Identical coupled task scheduling problem: the finite case

Michaël Gabay, Gerd Finke, Nadia Brauner

▶ **To cite this version:**

Michaël Gabay, Gerd Finke, Nadia Brauner. Identical coupled task scheduling problem: the finite case. [Research Report] –. 2011. hal-00627902v2

HAL Id: hal-00627902
https://hal.science/hal-00627902v2

Submitted on 7 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Identical coupled task scheduling problem: the finite case

Michaël Gabay*, Gerd Finke*, Nadia Brauner*

Rapport de recherches − 29 september 2011 − 30 pages

Abstract: Coupled tasks belong to the class of multi-operation tasks, where two consecutive operations are separated by a certain time interval of fixed duration. The objective is to schedule the tasks on one machine in order to minimize the makespan. Such problems occur especially in the case of the management of radar systems.

The complexity status of this problem has been settled for all particular cases, except the case of the identical coupled task problem, where one has multiple copies of a single coupled task. This scheduling problem seems to be simple at first glance. However, its complexity status (for both, the cyclic and the finite case) has remained open for many years. Recently, Vassilissa Lehoux-Lebacque, Gerd Finke and Nadia Brauner proved that the cyclic case is polynomial. The aim of this work was to study the finite case by using some of the results and structures known from the cyclic case.

We have been able to see that the finite case strongly differs from the cyclic case. Using integer programming and the commercial solver `CPLEX`, we have obtained a series of optimal solutions for small instances. The structures underlying these solutions are much more complicated than those of the cyclic case. Eventually, we have obtained results that let us think that this problem is not in $\mathcal{NP}$.

Keywords: Scheduling, Coupled tasks, Complexity, High multiplicity.

Résumé : Les tâches couplées sont des ttâches multi-opérations pour lesquelles deux opérations consécutives sont séparées par une durée fixe. L'objectif est de trouver un ordonnancement sur une machine de durée totale minimale. Ces problèmes surviennent lors de la mise en place de systèmes radars.

La complexité de ce problème a été déterminée dans tous les cas, à l'exception du cas des tâches couplées identiques. Récemment, Vassilissa Lehoux-Lebacque, Gerd Finke et Nadia Brauner sont parvenus à prouver que ce problème est polynomial dans le cas cyclique. L'objectif de ce travail est d'étudier le cas fini en se servant des structures utilisées dans le cas cyclique.

Durant cette étude, nous avons pu observer que le cas fini est très différent du cas cyclique. à l'aide de la programmation lineaire en nombres entiers et du solveur commercial `CPLEX`, nous avons obtenu un catalogue de solutions optimales pour des petites instances. Les structures de ces solutions sont beaucoup plus compliquées que celles du cas cyclique. Nous avons enfin obtenu de nombreux résultats (solutions, configurations et propriétés) qui nous laissent penser que ce problème est probablement non $\mathcal{NP}$.

Mots-clés : Ordonnancement, Tâches couplées, Complexité, Haute multiplicité.

*Laboratoire G-SCOP 46, avenue Félix Viallet - 38031 Grenoble Cedex 1 - France

Introduction

The coupled task problem arises in the case of scheduling tasks in radar systems: to detect an object, the radar transmitter emits a pulse in some direction and then, after a fixed amount of time, the receiver tries to receive the reflection of the first signal. If some reflection is received, the radar system can compute the position, the direction and the speed of the object using the transit time of the pulse and the doppler effect.

Formally, there is one machine: the radar system. There is a first operation of duration a: emitting a pulse, then a fixed duration L and eventually an operation of duration b: receiving the pulse. Moreover, the idle time of duration L can be used to schedule other tasks. In the general case there can be many different tasks with different durations. For instance when tracking an object, this object can be more or less close to the radar system and hence the transit time can vary. A coupled task scheduling problem is then given by a set of tasks of the form (a_i, b_i, L_i) and the aim is to minimize the makespan (the total duration of the scheduling) on a single machine (the radar). The real problem is an online scheduling problem but here we consider only the offline version. The complexity of all cases is known except when $\forall i\ a_i = a$, $L_i = L$ and $b_i = b$. This problem is called *identical coupled task scheduling problem*. Recently, Vassilissa Lehoux-Lebacque, Gerd Finke and Nadia Brauner have proven in [1] that this problem is polynomial in the cyclic case. However, the complexity status of the problem remains unknown for the finite case.

One can think that this problem is simple but it will be explained in the next section why this is not true. Moreover, we would like to point out that the study of this problem is not meaningless. As described in [2], a radar system executes different kinds of tasks (surveillance, tracking, etc.), either all done on the same system (for recent combat radars) or on different systems (for classic radars). For the surveillance task, in order to detect whether an object enters an airspace, all of the tasks are roughly of the same duration, hence can be supposed to be identical, and the problem is cyclic. In order to maximize the chances to detect the object, the cycle time rate has to be minimized. This corresponds to the cyclic identical coupled task scheduling problem.

The aim of our work was to study the finite case in order to analyze its complexity. To reach this goal, we tried to use the patterns known from the cyclic case to build optimal schedules in some particular cases and collect information and properties about the general finite case.

In this report, we will first explain the problem, its difficulties and what is known about it. Then we will present some positive and negative results obtained and point out what remains to be done.

1 Problem overview

In this section, we will formally describe the considered problem and give its state-of-the-art.

1.1 Problem description

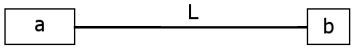
The problem is the following:
Input: 4 positive integers a, b, L, n.
Output: a schedule of n identical coupled tasks on a single machine minimizing the makespan.

A coupled task consists of a first operation a, followed by an operation b in exactly L units of time after the accomplishment of a as represented figure 1. n is the number of identical tasks to schedule. Operations cannot overlap while operations belonging to other tasks can be scheduled during the separation length L.

We call a *task* a coupled task and we call an *operation* one of the two parts of a task, i.e. a or b.

We will assume that $L \geq a+b$ (otherwise the problem is trivial) and that $a > b$. Such assumption can be made because Orman and Potts proved in [3] that if $a = b$ a greedy schedule (placing all a's

Figure 1: A single coupled task



as soon as possible) is optimal and if $b > a$ then we can compute the schedule by inverting a and b and reverse it.
Moreover, we can assume that a, b, L are integers (as we do not manipulate irrational numbers, a, b, L could be rational but multiplying them by the least common multiple of their denominators will make them integers without changing the problem).

We have stated the problem as an optimization problem, we will now state it as a decision problem:
IDENTICAL COUPLED TASKS SCHEDULING
Instance: 4 positive integers a, b, L, n and a positive number C_{max}.
Question: Does there exists a schedule of makespan smaller or equal to C_{max} ?

Using Graham's notation − as done in [3] − this problem can be stated as $1|Coup-task,\ a_i = a,\ L_i = L, b_i = b|C_{max}$.

1.2 Problem characteristics

At first glance, this problem could seem to be easy. However, it is in fact quite different to the other special cases of the coupled task scheduling problems. When some of the durations are fixed - but not all - the input is at least $n+2$ integers (at most 2 durations are fixed and n operations of non-fixed duration remain). Hence, the length of an instance is $O(n\log(h))$, where h is the maximum of all given integers. In the case of the identical coupled task scheduling problem, the input is a, b, L, n - 4 integers. Hence the length of the instance is $O(\log(\max(a,b,L,n))$. More precisely, as $L \geq a+b$, the length of the instance is: $O(\log(\max(L,n))) = O(\log(L)+\log(n))$. Such a problem is called a *high multiplicity scheduling problem.* The reader can refer to [4] and [5] for more details about these problems.

An algorithm that would be polynomial in n is exponential in $\log(n)$. Moreover, let us recall the definition of the class $\mathcal{NP}$ (as stated in [6]):

Definition 1. A language L is in $\mathcal{NP}$ if and only if there exists polynomials p and q, and a deterministic Turing machine M, such that:

- $\forall\ x,y$ the machine M runs in time $p(|x|)$ on input (x,y)
- $\forall\ x \in L$, there exists a string y of length $q(|x|)$ such that $M(x,y) = 1$
- $\forall\ x \notin L$ and all strings y of length $q(|x|)$, $M(x,y) = 0$

In other words, there exists a certificate that is polynomial in the length of the instance and can be verified in polynomial time. In our case, the problem comes from this certificate: a complete schedule is not a certificate that is polynomial in the length of the instance. This is the difficulty of the problem: in other special cases, a complete schedule (specifying for instance all of the starting times of the n coupled tasks) can be given as a certificate. In our case, it cannot. The solution needs to be described in terms of structures - as done in the cyclic case exposed section 1.4. Hence, even proving that the problem is or is not in $\mathcal{NP}$ may be difficult.

One can think that placing as many coupled tasks in succession as possible with no unnecessary idle time is enough to have an optimal schedule, but this is not the case as can be seen figure 2 and 3: with the same instance $a = 8$, $b = 4$, $L = 28$, $n = 6$, a greedy schedule gives a makespan of 112 time units while another schedule gives a makespan of 104 time units.

Figure 2: A greedy schedule for $a = 8$, $b = 4$, $L = 28$, $n = 6$, makespan: 112

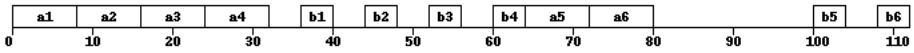
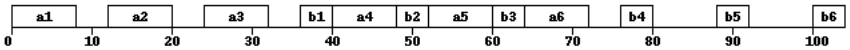


Figure 3: An optimal schedule for $a = 8$, $b = 4$, $L = 28$, $n = 6$, makespan: 104



1.3 State-of-the-art

The coupled task scheduling problem was originally introduced by Shapiro in [7]. In this article, he proved that the general problem is $\mathcal{NP}$-complete and proposed 3 heuristics. For more details about the theory of $\mathcal{NP}$ completeness, the reader can refer to [8]. Later on, Orman and Potts demonstrated in [3] the complexity of almost all cases (except $a_i = a$, $L_i = L$ and $b_i = b$) where some of the durations are fixed. The table 1, from [3], sums up all of these results (they are symmetric: $a_j = a$, $L_j = L$, b_j is the same as a_j, $L_j = L$, $b_j = b$). The borderline $\mathcal{NP}$-hard and polynomial cases are represented in this table. All of the other complexities can be derived from them.

Table 1: Complexity of coupled task scheduling problems

Strongly $\mathcal{NP}$-Hard	a_j; L_j; b_j
	$a_j = L_j = b_j$
	$a_j = a$; L_j; $b_j = b$
	$a_j = a$; $L_j = L$; b_j
Open	$a_j = a$; $L_j = L$; $b_j = b$
Polynomial	$a_j = L_j = p$; b_j
	$a_j = b_j = p$; $L_j = L$

In the case where the duration separating operations is not fixed, Gupta demonstrated in [9] that multioperation scheduling problems with time-lags are $\mathcal{NP}$-hard in the strong sense. More specifically, he demonstrated that the coupled task scheduling problem with time-lags is $\mathcal{NP}$-hard in the strong sense.

In his PhD thesis [2], Cyril Duron, studied the case of the online insertion of a random task (a simple or a coupled task) in a schedule with the criterion of minimizing the sum of delays. He exposes in his thesis applications of coupled task scheduling problems, especially in the case of multifunction radar systems, including battle radars.

Brauner et al. demonstrated in [10] that the coupled task scheduling problem is equivalent to a one-machine no-wait robotic cell problem.

In [11], an exact algorithm using graphs for the identical coupled task scheduling problem is presented. Its complexity is $O(nr^{2L})$ where $r \leq \sqrt[a-1]{a}$ holds. Baptiste improved this result by proving in [12] that for fixed a, b and L, the problem can be solved in $O(log(n))$. However, the constant is exponential in L and therefore this algorithm cannot be used in practice, as the computational experience in [10] shows.

Baptiste also proved in [12] that there always exists an optimal integer solution for the coupled task scheduling problem (in the general case) by remarking that the constraint matrix of the linear problem corresponding to minimizing the makespan while knowing the order of the tasks is totally unimodular.

Recently, Blazewicz et al. demonstrated in [13] that the identical coupled task scheduling problem with unit processing times ($a = b = 1$) and strict precedence constraints is $\mathcal{NP}$-complete in the strong sense. Similar results are shown by Simonin et al. in [14].

