditions are also sufficient[8].

The following proposition goes one step futher (for simplicity, we suppose that there are only non-strict inequalities). It is a modest consequence of decomposition. As far as we know, the sufficiency of this condition with three tasks is new. If C is a circuit, we write $g_C = \gcd((g_{i,j})_{(i,j)\in C})$. We have :

Proposition 2. *If N tasks are schedulable on the same processor, then we have :*

$$\left\lceil \frac{\sum_{(i,j)\in C} l_{i,j}}{g_C} \right\rceil + \left\lceil \frac{\sum_{(i,j)\in C} l_{j,i}}{g_C} \right\rceil \leq \frac{\sum_{i\in C} g_{i,j}}{g_C}, \qquad \forall \text{ el}^{\text{t}}\text{circuit } C \tag{48}$$

If there are at most three tasks, the converse is true.

Proof. Since, we only consider non-strict inequalities, the set of feasible quotients for system (4,6) is defined by (44) and (6). From these two constraints, we get $\sum_{(i,j)\in C} l_{i,j} \leq \sum_{(i,j)\in C} g_{i,j}q_{i,j} \leq \sum_{(i,j)\in C} (g_{i,j} - l_{j,i})$. We can divide this expression by g_C. Since the middle term remains integer, we can round the left and the right term, and remove the middle term, which gives (48). Conversely we want to show that, if there are only 3 tasks, and if this condition is satisfied, system (44,6) is satisfiable. We need to consider only one circuit $C = (i, j, k)$ of size 3 (the reversed circuit is handled simultaneously). If condition (48) is satisfied for this circuit, we can find an integer value q such that $\sum_{(i,j)\in C} l_{i,j} \leq g_C q \leq \sum_{(i,j)\in C} (g_{i,j} - l_{j,i})$. Using Euclid algorithm, we can find values $q_{i,j}$, $q_{j,k}$ and $q_{k,i}$ such that $g_C q = g_{i,j}q_{i,j} + g_{j,k}q_{j,k} + g_{k,i}q_{k,i}$. The remaining variables $q_{j,i}$, $q_{k,j}$ and $q_{i,k}$ are defined in order to satisfy (6). By these choices, constraints (44) associated with C and its reverse are satisfied. Moreover, since condition (48) is satisfied for each circuit of size 2, we get $2\lceil (l_{i,j} + l_{j,i})/g_{i,j} \rceil \leq 2$, hence $l_{i,j} + l_{j,i} \leq g_{i,j}$. This implies that all constraints (44) associated with circuits of size 2 are satisfied. □ □

In the case of processing times, a simple rearrangement gives the following corollary :

[8] If both $l_{i,j}$ and $l_{j,i}$ are zero, then $t_i = t_j = 0$ is a feasible solution. Otherwise, suppose w.l.o.g. that $l_{i,j} > 0$ and define $t_i = 0$ and $t_j = l_{i,j}$. Then $(t_j - t_i) \bmod g_{i,j} = l_{i,j}$ (since $l_{i,j} \in [0, g_{i,j})$) and $(t_i - t_j) \bmod g_{i,j} = g_{i,j} - l_{i,j} \geq l_{j,i}$. This shows that i and j are schedulable.

Corollary 1. *Let g be the* gcd *of all the periods. If N tasks are schedulable on the same processor, then we have :*

$$\sum_i p_i \leq g \cdot \left\lfloor \frac{1}{2} \cdot \frac{g_{1,2} + g_{2,3} + \cdots + g_{N,1}}{g} \right\rfloor \tag{49}$$

If there are at most three tasks, the converse is true.

Remark 2. *We can compare the necessary condition given by* (49) *with the obvious sufficient condition $\sum_i p_i \leq g$ (in fact, if the latter condition is satisfied, we can solve the monoperiodic instance with period g, and* a fortiori *the original multiperiodic instance).*

3.2 Decomposition of the optimization problem

3.2.1 Local optimum in the fractional case

Let us consider the optimization problem (22-26) with fractional offsets, defined in section 2.2.1. In order to simplify, let us suppose that we kept the two variables $q_{i,j}$ and $q_{j,i}$, linked through (6). If these quotients are fixed, the remaining problem has the following form :

$$\begin{aligned}
\max \quad & \alpha & & (50)\\
\text{s.t.} \quad & t_j - t_i \geq l_{i,j}\alpha - g_{i,j}q_{i,j} & \forall (i,j) \in \mathcal{G} \quad & (51)\\
& t_i \in \mathbb{Q} & \forall i \quad & (52)\\
& \alpha \geq 0 & & (53)
\end{aligned}$$

In a multiprocessor context, we suppose that the assignment (a_i) is fixed too. Hence, the resulting problem is similar, except that we only consider the arcs $(i,j) \in \mathcal{G}$ such that $a_i = a_j$. An optimal solution to this problem will be called a *local optimum*. This subproblem is in fact the dual of the *minimum cost-to-time ratio cycle problem*. The resolution of this problem is out of the scope of this paper, but we can sketch a polynomial algorithm which performs a dichotomy on the α-value. Indeed, for a given value of α, constraints (51) define the dual of a longest path problem. It can therefore be solved by the Bellman-Ford algorithm. If the latter succeeds, we can use the solution to improve the current lower bound on α (primal-dual step). Otherwise, it is well known, using Farkas lemma, that there is an elementary circuit C with a strictly positive cost : $\sum_{(i,j)\in C} l_{i,j}\alpha - g_{i,j}q_{i,j} > 0$. We deduce that a feasible solution should satisfy $\alpha \leq \frac{\sum_{(i,j)\in C} g_{i,j}q_{i,j}}{\sum_{(i,j)\in C} l_{i,j}}$. This gives us a new upper bound for the α-value. From the previous reasoning, we also deduce a simple optimality criterion :

Proposition 3. *A fractional feasible solution is locally optimal* iff *there is a* critical circuit *C such that $\alpha = \frac{\sum_{(i,j)\in C} g_{i,j}q_{i,j}}{\sum_{(i,j)\in C} l_{i,j}}$.*

3.2.2 Local optimum in the integral case

If the offsets are integer, we first need to round the right-hand side of (51) before we use a Bellman-Ford algorithm. Hence, we work on constraints of the form $t_j - t_i \geq \lceil l_{i,j}\alpha \rceil - g_{i,j}q_{i,j}$. Given this, a dichotomous approach is still possible. However, the optimality criterion is not valid anymore. Indeed, with integer offsets, the above program is now equivalent to the following :

$$\max \quad \alpha \tag{54}$$

$$\text{s.t.} \sum_{(i,j)\in C} g_{i,j}q_{i,j} \geq \sum_{(i,j)\in C} \lceil l_{i,j}\alpha \rceil, \ \forall \text{ el}^{\text{t}}\text{circuit } C \tag{55}$$

$$\alpha \geq 0 \tag{56}$$

This is not a linear program anymore. At least, fractional and 'integral' local optima are related :

Proposition 4. *If α^* is the fractional local optimum, then the 'integral' local optimum is at least $\min_{i,j} \frac{\lfloor l_{i,j}\alpha \rfloor}{l_{i,j}}$. In particular, it belongs to $(\alpha^* - \frac{1}{l_{\min}}, \alpha^*]$ where $l_{\min}$ is the smallest delay*[9].

