Treball de Fi de Màster

**Master's Degree in Automatic Control and Robotics (MUAR)**

# General Solution for the Multistage Scheduling Problem with Time and Resource Constraints

# MEMÒRIA


**Autor:** Aitor Val Balmaña

**Director:** Dr. Vicenç Puig Cayuela


**Convocatòria:** June 2020



Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona


# Contents

## List of Figures

# 1 Introduction

## 1.1 Motivation

Production planning and scheduling is a crucial part of any manufacturer, and good practises in this field are a key aspect for the overall well-functioning of any industry. In the recent years, there has been some research on how to handle this problem more efficiently by means of optimization tools. In particular, the proposed solution algorithms were based on mixed integer linear programming frameworks. But all the existing methods have one drawback, they offer a very specific solution for a specific plant or they fall under the trap of being too academic and abstract, and therefore, the proposed solutions are complex and cannot be applied.

A lot of commercial optimizers have been developed to solve the scheduling problem. The complexity of suiting all the different plants is daunting, and usually they offer personalized customization. Of course, this is not cheap. The rise of the open source philosophy has provided a lot of useful tools that match with the commercial options. This project intents to offer a solution relaying only on open source tools. Moreover, this work form part of a bigger project that intends to develop a platform with industry 4.0 tools working very tightly with each other. The scheduler developed will form part of this bigger platform.

This project has raised mainly with the approach presented in [4]. In this work, it has been proved that a constraint satisfaction framework accompanied by pre-processing is not only possible, but better suited in order to be more adaptable. And, with the rise of IA techniques, new tools on this field can be developed. Therefore, reinforcement learning (an IA technique) will be tested on this particular problem. In conclusion, this work has been motivated by the opportunity to take a modern approach of the problem that promises very good results.

## 1.2 Objectives

The aim is to get the most general possible solution for the multistage scheduling problem using an innovative and modern approach. The goal is to optimize, not only the makespan, but other aspects of the schedule as the production costs. The proposed solution will take the information of the plant (machines, formula for each product, orders, etc...) and give a feasible schedule that minimizes time, cost and other aspects. Moreover, the machines will have maintenance/cleaning needs that must be respected. Orders will have a due date associated that should be fulfilled. And, the most important feature that the program must have is adaptability. It has to be able to describe and solve a large pool of different plants, all of them with its own particularities.

This has to be achieved with an open source constraint satisfaction solver relying heavily on pre- and post-processing along with possible artificial intelligence techniques. The scheduling tool is developed with Python and should be integrated as a service inside an API, this will ease the integration inside the industry 4.0 platform. This way of proceeding adapts the traditional scheduling problem to the new software developing philosophy with modern architecture.

The principal purposes of the project are two. Proving that a new approach on the scheduling problem not only is possible, but better. And finally, exploring possible useful AI techniques applied to the problem.

### 1.3 Thesis structure

This thesis is structured as follows. First, in Section 2, some past works that have helped the development of this project are reviewed. Then, there is a short explanation on how the problem can be described and codified mathematically as a constraint satisfaction problem in Section 3. Then, the main scheduler is broken down into its different parts along with detailed explanations about the most interesting algorithms especially designed for its operation as pre- and post- processing on Section 4. The codification explained in Section 3 is used in this section. Then, the used AI techniques explained in Section 6 are adapted for the current scheduling problem. The inclusions of maintenance tasks in Section 5 are included in the scheduler since their inclusion lead to some adaptations of the general scheduling algorithm. Section 7 discusses the adaptation of the scheduler as a service by implementing it as an API. This section also shows how the scheduler works as a standalone application. Finally, the scheduler is tested on a real milk production plant under different conditions in Section 8. This section also shows how to codify and extract the needed information of a plant in order to use the scheduler. Section 9.2 presents the socio-economic impacts. Finally, Section 10 draws the main conclusions.

## 2 Background

There has been multiple efforts related to solving the scheduling problem and some historical background of this topic is needed for the development of the project. The problem of developing a generic scheduling tool is very complex. This tool not only has to be able to adapt to different production lines but it has to overcome the computational complexity.

On regard of adaptability, there is a lot of human factors and technicalities that have to be taken into account, as e.g., variations of processing times, input of resources requirements, etc. All of these aspects are very well assessed in [6]. So, trough the development of this project, adaptability and easy usage has always been a priority.

Most of the solvers used nowadays for scheduling are based on Mixed Integer Linear Programming (MILP) algorithms that consider 0-1 variables for decision-making. This way of addressing scheduling leads to high dimensionality optimization problems and the computing times are excessive, making the whole implementation very difficult to apply in real cases. In [7], [8] and others, the use of different MILP approaches is presented in severeal applications as e.g. a pharmaceutical plant with due dates constraints and multistage processes. MILP frameworks with decision-making variables can be very robust but its complexity and mathematical approach can make the implementation very difficult for complex problems. And the major weakness is that for each plant it is presented its own different MILP problem, making the solution hardly adaptable. [1] shows methods developed to asses excessive computational times when dealing with big and complex problems.