During her PhD thesis [15], Vassilissa Lebacque started to work on the identical coupled task scheduling problem and devoted to this topic a chapter of her thesis which opened the way to find the solutions of this problem in the cyclic case.
Recently, Lebacque et al. demonstrated in [1] that the identical coupled task scheduling problem is polynomial in the cyclic case. The next section describes their work.

The coupled task scheduling problem is a model for scheduling operations on an radar system using a mechanically steered antenna. Recent radar systems may use an Electronically Steered Antenna instead. In such a case, the operations are faster as the beam can be moved instantaneously in another direction and its waveform can be changed instantaneously too. Hence, there is no longer a fixed duration between two tasks but rather minimal and tight times and costs. Concerning those systems, Emilie Winter and Philippe Baptiste proposed in [16] a framework for the analysis, demonstrated that the problem is $\mathcal{NP}$-Hard in the strong sense and proposed a few heuristics to solve this problem.

1.4 The cyclic case

In this section, we will describe the solution of the identical coupled task problem in the cyclic case. All the definitions and results here come directly from [1].

Definition 2 (Cyclic identical coupled task scheduling problem)**.** Let three positive integers a, b and L be given. One has multiple copies of a single coupled task $(a; b; L)$, consisting of two *operations* (or *parts*), the first having length a and the second length b, and both operations are exactly separated by L time units. A *cycle* C is a finite sequence of a's, b's and idle times that can be appended repeatedly to each other, to form a feasible placement of the coupled tasks (*i.e.* without overlapping of operations). C contains necessarily the same number of a's and b's.
The cycle time $\lambda(C)$ is the ratio of the length of C and the number of coupled tasks in C (*i.e.* the number of elements in C divided by 2). We call two cycles C_1 and C_2 *equivalent* if they have the same cycle times, $\lambda(C_1) = \lambda(C_2)$. The cycle C_1 *dominates* C_2 if their cycle times verify $\lambda(C_1) \leq \lambda(C_2)$. The dominance is *strict* if $\lambda(C_1) < \lambda(C_2)$. A cycle is *optimal* for the identical coupled task problem if and only if it dominates all other cycles.

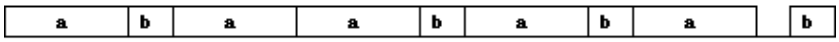
As can be understood in that definition, in the cyclic case, the aim is to find a cycle C that minimizes the cycle time $\lambda(C)$.

In [1], a profile is defined as follows:

Definition 3 (Profile, Window)**.** Let C be a cycle and $W_S = [a_S, L, b_S] = a_S S b_S$ a window of C (S is the set of operations between a_S and b_S). Obtain $\bar{S}$ by converting in S all subsequences bb to $\bar{a}b$. Should S terminate with a b, then also this b becomes $\bar{a}$ since it is followed by b_S. Count the number β of terms (ba) occuring in $\bar{S}$, then count the number α of remaining a's and $\bar{a}$'s. We call (α, β) the profile of the window W_S.

The figure 4 illustrates an example window, the configuration of this window is:
$a_0baababab_0 = a_0(ba)a(ba)(ba)b_0$, hence there are three ba's and one a in this window, which corresponds to a profile $(1,3)$.

Figure 4: An example $(1,3)$ window



Consider a second example. Given a window starting with a_1: $a_1babbaabaabb_1$, we have: $a_1babbaabaabb_1 = a_1(ba)\bar{a}(ba)a(ba)a\bar{a}b_1$. Hence the profile of this window is $(4,3)$.

Remark 1. An $\bar{a}$ is always counted in α as given an $\bar{a}_0$, it was either $bb_0b = \bar{a}\bar{a}_0b$ or $ab_0b = a\bar{a}_0b$. In both cases, it is not a ba.

A window contains $\alpha+2\beta$ operations and we have $\alpha a+\beta(b+a) \leq L$. Then take the slack variable γ as: $\gamma = L-\alpha a-\beta(b+a)$, where γ is the total idle time of a window. These parameters constitute an *extended profile* of the cycle: (α,β,γ) - the *profile* is simply constituted of two parameters: (α,β).

A cycle $C(\alpha,\beta)$ can be constructed based on a profile (α,β). This can be achieved by normalizing the order of the elements in a window to: $W = aa^{\alpha}(ba)^{\beta}b = a^{\alpha+1}(ba)^{\beta}b$. Then extending this sequence to $Z = a^{\alpha+1}(ba)^{\beta}b^{\alpha+1}(ab)^{\beta}$. Z contains $1+\alpha+2\beta$ coupled tasks in the cyclic sense. Whatever the idle times in the first window are, constructing a schedule by following Z using an earliest placement strategy (a new a is placed without idle time following a b or an a), gives the cycle $C(\alpha,\beta)$ after $(\beta+1)$ repetitions of Z.

They also define the notions related to *blocks*:

Definition 4 (Blocks)**.** Let (α^N,β^N) be the optimal cyclic profile, we define $M = \alpha^N+2\beta^N$. A block B is a sequence of $2(M+1)$ consecutive operations. A block is *tight* if all of its windows are *tightly-packed* (that is they contain a maximum number of operations: M). A block B is *short* if its length satisfies: $|B| \leq 2L+a+b$ and *long* otherwise. The *gain* of block B is defined as $\Delta(B) = (a+2L+b)-|B|$.

M satisfies the following integer program:

$$\begin{aligned} & \max M \\ \text{subject to: } & M = \alpha+2\beta \\ & \alpha a+\beta(a+b) \leq L \\ & M,\ \alpha,\ \beta \in \mathbb{N} \end{aligned}$$

It is then proven, that the optimal cycle is $C(\alpha^N,\beta^N)$ with α^N, β^N and the cycle time $\lambda(C)$ given by the polynomial algorithm 1 of complexity $O(log(L)^2)$.

Algorithm 1 Best cyclic profile

Input: $a,\ b,\ L\ (a>b,\ L \geq a+b)$
Output: Best profile $(\alpha^N,\beta^N,\gamma^N)$ and cycle time λ

Let $\beta^N = \lfloor \frac{L}{a+b} \rfloor$ and $R = L-\beta^N(a+b)$
if $R<a$ **then**
  $\alpha^N = 0$ and $\gamma^N = R$
else
  $\alpha^N = 1$ and $\gamma^N = R-a$
end if

if $\gamma^N \geq (\beta^N+1)(a-b)$ **then**
  $\alpha^N = \lfloor \frac{L}{a} \rfloor$, $\beta^N = 0$ and $\gamma^N = L-\alpha^N a$ {here γ^N satisfies $b>\gamma^N>a-b$}
end if

$\lambda = \frac{(\beta^N+1)(2L+a+b)-\gamma^N}{(\beta^N+1)(1+\alpha^N+2\beta^N)}$

return $(\alpha^N,\beta^N,\gamma^N),\lambda$

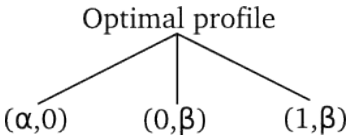


Figure 5: Optimal cyclic profiles

From this algorithm, we can immediately see that there are only three kinds of optimal solutions:

In [1], the following property is also proven:

Property 1. *A cycle* $C(\alpha,\beta,\gamma)$ *is dominated by:*

1. $C(\alpha+2,\beta-1,\gamma-(a-b))$ *if* $\beta \geq 1$ *and* $\gamma \geq (\beta+1)(a-b)$
2. $C(\alpha-2,\beta+1,\gamma+(a-b))$ *if* $\alpha \geq 2$ *and* $\gamma \leq (\beta+1)(a-b)$

This dominance is strict when $\gamma > (\beta+1)(a-b)$ *in 1 or* $\gamma < (\beta+1)(a-b)$ *in 2.*

This is a dominance rule, hence this proves that given the optimal profile $(\alpha^N,\beta^N,\gamma^N)$, when $\gamma^N = (\beta^N+1)(a-b)$, for a positive integer i, all cycles having a profile $(\alpha^N+2i,\beta^N-i,\gamma^N-i(a-b))$ with $\beta^N - i \geq 0$ or a profile $(\alpha^N-2i,\beta^N+i,\gamma^N+i(a-b))$ with $\alpha^N \geq 2i$ are optimal.

Eventually, this points out a link between the profiles having the maximum number of operations as all such profiles are $(\alpha^N+2i,\beta^N-i,\gamma^N-i(a-b))$ with $\beta \geq i$ and $\gamma^N - i(a-b) \geq 0$ or $(\alpha^N - 2i,\beta^N+i,\gamma^N+i(a-b))$ with $\alpha^N \geq 2i$.

2 Objectives and plan

The objective of this study was to use the results shown in the cyclic case to find results and properties for the finite case. Especially, we first try to generalize the structure known from the cyclic case in order to apply them in the finite case.

As will be seen in the following, the solution structures are far more complicated than in the cyclic case. In order to generate a catalog of solutions, we will first develop the methods we use to solve instances of the problem.

3 Solving instances for the finite case

The methods proposed by Ahr et al in [11] cannot be used for practical computations (the running time is highly exponential in L). We decided therefore to use two other approaches, namely Integer Programming and Dynamic Programming, to solve instances of the problem.

3.1 Integer Programming

In order to solve instances of the problem, we wrote a mixed integer linear program and implemented it on `OPL` (IBM - ILOG) modeling language to solve instances using the `CPLEX` solver for integer programming (IP) (more information at: `http://www-01.ibm.com/software/websphere/products/optimization/`). To solve problems, the `CPLEX` algorithm relies on the (dual) simplex algorithm, barrier interior point method and branch and bound.
In this section, we will expose the basic model and some of the improvements made.

The basic model is the following mixed integer program (MIP):

$$\min\ t_n \tag{1}$$

$$\text{subject to: } t_i \geq t_{i-1} + a \qquad \forall i \in \{2,\dots,n\} \tag{2}$$

$$t_j - t_i \leq L + M y_{i,j} \qquad \forall i,j \in \{1,\dots,n\} \text{ s.t. } i < j \tag{3}$$

$$t_j - t_i \geq (a + L + b) y_{i,j} \qquad \forall i,j \in \{1,\dots,n\} \text{ s.t. } i < j \tag{4}$$

$$t_i \in \mathbb{R}^+ \qquad \forall i \in \{1,\dots,n\} \tag{5}$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \in \{1,\dots,n\} \tag{6}$$

t_i are the decision variables corresponding to the starting times of the tasks. Remark that as all tasks are the same, we can fix their starting, i.e. if $i < j$ then the task i starts before the task j. $y_{i,j}$ are boolean decision variables needed to ensure that an a never overlaps with a b.
The objective (1) is to minimize the makespan, hence to minimize the starting time of the last coupled task t_n. The first constraint (2) corresponds to the duration of operations a's and prevents two a's from overlapping. Moreover, as two a's cannot overlap, two b's cannot overlap. The only case remaining is the overlapping of an a and a b.
The second and the third constraints (3, 4) prevent an a and a b from overlapping: for $i < j$, either the task j starts before the task i ends, hence $t_j \leq t_i + a + L - a = t_i + L$ and the constraint (3) is activated ($y_{i,j} = 0$) and the constraint (4) is trivially verified. Or the task j starts after the task i ends, hence $t_j \geq t_i + a + L + b$ and the constraint (4) is activated ($y_{i,j} = 1$) and the constraint (3) is trivially verified provided that M is large enough.

This model is a good start for small instances but it has to be improved to work with larger instances. Moreover, M needs to be fixed to a reasonable value.