Proof. A sufficient condition for α to be feasible for the program with integer offsets is given by $\lceil l_{i,j}\alpha \rceil \leq l_{i,j}\alpha^*, \ \forall (i,j)$. Indeed, in this case, all the circuit constraints (55) are satisfied. This is equivalent to $\alpha \leq \lfloor l_{i,j}\alpha^* \rfloor / l_{i,j}, \ \forall (i,j)$. Hence $\min_{i,j} \lfloor l_{i,j}\alpha^* \rfloor / l_{i,j}$ is feasible for the program with integer offsets. Note finally that $\alpha^* - 1/l_{i,j} < \lfloor l_{i,j}\alpha^* \rfloor / l_{i,j} \leq \alpha^*$. □ □

4 Optimizing the offset of one task

Since optimizing all the tasks together is hard, we first try to optimize the assignment a_i and offset t_i of one task i, while the other assignments and offsets $(a_j^*, t_j^*)_{j\neq i}$ are fixed. Because of its obvious analogy with game theory, where agents try to optimize their actions given the actions of other agents, the problem studied in this section is called the best response problem [1, 2]. In the following, we first present a method, called the best offset procedure, which allows to find the best location for a task on a given processor. Using this procedure, we then easily define the best response procedure, which both finds the best processor and the best location.

4.1 The best offset procedure on a given processor

For each task i, and each processor p, we first define a method $\textsc{BestOffset}_i^p$ which returns the best possible offset for task i on processor p. More formally,

[9] The difference between the two values can be reached asymptotically : take N unitary tasks ($l_{\min} = 1$) with the same period $T = 2N - 1$. Then the optimal fractional solution is $\alpha^* = 2 - 1/N$ (hence $\alpha^* - 1/l_{\min} = 1 - 1/N$), and the optimal integral solution is $\alpha = 1$.

the BESTOFFSET$_i^p$ procedure consists in solving the following program :

$$(BO_i^p) \quad \max \quad \alpha_i \tag{57}$$

$$\text{s.t.} \quad l_{j,i}\alpha_i \leq (t_i - t_j^*) \bmod g_{i,j} \leq g_{i,j} - l_{i,j}\alpha_i \quad \forall j \in \mathcal{G}(i),\ a_j^* = p \tag{58}$$

$$t_i \in \mathbb{Q} \qquad \text{or } t_i \in \mathbb{Z} \tag{59}$$

$$\alpha_i \geq 0 \tag{60}$$

Here, instead of using a binary vector $(a_{i,k})$, we represent an assignment more compactly by a variable $a_i \in [1, P]$. Contraints (58) are the non-overlapping contraints of task i with other tasks j currently on this processor ($a_j^* = p$). We define N_p to be the number of such contraints. In particular, when optimizing the offset of a task, we do not consider constraints linking two other tasks. Note that we deliberately use a formulation involving a modulo : in the following, quotients will be handled completely implicitely. This amounts to working on a two-dimensional projection : the only relevant variables are t_i and α_i, since the other offsets and assignments $(t_j^*, a_j^*)_{j \in \mathcal{G}(i)}$ are parameters. Moreover, following the discussion in sections 2.1.4 and 2.2.2, we can add the following bounds :

$$T_i^p = \operatorname{lcm}((g_{i,j})_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}}) \quad \text{and} \quad \alpha_{i,\max}^p = \min_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}} \alpha_{\max}^{i,j} \tag{61}$$

Since (BO_i^p) is T_i^p-periodic (if t_i is a solution, then $t_i + T_i^p$ is also a solution), we can impose t_i to belong to $[0, T_i^p)$.

4.1.1 Structure of the solution set

If we try to draw the function $\alpha_i = (t_i - t_j^*) \bmod g_{i,j}/l_{j,i}$ representing the optimal value of α_i respecting one constraint $(t_i - t_j^*) \bmod g_{i,j} \geq l_{j,i}\alpha_i$, 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 $\alpha_i = (t_j^* - t_i) \bmod g_{i,j}/l_{i,j}$, 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, which gives a continuous curve[10]. Hence, the set of points (t_i, α_i) satisfying one constraint (58) consists in all the points between this curve and the horizontal axis (see Figure 3).

Given some fixed offsets $(t_j^*)_{j \in \mathcal{G}(i)}$, the set of solutions (t_i, α_i) of (BO_i^p) is the intersection of the sets decribed above for each $j \in \mathcal{G}(i),\ a_j^* = p$ (see Figure 4).

It 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 are $T_i^p/g_{i,j}$ zero points in $[0, T_i^p)$, hence n_{poly} is bounded by $T_i^p \sum_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}} \frac{1}{g_{i,j}}$. This upper bound can be reached when

[10] Note that if $\mathcal{G}$ was not symmetric, we would need to consider some degenerate cases were one of the slopes (increasing or decreasing) would be infinite (since $l_{i,j} = 0$ or $l_{j,i} = 0$).

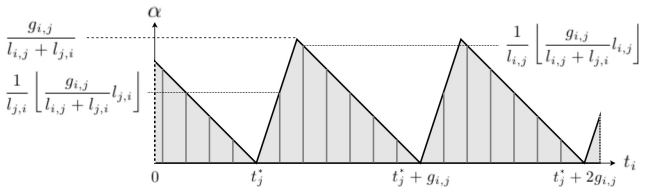

Figure 3: Possible values for (t_i, α_i) when constrained by a single task j

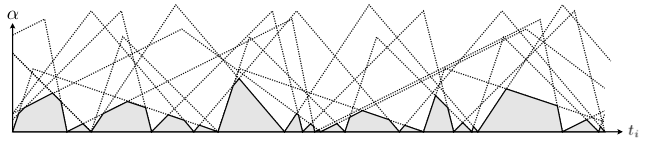

Figure 4: Possible values for (t_i, α_i) constrained by all tasks $j \in \mathcal{G}(i),\ a_j^* = p$

the offsets are fractional, since in this case, we can always choose the offsets t_j^* such that the sets of zero points $(t_j^* + g_{i,j}\mathbb{Z})_{j\in\mathcal{G}(i)}$ are all disjoint. In the case of integer offsets, there are obviously at most T_i^p zero points in the interval $[0, T_i^p)$ and therefore, at most T_i^p polyhedra.

4.1.2 Solving the best offset problem

When solving the best offset problem, we try to find the new location t_i which maximizes the associated α_i-value. In the case of integral offsets, we can trivially solve this program by computing the α_i-values for each of the offsets $[\![0, T_i - 1]\!]$, using the following expression, and selecting the best one.

$$\alpha_i = \min_{\substack{j\in\mathcal{G}(i)\\ a_j^*=p}} \min\left(\frac{(t_i - t_j^*) \bmod g_{i,j}}{l_{j,i}}, \frac{(t_j^* - t_i) \bmod g_{i,j}}{l_{i,j}}\right) \tag{62}$$

We aim at finding a faster procedure as the best response problem has to be solved repeatedly inside a local search method. In [2], the authors propose a method, which also works in the case of fractional offsets, consisting in computing the α_i-values only for a set of precomputed intersection points (since a fractional optimum always lies at the intersection of an increasing and a decreasing line). Here, we present a much more efficient method which relies on the fact that (BO_i^p) is locally a two-dimensional linear program.

In order to solve (BO_i^p), we need to check all the possibilities in the interval $[0, T_i^p)$. More generally we can start at any initial offset t_i^{start}, for example the current position of task i, and run on the right until we reach the offset $t_i^{\text{end}} = t_i^{\text{start}} + T_i^p$. Starting with $t_i^{\text{ref}} = t_i^{\text{start}}$, we can compute the local polyhedron which contains this reference offset (see Figure 5).