The appearance of commercial constraint/optimization tools has yield to new simpler ways of dealing with these type of problems. In particular, in [4], they use IBM ILOG Optimization Suite which is a solver specially thought for scheduling purposes that comes from IBM CPLEX. Recently, these type of problems are starting to be studied under constraint/optimization solvers relaying on pre- and post-processing, being a current topic of research. This different framework opens the door to IA techniques and other tools to gain adaptability. On regards of the solver used on this work, it has been chosen to use the open source OR-Tools CP solver from Google [5]. There are studies showing the efficiency of this solver against the most robust one to date (IBM CPLEX) being OR-Tools a solid choice [3].

In the project in hands, some ideas from [1] and [2] have been taken (generative and constructive step) to deal with multistage problems with high dimensionality; and from [4] it has been taken inspiration to codify the problem without decision variables and handling of maintenance tasks.

The scheduler presented uses a pure constraint/optimization solver with high pre- and post-processing as said, in combination with reinforcement learning algorithms. The introduction of reinforcement learning algorithms in this particular type of problem is a current field of research.

# 3 Problem Formulation

## 3.1 Overview

The schedule consists on providing an optimal schedule of tasks in order to fulfill a set of orders. This schedule has to take into account the formula for each product, the specifications of the cleaning of each machine, the availability of each machine and, of course, the orders.

This schedule consists on different tasks that are processed on the machines and needs a certain time to be completed. The products are completed after a certain sequence of these tasks. For example, after three different tasks a certain amount of product is completed. These sequence of tasks is called formula, and each product has, at minimum, one. Although, there may be different ways to make the same product. Every task that comes from a formula has to be scheduled on a specific designated machine. These machines have its own particularities (availability, maintenance/cleaning special tasks, etc...), and usually are associated with different products and formulas making the whole process of scheduling complex to deal with. These problem can be described as a constraint satisfaction framework and it is presented in the following Section. The problem variables can be described through the following items $Fo$ (formulas), $Or$ (orders) and $Ma$ (machine) and they are explained next.

## 3.2 Order formulation

The $Or$ item (1) stores the information of the orders and contains information about the product and its quantity along with the due date

$$Or = (or\_id, ddate, q, pro\_id); \tag{1}$$

where $or\_id$ is the unique order id, $ddate$ is the delivery date, $q$ is the quantity asked and $pro\_id$ is the product id. So, each item $Or$ will describe an order containing an unique product, delivery date and quantity. Notice that the $or\_id$ does not identify an order coming from the same costumer with different products. These types of orders have to be decomposed in a set of orders with an unique $or\_id$ each, and probably, the same $ddate$. An order id can only be associated to a determined product.

## 3.3 Formula formulation

The $Fo$ item (2) stores the information of all the different ways available to produce each product. It contains the precedence of tasks, its connections, the time needed and the cost associated of each task

$$Fo = (fo\_id, pro\_id, step, stp\_pre, ma\_id, type, max\_output, pro\_del, c); \tag{2}$$

where $fo\_id$ is the formula id which must be unique for each entry and $pro\_id$ is the product identifier. $step$ determines a unique process inside the production. $stp\_pre$ is a list of integers

in which the step precedence is defined. The tasks scheduled for this particular $fo\_id$ will be placed after the steps defined on this attribute. $ma\_id$ is the id of the machine on which this step is performed. $type$ defines how the task is performed (explained shortly after). $max\_output$ is the maximum throughput of this particular step on this machine and $pro\_del$ is the time consumed by this machine in order to achive this step. So, each $Fo$ item will be a possible step (can be the only one possible) inside the complete sequence in order to complete the $pro\_id$.

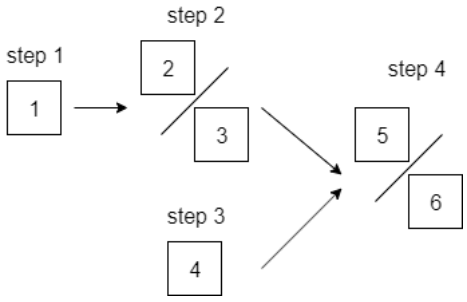
There are three $type$ possibilities:

- Time and cost are affected by the quantity that is produced.
- Time and cost are not affected by the quantity that is produced.
- Task is parametrized as a production ratio (as e.g., kilograms per hour).

The field $type$ is introduced in order to encapsulate different type of production plants and processes. For example in a brewery production plant, the time and costs are not affected by the quantity amount produced in the fermentation step. But, in a car production plant, time and costs are affected. The third type introduced is when the plant has stages that produce a lot of quantity or the product is a gas or liquid and it is more useful to represent the step production as a ratio.

An example of a production process containing all the particularities possible to codify is shown next. This example contains multistage steps (step 2 and 4) and the formula also contains branching. This example illustrates how the relation of different steps can be codified.

| Fo | | | |
|---|---|---|---|
| fo_id | step | stp_pre | ma_id |
| 1 | 1 | [0] | 1 |
| 2 | 2 | [1] | 2 |
| 3 | 2 | [1] | 3 |
| 4 | 3 | [0] | 4 |
| 5 | 4 | [2,3] | 5 |
| 6 | 4 | [2,3] | 6 |




As can be seen, this approach to codify the formulas yields to the ability to solve very complex and different problems.