In order to improve the model, we did the following modifications:

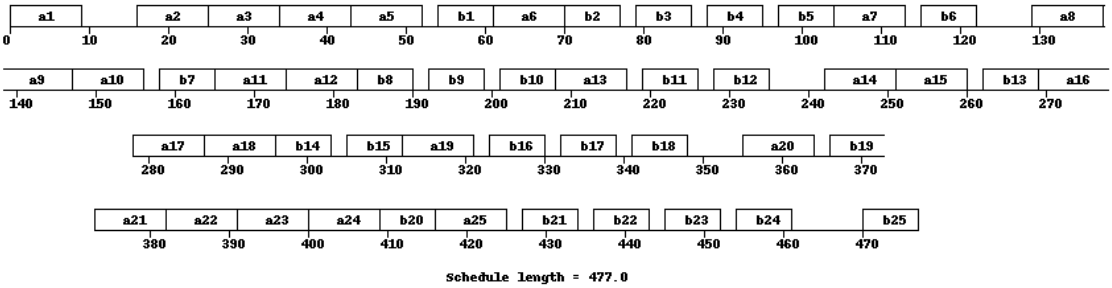
- Adding constraint $t_1 = 0$ (it is obvious that the optimal solution will respect this but if not specified, other cases will be considered).
- Fixing $y_{i,j} = 0\ \forall i \geq j$. It prevents the algorithm from branching although it is not necessary (but in fact `CPLEX` seems to detect it too).
- Adding constraint $t_i \leq t_{i-1} + a + L + b$: this correspond to a "non-separation": if the following task is placed $a + L + b$ units later, this is as if a new schedule begins. Hence there is no point starting later than the starting time of the new schedule.
- Remark that if an a is after a $'b'$, then all following a's will be after this b. This can be expressed through the constraint $y_{i,j} \geq y_{i,j-1}$.
- Having an initial solution of makespan $C_{\max}^{\text{init}}$ helps to eliminate more nodes at the beginning of the branching by searching only better solutions. This can be accomplished by adding the constraint $t_n \leq C_{\max}^{\text{init}} - b - L - a - 1$. When using such a constraint, M can be set to $C_{\max}^{\text{init}}$ for instance. In order to set M and use this constraint, we can compute the greedy schedule value.
- For two tasks i, j, with $i < j$, we have $t_j \geq t_i + (j-i)a$. Hence, whenever $(j-i)a \geq a + L + b$ then b_i cannot overlap with a_j and therefore, the non a-b overlapping constraints are not necessary and should be removed. Adding the non-overlapping constraints only when necessary strongly improves the computation time.
- Transforming the Mixed Integer Program into an Integer Program by changing the domain of t_i into $\mathbb{N}$. Though it results in having more nodes during the computation, the computation speed is slightly improved.

- Removing y and expressing a-b overlap constraints as logical constraints: for $i<j$, $t_j \leq t_i + L$ `OR` $t_j \geq t_i + a + L + b$ (this is not an IP any more though it may be linearized). This did not help the solving process.

In order to illustrate those optimizations, we have run some tests on an instance where $a=9$, $b=7$, $L=45$ and $n=25$. For this instance, the optimal cyclic profile is $(1,2,4)$. The greedy schedule has a makespan of 485 and the schedule using the cycle $C(1,2)$ until enough tasks are accomplished is optimal. The optimal makespan is 477 and the optimal schedule is represented fig. 6.

The tests have been run on a dual core 2 processor at 2×2.26GHz, using `CPLEX 12` and run on only one thread (hence using only one core) in order to be fair and deterministic. The results are available table 2. In this table, from one line to the next, all of the previous optimizations have been kept. The efficiency column, corresponds to the base time (using the base MIP), divided by the time needed using the model on the line.
Please notice that the illustrated tests have been run on only one instance and hence some more exhaustive tests may be needed to really be able to compare the optimizations. We have executed more tests while making the models and the trends illustrated table 2 were confirmed.

Figure 6: Optimal solution of the benchmark instance



Unfortunately, we cannot add more cuts because we do not know enough about the structures of optimal solutions in the finite case. Moreover, as will be shown afterwards with some surprising solutions, there seem to be no clear solution structure that could be used to generate cuts.

After more experiments, it appears that the complexity of the solving process increases exponentially with the number of blocks $\lceil \frac{n}{M+1} \rceil$. We have managed to compute the optimal solution up to $a=10$, $b=9$, $L=82$ and $n=37$ ($\lceil \frac{n}{M+1} \rceil = 5$) in 4h10, without the best model and with `CPLEX 10`. However, we have not been able to compute the optimal solution for $a=10$, $b=9$, $L=82$ and $n=46$ ($\lceil \frac{n}{M+1} \rceil = 6$) after more than two weeks of computation (with `CPLEX 10`).

3.2 Dynamic Programming

In order to try to have a faster solving method, we implemented a dynamic programming method in `C++`.
The method is constructive: when an a is placed, there are only b's that are already placed afterwards (the b's corresponding to this a and some previous a's).
Let $C^*_{\max}$ be the optimal makespan, $f(i,S)$ the function computing the optimal makespan with i operations a to place and S a set of times $\{t_1,\ldots,t_{k_S}\}$ (related to the current time) on which begin b's (corresponding to some a's previously placed). Moreover, S is such that $t_1 \geq a$. We denote $|S| = t_{k_S} + b$.

Table 2: Optimization benchmark

Additional Constraints	Computation time (in seconds)	Node count	Efficiency
None (MIP)	229,9	535200	100%
t_i positive integers (IP)	222,8	656000	103%
$t_1 = 0$	122,8	482700	187%
$y_{i,j} = 0 \ \forall i \geq j$	122,8	482700	187%
$t_i \leq t_{i-1} + a + L + b$	179,9	725600	128%
$y_{i,j} \geq y_{i,j-1}$	152,2	507300	151%
$t_n \leq t_n^{\text{greedy schedule}}$	91,3	338500	252%
$t_n \leq t_n^{\text{cyclic solution}}$	91,1	335100	252%
Removing constraints 3 and 4 when i is far enough from j	68,8	321000	334%
Replacing y by logical constraints	243,0	538500	95%
Removing logical constraints when i and j are far enough	99,5	472500	231%

The recursive formula is the following:

$$C^*_{\max} = f(n, \emptyset) \tag{7}$$
$$f(i, \emptyset) = f(i-1, \{L-a\}) + a \tag{8}$$
$$f(i, S) = \min(f(i, S^{\leftarrow}) + [S| - |S^{\leftarrow}|, f(i-1, S^{\rightarrow}) + a + L - |S^{\rightarrow}|) \tag{9}$$
$$f(0, S) = |S| \tag{10}$$

Let l be the smallest integer such that $l \geq 1$ and $\forall i \in \{1, \ldots, k_S\}$, $t_i + b \leq l$ or $t_i \geq l + a$. We have $l \leq t_{k_S} + b$. Then, let j be the smallest integer such that $t_j \geq l$, we have: $S^{\leftarrow} = \{t_j - l, \ldots, t_{k_S} - l\}$. In other word, $S^{\leftarrow}$ is the nearest set of b's "after" S such that there is an idle time of at least a. Let $S' = \{t_1 - a, \ldots, t_{k_S} - a, L\}$, l the smallest integer such that $l \geq 0$ and $\forall i \in \{1, \ldots, k_S\}$, $t_i + b \leq l$ or $t_i \geq l + a$. Let j be the smallest integer such that $t_j \geq l$, we have: $S^{\rightarrow} = \{t_j - l, \ldots, t_{k_S} - l, L - l\}$. In other words, $S^{\rightarrow}$ is the nearest set of b's "after" S' such that there is an idle time of at least a.

This is the dynamic programming approach we have implemented. The previous definition of $S^{\leftarrow}$ and $S^{\rightarrow}$ may seem to be complicated, but in fact, computing them is done with a greedy algorithm of cost $O(\#S)$ (where $\#S$ is the number of elements in S).

This recursive formula tries all of the possible placements (except those with an a separated from the next a by more than $L + b$, but we have explained in the previous subsection why it is not necessary) and returns the best, which is the optimal solution.

However, the number of combinations of S is exponential and hence it is impossible to keep everything in the computer memory. In order to tackle with this issue, we have implemented a partial memoï¿½zation approach: sets S are memoized only for some S such that $\#S < j$ is fixed. Eventually, in order to speedup the computation, we compute a lower bound at each iteration and cut the branch if the current time plus the bound is greater than the current best makespan found.

This program is efficient when n is small, yet as n increases, it is necessary to increase the number of memoized sets (a good ratio is to memoize up to $k(M+1)$ operations with $k = \max(\lfloor \frac{n}{M+1} \rfloor - 1, 0)$). However, the memory space needed is highly exponential and the solving time grows exponentially. Using this program, we have not been able to reach better performances than using `CPLEX` with the most efficient model we had developed.

4 Definitions

In this section, we will introduce definitions through the study of examples. We will show patterns for some instances where we will see that it will not be possible to consider only the optimal cyclic profiles.

As in the cyclic case, we denote $(\alpha^N, \beta^N, \gamma^N)$ the optimal cyclic profile and we define M as the maximum number of operations contained between an a and its corresponding b: $M = \alpha^N + 2\beta^N$.

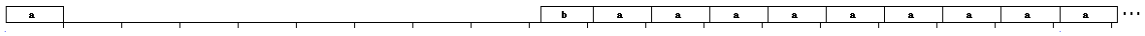
Remark 2. Remark that if two b's are scheduled consecutively, then there is an idle time of at least $a - b$ between those two b's. Such idle times are called *intrinsic*. Moreover, we can consider that the first b takes the time of an a. We shall denote again such b's as $\bar{a}$.

We will have saturated and tight profiles that we define as follows:

Definition 5 (Saturated, Tight)**.** A profile (α, β) is *saturated* if for β fixed, α is maximal (α cannot be increased).
Recall that M is the integer that maximizes $\alpha + 2\beta$. A profile (α, β) is *tight* if $\alpha + 2\beta = M$.

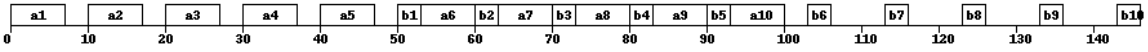
When we consider optimal solutions, we first assume that coupled tasks are shifted as much as possible to the left (*left-shifted*). Moreover, we assume that the schedule is *compact* which means that none of its tasks can be placed earlier without delaying any task.
Considering compact and left-shifted schedules only is a very important choice as some properties (such as the β increasing property) might not hold if the schedule is not compact.
We can make this choice because there always exists an optimal compact left-shifted schedule: given a non-compact schedule, we can make it compact by beginning some tasks earlier without lengthening the schedule. Moreover, if a schedule is not left-shifted, shifting some tasks to the left cannot lengthen the schedule. Hence there always exists a compact left-shifted optimal schedule.
The sequence F (fig. 7) is an example of non compact sequence.

Figure 7: A non compact sequence, $a = 10$, $b = 9$, $L = 82$



It is clear that there will never be any b in the first window. Yet, these b's do exist in a way as their place is "reserved", in order to be able to put ab's afterwards. Those b's, whose place are reserved but cannot be placed are called *ghost* b's. Moreover, it is the same with the a's in the last window of the schedule. In order to have the real profile of a window, it is needed to count ghost b's and ghost a's. This profile, including *ghost* a's and b's is called a *shadow profile*.

Figure 8: The optimal solution for $a = 7$, $b = 3$, $L = 43$, $n = 10$, optimal profile: $(0, 4, 3)$

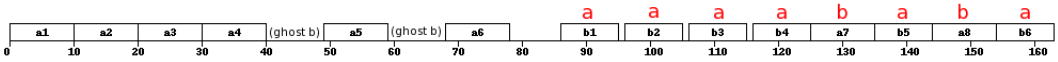


As seen in the cyclic case, there is a duality between a's and b's, this duality can be used to determine the profile of sets of tasks which begin with a b:

Definition 6 (Duality)**.** The dual of an operation a is b and the dual of b is a. The dual of a set of operations is the dual of all of its operations.

In order to determine the profile of a set of $M + 2$ tasks beginning with a b, we should determine the profile of its dual. The figure 9 illustrates a dual window $(3, 2)$.

Figure 9: Windows and dual window $(3,2)$, for $a=10$, $b=9$, $L=76$, the dual is written above the operations, in red.



4.1 Pure strategies

A first obvious strategy, called *pure strategy* is to generate schedules using a unique (shadow) profile, i.e. following exactly the cycle pattern of $C(\alpha,\beta)$ for n tasks. We shall concentrate on saturated profiles only. An example of such a strategy, using a $(0,4,3)$ profile, is given fig. 8.

We know that pure strategies place $2(\alpha+2\beta+1)$ elements (a's or b's) on a duration $a+2L+b-\delta$, with $0\leq\delta<a$ for saturated profiles. For pure strategies, we define a new kind of blocks:

Definition 7 (Profile block)**.** Given a saturated profile (α,β,γ), a profile block $B(\alpha,\beta;\delta)$ is a sequence of $2(1+\alpha+2\beta)$ consecutive elements. Starting with the first element, counting all non intrinsic idle times and the idle time following the last element gives the length of the profile block $|B(\alpha,\beta;\delta)|$. Then $\delta=(a+2L+b)-|B(\alpha,\beta;\delta)|$. We call $\alpha+2\beta+1$ the *profile block number*, i.e. half of the number of operations of the profile block.