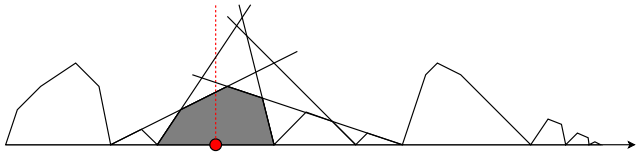

Figure 5: Selection of the polyhedron containing a reference offset t_i^{ref}

Using a dedicated linear programming algorithm, we can then find the optimal solution in this local polyhedron (see section 4.1.3). After this local optimization, we obtain a current best incumbent t_i^{lb} with value α_{lb}, which becomes a lower bound. In the rest of the procedure, we are only interested by the polyhedra which could improve this value. In order to reach the next reference offset, we solve another problem which can be implemented as a propagation procedure (see section 4.1.4). We obtain a new polyhedron and we restart the local optimization at this point, which gives us a better value. We continue until the propagation mechanism reaches t_i^{end} (see Figure 6). In the end, we obtain a best offset $t_i \in [t_i^{\text{start}}, t_i^{\text{end}})$. If needed, we can consider $t_i \bmod T_i^p$ which is an equivalent solution in $[0, T_i^p)$.

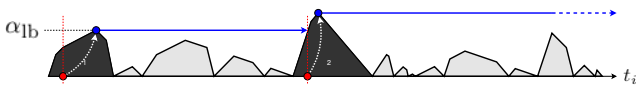

Figure 6: Propagating up to the next strictly improving offset

We will see in section 4.1.4 that the overall propagation on the interval $[t_i^{\text{start}}, t_i^{\text{end}})$ runs in $O(N_p n_{\text{poly}})$. During this process, there are at most n_{poly} additional local optimizations. Using Megiddo algorithm [13], each one of these optimizations can be performed in $O(N_p)$. Hence the overall procedure runs in $O(N_p n_{\text{poly}})$. Note that in practice the number of local optimizations is much lower : as α_{lb} is improved, more and more polyhedra lie completely below this level and are skipped (see the lighter polyhedra in Figure 6).

4.1.3 Solving the local best offset problem

In the following, we will need a simple result about modulo equations :

Proposition 5. *(1) The largest $y \leq a$ such that $(y-b) \bmod c = 0$ is given by $y = a - (a-b) \bmod c$. (2) The smallest $y > a$ such that $(y-b) \bmod c = 0$ is given by $y = a + c - (a-b) \bmod c$.*

Proof. We want to find the largest $y \leq a$ in the set $b + c\mathbb{Z}$. Hence $a - y$ is the smallest positive representative of the set $a - b + c\mathbb{Z}$, which is $(a-b) \bmod c$. Therefore $y = a - (a-b) \bmod c$. Since $y + c$ is the next solution in the set $b + c\mathbb{Z}$, it is the smallest solution strictly greater than a. □ □

We want to compute the local polyhedron which contains a reference offset t_i^{ref} (see Figure 5). Locally, the constraint $(t_i - t_j^*) \bmod g_{i,j} \geq l_{j,i}\alpha_i$ is linear, of the form $t_i - o_j \geq l_{j,i}\alpha_i$. Here, o_j is the largest $\tau \leq t_i^{\text{ref}}$ such that $(\tau - t_j^*) \bmod g_{i,j} = 0$. In the same way, we can compute the decreasing constraint $o'_j - t_i \geq l_{i,j}\alpha_i$ where o'_j is the smallest $\tau > t_i^{\text{ref}}$ such that $(t_j^* - \tau) \bmod g_{i,j} = 0$. By Proposition 5, we have :

$$o_j = t_i^{\text{ref}} - (t_i^{\text{ref}} - t_j^*) \bmod g_{i,j} \quad \text{and} \quad o'_j = o_j + g_{i,j} \tag{63}$$

Note that when $(t_i^{\text{ref}} - t_j^*) \bmod g_{i,j} = 0$, we have implicitely choosen the polyhedron on the right because it corresponds to the moving direction (see Figure 7).

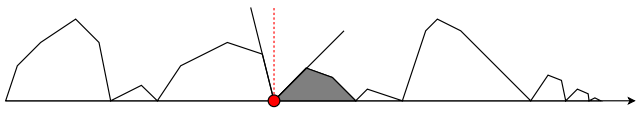

Figure 7: The polyhedron on the right is choosen in the degenerate case $\alpha_i = 0$

Once the local polyhedron has been defined, the problem is now to solve the following problem :

$$(Loc{-}BO_i^p) \quad \max \quad \alpha_i \tag{64}$$

$$\text{s.t.} \quad t_i - l_{j,i}\alpha_i \geq o_j, \qquad \begin{smallmatrix}\forall j \in \mathcal{G}(i) \\ a_j^* = p\end{smallmatrix} \tag{65}$$

$$t_i + l_{i,j}\alpha_i \leq o'_j, \qquad \begin{smallmatrix}\forall j \in \mathcal{G}(i) \\ a_j^* = p\end{smallmatrix} \tag{66}$$

$$t_i \in \mathbb{Q} \quad \text{or } t_i \in \mathbb{Z} \tag{67}$$

In the fractional case, we can use any available method of linear programming. However, since the problem is a particular two dimensional program, we can use special implementations of these methods. From a theoretical perspective, the best approach is given by Megiddo algorithm [13], which allows to find a solution in $O(N_p)$. However, in practice, it is outperformed by the following primal simplex approach, which runs in $O(N_p^2)$ in the worst case (this theoretical extra cost is cheap in practice especially as most of the polyhedra are skipped by the propagation mechanism which still runs in $O(N_p n_{\text{poly}})$). Note first that a local polyhedron is delimited by increasing and decreasing lines. Hence, the fractional optimum is at the intersection of two such lines. Therefore a natural way to find the optimum is to try all the possible intersections between an increasing line, of the form $t_i - l_{j,i}\alpha_i = o_j$, and a decreasing line, of the form $t_i + l_{i,k}\alpha_i = o'_k$, and to select the one with the smallest α_i-value. The coordinates of these intersection points are given by[11] :

$$t_i = \frac{l_{j,i}o'_k + l_{i,k}o_j}{l_{j,i} + l_{i,k}} \qquad \text{and} \qquad \alpha_i = \frac{o'_k - o_j}{l_{j,i} + l_{i,k}} \tag{68}$$

[11] Note that in the computation only the α_i-coordinate is needed since the t_i-coordinate of the selected intersection point can be computed afterwards by $t_i = o_j + l_{j,i}\alpha_i$

Since there are N_p lines of each kinds, this algorithm runs in $O(N_p^2)$. In practice, a better approach is to start with a couple of increasing and decreasing lines, and alternatively to try to improve the decreasing line (see Figure 8), then the increasing one (see Figure 9), and so on, until no improvement is made.

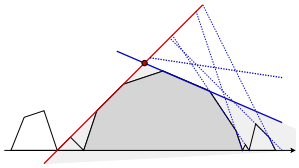

Figure 8: The lowest intersection point of a fixed increasing line with decreasing lines

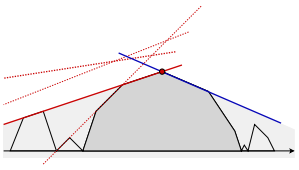

Figure 9: The lowest intersection point of a fixed decreasing line with increasing lines

The overall solving process is illustrated on Figure 10. We call this a primal approach since this is essentially the application of the primal simplex algorithm. However the primal simplex is not applied on $(Loc{-}BO_i^p)$ but on its dual. A dual approach is illustrated on Figure 11 and presented in a technical report [14].