### 3.4 Machine formulation

The item $Ma$ contains the information needed for each machine.

$$Ma = (ma\_id, cl\_delay, cl\_cost, cl\_every) \tag{3}$$

where $ma\_id$ is the unique machine identification for each machine on the plant, $cl\_delay$ is the time that is required for the cleaning/maintaining of the machine (cip tasks), $cl\_cost$ is the

cost of this process, $cl\_every$ specifies the working time after which a cleaning/maintenance operation is required. Note that cleaning tasks are not periodic and depend on the working time accumulated on each machine. This fact makes this constraint really difficult to respect with a constraint satisfaction solver and is exhaustively discussed in Sections 4 and 5.

### 3.5 Mathematical formulation of the problem

All of this information is considered as an input and processed in order to define a constrained satisfaction problem. The variables for this problem will be the starting time ($T.start$) and ending time ($T.end$) of each task to be scheduled. These two variables create an interval that reduce the dimensionality of the problem, but the declaration of both is very useful to define other needed constraints. In fact, as these two variables creates an interval the following constraint is defined.

$$T.end_i + T.start_i = pro\_del \qquad \forall i \tag{4}$$

where $i$ refers to the whole set of tasks.

The principal constraints that have to be imposed are:

- Precedence: if $j$ is a sequence of tasks that have all the required steps in order to produce a certain amount of product. The condition of precedence is formulated as

$$T.end_i^j \leq T.start_{i+1}^j \qquad \forall i \in j \tag{5}$$

- No Overlap: if $m$ define in which machine the task is scheduled. The condition of no overlapping is formulated as

$$T.end_i^m \leq T.start_{i+1}^m \qquad \forall i \in m \tag{6}$$

- Due Date: if $o$ is the order associated with the task, the due date constraint is:

$$T.end_i^o \leq ddate^o \qquad \forall i \in o \tag{7}$$

- Cip tasks: This is by far the most difficult constraint. And mathematically can be written as:

$$When \sum_{i=k}^{n} (T.end_i^m - T.start_i^m) = cl\_every^m \rightarrow T(cip).start_{n+1}^m = T.end_n^m \quad \forall i \in m \tag{8}$$

  where $k$ is 0 when the schedule is starting and $n + 2$ for the rest of the schedule. The implementation of this constraint has been made with pre- and post-processing heuristics. In this work, different approaches are considered in order to fulfill this constraint, as it will be explained in detail in Section 5.

Due to the architecture of the program, two different objective functions must be defined. First, the most simple one is defined as follows

$$J = max(T.end_i) \qquad \forall i \tag{9}$$

where $max(T.end^i)$ is the makespan of the schedule.

Moreover, in order to take into account the minimization of the production costs and other goals, the following multi-objective function is considered:

$$J = min(w_1 max(T.end_i) + w_2 \sum_{i=0}^{N} T.cost_i + w_3 n(Ma) + w_4 \sum_{i=0}^{N} (T.end_i - ddate_i) \qquad \forall i \tag{10}$$

The second term references the minimization of the production cost and $w_2$ is the corresponding weight. On the other hand, the third and fourth objectives with their weights associated $w_3$ and $w_4$ are respectively related to the minimisation of the number of machines used and the satisfaction of the due date. In particular, the term associated with $w_3$ aims to achieve a schedule that requires less machines ($n(Ma)$ is the number of machines implicated), which means normally less resources needed. The cost of starting up the machine, the resources to maintain the machine idle or the personnel, are not reflected on the total cost, as the cost is linked to the particular tasks. Finally, the last term is introduced to penalize the solutions that does not fulfill the due dates. Notice that a solution that does not respect the due dates can still be the best one because it improves other aspects of the schedule, this degree of freedom is controlled with the term $w_4$.

The distinction of these previous two objective functions is very important for the scheduling algorithm. The used constraint satisfaction solver is only able to take into account Eq.9, but in the reinforcement learning phase Eq.10 is considered. This approach makes the implementation faster as the problem to be solved by the constraint satisfaction solver is simpler. There are other aspects of the architecture that yield to this decision and are detailed in the following.

# 4 Scheduler

## 4.1 Introduction

The proposed scheduler relies heavily on pre-processing of the data and post-processing of the solution to minimize complexity and provide flexibility. The scheme of the main scheduler process is represented graphically in Figure 1.