A profile block corresponds, in fact, to a sequence Z of a cycle $C(\alpha,\beta)$. Fig. 8, the first block is $a_1\dots a_5(idle=3)b_1\dots b_5$. This is a profile block $B(0,4;3)$. Fig. 9, a profile block $B(3,2)$ is illustrated.

We can remark that given a profile (α,β,γ), for a profile block $B(\alpha,\beta,\delta)$, we have $0\leq\delta\leq\gamma$. Indeed: the gain of this profile block is at most γ. Moreover, as it represents a profile, it contains exactly $2(\alpha+2\beta+1)$ operations and the first window is followed by $\alpha+2\beta$ operations, with idle times at most γ, hence the length of the profile block is at most $(a+L+b)+(\alpha a+\beta(a+b)+\gamma)=(a+L+b)+L=a+2L+b$, hence $\delta\geq 0$.

Pure strategies use a single (saturated) profile. It is quite natural to consider particular profile blocks containing the maximum number of tasks $2(M+1)$. These profile blocks are associated with tight profiles. This allows us to extend the definitions of *saturated* and *tight* to profile blocks: if the profile is *saturated*, then the profile block is *saturated*. The same goes for *tight* profile blocks.

Now that we have described the interesting structures for pure strategies, we can make a list of feasible solutions for an example. Table 3, we consider an example where $a=5$, $b=3$, $L=100$ and n is small. All of the saturated profiles are represented in the first column of the table. When n is smaller than M, the pure strategy using the saturated profile whose block number is the closest to n and placing the maximum number of a's for this block number is optimal. Remark that in this case, it is very important that the profile block is $a(ba)^\beta a^\alpha(ab)^\beta b^\alpha$ rather than $aa^\alpha(ba)^\beta b^\alpha(ab)^\beta$ because in the first case, the last a's will be placed earlier. Table 3 illustrates this evolution of the pure strategy solutions for an instance with small n and a big L (OPT means that it is a real optimum, over all possible placements).

Let us analyze the example table 3. There are 13 saturated profiles and the profile block number goes from 21 to 25. The first three top profiles are tight $(1+\alpha+2\beta=M+1=25)$. The optimal cyclic profile is $(0,12,4)$.
The optimal solutions OPT are found with `CPLEX`. They are at least until $n=30$ profile block solutions. For $n\leq 21$, the greedy earliest placement strategy is optimal (profile $(20,0)$). It should be noted that in the finite case, the order of the terms (ba) and a are important. To get the optimal profile block solution of table 3 for $n\leq 30$, all ba's are to be placed before all a's.
Remember that in the cyclic case, all cycles are equivalent, independent of the placements of the β terms ba and the α terms a.
As will be seen later, the order of terms (ba) and a in profile blocks is essential for the solution

Table 3: Optimal pure strategy for $a=5$, $b=3$, $L=100$, n small

α	β	γ	Block number $1+\alpha+2\beta$	$n \leq 21$	22	23	24	25	26	27	28	29	30
0	12	4	25 (tight)						OPT				
2	11	2								OPT	OPT		
4	10	0						OPT				OPT	OPT
5	9	3	24										
7	8	1					OPT						
8	7	4	23										
10	6	2											
12	5	0				OPT							
13	4	3	22										
15	3	1			OPT								
16	2	4	21										
18	1	2											
20	0	0		OPT									

quality.
For the example and $n \leq 30$, all optimal solutions OPT are described as pure strategy solutions. As we shall see later, pure strategies are not sufficient for larger n and other instances.

4.2 General blocks

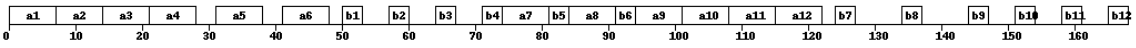
Consider any solution. We define a *block* as a sequence of exactly $2(M+1)$ elements (including ghosts) and the idle time after the last element.
In order to study solution structures, we will aggregate operations into blocks. The remaining set of operations is an incomplete block and will be called the *extension*.
We also extend the definition of *tight* and *saturated* to a block: a block is *tight* if all of the windows (including dual windows) contained in this block are tight and a block is *saturated* if all of the windows (including dual windows) contained in this block are saturated.

In a profile block, δ defines the gain of this profile block, compared to $a+2L+b$. Corresponding to a block, we define the gain as follows:

Definition 8 (Gain)**.** The *gain* of a block B is $\Delta(B) = 2L+a+b-|B|$. If $\Delta(B) \geq 0$, the block is *short*, otherwise, it is *long*.

In order to minimize the makespan, the gain of the schedule has to be maximized. Hence, in a pure strategy using profile (α,β,γ), the first block has to be a profile block $B(\alpha,\beta;\gamma)$.
For a profile block $B(\alpha,\beta,\delta)$ such as $\alpha+2\beta=M$, the profile block is tight and hence, $\delta=\Delta(B)$. The fig. 8 illustrates this case. However, this is not always the case. For instance, fig. 10, the first operations constitute a profile block $B(3,2,3)$. However, $M=8>3+2\times 2$. Hence the first block is $a_1 \ldots a_{10}$ and $\Delta = 7+2\times 43+3-108=-12$. Therefore, the first block is long.

Figure 10: The optimal solution for $a=7$, $b=3$, $L=43$, $n=12$, optimal profile: $(0,4,3)$



4.3 Pure strategy approximation

One may wonder how good are pure strategy solutions for large n. We have the following approximability theorem:

Theorem 1 (Asymptotic guarantee)**.** *The pure strategy using the optimal cyclic profile is asymptotically optimal.*

Proof. Given an instance I, with $n = k(\beta^N+1)(\alpha^N+2\beta^N+1) = k(\beta+1)(M+1)$. Denote $l_C = (a+2L+b)(\beta^N+1)-\gamma^N$ the optimal cycle length. We have: $Opt(n,I) \geq kl_C$ (otherwise a better cycle would exist). The pure strategy (of length $P(n,I)$) is made of k cycles and an extension in order to finish the tasks started in the last window. Hence, $Opt(n,I) \leq P(n,I) \leq kl_C + L$. Therefore:

$$\frac{P(n,I)-Opt(n,I)}{Opt(n,I)} \leq \frac{L}{kl_c} \underset{k\to\infty}{\longrightarrow} 0$$

Eventually, if n is different, let $k = \lfloor \frac{n}{(\beta+1)(M+1)} \rfloor$, we have:

$$\frac{P(n,I)-Opt(n,I)}{Opt(n,I)} \leq \frac{P((k+1)(\beta+1)(M+1),I)-P((k-1)(\beta+1)(M+1),I)}{Opt(n,I)} \leq \frac{2l_C}{kl_c} \underset{k\to\infty}{\longrightarrow} 0$$

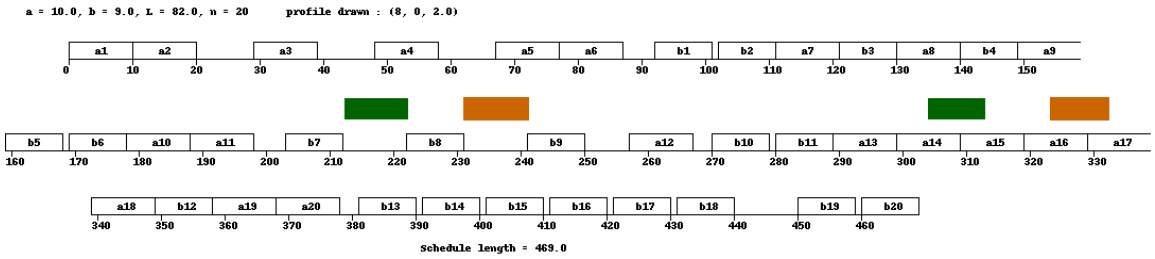
□

This is an important result, and following it, one may believe that for n sufficiently large, the best pure strategy will eventually be optimal. It will be shown later that there are instances where the pure strategy never yields an optimal solution (see section 6, page 17).

5 General results

We can first notice that while ghost a's can only appear in the last window, ghost b's may appear in other places. The fig. 11 illustrates why ghosts a's cannot exist anywhere else: the green and orange boxes illustrate two idle times of length a. Remark that counting a ghost a does not make sense as their b's would overlap with some other operations. On this figure, you can see that no ghost b could be placed into these idle times too because to place a ghost b between two existing b's requires the sequence $b(idle = a-b)(ghost\ b)(idle = a-b)b$. Hence the minimum space needed to place a ghost b is $(a-b)+b+(a-b) = 2a-b$.

Figure 11: An optimal solution in the case $a = 10$, $b = 9$, $L = 82$



Back to the cyclic case, with the previous definitions, remark that the optimal cyclic profile is tight and saturated. We can also immediately notice that a tight profile is saturated and that a saturated profile may not be tight, as illustrated fig 12, 13, 14 (the optimal profile for $a = 7$, $b = 3$, $L = 24$ is $(0,2,4)$).

We shall then summarize other general properties concerning profiles, profile blocks and blocks.

Figure 12: A tight block for $a = 7$, $b = 3$, $L = 24$

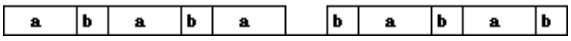
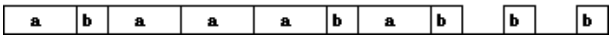


Figure 13: Another tight block for $a = 7$, $b = 3$, $L = 24$



Property 2.

- *A tight profile is saturated but a saturated profile is not necessarily tight.*
- *By definition, a profile block, based on profile (α, β) contains $2(1+\alpha+2\beta)$ elements. This implies that a profile block is a block if and only if its profile (α, β) is tight.*
- *Extending a profile block, based on a saturated non-tight profile, to a full block requires two more elements. The additionnal length is therefore at least $(a+b)$. Hence the resulting block is long with $\Delta \leq (a+2L+b)-(a+2L+b-\delta+a+b) < -b$.*
- *Consider a general block that is neither tight (and short) nor an extension of a saturated but non tight profile block (which is always long). Let us call such irregular block transition blocks. Such blocks are all long, except in a very particular case (see [1]): suppose the profile of the form $(0, \beta)$ is feasible and tight. Drop from the tight block $B(0, \beta)$ a single (suitable) element and extend the sequence to a full block. Such a block may be short with $\Delta \leq a - b$.*

6 Mixed profile strategies

In order to generate schedules, based on profiles blocks, but with changing profiles, we can do the following: replace an $'a'$ by a ghost b, if it is possible and feasible.

As explained in the previous section, this replacement is not possible in a sequence of the form $b(\epsilon_1)a(\epsilon_2)b$, with idle times such that $\epsilon_1 + \epsilon_2 < a - b$.

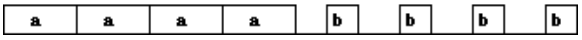
Two consecutive windows $W_1 = [a_1, L, b_1]$ and $W_2 = [a_2, L, b_2]$ are neighboring if there are no further a between a_1 and a_2. Then, the following result seems to hold:

***Conjecture* 1.** *Given two neighbouring tight windows W_1, W_2 of shadow profiles (α_1, β_1) and (α_2, β_2), we have: $\beta_1 \leq \beta_2$.*

Argument. All of the b's in W_2 are fixed by the a's of W_1 and earlier. As a consequence, the only way to change profile is to modify a's. Removing an a would result in a non tight window, hence the a's are either moved or replaced by ghost b's. Moreover, moving an a has to be done in a margin related to its surrounding idle times and the surrounding transformed a's. Therefore, the only way to change the profile is to transform some a's into ghost b's.
We will show that any transformation is either impossible or increases β_2 (we denote (α_2, β_2) the original profile and (α'_2, β'_2) after the transformation):

- $aaa \to a(b)a$: $\alpha'_2 = \alpha_2 - 2$, $\beta'_2 = \beta_2 + 1$, $\alpha'_2 + 2\beta'_2 = \alpha_2 + 2\beta_2$ the profile is still tight.
- $aab \to a(b)b = a\bar{a}b$: the profile is unchanged.
- $baa \to b(b)a = \bar{a}ba$: the profile is unchanged because in ba_1a_2, a_2 necessarily contributes in α and in $\bar{a_3}ba_4$, $\bar{a_3}$ necessarily contributes in α because a b before $\bar{a_3}$ would also be an $\bar{a}$.