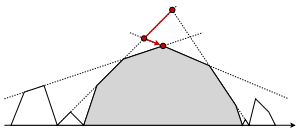

Figure 10: Finding the fractional optimum with a primal simplex approach

Figure 11: Finding the fractional optimum with a dual simplex approach

Remark 3. *Suppose we just improved the decreasing line (Figure 8) and we get a new intersection point. We know that the whole local polyhedron lies inside the cone oriented below and defined by the increasing and the decreasing line. Then all the decreasing lines with a smaller slope than the new decreasing line,* i.e. *with a larger delay $l_{i,j}$, lie completely outside this cone, and therefore cannot be active at the optimum. We can therefore drop these lines in the subsequent phases. In other words, the delay $l_{i,j}$ of the decreasing line strictly decreases with time (except for the first improvement, since the initial decreasing line can be any line). The same is true for the increasing line.*

If we define a phase to be the computation of a new increasing or decreasing line, then the previous remark implies that the number of "decreasing" phases (*resp.* "increasing" phases) is bounded by the number of distinct delays $l_{i,j}$ (*resp.* $l_{j,i}$). Since a phase runs in $O(N_p)$, the method runs in $O(N_p^2)$ in the worst case of distinct delays. However, the number of phases can be much lower. In particular, in the case of processing times, all the delays $l_{i,j}$ are identical (to p_i), which means that all the decreasing lines have the same slope $-1/p_i$ (see

Figure 12). Hence, the primal simplex method runs in $O(N_p)$ since it stops after three phases : even if initially there are N_p decreasing lines with equations $o'_j - t_i = \alpha p_i$, the one with the smallest o'_j will be selected after the first phase. At the end of the second phase, we are at the optimum : the third phase can be dropped since no better decreasing line will be found.

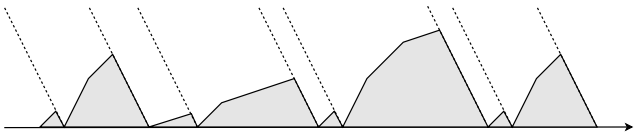

Figure 12: All the decreasing lines are parallel in the case of processing times

In the integral case, we obtain a method with the same complexity, since once a fractional solution has been found, we can deduce an integral solution with an additional computation in $O(N_p)$. Indeed, if t_i is integer, then (t_i, α_i) is the desired solution. Otherwise we can compute the α_i-values α_i^- and α_i^+ associated with $\lfloor t_i \rfloor$ and $\lceil t_i \rceil$ and take the largest one. Note that since $\lfloor t_i \rfloor$ (*resp.* $\lceil t_i \rceil$) is on the increasing phase (*resp.* decreasing phase), only the corresponding constraints are needed to compute α_- (*resp.* α_+) :

$$\alpha_i^- = \min_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}} \frac{\lfloor t_i \rfloor - o_j}{l_{j,i}} \quad \text{and} \quad \alpha_i^+ = \min_{\substack{k \in \mathcal{G}(i) \\ a_k^* = p}} \frac{o'_k - \lceil t_i \rceil}{l_{i,k}} \tag{69}$$

4.1.4 Finding the next improving offset

After a local optimization, we obtain a solution t_i^{lb} with value α_{lb}, which becomes a lower bound. We want to find a new solution t_i greater than t_i^{lb} which strictly improves its value. Hence, this solution should satisfy :

$$\alpha_{\text{lb}} l_{j,i} < (t_i - t_j^*) \bmod g_{i,j} < g_{i,j} - \alpha_{\text{lb}} l_{i,j}, \quad \substack{\forall j \in \mathcal{G}(i) \\ a_j^* = p} \tag{70}$$

Moreover, among all the possible solutions, we would like to find the smallest one. In the case of integral offsets, the middle expression gives an integer value. Hence we can round the bounds and obtain the equivalent interval constraint (72). Therefore, we want to solve :

$$\min \ t_i \tag{71}$$

$$\text{s.t.} \ \lfloor l_{j,i} \alpha_{\text{lb}} \rfloor + 1 \leq (t_i - t_j^*) \bmod g_{i,j} \leq g_{i,j} - \lfloor l_{i,j} \alpha_{\text{lb}} \rfloor - 1 \quad \forall j \in \mathcal{G}(i), \ a_j^* = p \tag{72}$$

$$t_i \in [\![t_i^{\text{lb}}, t_i^{\text{end}} - 1]\!] \tag{73}$$

In the case of fractional offsets, there is no such smallest solution to equation (70), because the left inequality is strict. We can however change the latter into

a non-strict inequality, which amounts to searching for the first offset greater than t_i^{lb} with value at least α_{lb} :

$$\min \ t_i \tag{74}$$

$$\text{s.t. } l_{j,i}\alpha_{\text{lb}} \leq (t_i - t_j^*) \bmod g_{i,j} < g_{i,j} - l_{i,j}\alpha_{\text{lb}} \quad \forall j \in \mathcal{G}(i),\ a_j^* = p \tag{75}$$

$$t_i \in [t_i^{\text{lb}}, t_i^{\text{end}}) \tag{76}$$

If t_i is the resulting offset, the point $(t_i, \alpha_{\text{lb}})$ will be on an increasing line (at least one left inequality will be active). But since the right inequality remains strict, we forbid this point to be on a decreasing line. Thus, we know that t_i will not be the local optimum of the new polyhedron. Therefore, this new local polyhedron will contain a solution with value strictly greater than α_{lb}.

In the following, we will focus on the resolution of the fractional case (the integral case is completely similar and explained in detail in [15]). In order to solve this program, we will start with $t_i = t_i^{\text{lb}}$ and propagate on the right until we find a feasible solution. On a given iteration, we choose a constraint j and check if it is satisfied. We could check the two inequalities of the interval constraint (75), however we can also remark that this is equivalent to :

$$(t_i - t_j^* - l_{j,i}\alpha_{\text{lb}}) \bmod g_{i,j} < g_{i,j} - (l_{i,j} + l_{j,i})\alpha_{\text{lb}} \tag{77}$$

Therefore we can compute $r = g_{i,j} - (t_i - t_j^* - \alpha_{\text{lb}} l_{j,i}) \bmod g_{i,j}$. If $r \leq (l_{i,j} + l_{j,i})\alpha_{\text{lb}}$, the constraint is violated. In this case, we compute the smallest offset τ strictly greater than the current one, and which satisfies the current constraint, *i.e.* we compute $\tau > t_i$ such that $(\tau - t_j^* - l_{j,i}\alpha_{\text{lb}}) \bmod g_{i,j} = 0$. By Proposition 5, we have $\tau = t_i + r$. For this offset τ, the current constraint is now satisfied. We set $t_i = \tau$ and continue the process with the next constraint. We cycle along the constraints, until no update is made during N_p successive rounds or the offset t_i becomes greater than t_i^{end}. In the former case, all the constraints are satisfied by the current offset, otherwise there is no solution (α_{lb} is optimal) and we return the special value $\varnothing$. This procedure is summarized in Algorithm 1. For convenience, we suppose here that the constraints $j \in \mathcal{G}(i),\ a_j^* = p$ are numbered from 0 to $N_p - 1$.