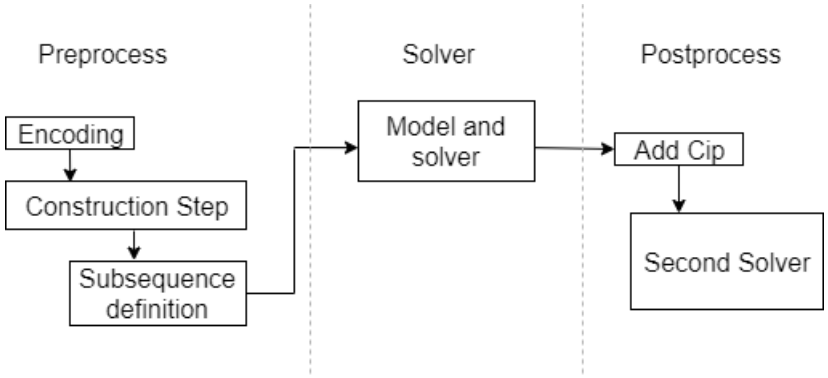



Figure 1: Flowchart of the main scheduler process

The solver used is an open source constraint satisfaction solver of OR-Tools from Google [5]. The use of a constraint satisfaction solver has its advantages and limitations. The advantages are the concise manner of defining the problem and the computational efficiency. The limitation is the difficulty to write the particularities of each production process, and for that, the pre- and post-processing are important. The principal things that need to be assessed with pre- and post-processing are:

- Multistage steps: the solver does not know which alternative will result in a better solution. This is decided on the constructive step with heuristics.
- Subsequences: The solver neither knows which tasks have to be preceded by others. This is a difficult key aspect of the scheduler as it has to take into account the step precedence defined on each formula and the quantity requirements. Once the precedence is defined, this information is passed to the solver as another constraint (see Eq.(5)).
- Cleaning/maintenance tasks (cip tasks): The solver is not able to schedule the cleaning tasks with the particular constraint associated and they have to be treated differently. After adding the cip tasks, another constraint satisfaction problem has to be solved (same problem but with different constraints). All of this processing is discussed later in this section.

## 4.2 Encoding

The encoding part consists of the declaration of the tasks that are needed to fulfill the orders. The defined set of tasks has to include task alternatives for the multistage processes. So, the set of tasks produced will not only have the required tasks, but all the possible alternatives for

those tasks too. The information stored in $Or$, $Fo$ and $Ma$ will be used to obtain these tasks. Each task will have the following information:

- task_id: Each task will be associated with an id. This is useful to characterize tasks that are equivalent and come from a multistage process, identifying them with the same task_id.
- cip: integer variable with 0 or 1. This number will identify the cip tasks with a 1.
- fo_id: From which formula identifier comes this task.
- order_id: id of the order that this task fulfill.
- ma_id: id of the machine where the task has to be performed.
- quantity: quantity of product made.
- cost: particular cost of this task. This will be affected by the type of formula.
- step: stage of the sequence for the product.
- start: starting time of the task.
- end: ending time of the task.
- dt: duration of the task. This will be affected by the type of formula that comes from.
- delivery_date: delivery date of the order that this task is associated with.

$Or$ , $Fo$ and $Ma$ come from a database or a JSON file. This database is read and it is transformed into a list of tasks as indicated in Algorithm 1.

**Algorithm 1** Encode T_single and multinfo list

**for all** $Or$ **do**
  delivery_date = Calculus of the delivery date (ddate)
  **for all** $Fo = pro\_id$ **do**
    Multistage detection
    **if** Multistage detection = False **then**
      Codify $T$ from $Fo$
      Add $T$ to T_single list
    **end if**
    **if** Multistage detection = True **then**
      Store the needed $fo\_id$ in $multinfo$
    **end if**
  **end for**
**end for**
**return** T_single, $multinfo$

Note that we are separating the multistage tasks and the single tasks. The single tasks are created directly but only the information of the multistage steps are stored in $multinfo$. Inside the list $multinfo$ is stored the $fo\_id$ along with the quantity, $or\_id$ and the step of these tasks. This distinction will be very useful because there are multiple ways to define the different alternatives that a multistage step has and is advisable to separate the creation of the two lists in two different algorithms. The calculus of the delivery date is simply the transformation of the variable $ddate$ that is a date into an integer taking into account the shifts duration and the actual date.

It is important to note that the tasks are created trying always to reach the max throughput of the machine. For example, the algorithm will create 3 tasks with a quantity of 100, 100 and 50, with an order of 250 in a step with 100 maximum throughput. This fact is really important to take into account because maybe there are plants that this policy does not yield to the optimal solution. But generally, this policy leads to quasi-optimal solutions.

The algorithm to create the multistage tasks takes the list $multinfo$ and creates the T_multi list. For now, it only creates the alternative that the maximum full output formula for a step can be obtained by means of the formulas of the same step (multistage). For example, if a multistage step has two machines with a different output of 100 and 75, the algorithm will create three multistage tasks: one describing the possibility of making 100 units on the first machine and two of 25 and 75 units on the second machine, being the last two equivalent to the first. In conclusion, the maximum possible output inside the multistage step will affect the other tasks. The encoding process for the multistage tasks is presented in Algorithm 2.