Figure 14: A saturated yet non tight block for $a=7$, $b=3$, $L=24$



- $bab \to b(b)b = \bar{a}\bar{a}b$ this transformation can be discarded since the resulting schedule would not be compact: the a correspond to this ghost b could be placed instead of this ghost b. Consider the whole transformed sequence: $a_1b^ja_0b^ka_2 \to a_1b^j(ghost\ b)b^ka_2$. The idle time around the transformed a_0 should be at least $a-b$ and, in order for the schedule to stay tight, b_0 is transformed into an a. L time units later, the sequence corresponding to $S_1 = a_1b^ja_0b^ka_2$ is $S'_1 = b_1a^jb_0a^kb_2$ and the sequence corresponding to $S_2 = a_1b^j(ghost\ b)b^ka_2$ is $S'_2 = b_1a^ja^*a^kb_2$. Remark that in the sequences S'_1 and S'_2, if an a is transformed into a ghost b then the property is verified. Hence, suppose that there is no transformation in these sequences (moreover, as it must remain tight, no task is removed).
  Remark that as the a in S'_2 will be replaced by a b, it gives an $(a-b)$ margin to place the operation. Therefore, in order to make the placement of b_0 impossible, the only way is to place the a's in the sequence S' and the b's in the sequence S such that b_0 overlaps with some task. Yet, the placement margins are strongly constrained by the fact that an a can be placed in S and in S'. Hence, we think that the sequence S' would not be compact because a^* could be placed instead of the ghost b. However, we have not been able to prove this very carefully yet.

□

Remark that given a window of profile (α,β,γ) (we ignore what happens before and after this window), if $\alpha \geq 2$ replacing an aa by a ba is feasible and results in a profile $(\alpha-2,\beta+1,\gamma+(a-b))$. Moreover, if $\beta \geq 1$ and $\gamma \geq a-b$, we can replace a ba by an aa (it may be needed to shift the schedule) and the profile will be $(\alpha+2,\beta-1,\gamma-(a-b))$. In both cases, the windows have the same number of elements as the original window, which is $\alpha+2\beta+2$. This gives the list of profiles table 3.
As stated in remark 1, an $\bar{a}$ always counts in α. Hence, if we have some $\bar{a}$, replacing an $\bar{a}_0 = b_0$ by an a is feasible and either does nothing to the profile (if we had ab_0b) or gives a profile $(\alpha-2,\beta+1,\gamma+(a-b))$ (if we had bb_0b).

Property 3. *Given a profile (α,β,γ), all of the profiles having the same number of operations are, for i a positive integer, $(\alpha+2i,\beta-i,\gamma-i(a-b))$ with $\beta \geq i$ and $\gamma \geq i(a-b)$ or $(\alpha-2i,\beta+i,\gamma+i(a-b))$ with $\alpha \geq 2i$.*

Proof. All such profiles are feasible and have the same number of elements as stated in the remark before. Moreover, there are no other profiles with $(\alpha+2i,\beta-i)$ with i an integer because either β or γ will not be large enough (and a negative β or γ does not make sense).
Given another feasible schedule $(\alpha',\beta') = (\alpha+j,\beta+i)$, its number of tasks in L is $\alpha'+2\beta'$. Therefore, in order for it to have the same number of tasks we should have $\alpha'+2\beta' = \alpha+2\beta \Leftrightarrow j = -2i$, hence the profile should be of one of the forms given before. □

Following that property, remark that for a profile $(\alpha,0)$, β can be increased to $\lfloor \frac{\alpha}{2} \rfloor$. Hence, the following corollary comes:

Corollary 1. *When the optimal cyclic profile is $(\alpha^N,0)$, all saturated profiles are tight.*

Proof. Given a feasible profile (α,β), we have: $\alpha+2\beta \leq M = \alpha^N$, hence $\alpha \leq \alpha^N - 2\beta$ which corresponds to the tight profile $(\alpha^N-2\beta,\beta)$. Hence, if (α,β) is saturated, then $\alpha = \alpha^N - \beta$. □

Table 4: A β increasing sequence for $a=10$, $b=9$, $L=80$, $n=46$

Block index	Block configuration	Profile	Gain
1	$aaa(ba)(ba)(ba)(idle=3)b-$	$(2,3)$	3
2	$a(ghost\ b)a(idle=4)(ba)(ba)(ba)b-$	$(0,4)$	0
3	$a(ba)(ba)(idle=4)(ba)(ba)b-$	$(0,4)$	0
4	$a(ba)(ba)(ba)(idle=4)(ba)b-$	$(0,4)$	0
5	$a(ba)(ba)(ba)(ba)(idle=4)b-$	$(0,4)$	4

The property 3 allows us to create easily lists of equivalent profiles for i, the number of operations in L, from 1 to M. However, due to the construction process, the length of the whole list is an $O(L)$ and the number of profiles with the same block number is an $O(a)$. Hence it is not polynomial in the length of the instance and therefore, in order to have a polynomial algorithm, this list cannot be enumerated.

In theorem 1, we have shown that the pure strategy using the optimal cyclic profile is asymptotically optimal and we wondered whether for n sufficiently large, the pure strategy is really optimal or not. Given the previous results, we can think that we can make the pure strategy evolve in order to reach higher gain and hence the pure strategy will never be optimal in some cases. Indeed, there exists such a case. For instance, when $a=10$, $b=9$ and $L=82$, the optimal cyclic profile is $(8,0,2)$. Yet, for $n=k(M+1)+1$, the pure strategy has a makespan of $k(a+2L+b-\gamma^N)+a+L+b=181k+101$, while the solution $B(8,0,2)^{k-1}B(0,4,6)$ has a makespan of value: $(k-1)(a+2L+b-\gamma^N)+(a+2L+b-6)+a+L+b=181k+97$. Increasing β helps improving the schedule.

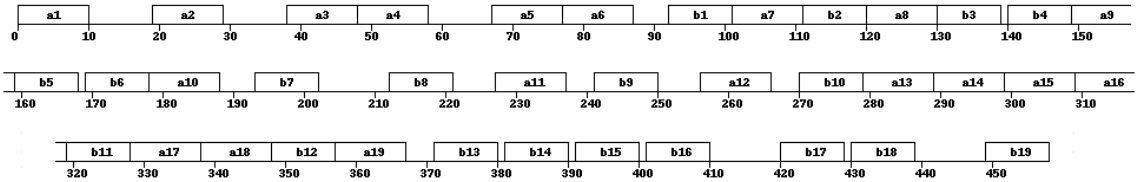
Therefore, in order to improve the pure strategy, we can make a β increasing strategy: using only tight blocks, the aim is to reach a maximal gain by increasing β. For instance, for $a=10$, $b=9$, $L=80$, the tight profiles are $(0,4,4)$, $(2,3,3)$, $(4,2,2)$, $(6,1,1)$, $(8,0,0)$. For 8 blocks ($n=73$), the pure strategy reaches an overall gain of 8. However, the schedule:
$B(4,2,2)B(4,2,0)B(2,3,0)B(2,3,3)B(2,3,0)B(0,4,0)B(0,4,0)B(0,4,4)$ is feasible and reaches a gain of $2+3+4=9$.
Table 4, another β increasing sequence is represented. In this example, the optimal cyclic profile is $(0,4,4)$, and the best pure strategy for $n=46$ uses profile $(2,3)$ and reaches a gain of 6. The β increasing sequence using tight profiles $(2,3)$ and $(0,4)$ reaches an improved gain of $3+4=7$.

Although this approach helps increasing the gain, finding the best β increasing sequence may be difficult: this is a knapsack problem (with structured data) and the list of elements has a length $O(a)$ (which is exponential in the length of the instance).

Let $n=k(M+1)+j$. Consider all tights profile $(\alpha_1,\beta_1,\gamma_1)\dots(\alpha_l,\beta_l,\gamma_l)$ such as $\forall i\in\{1,\dots,l\}\ \alpha_i+1\geq j$. The knapsack problem is the following: for each profile i, take one element of volume β_i+1 (the cycle length) and price γ_i. The volume of the bag is $k-1$. The aim is to take the objects that maximize the total value of the bag. The optimal β increasing sequence is then a sequence of profile blocks corresponding to those profiles, with a last tight block whose profile is (α,β) with $\alpha+1\geq j$ and $(\alpha-2,\beta+1)$ not tight or $\alpha-1<j$. The profile blocks can be taken of the form $a^{\alpha+1}(ba)^{\beta}b^{\alpha+1}(ab)^{\beta}$ with the gain maximal in the first block and β increased after β occurrences of the block.

Moreover, there is another problem which is that the optimal solution might not at all use any tight profile. Such an example is represented fig. 15. For this case, the best β increasing sequence gives a makespan of $2\times 181+97=459>458$. This was a very astonishing result for us and the reason why we started to think that this problem is not $\mathcal{NP}$. In this example, the optimal strategy uses a "transition" in order to reach a full gain in both of the profile blocks. The structure of the solution is $B(2,3,5)WB(4,2,4)$ with W a window that "collects" the b's from the tasks started in

Figure 15: The optimal solution for $a = 10$, $b = 9$, $L = 82$, $n = 19$ − makespan: 458



the previous block and starts the tasks to finish in the following block. We will call such windows transitions because they allow to change profiles (possibly with decreasing β) while preventing from finishing a whole cycle before gaining γ again.

Those transitions prevents us from considering "structured" blocks of the form $a^{\alpha+1}(ba)^{\beta}b^{\alpha+1}(ab)^{\beta}$ or $a(ba)^{\beta}a^{\alpha}b(ab)^{\beta}b^{\alpha}$ only because we have been able to prove that in some cases there exists no optimal solution starting with a block of one of the previous forms. We have done that by adding constraints to the integer program for the case $a = 10$, $b = 9$, $L = 82$, $n = 19$. In this case, no structured optimal solution exists.

Finding those transitions is not an easy task and we analyze it more precisely at the end of the subsection 7.2. Moreover, this combined with the fact that sometimes (such as represented fig. 10) the optimal solution do not use tight profiles, shows that this problem is far from being easy to solve, even without considering the complexity issues induced by the length of the instance.

By extension, we will call transition blocks the non-saturated blocks. Remark that a transition block starting with an a is long, because it begins with a window (and hence a fixed length $a+L+b$) followed by at least $M+1 > M$ operations. Hence, those operations need strictly more than L to be done. Therefore, the transition block length is strictly superior to $(a+L+b)+L$. Hence it is long.

Occasionally, transition blocks can be short when they begin with a b, $(0,\beta)$ is tight and $\gamma \geq b$: if we remove an a in a block in which the gain is $\geq b$, the task following the end of the block is a b that can be included in the block to replace the removed a without making the block long.

Remark that if $(1,\beta^N)$ is the optimal cyclic profile, $\gamma < b$ and $(0,\beta)$ is not tight. Hence, the short blocks are the saturated blocks.

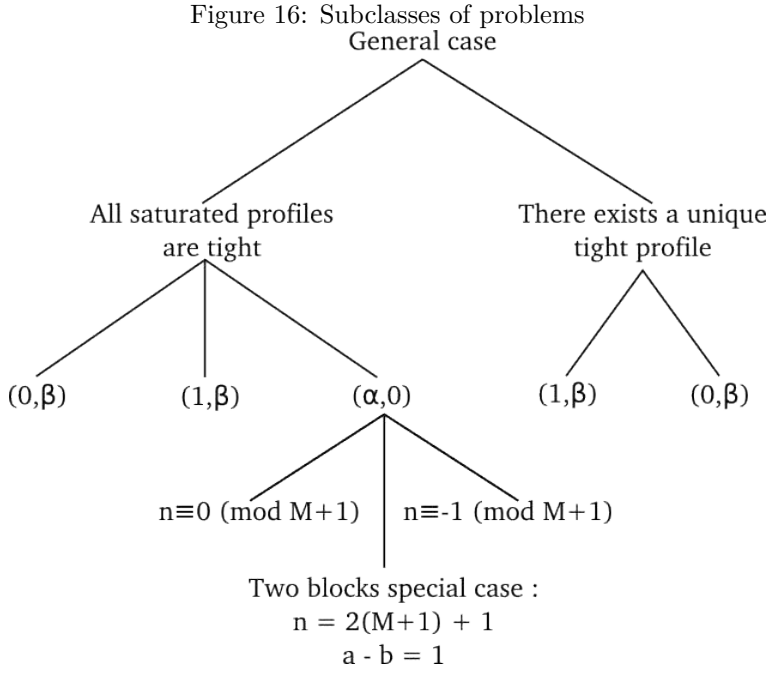
7 Special subclasses of instances

As the general case seemed to be hard to tackle with, we decided to isolate subclasses of the problem in order to solve them or prove them to be very hard to solve. The diagram 16 illustrates some of the studied subclasses.