Proposition 6. *The overall propagation from t_i^{start} to t_i^{end} runs in $O(N_p n_{poly})$.*

Proof. We will show that the propagation progresses by one polyhedron every N_p iterations, hence the end of the period is reached after at most $N_p n_{\text{poly}}$ iterations, each running in $O(1)$. Let us define a cycle to be N_p consecutive iterations and let us show that after two cycles, the current offset t_i does not lie in the same local polyhedron. Note that after a cycle, the algorithm either stops or an update has been made. Consider a constraint j updated during the second cycle. If j was not updated during the first cycle, then this contraint was satisfied during the first cycle (hence the current offset was in a given polyhedron

Algorithm 1 A propagation procedure to find an improving fractional solution

```
1: procedure FindImprovingFractionalSolution(α_lb, t_i^lb, t_i^end)
2:     t_i ← t_i^lb                                   ▷ The current offset
3:     N_sat ← 0                                      ▷ The number of satisfied constraints
4:     j ← 0                                          ▷ The current evaluated constraint
5:     while N_sat < N_p do
6:         r ← g_{i,j} − (t_i − t*_j − l_{j,i} α_lb) mod g_{i,j}
7:         if r > (l_{i,j} + l_{j,i}) α_lb then       ▷ If constraint j is satisfied
8:             N_sat ← N_sat + 1                      ▷ One more constraint is satisfied
9:         else
10:            t_i ← t_i + r                          ▷ We go to the first feasible offset
                                                      ▷ If we reach the end without solution, we stop
11:            if t_i ≥ t_i^end then return ∅
12:            N_sat ← 1                              ▷ We restart counting the satisfied constraints
13:        end if
14:        j ← (j + 1) mod N_p                        ▷ We consider the next constraint
15:    end while
16:    return t_i                                     ▷ We return an improving offset
17: end procedure
```

of Figure 3), but violated during the second. Therefore, at the end of second cycle, the offset has been pushed to the next polyhedron of Figure 3. If j was already updated during the first cycle, then the current offset has been pushed forward to the next feasible solution two times for this constraint. Therefore, the local polyhedron has also changed. □ □

4.2 The multiprocessor best response method

Given the BestOffset_i^p procedure, we define a method BestResponse_i which returns the best assignment and offset for the task i, given the current assignments and offsets (t_j^*, a_j^*) of all the tasks. In order to choose the assignment, an agent simply computes the best offset on each processor and select the best one. It starts with the current processor a_i, which has priority in case of equality. Some of the next processors can sometimes be skipped. Indeed, for a given processor p, equation (61) defines an upper bound $\alpha_{\max}^p$ on the objective. Therefore, if the current best solution found on previous processors is already better than this upper bound, there is no way to improve the current solution with processor p, which can be skipped.

Since the BestOffset_i^p method runs in $O(T_i N_p)$, and since we have $\sum_p N_p = |\mathcal{G}(i)| \leq N - 1$, the BestResponse_i method runs in $O(T_i N)$.

5 An equilibrium-based heuristic

By moving the tasks round after round, we increase the space between them. Iterating this process, we expect to reach an equilibrium. This is essentially the principle of the heuristic presented in this section.

Definition 1. *A solution $(t_i, a_i)_{i \in I}$ is an equilibrium* iff *no task i can improve its assignment or offset using the* BestResponse_i *procedure.*

5.1 Equilibrium in the integral case

In order to find an equilibrium, the heuristic uses a counter N_{stab} to count the number of tasks known to be stable, *i.e.* which cannot be improved. It starts with an initial solution (for example randomly generated) and tries to improve 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 apply BestReponse_i. If no improvement was found, then one more task is stable, otherwise we update the assignment and offset of task i and reinitialize the counter of stable tasks. We continue until N tasks are stable. This is summarized in Algorithm 2 (the ϵ value is used in the fractional case; choose $\epsilon = 0$ in the integral case).

Algorithm 2 The heuristic

procedure ImproveSolution($\epsilon, (t_j)_{j\in I}, (a_j)_{j\in I}$)
  $N_{\text{stab}} \leftarrow 0$ ▷ The number of stabilized tasks
  $i \leftarrow 0$ ▷ The task currently optimized
  while $N_{\text{stab}} < N$ **do** ▷ We run until all the tasks are stable
    $(t'_i, a'_i, \alpha'_i) \leftarrow \text{BestResponse}_i((t_j)_{j\in I}, (a_j)_{j\in I})$
    if $\alpha'_i > \alpha_i + \epsilon$ **then** ▷ We have a strict improvement
      $t_i \leftarrow t'_i$; $a_i \leftarrow a'_i$; $\alpha_i \leftarrow \alpha'_i$
      $N_{\text{stab}} \leftarrow 1$ ▷ We restart counting the stabilized tasks
      $\alpha \leftarrow \alpha_i$
    else ▷ We do not have a strict improvement
      $N_{\text{stab}} \leftarrow N_{\text{stab}} + 1$ ▷ One more task is stable
      $\alpha \leftarrow \min(\alpha_i, \alpha)$
    end if
    $i \leftarrow (i+1) \bmod N$ ▷ We consider the next task
  end while
  return $(\alpha, (t_j)_{j\in I}, (a_j)_{j\in I})$
end procedure

Remark 4. *In this procedure, randomness only occurs in the construction of the initial solution. The tasks are cyclically moved in a fixed order. While this choice seems to favor the first tasks, so far, we did not find improvements by introducing randomness in this order.*

If the algorithm stops, all the tasks are stable, hence we have reached an equilibrium. The problem is to prove the convergence of the heuristic. The main reason for the use of integers is that it ensures the convergence in a finite number of steps. The termination proof relies on the fact that the underlying game is an ordinal potential game. Given a solution $(t_j, a_j)_{j\in I}$, we define the values $(\alpha_i)_{i\in I}$ by expression (62), *i.e.* $\alpha_j = \min_{\substack{k\in\mathcal{G}(j)\\ a_k=a_j}} \alpha_{j,k}$ where :

$$\alpha_{j,k} = \min\left(\frac{(t_k - t_j) \bmod g_{j,k}}{l_{j,k}}, \frac{(t_j - t_k) \bmod g_{k,j}}{l_{k,j}}\right) \tag{78}$$

Let $\sigma : [\![1, N]\!] \to I$ be a permutation sorting these values by increasing order. Then, the potential of the solution is the vector :

$$V((t_j, a_j)_{j\in I}) = (\alpha_{\sigma(1)}, \cdots, \alpha_{\sigma(N)}) \tag{79}$$

Proposition 7. *When an update is made, the potential* (79) *strictly increases for the lexicographic order, preventing the algorithm from cycling.*

Proof. Let $(\alpha_j, t_j, a_j)_{j\in I}$ (*resp.* $(\alpha'_j, t'_j, a'_j)_{j\in I}$) be the data before (*resp.* after) the improvement of a task i. Consider first a task j such that $\alpha_j < \alpha_i$. There is a task $m \in \mathcal{G}(j)$ such that $a_m = a_j$ and $\alpha_j = \alpha_{j,m}$. If $a_i = a_j$, we have $\alpha_{j,i} \geq \alpha_i > \alpha_j$, hence this task m cannot be i. Note that for all $k \neq i$ (in particular for $k = m$), the offsets and assignments are unchanged ($t'_k = t_k$ and $a'_k = a_k$), hence $\alpha'_{j,k} = \alpha_{j,k}$. Moreover, if $a'_i = a_j$, we have $\alpha'_{j,i} \geq \alpha'_i > \alpha_i > \alpha_j = \alpha_{j,m}$. This shows that $\alpha'_j = \min_{\substack{k\in\mathcal{G}(j)\\ a'_k=a_j}} \alpha'_{j,k} = \alpha_{j,m} = \alpha_j$. In other words, the values strictly smaller than α_i are unchanged. Suppose now that $\alpha_j > \alpha_i$. For each task $k \in \mathcal{G}(j) \setminus \{i\}$ such that $a_j = a_k$, we have $\alpha'_{j,k} = \alpha_{j,k} \geq \alpha_j > \alpha_i$. Moreover, if $a'_i = a_j$, we also have $\alpha'_{j,i} \geq \alpha'_i > \alpha_i$. This shows that $\alpha'_j = \min_{\substack{k\in\mathcal{G}(j)\\ a'_k=a_j}} \alpha'_{j,k} > \alpha_i$: the values strictly greater than α_i remain so. Finally, consider the set of tasks j such that $\alpha_j = \alpha_i$. The same reasoning shows that these values remain at least greater or equal than α_i. Since i is one of these tasks, and since $\alpha'_i > \alpha_i$, at least one of them becomes strictly greater. Hence, there are strictly less values equal to α_i among (α_j) than among (α'_j). These three points allow to conclude that the potential (79) has increased. □ □

In the integral case, α_i is bounded and belongs to $\frac{1}{l}\mathbb{Z}$ for some delay $l = l_{i,j}$ or $l = l_{j,i}$. Therefore, the potential can only take a finite number of values, and the algorithm converges after a finite number of steps (we refer to [1] for the original proof of termination and correction).