**Algorithm 2** Encode T_multi

```
for all multinfo do
  delivery_date = Calculus of the delivery date with or_id stored
  acumqor = 0; quantity = quantity of the order; max_output = max output of the fo_id
  while acumqor ≠ quantity do
    if quantity-acumqor < max_output then
      quantity of the set of tasks = quantity - acumqor
    end if
    for Fo ∈ multinfo do
      Codify set of T with sum(q) = quantity of the set of tasks
      Add set of T to T_multi list
      Update acumqor with quantity of the set of tasks
    end for
  end while
end for
return T_multi
```

T_multi list has a particularity: equivalent tasks will be encoded with the same $task\_id$ as said and recognized between them with an extra parameter called $opt$. With this structure all the different options for the same $task\_id$ have to have the same accumulated quantity. So, recov-

ering the last example, the first task of 100 will have the option 1, and the other two tasks the option 2, all of them having the same $task\_id$.

The advantage of this way of proceeding is that with the information of $multinfo$, it is possible to produce a $T\_multi$ list using a creation algorithm with different criteria and thus making the scheduler more adaptable to different production processes.

### 4.3 Constructive Step

In this part, tasks from the $T\_multi$ are selected in order to form a single list of T. This algorithm follows an heuristic based on selecting the option inside T_multi that will produce the least $horizon$. This $horizon$ is calculated as the maximum working time scheduled for the machines implicated on the different options.

The constructive step is described in Algorithm 3.

**Algorithm 3** Constructive Step

```
Calculate M_Duration with T_single
for each unique task_id ∈ T_multi do
  for all opt do
    new_horizon = Calculus new_horizon (M_Duration)
    if new_horizon < horizon then
      Store best option
      horizon = new_horizon
    end if
  end for
  Add best option to T_single
  Update M_Duration
end for
return T list
```

M_Duration is a list with the working times of each machine with the current tasks selected. Note that this list has to be updated each time a decision is made. Finally, the output of this part will provide a set of tasks that fulfill the orders taking into account that each step has different alternatives. All of these decisions and possibilities are stored and then used by the reinforcement learning step (see Section 6) as that part tries to improve the selection of the multistage tasks presented in this section.

### 4.4 Subsequence definition

Now that the set of tasks is defined, the relation between them needs to be imposed. This information is key for defining the constraint of precedence (12). Two major aspects need to be respected. First the $stp\_pre$ that comes from the $fo\_id$ needs to be taken into account, and secondly, a task from further steps cannot be initiated if the quantity from previous steps is not produced. So, apart from the list of tasks T, the solver will need the information of precedence

that will be stored in the list $Subsequence$. This list will have the next structure:

$$Subsequence = [(pretasks, postasks)] \tag{11}$$

Each element of the $Subsequence$ list will store the information of post-tasks that need to be preceded by pre-tasks. In other words, the solver will transform this information to the following constraint

$$T.end_{pretasks} \leq T.start_{postasks} \tag{12}$$

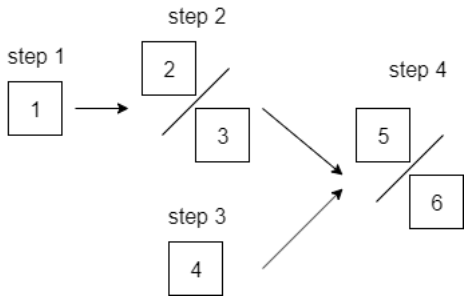
This part may seem trivial but it is not. And, it is a complex key part for the correct operation of the scheduler. The calculus of the subsequence list have to be smart and consistent as a bad subsequence declaration can yield to inefficient schedules. First, we need to extract the information needed for this calculus. For that, a new list is created. This list has three dimensions. The first one is the different steps on the set of tasks. The second dimension corresponds to the tasks itself and the last one is the information of each task. The information needed for this calculus is the set of $t\_id$, its quantity, if it has been pre-preceded or post-preceded using two Boolean values, and the length of this set of tasks. The length is needed because the algorithm can blend two tasks in one and this information is crucial for ensuring the correct operation. Note that this list has to be created from each order.

Now that the important information is stored and sorted. The next step is based on expanding the last list mentioned to have the step precedence requirements. In order to do that, a new dimension is added on top of the others. This new dimension contains the relations between steps. For an easy explanation the example presented in 3.3 is used.

The next table shows the information of the added dimension to this list for this particular example. This new dimension will have the tasks of the steps that have to be related.

| tinfo | |
|---|---|
| new dimension | step |
| [tasks step 1 , tasks step 2] | 2 |
| [tasks step 2 , tasks step 4] | 4 |
| [tasks step 3 , tasks step 4] | 4 |

Table 1: First dimension of tinfo and the step that is referenced.




The algorithm that determines the precedence between tasks has the tinfo list as input and the subsequences defined in Eq. (11) as output. The algorithm can be broken down into two searches inside each new dimension of tinfo, called minus search and plus search. The general scheme of this procedure is shown in Algorithm 4.

**Algorithm 4** Subsequencence definition

```
for all tinfo items do
  c = 0
  while c == 0 do
    Select candidate to be pre-tasks
    Minus search for post-tasks
    if post-tasks not found then
      Plus search (blending of pre-tasks)
    end if
    if No post-tasks available then
      c = 1
    end if
  end while
end for
return subsequence
```

The minus search is an algorithm that searches the most suitable post-task for the pre-tasks candidates. And if no possible subsequence is found the algorithm does a plus search that consist on blending the actual candidates of pre-tasks to other pre-tasks generating new candidates. The selection of the candidates pre-tasks and the different searches have to follow a very well defined priority in order to produce an optimal subsequence.