There is another subclass which is trivial, that is if $n \leq \frac{L}{a}+1$, the greedy schedule is optimal as it places all a's with no idle times.

In the following subsections, we will take a closer look at some of these subclasses.

Figure 16: Subclasses of problems



7.1 Unique tight profile

We will consider the case where there exists a unique tight profile. There exists a unique tight profile when the optimal profile is $(\alpha^N, \beta^N, \gamma^N)$ with $\alpha^N \in \{0,1\}$ and $\gamma^N < a-b$. Moreover, if $\alpha^N = 1$, $\gamma^N < b$.

We focused on the case where $(0, \beta^N, \gamma^N)$ is the unique tight profile. That is: $(0, \beta^N, \gamma^N)$ is the optimal cyclic profile and $\gamma^N < a-b$. This case will be the one considered afterwards.

Given a block $B(0, \beta^N)$, all of the b's are fixed in the following window. It remains to place the a's but those a's are constrained by the positions of b's. Hence, there exists no transition respecting the structure conditions (exposed remark 4 and illustrated fig. 19) that allows to reach a full gain before and after it.
Therefore, in this case, the problem is not the transitions but rather to know if the optimal solution will use the tight profile or not.

Lemma 1. *Given a solution, divide it in blocks by starting from the first task and counting blocks. Suppose one of these block is tight, take the first tight block of this division. Making all of the previous block tight improves the schedule.*

Proof. This is the first tight block. Hence, all of the previous blocks are non tight. Therefore, in this case, they are long. Replacing all of them by tight blocks (by extending the first tight block to the left) makes them all short and hence improves the makespan. □

Remark that for the optimal solution, if it admits a tight block then the first block of the optimal solution is tight because suppose the tight block is not the first block, transforming the previous block into tight blocks strictly improves the schedule, which is absurd.

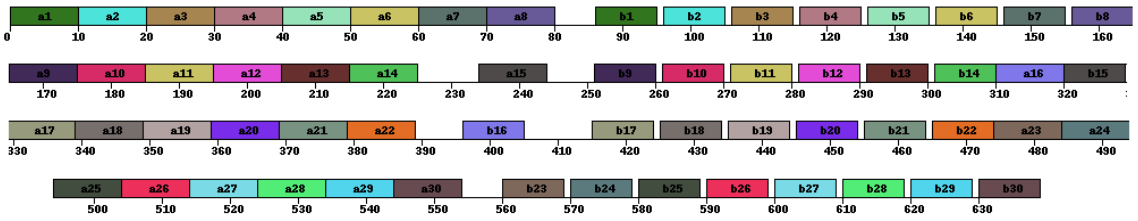
Moreover, once there is a tight block in the optimal solution, it is very costly to change profile therefore, it seems that either the optimal solution is the pure strategy using the optimal cyclic profile or none of its windows is tight. One can wonder if an optimal solution without tight windows

exists. Unfortunately, this is the case: fig. 10 the optimal solution for $a=7$, $b=3$, $L=43$, $n=12$ is represented. The optimal cyclic profile is $(0,4,3)$. Hence $M=8$ and this case contains one block and an extension. The optimal solution uses the profile $(3,2,3)$ of block number 7. The makespan is 168 while the makespan of the pure strategy is 169. This optimal solution is remarkable for two reasons: the first one is that in the second block, all a's are placed consecutively, with no idle times and without being separated by some b's. The second reason is that the gain made in the first block is not "lost" afterwards: the gain in the first block is the idle time before the first b of the schedule. However, the first b of the second block will be preceded by an idle time of the same duration. Yet, in this case, there are no a starting after the first b of the second block. Hence, there is no loss due to this idle time.

Concerning this case, we have run many instances and most of the time the optimal solution is the pure strategy using the optimal profile. Moreover, provided that n is large enough, it will not be affordable to lose tasks in all blocks to increase the gain and hence the pure strategy will be used. However, deciding whether the pure strategy is optimal or not seems to be complicated.

In order to simplify this case we tried to find properties in the case where $\gamma^N=0$ and hoped that in such a case, the pure strategy could always be optimal. However, we found a very interesting counter example: for $a=10$, $b=9$, $L=76$ and $n=30$, the optimal cyclic profile is $(0,4,0)$. Hence, for $n=30$, there are 3 blocks and an extension. This may seem to be large and so one can think that the optimal cyclic profile will be used but in fact it will not. The length of the pure strategy is 646 while the length of the optimal solution is 639. The optimal solution is illustrated fig. 17. It is not compact but can be made compact by placing the operation a_{24} before b_{17}.
In the compact equivalent schedule, the solution starts with a block $(7,0)$, then β is increased and all of the remaining blocks have a profile $(5,1)$. Therefore, the optimal solution is not complicated, yet we have to find the best β increasing sequence of blocks which is a hard task.
Moreover, for n small, there is often an optimal strategy which do not uses the tight profile. For instance, when $a=10$, $b=9$, $L=76$, for $n=11$ to $n=26$, the pure strategy is optimal only 5 times.

Figure 17: An optimal schedule for $a=10$, $b=9$, $L=76$ and $n=30$



Even in the more specific case $\gamma^N=0$, it seems to be quite complicated to figure out whether the optimal solution uses the pure strategy or not and if not how is the optimal solution.

7.2 All saturated profiles are tight

In this case, the optimal profile can take all of the three possible shapes.

We started by working on the case where $(\alpha,0)$ is optimal (hence all saturated profiles are tight, from 1). We thought that as the optimal cyclic profile is the greedy solution, it could be easy to generalize. In fact it is the opposite: as all saturated profiles are optimal, transition windows can occur, making this case very complicated to deal with.

In this case, we still have a useful structure properties:

Property 4. *The first block of an optimal schedule is tight.*

Proof. As all saturated profiles are tight $(\alpha^N + 2\beta^N, 0)$ is feasible and tight (and hence short). Whatever the first block is, it starts with an a and can be replaced by $B(\alpha^N + 2\beta^N, 0)$ because this block is not constrained by a previous block (no task previously started has to end in it) and completes $M+1$ whole tasks while any other block completes less tasks (either it is not compact and it may accomplish $M+1$ whole tasks or it has ghost b's and accomplishes strictly less). Hence, if the first block is not tight, then it is long and replacing it by $B(\alpha^N + 2\beta^N, 0)$ (which is short) will shorten the schedule. Therefore, the first block of an optimal schedule is tight. □

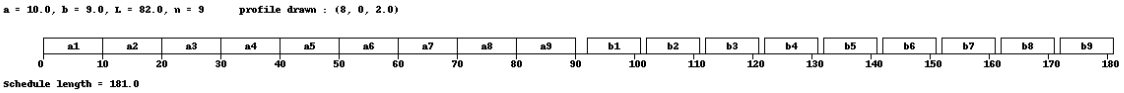
In the next part of this section, we will consider the case in which the profile $(\alpha, 0)$ is optimal in the cyclic case. This can be determined in $O(\log(L)^2)$ time (using algorithm 1) which is polynomial in the length of the instance.

We denote B the block of profile $(\alpha^N, 0, \gamma^N)$ with its idle time before the first b:

$$B = a^{\alpha^N+1}(idle = \gamma^N)b^{\alpha^N+1}$$

The length of the block B is $|B| = a + 2L + b - \gamma^N$. This block is illustrated fig. 18.

Figure 18: Block B in the case $a = 10$, $b = 9$, $L = 82$



In [1], it is shown that when the profile $(\alpha^N, 0, \gamma^N)$ is optimal, we have: $b > \gamma^N \geq a - b$.

Remark 3 (Separability). The block B is *separable*, which means that given any feasible schedule containing a block B, this block can be removed without changing anything in any other block and the resulting schedule is still feasible. This comes from the fact that as there is no b in the first part of the block B, there is no a in the previous block that should finish in the beginning of the next block and as there is no a in the second part of B, there is no b that should finish in the beginning of the following block. So there is no a in the end of the previous block and there is no b in the beginning of the following block. Therefore, those two blocks can be put one after the other without any modification.
Hence, block b splits the schedule, if there is a block B at some point in the optimal solution, this means that the previous and the following schedules are both optimal for their respective number of tasks.

Property 5. *For $n \equiv 0 \mod M+1$, the optimal solution is $B^{\frac{n}{M+1}}$.*

Proof. As B is an optimal cycle in the cyclic case, $B^{\frac{n}{M+1}}$ is also an optimal cycle. Suppose we have a solution S such that $|S| < |B^{\frac{n}{M+1}}| = \frac{n}{M+1}(a + 2L + b - \gamma^N)$, then S is a strictly better cycle than $B^{\frac{n}{M+1}}$ which is absurd. Hence there is no such S. □

Please note that this proof works because there is no extension to $B^{\frac{n}{M+1}}$. In the general case, repeating a cycle does not lead to a solution: in the finite case, the a's started at the end of the last block need to have their b's finished.

Studying this case, we have observed that for pure strategies using saturated profiles, it seemed that the best schedule was given by a decreasing sequence of α which is reinitialized every $M+1$ tasks. We have computed pure strategies values for saturated profiles in the example $a = 10$, $b = 9$, $L = 82$ and increasing n. Their values are given table 5. You can observe this decreasing sequence of α with the best pure strategy, it is: $\alpha^{OPT} = 0, 2, 2, 4, 4, 6, 6, 8, 8, 0, 2, 2, 4, \ldots$.

Table 5: Pure strategies values for $a = 10$, $b = 9$, $L = 82$ and small n

Task number \ Profile	(8, 0, 2)	(6, 1, 3)	(4, 2 , 4)	(2, 3, 5)	(0, 4, 6)	Optimal value
6	**151**		160	178	202	151
7	**161**		179	222	221	161
8	**171**	180	242	241	240	171
9	**181**	262	261	260	259	181
10	282	281	280	279	**278**	278
11	292	291	290	**289**	303	289
12	302	301	300	**299**	322	299
13	312	311	**310**	323	341	310
14	322	321	**320**	342	360	320
15	332	**331**	343	361	379	331
16	342	**341**	362	400	404	341
17	**352**	363	421	424	423	352
18	**362**	442	444	443	442	362
19	463	464	463	462	461	**458**
20	473	474	473	472	480	**469**
21	483	484	483	482	505	**479**
22	493	494	493	501	524	**490**
23	503	504	503	525	543	**501**
24	513	514	522	544	562	**511**
25	523	524	545	583	581	**522**
26	**533**	543	604	602	606	533
27	**543**	625	623	626	625	543
28	644	644	646	645	644	**638**
37	825	827	825	828	827	**819**

Table 6: β increasing strategies values for $a = 10$, $b = 9$, $L = 82$ and small n

Task number \ Profile	(8, 0, 2)	(6, 1, 3)	(4, 2 , 4)	(2, 3, 5)	(0, 4, 6)	Optimal value
19	462				459	**458**
20	472			470		**469**
21	482			480		**479**
22	492		491			**490**
23	502		**501**			501
24	512	512				**511**
25	522	**522**				522
26	**533**					533
27	**543**					543
28	644				640	**638**
37	825				821	**819**

This seemed to be a very encouraging result and we hoped that managing the end well (by making the best β increasing sequence) could lead us to an optimal solution. In fact, we thought that as all blocks should be short (this is our mistake), they must be tight and hence, β can only increase.
You will find fig. 6 the values of the solutions when β is increased in the last block only (there may exist better β increasing sequences for $n = 28$ and 37).

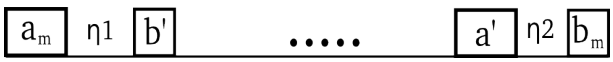
In the case $n = 19 = 2(M+1)+1$, there are two blocks and one task, hence it is possible to increase β only once. The solution would then be $B(8,0)B(0,4)a$ and the makespan 459 (overall gain: $\Delta = 2+6 = 8$). However, after being unable to prove this theoretically, we have investigated this case deeper and proved, using `CPLEX`, that the optimal makespan value is 458. The corresponding solution is represented fig. 15.

The structure of the optimal solution is the following: $B(2,3;5)WB(4,2;4)$, with an overall gain of 9. $W = [a_{10}, L, b_{10}]$ will be called a *transition* window. It allows to change profile block in order to reach a full gain in both of them. If we count blocks in the optimal solution, the second block is long. However, the counterpart is such that it is worth delaying some tasks.