5.2 Equilibrium in the fractional case

In the fractional case, Proposition 7 remains valid. However, the number of different values for α_i is not finite. To ensure the finiteness of the process, we introduce a small quantity $\epsilon > 0$ such that a task i can be moved only if its value α_i is improved by at least ϵ (see Algorithm 2). Given this modification, we easily see that when an update is made, the following potential (which only admits a finite number of values) strictly increases for the lexicographic order :

$$V_\epsilon((t_j, a_j)_{j\in I}) = (\epsilon\lfloor\alpha_{\sigma(1)}/\epsilon\rfloor, \cdots, \epsilon\lfloor\alpha_{\sigma(N)}/\epsilon\rfloor) \tag{80}$$

Without this trick, *i.e.* with $\epsilon = 0$, the following example shows that, the heuristic is in fact unlikely to converge in a finite number of steps. Let us define three tasks with period $T = 4$ and unitary processing times. Initially, we set $t^0_1 = 0$, $t^0_2 = 2$ and $t^0_3 = 3$ (see Figure 13). Note that it is an integral equilibrium, but not a fractional one : we can improve position of task 1 by moving it 1/2 clockwise. Then task 2 can be moved 1/4 counterclockwise, and task 3 1/8 clockwise. After the first improvement, task 1 was in equilibrium, but since then, task 2 is now closer by 1/4 and task 3 is closer by 1/8. Hence

task 1 compensates by moving 1/16 counterclockwise, task 2 by moving 1/32 clockwise and task 3 by moving 1/64 counterclockwise, and so on. A careful analysis shows that the offsets of the tasks follow the following sequences :

$$t_1^n = 0 + \frac{1}{2}\sum_{k=0}^{n-1}(-1/8)^k = \frac{4}{9} - \frac{4}{9}(-1/8)^n$$

$$t_2^n = 2 - \frac{1}{4}\sum_{k=0}^{n-1}(-1/8)^k = \frac{16}{9} + \frac{2}{9}(-1/8)^n$$

$$t_3^n = 3 + \frac{1}{8}\sum_{k=0}^{n-1}(-1/8)^k = \frac{28}{9} - \frac{1}{9}(-1/8)^n$$

These sequences converge to 4/9, 16/9 and 28/9 respectively, and these values form a fractional equilibrium.

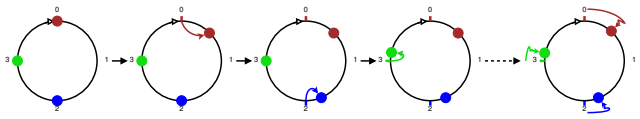

Figure 13: Convergence to an equilibrium in the fractional case

Remark 5. *This example shows that rational numbers are inappropriate to implement the heuristic, which is confirmed by experiments. Indeed, the term* t_1^n *has the form* $\frac{(8^n-(-1)^n)}{3^2\cdot 2^{3n-2}}$*. The numerator is always odd, therefore the term* 2^{3n-2} *in the denominator cannot be simplified. Hence whatever the size of the representation we select for the rational numbers, we quickly reach maximum capacity. The heuristic is therefore implemented with floating point numbers, even when all the data are rationals.*

In the previous example, the offsets of the tasks converge. However, a variation on this example shows that this is not always the case (at least when we consider latency delays and not simply processing times). Let us consider the same three tasks, with unitary delays between them, and let us add a fourth task at initial position $t_4^0 = 1$ (see Figure 14), such that all the delays to and from this task are small enough (less than 1/3). Because of this choice, we can check inductively that task 4 has no impact on the other tasks. Their offsets still follow the same trajectories. When task 4 is improved, it can go either between tasks 1 and 2 (whose mutual distance is $t_2^n - t_1^n = \frac{4}{3} + \frac{2}{3}(-1/8)^n$), or between tasks 2 and 3 ($t_3^n - t_2^n = \frac{4}{3} - \frac{1}{3}(-1/8)^n$), or between tasks 3 and 1 ($t_1^n + 4 - t_3^n = \frac{4}{3} - \frac{1}{3}(-1/8)^n$). Hence, when n is even, it goes between tasks 1 and 2, at offset $t_4^n = (t_1^n + t_2^n)/2$ and when n is odd, it goes between tasks 2 and 3, at offset $t_4^n = (t_2^n + t_3^n)/2$.

At least, we can garantee the following :

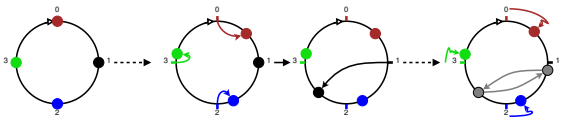

Figure 14: Offsets can diverge in the fractional case

Proposition 8. *If the offsets converge, then the result is an equilibrium.*

Proof. This is a simple consequence of Berge maximum theorem. Let $BR_i : \prod_{j\neq i}[0,T_j] \to \mathbb{P}([0,T_i])$ be the best offset correspondence, *i.e.* the map which associates with each parameters $(t_j^*)_{j\neq i}$ the set of optimal solutions for the best offset problem. Then, the maximum theorem tells us that this correspondence is upper hemicontinuous. In particular, its graph is closed. Hence, if the offsets converge to a solution (t_j), this solution satisfies $t_i \in BR_i((t_j)_{j\neq i})$ for all i, and therefore, this is an equilibrium. □ □

5.3 Final improvement and multistart

The following proposition shows that in the fractional case, an equilibrium is always at least locally optimal :

Proposition 9. *A fractional equilibrium is a local optimum.*

Proof. Suppose that (t_i) is an equilibrium : for each task i, there is some α_i such that (t_i, α_i) is the optimal solution for the best offset problem (BO_i). Let us define $\alpha = \min_i \alpha_i$ and let us choose a task i_1 such that $\alpha_{i_1} = \alpha$. Since (t_{i_1}, α) is optimal for (BO_{i_1}), there is a decreasing line active at t_{i_1}. Hence, there is a task i_2 such that :

$$t_{i_2} - t_{i_1} = \alpha l_{i_1,i_2} - g_{i_1,i_2} q_{i_1,i_2} \tag{81}$$

Since (t_{i_2}, α_{i_2}) is optimal for (BO_{i_2}), there is a task i_3 such that $t_{i_3} - t_{i_2} = \alpha_{i_2} l_{i_2,i_3} - g_{i_2,i_3} q_{i_2,i_3}$. Moreover, since (t_{i_2}, α_{i_2}) is feasible for (BO_{i_2}), we have in particular $t_{i_2} - t_{i_1} \geq \alpha_{i_2} l_{i_1,i_2} - g_{i_1,i_2} q_{i_1,i_2}$. Combined with (81), we deduce that $\alpha \geq \alpha_{i_2}$, and since α is minimal, we have in fact $\alpha_{i_2} = \alpha$. Therefore :

$$t_{i_3} - t_{i_2} = \alpha l_{i_2,i_3} - g_{i_2,i_3} q_{i_2,i_3} \tag{82}$$

Iterating this process, we find a sequence of tasks (i_n) such that $t_{i_{n+1}} - t_{i_n} = \alpha l_{i_n,i_{n+1}} - g_{i_n,i_{n+1}} q_{i_n,i_{n+1}}$. After a finite number of steps, we reach a node i_k already visited. Therefore, we obtain a circuit C. When summing the previous equations for all the couples of the circuit, the left part becomes zero and we obtain $\sum_{(i,j)\in C} g_{i,j} q_{i,j} = \alpha \sum_{(i,j)\in C} l_{i,j}$. By Proposition 3, we are at a local optimum. □ □

In the fractional case, the result of the heuristic is only an ϵ-equilibrium. We can however expect to be near an equilibrium, hence near a local optimum.