The selection of the pre-tasks candidate has to follow two criteria. The first priority is selecting the pre-tasks with less tasks blended (length information) and then selecting the pre-tasks with more throughput associated inside the first group.

When searching for the optimal post-tasks (minus search), it is important to note that the algorithm can reach a suitable solution but not the optimal one. That is why inside the minus search there is an extra search trying to match the post-task to the optimal pre-tasks (inverse search). The inverse search is done with the objective of finding the best solution possible and not a feasible one. The scheme of the minus search is shown in Algorithm 5.

When this minus search fails to declare a new subsequence, a plus search is needed. This plus search will simply blend the current pre-tasks with the pre-task that have more quantity independently of its length. The beauty of this algorithm is that it ensures that the task blended does not have an optimal match. And if these two tasks are blended, the resulting set can be more optimally paired than separately. Moreover, this new set will be evaluated in the next minus search if an optimal solution is not found with the sets with less length. Next an example of how a solution looks is shown in Table 2 and 3 .

**Algorithm 5** Minus search

```
Sort post-task with descending throughput
for all post-tasks available do
  result = q(pre-tasks) - q(post-tasks)
  if result == 0 then
    Store subsequence = (pre-tasks,post-tasks)
  end if
  if result > 0 then
    Search better set of pre-tasks
    if Better pre-task found then
      Store subsequence = (pre-tasks,post-tasks) with best pre-tasks found
      break
    end if
    Store post-task; update result
    while result > 0 do
      Expand the set of post-tasks; update result
    end while
    Store subsequence = (pre-tasks,post-tasks)
  end if
  if result < 0 then
    Search better set of pre-tasks(more length than the current set) for the actual post-tasks
    if found then
      Store subsequence = (pre-tasks,post-tasks) with best pre-tasks found
    end if
  end if
end for
return Subsequence
```

Task information with the step precendence

| prestep | poststep |
|---|---|
| [0,20];[1,100];[2,100];[3,100];[4,100];[5,100];[6,100] | [10,20];[11,300];[12,300] |
| [7,300];[8,20];[9,300] | [10,20];[11,300];[12,300] |
| [10,20];[11,300];[12,300] | [13,150];[14,150];[15,20];[16,150];[17,150] |

Table 2: List of tasks with his t_id and quantity, and presented with its step precedence.

| Subsequence | |
|---|---|
| pretask | postask |
| 0 | 10 |
| 1,2,3 | 11 |
| 4,5,6 | 12 |
| 9 | 11 |
| 7 | 12 |
| 8 | 10 |
| 11 | 13 14 |
| 12 | 16 17 |
| 10 | 15 |

Table 3: Result of the subsequence algorithm for the tasks on Table 2

These subsequences have to be constant trough the entire run of the program, as the problem is solved multiple times.

Note that if a task is changed during the reinforcement learning step, it will not affect the subsequence definition. As, by construction, the tasks changed will have same task_id and quantity, but different time, cost and machine.

### 4.5 Model and Solver step

In this this step, the constraint satisfaction problem is defined and solved with the set of tasks and the subsequence information calculated in the previous steps. The solver is the CP-SAT solver from OR TOOLS from Google as previously mentioned [5].

#### 4.5.1 Initialize model. Variable domain declaration

Here the $T.start$ and $T.end$ variables for each task are created. For the regular tasks, these variables will be integers that go from 0 to the horizon. The definition of this horizon is key issue in order to propose a model that the solver is able to know if it is feasible or not. The horizon definition also affects to the performance of the solver due to the fact that a larger horizon means more domain to explore.

This horizon is defined depending on the particular application case. There are tree main scenarios: when there is no due date, when there is due date and when we solve the new problem adding the cip tasks (each one with a different horizon definition):

- **No due date**. The horizon is set as the sum of all the time required to do all the tasks as if all the tasks where performed sequentially. This option is important because maybe the due dates makes the problem unfeasible and some of them have to be deleted.

$$Horizon = \sum_{i=0}^{N} T.dt \tag{13}$$

- **Due date required**. The horizon is set as the most far due date of the set of orders.

$$Horizon = max(T.delivery\_date_i) \quad \forall i \tag{14}$$

- **Adding cip**. The horizon is set as the previous makespan plus the maximum possible delay added due to cip tasks:

$$Horizon = Makespan + \frac{Makespan}{min(Ma.cl\_every_i)} \cdot max(Ma.cl\_delay_i) \quad \forall i \tag{15}$$

### 4.5.2 Constraint definition

The constraints presented in Section 3.3 are imposed. Here, the solver tries to fulfill all the constraints; but, after some critical time, or, if the scheduler is not able to find a solution (unfeasible problem), the constraint of the due date is dropped and the solver tries to find a optimal solution without the due date constraints. The elimination of the due date constraint takes the assumption that every problem is feasible with an infinite horizon or, in our case, if every task is programmed one after the other according to Eq. (13). This decision is made under the philosophy that it is better to provide some feasible schedule than not providing any.