The structure of a transition is illustrated fig. 19.

Remark 4. Remark that given a profile block $a_0 a^j b_{-t} \dots b_{-1} a^k b_0 \dots$ (with j,k positive integers), its gain is the idle time that was before b_0. Hence, in the next block, this idle time comes before b_{-t}, while in the previous block, this idle time was before b_{-1}.
Therefore, given two blocks B_1 (whose first b in the transition window is denoted b') and B_2 (whose last a in the transition window is denoted a'), in order to reach a gain $\delta_1 + \delta_2$, we have to be able to make a transition such that: $\eta_1 = ja + \delta_1$ and $\eta_2 = pa + \delta_2$ with j,p positive integers.

Figure 19: The structure of a transition



This example rises some huge problems and cuts out our hopes that this problem may be polynomial:

- In this solution, there are several profiles and a window. How can we describe the solution so that it is polynomial in $log(n)$ and $log(L)$? The number of blocks is $\frac{n}{M} > \frac{n}{L/b} = \frac{bn}{L} > O(\frac{n}{L})$

which is exponential in n. Can we find some structures that would describe the solution polynomially ?

- Do transitions exist ? Otherwise, is the optimal solution using short blocks only ? If it is the case, how can we find the best β increasing sequence ? − if we change only the last block, it is easy − in the general case, finding a sequence may be tough in its combinatorics. Especially, how could we do it in $O(log(L))$?
- How can we find the transitions configurations ? Given two blocks B_1 and B_2 that are not trivially incompatible (for instance if $\beta_1+\beta_2 \geq M$). This can be modeled as a complicated flow problem with a graph exponential in the length of the instance. More precisely, given a window of profile (α,β) the number of possible combinations of these elements is $\binom{\alpha+\beta}{\alpha} = \binom{\alpha+\beta}{\beta}$. Hence, enumerating all possible transitions cannot be done in polynomial time.

All of these results incite us to think that even this case cannot be solved or described in polynomial time. However, the case $n \equiv -1 \mod (M+1)$ is quite interesting as it seems that the optimal solution is $B^{\frac{n+1}{M+1}}$ without the last task. This is not surprising because $(\alpha^N,0)$ is the only tight profile in which $M-1$ a's can be started in L, hence, as the last B loses only one task, it cannot be replaced by another block in order to increase the gain. Therefore, $B^{\frac{n+1}{M+1}}$ minus the last task should be the optimal solution. We have not proved it yet but tests on other examples did not invalidate this conjecture.

The following section concerns a particular, yet very interesting, subcase of this class of problems.

7.3 An interesting subclass of instances

The last class of instances is very particular and help us emphasize the singularity of optimal solutions in the finite case.

We consider the following problems: $a-b=1$, $(\alpha^N,0,\gamma^N)$ is the optimal cyclic profile (hence all saturated profiles are tight). $M=\alpha^N$ and $n=2(M+1)+1$. In other words, there are two blocks and a window to schedule.
The tight profiles are $(\alpha^N-2i,i,\gamma^N+i)$, for $i \in \{0,\dots,\beta_{max}=\lfloor\frac{\alpha^N}{2}\rfloor\}$. The length of the tight profiles list is $O(a)$.

The fig. 15 illustrates the optimal solution for the case $a=10$, $b=9$, $L=82$, $n=19$. On this example, the optimal solution is $B(2,3,5)BE$ where B is a transition block and E the extension. However, dividing another way, the optimal solution is $B(2,3,5)WB(4,2,4)$ where W is a window in which the tasks of the previous block end and those of the next block start. Moreover, it allows to retrieve a full gain in the second block. This is an interesting and original structure. Hence one can wonder whether it is common to all of the solutions of the current subclass.

We define the *standard block* $\bar{B}(\alpha,\beta,\Delta)$ as a sequence of operations $a_1\dots a_2(idle=\Delta)b_1\dots b_2$ (there may be any kind of operations between a_1 and a_2 and b_1 and b_2 but no $\bar{a}$ between a_1 and b_1) such as the shadow profile of the block is (α,β,γ) and $\Delta=\gamma$. As $\Delta=\gamma$, those blocks may be denoted $\bar{B}(\alpha,\beta)$ too.

Theorem 2. *There exists an optimal solution of the form* $\bar{B_1}(\alpha_1,\beta_1;\gamma_1)W\bar{B_2}(\alpha_2,\beta_2;\gamma_2)$ *with* $W=[a_m,L,b_m]$ *a transition window and both blocks are tight.*

The total length of the schedule is then $(a+2L+b-\gamma_1)+(a+L+b)+(a+2L+b-\gamma_2) = C-(\gamma_1+\gamma_2)$ where $C=3a+5L+3b$ is a constant. We denote $\Delta=\gamma_1+\gamma_2$. Hence, finding the optimal schedule is finding a schedule that maximizes Δ.

Proof of theorem 2. The solution $\bar{B}(\alpha^N,0)W\bar{B}(\alpha-2\beta_{max},\beta_{max})$ is always feasible and has a gain $\Delta=\gamma^N+\gamma^N+\beta_{max}$.

Given an optimal solution, it can be seen as a B_1WB_2 (it is just a matter of ghosts). B_1 and B_2 are short because otherwise they could be replaced by $\bar{B}(\alpha^N,0)$. Moreover, B_1 and B_2 are tight because there exists only one type of non tight block and its gain is $(a-b) \leq \gamma^N$.
As B_1 is the first block, it is already in the standard form $\bar{B_1}$. B_2 can be modified to be in standard form, without being lengthened. $B_2 = a_1 \dots a_2 \dots b_1 \dots b_2$. In $a_2 \dots b_1$ there are only b's and in $b_1 \dots b_2$ there are only b's and ghost a's.

- If there is an $\bar{a}$ in $a_1 \dots a_2$, the corresponding coupled task can be removed and an a can be placed instead of the $\bar{a}$. There will be no conflict with its b in the last window and the schedule length is unchanged.

- If there is a b in $a_2 \dots b_1$ then there is an $\bar{a}$ and B_2 is preceded by an a. Remove this a and shift the task (a_1, b_1) to the left. Insert this a after a_1. The length of the schedule is unchanged.

The first block in an optimal solution (left-shifted) is some $\bar{B_1}(\alpha_1, \beta_1, \Delta_1)$ with $\Delta_1 = \gamma_1$. The block contains $1 + \alpha_i + 2\beta_i$ operations a. Consider the schedule of the remaining $(K+1)$ coupled tasks, the first being a_m. We know the optimal solution for $(K+1)$ coupled tasks has a length $a + L + b + |B(\alpha^N - 2\beta_{max}, \beta_{max}, \gamma^N + \beta_{max})|$. This means that b_m must be before B_2 starts, otherwise there would be a better solution.
Eventually, the gain in the last block is γ_2 because the schedule is left shifted and hence if there is an idle time, it is placed before a $'b'$. Therefore, when considering an integer optimal solution (which always exists) there is an idle time of duration at least 1 followed by a $'b'$ which is a total duration of at least a. Hence we can replace it by an a and shift the schedule to the left. The resulting schedule has a smaller makespan.
Therefore, there exists an optimal solution of the form $\bar{B_1}(\alpha_1, \beta_1; \gamma_1) W \bar{B_2}(\alpha_2, \beta_2; \gamma_2)$ with $\bar{B_1}$ and $\bar{B_2}$ tight. □

Remark that for $(\alpha - 2\beta_{max}, \beta_{max})$ there exists no transition window that allows us to reach a maximal gain on both of its sides because the a's in B_1 fix the positions of the B's in B_2 and hence lock the positions of the a's in W. Hence, the overall gain value is strictly inferior to $2\gamma^N + \beta_{max}$. Therefore, the objective is to find the two blocks and transition window that allow us to reach a maximum gain. This gain is in $\{2\gamma^N + \beta_{max}, \dots, 2\gamma^N + \beta_{max} - 1\}$.

In this case, the certificate should be a compact description of the transition window (which is a list of length $O(L)$). For instance, the starting points of a's and b's in the window section L could be given. This certificate could be verified in polynomial time (in L): the first step is to check that there is no overlap in order to verify the feasibility. The second step is to count a's and b's: if there are β_1 b's and β_2 a's, then this means that $B_1 = B(\alpha^N - 2\beta_1, \beta_1, \gamma^N + \beta_1)$ and $B_2 = B(\alpha^N - 2\beta_2, \beta_2, \gamma^N + \beta_2)$. Moreover, it is infeasible if $\beta_1 + \beta_2 \geq M$. Eventually, the order and the feasibility of the operations in $\bar{B_1}$ and $\bar{B_2}$ can be retrieved by comparing the positions of the operations in the window with the positions in the standard list. For instance, for B_1 the position of the first b in the window should be in γ_i, or $\gamma_i + a$, or $\gamma_i + 2a$, ...
Even though this certificate can be verified in polynomial time (pseudo-polynomial in the length of the instance), its length is still exponential in the length of the instance. Hence, in order to have a polynomial certificate we need to find more structure on the optimal solution.

In order to maximize Δ we can use a binary search on the value of Δ: we know that $\Delta \in \{2\gamma^N + \beta_{max}, \dots, 2\gamma^N + 2\beta_{max} - 1\}$. Moreover, given a value of Δ, the possible profile combinations can be determined in linear time. Hence, the problem is to determine the feasibility of a Δ. However, we cannot enumerate all of the $\binom{\alpha+\beta}{\alpha}$ possible combinations to achieve this goal. Therefore, we have to hope that there are some patterns in the optimal solutions.

Fig. 15, illustrates that being able to use a simple structured pattern as $a^\alpha (ba)^\beta$ in this case is hopeless. On this example, the combined profiles are $(2,3)$ and $(4,2)$ and the overall gain reached is $\Delta = 9$.

Yet, we have been studying more precisely lots of examples (171). Table 7 illustrates some optimal profiles for $a = 10$ and $b = 9$, with differents L and n but $n = 2M + 3$. Analyzing all of the examples, we remarked that in all cases, the two optimal profiles are (α_1, β_1) and (α_2, β_2) with (α_1, β_1) the tight profile whose α_1 is maximal such that $\alpha_1 \leq \beta_1$ and (α_2, β_2) the tight profile whose α_2 is minimal such that $\alpha_2 \geq \beta_2$. Hence $\alpha_1 = \beta_2$ and the two optimal profiles can be written as $(\alpha^N - 2x, x)$ and $(4x - \alpha^N, \alpha^N - 2x)$ with $x = \lceil \frac{\alpha^N}{3} \rceil$. Computing those profiles is simple arithmetic and requires the optimal cyclic profile to have been computed previously. Hence, the complexity is $O((log\ L)^2)$ which is polynomial in the length of the instance.

Theorem 3. *Denote $x = \lceil \frac{\alpha^N}{3} \rceil$, there is an optimal solution using profiles $(\alpha^N - 2x, x)$ and $(4x - \alpha^N, \alpha^N - 2x)$ in respectively the first and second block and those blocks are separated by a transition window. The gain of this schedule is $2\gamma^N + \lfloor \frac{2\alpha^N}{3} \rfloor$ and its length is $3a + 5L + 3b - 2\gamma^N - \lfloor \frac{2\alpha^N}{3} \rfloor$*

Lemma 2. *Let $x \in \mathbb{N}$, $x - \lceil \frac{x}{3} \rceil = \lfloor \frac{2x}{3} \rfloor$.*

Proof. If $x \equiv 0 \mod 3$, $\lceil \frac{x}{3} \rceil + \lfloor \frac{2x}{3} \rfloor = x$.
If $x \equiv 1 \mod 3$, $x = 3k + 1$ and $\lceil \frac{x}{3} \rceil + \lfloor \frac{2x}{3} \rfloor = (k + 1) + 2k = x$.
If $x \equiv 2 \mod 3$, $x = 3k + 2$ and $\lceil \frac{x}{3} \rceil + \lfloor \frac{2x}{3} \rfloor = (k + 1) + (2k + 1) = x$.