We can therefore complete the heuristic by computing the local optimum in the polyhedron containing the current solution (first compute the quotients with (3) and then solve the problem in section 3.2.1). Note that contrary to the heuristic, this procedure can be implemented with rational numbers when all the data are rationals.

In the integral case, as Propositions 4 and 9 suggest, an equilibrium is not far from a local integral optimum (see section 3.2.2). But no concept implies the other. Therefore, when an equilibrium is reached, we can also complete the heuristic by computing the local integral optimum in the polyhedron currently containing the solution. The result is not necessarily an equilibrium anymore, but we can restart the heuristic with this new point. We continue until the equilibrium returned by the heuristic is also a local optimum. This process also converges in a finite number of steps since the computation of the local optimum strictly increases the potential used to prove the convergence of the heuristic.

An equilibrium is only an approximate notion of optimum. Hence, in order to find a real optimum, the idea is now to run the previous heuristic several times with different randomly generated initial solutions, and to keep the best result, following a standard multistart scheme.

6 Results

We have tested the method on non-harmonic instances generated using the procedure described in [8] : 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 inverse exponential distribution and averaging at about 20% of the period of the task. The experiments were performed on an AMD Athlon 64 X2 5200+ 2.7GHz with 2GB of memory. Note that, compared to [15], the results are different mainly for two reasons : we changed the hardware, and we added the small improvement described in section 5.3 (a solution of the heuristic is now both an equilibrium and a local optimum, hence, individually, each execution of the heuristic is a bit slower, but less iterations are needed in order to obtain the best solution). To obtain more reproducible results, we have also forced the use of a single thread, both for the MILP and for the heuristic.

Table 1 presents the results of our heuristic (in the integral case), the MILP formulation and the original heuristic [2], on 15 instances with $N = 20$ tasks and $P = 4$ processors. Columns 6-10 contain the results of our new version of the heuristic. The value $\mathbf{start}_{10s}$ represents the number of times the heuristic was launched with different initial solutions during 10s. From this value, we compute $\mathbf{time}_{single} = 10/\mathbf{start}_{10s}$ which represents the average time needed for one execution of the heuristic. The value $\mathbf{start}_{sol}$ represents the number of starts needed to obtain the best result, and $\mathbf{time}_{sol}$ is the corresponding time.

Column $\mathbf{time}_{sol}$ of the MILP formulation (columns 2-5) represents the time needed by the Gurobi solver to obtain the best solution during a period of 600s

(10min)[12]. We have tested two versions of the MILP : one with continuous variables $x_{i,j}$ (see (36) in section 2.2.3), the other with binary variables $x_{i,j} \in \{0,1\}$. In addition to being much faster, the heuristic sometimes obtains better results (see instances 1, 3 and 11).

This table also includes the results of the original heuristic presented in [1, 2] (columns 11-12). The field $\mathbf{time}_{\text{single}}$ represents the time needed for a single run of their version of the heuristic. Averaging at 1.95s, these values are compatible with the results presented in [2]. The field $\mathbf{gain}_{\text{speed}}$, which is the ratio of the two values $\mathbf{time}_{\text{single}}$, shows that our version of the heuristic is incomparably faster (about 1550 times on these instances).

These good results have encouraged us to perform additional tests on large instances (50 processors, 1000 tasks). Table 2 presents the results (in the integral case) for 10 instances, where **starts** is the number of runs started during 1800s (30min), $\mathbf{time}_{\text{total}}$ is the actual time needed to perform these runs (since we do not stop a run once it is started), $\mathbf{time}_{\text{single}} = \mathbf{time}_{\text{total}}/\mathbf{starts}$ is the average time for one run, $\mathbf{starts}_{\text{sol}}$ is the round during which the best solution was found, and $\mathbf{time}_{\text{sol}}$ is the corresponding time. This shows that our heuristic can give feasible solutions ($\alpha \geq 1$) in about 1min, while these instances cannot even be loaded by the MILP solver (since they involve millions of variables and constraints). In order to evaluate the contribution of the propagation mechanism to the solving process, we also present results where the propagation of section 4.1.4 has been replaced by a simpler mechanism : once we are at a local optimum, we follow the decreasing line active at this point, until we reach the horizontal axis; this gives us a next reference offset and therefore a next polyhedron. While the impact of the propagation is quite small in the case of small instances, we see on these large instances that the propagation mechanism accelerates the process by a factor of 23 (average ratio of the two $\mathbf{time}_{\text{single}}$ values).

7 Conclusion

In this paper, we have proposed an enhanced version of a heuristic, first presented in [1, 2], and allowing to solve a NP-hard strictly periodic scheduling problem. More specifically, we have presented an efficient way to solve the best response problem. This procedure alternates between local optimizations and an efficient propagation mechanism which allows to skip most of the polyhedra. The results show that the new heuristic greatly improves the original one and compares favorably with MILP solutions. In particular, it can handle instances out of reach of the MILP formulation.

[12] We fixed a timeout of 600s because the solver almost never stopped even after 1h of CPU time. In the case of binary variables $x_{i,j}$, the only exceptions were instances 5 and 8, for which optimality has been proved in $71s$ and $470s$ respectively. In the case of continuous variables $x_{i,j}$, optimality has been proved for instances 5 and 9 in $47s$ and $543s$ respectively.

Appendix on monoperiodic and disjunctive scheduling

In this section, we highlight the difficulty of the problem by showing the link with disjunctive scheduling. Compared to a classical system of precedence constraints, our scheduling problem is made hard by the introduction of additional integer quotients variables. Minimally, these quotients induce an order between tasks. Indeed, given offsets (t_i), we can forget the absolute positions and only retain their relative order : $i \succ j \iff t_i > t_j$. On $\mathcal{G}$, this relation can be defined by the quotients alone : $i \succ j \iff q_{i,j} \geq 1$. Property $q_{i,j} + q_{j,i} = 1$ simply says that $\succ$ defines an orientation on $\mathcal{G}$: we have either $i \succ j$ or $j \succ i$ but not both. Contraints (44) and (45) imply in particular that for each circuit C, we have $\sum_{(i,j)\in C} g_{i,j} q_{i,j} > 0$. This implies in turn that the orientation given by $\succ$ is acyclic[13]. For example, if $\mathcal{G}$ is the complete graph, then $\succ$ simply defines a total order. Generally, quotients code for more than an order. However there is a special case for which they do no more. In the monoperiodic case, we saw that the variables $(q_{i,j})$ can be supposed to be binary. Moreover circuit contraints (44) have the following form [16] :

$$\sum_{(i,j)\in C} q_{i,j} \geq \frac{1}{T} \sum_{(i,j)\in C} l_{i,j}, \qquad \forall \text{ el}^{\text{t}}\text{circuit } C \tag{83}$$

If the period T is large enough, then the right-hand side is always less than 1. Hence this is equivalent to $\sum_{(i,j)\in C} q_{i,j} \geq 1$, which simply forbids circuits. In this case, $(q_{i,j})$ is a binary vector representing an acyclic orientation.