The constraints of precedence and no overlap (Eqs. (5) and (6)) are straightforward to declare but the other two have some difficulties and cannot be implemented right away. The cip constraint is imposed after a first solution (as explained in Section 4.1), but here a important problem arises: the addition of cip tasks will directly affect the other constraints and the whole schedule. The first two constraints will be solved imposing them to the second scheduling problem (precedence and no overlap). But, the due date constraint **must** be declared on the first schedule, thus it cannot be ensured on the final solution. This is why the due date constrain 7 has to be modified as follows

$$T.end_i^o \leq ddate^o - wt_{cip}^m \qquad \forall i \in o \quad \text{that finishes a product} \wedge \forall m \in o \tag{16}$$

where $o$ are the different orders and $m$ the different machines, and $wt_{cipp}$ is the working time introduced by the cip tasks. The calculation of the term $wt_{cip}$ is not an easy task because it is impossible to know the exact time a particular task will be delayed by cip tasks before the schedule is calculated. Thus the approach followed to consider this issue is the following.

For each final task (task that completes a certain amount of product), it is computed the necessary machines implicated with the associated $wt\_cip^m$ of each machine, this includes all the machines that precedent the final tasks. This $wt\_cip^m$ for each machine is computed with:

$$wt_{cip}^m = \frac{sum(T.dt^m)}{Ma.cl\_every^m} \tag{17}$$

Initially, it was thought that the sum of this $wt\_cip$ should be enough as the final task will be affected by all the delays introduced by the precedent steps. But, the results obtained were

really bad. The correction introduced to the due date constraint was too big. So, a refining of this correction is necessary.

Two different scenarios have to be kept in mind: the multistage steps and the branches of the production. The introduction of cip tasks to this two cases do not affect equally to the schedule as a purely sequential formula. The policy considered is as follows: when a multitask step is detected, the machine with more $wt\_accumulated = wt\_tasks + wt\_cip$ is considered and the $wt\_cip$ is updated only with the $wt\_cip$ of the machine selected. This policy is well suited because it only takes into account the branch or the multistage tasks that will take longer to accomplish. In other words, the branch or the multistage task that requires less time to accomplish will not delay the final step.

This calculus of the correction does not consider the superposition of the cip tasks that may occur and effect this correction. This effect is difficult to predict and affects positively to the schedule as a superposition of cip tasks will make the schedule shorter. Consequently, this effect is not considered for the correction. The algorithm to implement this correction is not shown as it is tedious. It is mainly a lot of list searching and list preparation with the only interesting part of the $wt\_accumulated$ policy explained above.

### 4.6 Solve model

CP-SAT solver returns the most optimal sequence found from all the set of feasible solutions. If the optimal solution is found, the scheduler stops before the time limit. Note that the simple objective function (Eq.9) is defined, not the complete one, minimizing only the makespan as the tasks has already been selected.

### 4.7 Add Cip tasks and second problem solved

After all the schedule is done, the cip tasks need to be added. In order to do that, first we add cip tasks to the actual schedule respecting the specific requirements of each machine. The procedure is simply following the constraint 8 and adding the cip tasks accordingly. But, it is really important to note that maybe, this modification of the initial schedule can break the other constraints. The due date constraint is already safe (as explained in Section 4.5.2) but the precedence inside a subsequence 5 is not. If a machine of the further steps is not affected by cip tasks, but the initial step is heavily affected, the tasks on the further steps may start before the tasks on the initial steps end. This is the reason that another simpler problem needs to be solved imposing again some constraints and only the addition of cip tasks is not a good choice.

This new problem will take the information of precedence between tasks inside the schedule of a machine from the current solution. This information is formulated as the following new constraint

$$T.end_i^m \leq T.start_{i+1}^m \qquad \forall i \in m \tag{18}$$

This new problem is far more simpler because the solver only have to move some tasks in order to ensure the constraint of precedence in Eq. (5). But, it does not have to specify in which order

the tasks have to be performed, or consider the due dates, as all of these constraints have already been considered and solved. Here, a key aspect of the method is shown.

The appearance of cip tasks makes the solution not proven optimal with this method, so even though the scheduler is able to provide a feasible solution, it is impossible to know if it is the optimal one. Maybe, the first solution is optimal and the second is optimal too, but maybe the final solution it is not, as the two solutions try to optimize the same but under two different set of constraints. Adding the different heuristics explained, makes the solution not proven optimal.

# 5 Different approaches for the cip tasks management

The way of handling the cip tasks has not been the same along the development. First, the cip tasks were implemented with the same method as [4]. But after some changes, a more robust solution was found. The final solution proposed is fully explained in Section 4 and forms part of the main scheduler. In this section, it is explained how the method of [4] to handle cip tasks is applied as well as the further refinement proposed.

## 5.1 Alternative approach

This approach consists on defining the cip tasks in the pre-process step. These cip tasks are defined as described in Algorithm 6. It is applied just after the constructive step.