□

Proof of theorem 3. There always exists a feasible solution using those profiles. This solution first block is:

$$a(baa)^{\alpha^N - 2x}(ba)^{3x - \alpha^N}(idle = \gamma_1)b(abb)^{\alpha^N - 2x}(ab)^{3x - \alpha^N}$$

Its last block is:

$$aa(ba)^{\alpha^N - 2x}a^{4x - \alpha^N}(idle = \gamma_2)bb(ab)^{\alpha^N - 2x}b^{4x - \alpha^N}$$

Where the b's in the first window and a's in the last window are ghosts and $\gamma_1 = \gamma^N + x$ and $\gamma_2 = \gamma^N + \alpha^N - 2x$.
The baa's in the first window let aa spaces in the transition window between two b's. These empty spaces can be used to place an a so that it matches a b in the first part of a second block. Moreover, as $\alpha_1 = \beta_2$, there are enough such spaces. Hence, the schedule is feasible and reaches maximum gain (related to the profile used) in the two blocks. This gain is $\gamma_1 + \gamma_2 = 2\gamma^N + \beta_1 + \beta_2 = 2\gamma^N + x + \alpha^N - 2x = 2\gamma^N + \alpha^N - \lceil \frac{\alpha^N}{3} \rceil = 2\gamma^N + \lfloor \frac{2\alpha^N}{3} \rfloor$ as stated in lemma 2. The length of this schedule is then $3a + 5L + 3b - 2\gamma^N - \lfloor \frac{2\alpha^N}{3} \rfloor$.

Theorem 2 implies that an optimal solution can be represented as $\bar{B_1}(\alpha_1, \beta_1; \gamma_1)W\bar{B_2}(\alpha_2, \beta_2; \gamma_2)$ with both blocks tight and reaching maximal gain in both of them. The problem we need solve is to maximize the gain of the two blocks: $G = \gamma_1 + \gamma_2 = 2\gamma^N + \beta_1 + \beta_2$. Writen as an integer program, this gives:

$$\begin{aligned}
\max \quad & \beta_1 + \beta_2 && (11)\\
\text{s.t.} \quad & \alpha_1 + 2\beta_1 = \alpha^N && (12)\\
& \alpha_2 + 2\beta_2 = \alpha^N && (13)\\
& \alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{N} && (14)
\end{aligned}$$

Yet, this model lacks constraints: we cannot combine any two profiles to reach a maximum gain. We have seen previously that the a's in the transition window need to be adjusted so that they can match b's in the second block. Hence, if we only use the ghost aa's in the transition window to place a's, we are limited to $\beta_2 \leq \min(\alpha_1, \beta_1 + \frac{\alpha_1 - \beta_1}{2})$ and we cannot reach a higher gain than in the solution above. In order to have $\beta_2 > \alpha_1$, we have to remove the offset γ_1 caused by the gain of

the first block. To achieve this goal, we need to place either $b(ghost\ a)$'s in the transition window corresponding to an aa in the second block (the offset decreases by one) or to place a's adjusted on two ghost a's in the transition window corresponding to a ba in the second block (this makes the offset increase by one). Once the offset is equal to $0 \mod a$ we can then place as much a's as possible in the transition window, separated by ghost or real b's. Remark that using both of these constructions is inefficient as each of them works in the opposite way of the other. Hence, we may assume that we will only use one of them.
Suppose that we place $b(ghost\ a)$'s in the transition window. In order to decrease the offset, we need to place aa's in their place in the second block. Moreover, we can assume that we will be placing these $b(ghost\ a)$'s consecutively starting from $t = a + 2L + b$ because any other placement of tasks can only increase or let the offset unchanged and hence will not allow us to increase β_2 to its highest. Hence, we will place γ_1 ba's and $\beta_1 \geq \gamma_1 = \beta_1 + \gamma^N$. Therefore, $\gamma^N = 0$ and $\beta_1 = \gamma_1$. Eventually, $\beta_2 \leq \frac{\alpha_1}{2} \leq \alpha_1$.
On the other hand, suppose that we use the other construction. We need to use only aa's and placing b's cannot help. Hence, we should place $a - \gamma_1$ aa's in the first block consecutively which has a length (including the idle time γ_1):

$$2a(a-\gamma_1)+\gamma_1 = 2a^2 - 2a\gamma_1 + \gamma_1 = (a-\gamma_1)(2a-1) + a = (a-\gamma_1)(a+b) + a$$

This means that using the idle time in the first block plus the time spent for the $2(a-\gamma_1)$ a's, we can place $(a-\gamma_1)$ ba's and one a which is one more operation than previously. Hence the first block would not be tight and the solution is not of the form we are looking for (and proved it exists).
Therefore, $\beta_2 \leq \alpha_1$. Moreover, as we cannot remove the offset, any b in the transition window has an aa overlaping its corresponding place in the second block and over those two a's, at least one is counted in α_2. Hence $\beta_1 \leq \alpha_2$.
Now, suppose that $\alpha_1 < \beta_1$ and $\alpha_2 < \beta_2$. Then we have $\beta_2 > \alpha_2 \geq \beta_1 > \alpha_1$ which is impossible because $\beta_2 \leq \alpha_1$. Hence, $\alpha_1 \leq \beta_1$ or $\alpha_2 \leq \beta_2$.
Eventually, on the overall gain, we have: $G \leq 2\gamma^N + \alpha_1 + \beta_1$ and $G \leq 2\gamma^N + \alpha_2 + \beta_2$ and since $\alpha_i + \beta_i = \alpha^N$ for both cases, the best bound is the one that minimizes α_i. Therefore, it respects $\alpha_i \leq \beta_i$. Taking these constraints into account, if the solution $(\alpha, \beta)W(\alpha^N - 2\alpha, \alpha)$ of the following integer program is feasible with a maximum gain, then it is optimal.

$$\begin{aligned}
\max \quad & \alpha + \beta && (15)\\
\text{s.t.} \quad & \alpha + 2\beta = \alpha^N && (16)\\
& 2\alpha \leq \alpha^N && (17)\\
& \alpha \leq \beta && (18)\\
& \alpha, \beta \in \mathbb{N} && (19)
\end{aligned}$$

We can then replace α using 16 which gives:

$$\begin{aligned}
\min \quad & \beta && (20)\\
\text{s.t.} \quad & 4\beta \geq \alpha^N && (21)\\
& 3\beta \geq \alpha^N && (22)\\
& \beta \in \mathbb{N} && (23)
\end{aligned}$$

Hence $\beta = \lceil \frac{\alpha^N}{3} \rceil$ and the previously given solution is such that $\beta_1 = \lceil \frac{\alpha^N}{3} \rceil$ and $\beta_2 = \alpha_1$, therefore, it is optimal.

□

This last result is polynomial. Yet, it emphasizes the complexity of this problem in both the description of the solutions (the certificate) and the solving process as it seems that there is no

Table 7: Optimal profiles for $a = 10$, $b = 9$ and $L = 82$, $n = 19 - L = 92$, $n = 21 - L = 104$, $n = 23$

Profil	Gain
(0,4)	6
(2,3)	**5**
(4,2)	**4**
(6,1)	3
(8,0)	2

Profil	Gain
(1,4)	6
(3,3)	**5**
(5,2)	4
(7,1)	3
(9,0)	2

Profil	Gain
(0,5)	9
(2,4)	**8**
(4,3)	7
(6,2)	**6**
(8,1)	5
(10,0)	4

recursivity and beeing able to find rules for each congruence class and each different number of blocks is out of question.

Conclusion

Given the results of the cyclic case, we thought that we could generalize the results of the cyclic case to the finite case and tried to demonstrate some properties. Then we worked on some examples and realized that many subtleties occur in the finite case. Eventually, we worked on a very specific case and realized that even in such a "simple" case, finding the optimal solution cannot be done easily.

Moreover, in order to be able to give a certificate polynomial in the length of the instance in such a high multiplicity scheduling problem, we need to be able to aggregate data and hence the optimal solutions need to be structured.

Ultimately, all of these results lead us to the following conjecture:

***Conjecture* 2.** *The identical coupled task scheduling problem is not in $\mathcal{NP}$*

It remains to prove this conjectures and some other results mentioned in this report.

Trying to prove that there exists no polynomial certificate may be very complicated as this is not because one has not found a polynomial certificate that no such certificate exists.
All of the results obtained during that study let us think that solving this problem is also exponential. In fact, this problem is probably in EXPSPACE which is known to be a strict superset of PSPACE (and hence of $\mathcal{NP}$). Therefore, a good approach to settle the complexity of this problem could be to try to find a reduction of some problem in EXPSPACE to this problem.

References

[1] Vassilissa Lehoux-Lebacque, Nadia Brauner, and Gerd Finke. Identical coupled task scheduling: polynomial complexity of the cyclic case. *to be published, déjà paru dans les Cahiers Leibniz n179*, 2011.

[2] Cyril Duron. *Ordonnancement en temps-réel des activités des radars.* PhD thesis, Université de Metz, December 2002.

[3] AJ Orman and CN Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.

[4] N. Brauner, Y. Crama, A. Grigoriev, and J. Van De Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of combinatorial optimization*, 9(3):313–323, 2005.

[5] N. Brauner, Y. Crama, A. Grigoriev, and J. Van De Klundert. Multiplicity and complexity issues in contemporary production scheduling. *Statistica Neerlandica*, 61(1):75–91, 2007.

[6] Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/NP_%28complexity%29.

[7] R.D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(3):489–498, 1980.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.

[9] J.N.D. Gupta. Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *Journal of Global Optimization*, 9(3):239–253, 1996.

[10] Nadia Brauner, Gerd Finke, Vassilissa Lehoux-Lebacque, Chris Potts, and Jonathan Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers & Operations Research*, 36(2):301 – 307, 2009. Scheduling for Modern Manufacturing, Logistics, and Supply Chains.

[11] D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research*, 59(2):193–203, 2004.

[12] P. Baptiste. A note on scheduling identical coupled tasks in logarithmic time. *Discrete Applied Mathematics*, 158(5):583–587, 2010.

[13] J. Blazewicz, K. Ecker, T. Kis, CN Potts, M. Tanas, and J. Whitehead. Scheduling of coupled tasks with unit processing times. *Journal of Scheduling*, pages 1–9, 2010.

[14] G. Simonin, B. Darties, R. Giroudeau, and J.C. K
"onig. Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. *Journal of Scheduling*, pages 1–9.

[15] Vassilissa Lehoux-Lebacque. *Théories et applications en ordonnancement : contraintes de ressources et tâches agrégées en catégories*. PhD thesis, Université Grenoble 1 - Joseph Fourier, September 2007.

[16] E. Winter and P. Baptiste. On scheduling a multifunction radar. *Aerospace science and technology*, 11(4):289–294, 2007.

[17] J. Békési, G. Galambos, M. Oswald, and G. Reinelt. Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters*, 37(2):93–96, 2009.

[18] P. Dodin, P. Minvielle, and J.P. Le Cadre. A branch-and-bound algorithm applied to optimal radar search pattern. *Aerospace science and technology*, 11(4):279–288, 2007.

Appendices

A Integer Programming model

Listing 1: "Integer Program for identical coupled task scheduling problem"

```
int a = ...;
int b = ...;
int L = ...;
int n = ...;
range Tasks = 1..n;
range MinusTasks = 2..n;

int s = a + L + b;

dvar int+ t[Tasks];
int greedy[Tasks];
int grd;
int M;
int G;
dvar boolean y[Tasks][Tasks];

// Computes greedy schedule value
execute GREEDY {
    var cour = 0;
    var deb = 1;
    for (var i in Tasks) {
        greedy[i] = cour;
        if (cour + 2 * a > greedy[deb] + a + L) {
            cour = greedy[i] + a + L + b;
            deb = i + 1;
        } else {
            cour = greedy[i] + a;
        }
    }
    grd = greedy[n];
    M = 2 * grd + L;
    G = L / a + 2;
}


minimize t[n];

subject to {
    t[1] == 0;
    t[n] <= grd;

    forall ( i , j in Tasks : i >= j) {
        y[i][j] == 0;
    }
```

```
45
    forall ( i in MinusTasks ) {
        orderingTasks:
            t[i] >= t[i-1] + a;
50      no_separation:
            t[i] <= t[i-1] + s;
    }

55  forall ( i , j in Tasks : i < j) {
        if (i + G > j) {
            init_y_pred:
                y[i][j] >= y[i][j - 1];
            overlap_down:
                t[j] - t[i] <= L + y[i][j] * M;
60          overlap_up:
                t[j] - t[i] >= s * y[i][j];
        }X
    }
65 }

execute DISPLAY {
    writeln("t = ", t);
    writeln("y = ", y);
70  writeln("Optimal = ", t[n]+a+L+b);
}
```