The previous case is related to the disjunctive scheduling problem. In this problem, starting dates are subject to the following disjunctive constraints $t_j - t_i \geq l_{i,j} \vee t_i - t_j \geq l_{j,i}$. However, once an order has been defined among the tasks, only one of the two inequalities remains. A natural way to express this disjunctive constraint as a MILP is to introduce a binary variable $q_{i,j}$ with value 1 if i starts after j. Then the first contraint is written $t_j - t_i + Tq_{i,j} \geq l_{i,j}$, while the second one is written $t_i - t_j + T(1 - q_{i,j}) \geq l_{j,i}$. In terms of modulo, the disjunction is represented by the two constraints $(t_j - t_i) \bmod T \geq l_{i,j}$ and $(t_i - t_j) \bmod T \geq l_{j,i}$. Here, T is simply a big-M constant choosen to be large enough, but it plays the role of a period. Graphically, this choice amounts to transforming a disjunctive problem into a monoperiodic problem (see Figure 15).

Acknowledgements

This work was funded by the French Midi-Pyrenee region (allocation de recherche post-doctorant n°11050523) and the LAAS-CNRS OSEC project (Scheduling in Critical Embedded Systems).

[13] On any circuit C, there is at least one couple (i, j) such that $i \succ j$. Since $\mathcal{G}$ is symmetric, there is also one such that $j \succ i$.

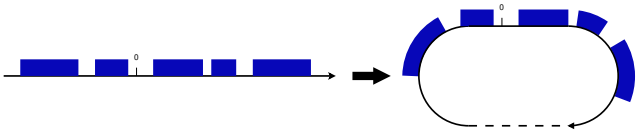

Figure 15: From disjunctive to monoperiodic scheduling

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] A. Bar-Noy, R. Bhatia, J.S. Naor and B. Schieber. Minimizing Service and Operation Costs of Periodic Scheduling. Mathematics of Operations Research, Vol. 27, N°3 pp. 518-544, 2002.

[4] S. Baruah, L. Rousier, I. Tulchinsky, D. Varvel. The complexity of periodic maintenance. In: Proceedings of the International Computer Symposium, 1990.

[5] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In International Conference on Real-Time Computing Systems and Applications, pp. 62-69, Hong Kong, 1999.

[6] W. Dauscha, H.D. Modrow, A. Neumann. On Cyclic Sequence Type for Constructing Cyclic Schedules. Zeitschrift für Operations Research, Vol. 29, N°1, pp. 1-30, 1985.

[7] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, A. Wiese. Scheduling periodic tasks in a hard real-time environment. In: Automata, languages and programming, pp. 299-311, 2010.

[8] 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.

[9] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In: Proceedings of the 12th IEEE Symposium on Real-Time Systems, pp. 129-139, 1991.

[10] J. Korst, E. Aarts, J.K. Lenstra, J. Wessels. Periodic multiprocessor scheduling. In: E.H.L. Aarts, M. Rem, J. van Leeuwen (eds.) PARLE 1991. LNCS, Vol. 505, pp. 166-178. Springer, Heidelberg, 1991.

[11] J. Korst. Periodic multiprocessors scheduling. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

[12] C.L. Liu, J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, N°1, pp. 46-61, 1973.

[13] 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.

[14] C. Pira and C. Artigues. An efficient best response heuristic for a non-preemptive strictly periodic scheduling problem. Technical Report 12668 LAAS-CNRS, Toulouse, 2012.

[15] C. Pira and C. Artigues. Line search method for solving a non-preemptive strictly periodic scheduling problem. Proceedings of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Ghent, Belgium, 27-30 August 2013, pp. 356-371, 2013.

[16] P. Serafini, W. Ukovich. A Mathematical Model for Periodic Scheduling Problems. SIAM J. Disc. MATH, Vol. 2, N°4, pp. 550-581, 1989.

	MILP (600s)				New heuristic (10s)				Original heuristic [2]	
	binary $x_{i,j}$		continuous $x_{i,j}$							
id	α_{MILP}	**time**$_{\text{sol}}$	α_{MILP}	**time**$_{\text{sol}}$	$\alpha_{\text{heuristic}}$	**starts**$_{^{10s}}$	**time**$_{\text{single}}$	**starts**$_{\text{sol}}$	**time**$_{\text{sol}}$	**time**$_{\text{single}}$	**gain**$_{\text{speed}}$
0	2.5	55	2.5	7	2.5	9418	$1.06190e^{-3}$	15	$41.379e^{-3}$	1.43	1347
1	2	103	2.01091	75	2.01091	9456	$1.05754e^{-3}$	15	$14.956e^{-3}$	3.27	3092
2	1.6	73	1.6	6	1.6	5415	$1.84696e^{-3}$	2	$2.484e^{-3}$	1.52	823
3	1.64324	87	1.6	15	1.64324	7840	$1.27555e^{-3}$	10	$11.783e^{-3}$	4.34	3403
4	2	17	2	66	2	7395	$1.35245e^{-3}$	1	$0.855e^{-3}$	3.48	2573
5	3	36*	3	34*	3	6192	$1.61515e^{-3}$	1	$1.363e^{-3}$	1.63	1009
6	2.5	57	2.5	7	2.5	9531	$1.04929e^{-3}$	15	$15.431e^{-3}$	1.44	1372
7	2	58	2	95	2	9601	$1.04166e^{-3}$	1	$1.220e^{-3}$	0.23	221
8	2.12222	41*	2.12222	70	2.12222	5604	$1.78462e^{-3}$	1	$1.767e^{-3}$	1.03	577
9	2	17	2	60*	2	8731	$1.14537e^{-3}$	3	$3.397e^{-3}$	2.42	2113
10	1.12	79	1.12	248	1.12	7510	$1.33175e^{-3}$	69	$92.174e^{-3}$	0.72	541
11	2.81098	67	1.92	11	2.81098	8579	$1.16572e^{-3}$	1	$1.208e^{-3}$	3.78	3241
12	1.5	15	1.5	39	1.5	6914	$1.44636e^{-3}$	3	$4.122e^{-3}$	0.27	187
13	1.56833	338	1.56833	530	1.56833	9125	$1.09590e^{-3}$	1	$1.182e^{-3}$	1.77	1615
14	2	35	2	92	2	6145	$1.62747e^{-3}$	2	$3.059e^{-3}$	1.85	1137

Table 1: Results of the MILP, the heuristic of [2], and the new version of the heuristic on instances with $P = 4$ processors and $N = 20$ tasks, with $timeout = 10s$

	With propagation						Without propagation					
id	$\alpha_{\text{heuristic}}$	**starts**	$\textbf{time}_{\text{total}}$	$\textbf{time}_{\text{single}}$	$\textbf{starts}_{\text{sol}}$	$\textbf{time}_{\text{sol}}$	$\alpha_{\text{heuristic}}$	**starts**	$\textbf{time}_{\text{total}}$	$\textbf{time}_{\text{single}}$	$\textbf{starts}_{\text{sol}}$	$\textbf{time}_{\text{sol}}$
0	1	211	1803	8.55	1	10.0204	1	13	1826	140.5	1	160.753
1	1.2	88	1803	20.49	14	305.218	1.16	6	2223	370.5	3	1144.65
2	1.15	46	1818	39.52	26	987.841	1.05424	4	2312	578	2	1155.82
3	1.2717	59	1852	31.39	55	1675.5	1.176	4	2320	580	1	197.652
4	1.12	75	1830	24.4	61	1468.3	1	2	2532	1266	1	258.884
5	1.66531	59	1834	31.08	3	133.925	1.64	2	1952	976	1	1230.31
6	1	211	1802	8.54	1	9.98876	1	14	1871	133.6	1	159.847
7	1.32	73	1817	24.89	17	425.38	1.22	6	1983	330.5	1	398.074
8	1.4007	59	1812	30.71	47	1537.07	1.25714	3	2508	836	1	889.750
9	1.28	59	1852	31.39	36	1219.36	1.24	3	2202	734	1	993.399

Table 2: Results of the new version of the heuristic on instances with $P = 50$ processors and $N = 1000$ tasks, with $timeout = 1800s$ (30min)