**Algorithm 6** Add cip tasks

```
Calculate M_Duration with T list
for all machine ∈ M_duration do
  ncip = ceil(working_time / clean_every)
  for k in ncip do
    T.start = (k + 1) * clean_every
    Add k cip tasks in T list
  end for
end for
return T list
```

So, the cip tasks will be managed as normal tasks but with different constraints. The cip tasks are forced to be periodic with some relaxation. As said, when defining the variables of the model, $T.start$ and $T.end$ of the cip tasks are defined within this domain:

$$T.start \in [T.start - \epsilon_{min}; T.start + \epsilon_{max}] \tag{19}$$

$$T.end \in [T.end - \epsilon_{min}; T.end + \epsilon_{max}] \tag{20}$$

where $\epsilon_{min}$ and $\epsilon_{max}$ are relaxation parameters in order to let the solver more freedom. These parameters are demonstrated to work properly when they are of the order of the most time consuming task scheduled for each machine. These parameters are part of the minimization function, because when near to zero the constraint in Eq. (8) is almost satisfied.

With this method constraint Eq. (8) is not fully ensured and further processing needs to be done. That is the reason an extra cleaning step is proposed to refine this approach.

## 5.2 Cleaning the solution and new problem solved.

After the first solution, there are cip tasks that are not needed and some filtering need to be applied. This filtering is done as follows:

1. Eliminate the consecutive cip tasks of the schedule without tasks in between.
2. Add a cip tasks at the end of the schedule.
3. Check for the constraint in Eq. (8) deleting all the cip tasks not needed.

The last step for cleaning the solution is done with a sliding window. Every time a cip task is found, it counts the working time of the tasks before and after the next cip task. If the working time is less or equal than the particular $clean\_every$, the middle cip task is deleted, and the window expands to the next cip task. If the cip task is not deleted, the next window starts from the middle cip task found. Figure 2 and 3 illustrate when the algorithm has to eliminate a cip task or when the cip task cannot be deleted.

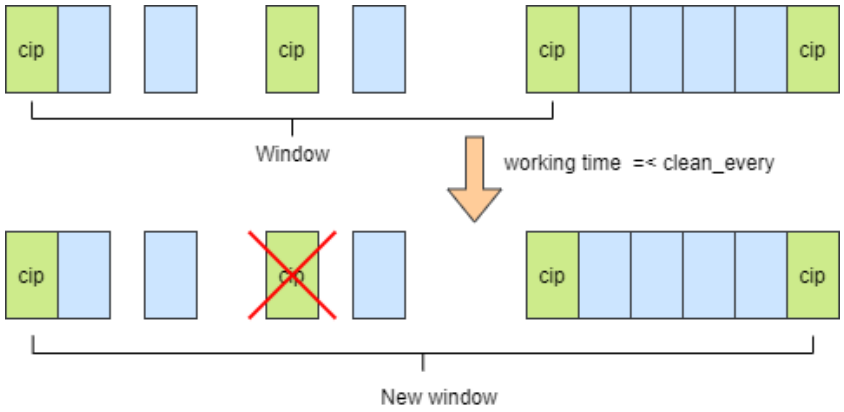



Figure 2: Graphic representation of how the algorithm for cleaning the solution works when deleting a cip task.

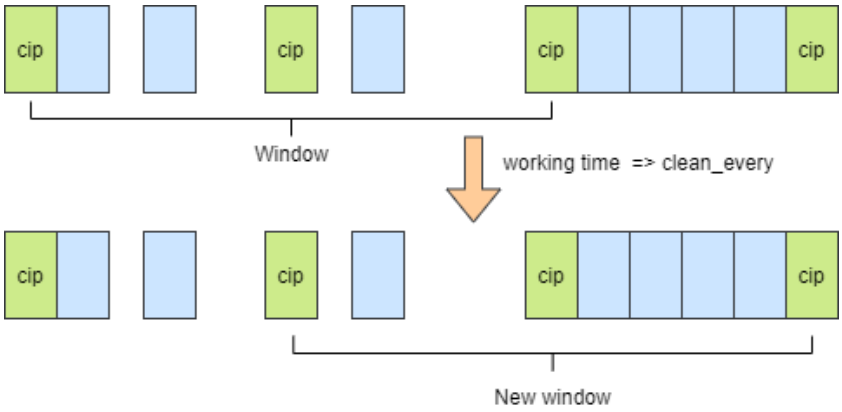
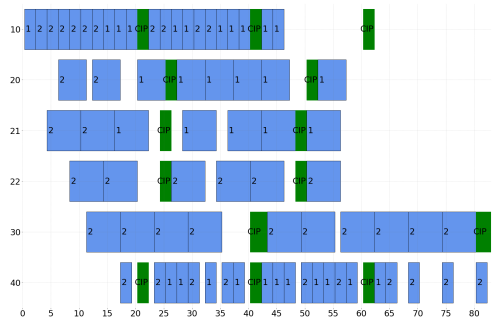



Figure 3: Graphic representation of how the algorithm for cleaning the solution works when a cip task is not deleted.

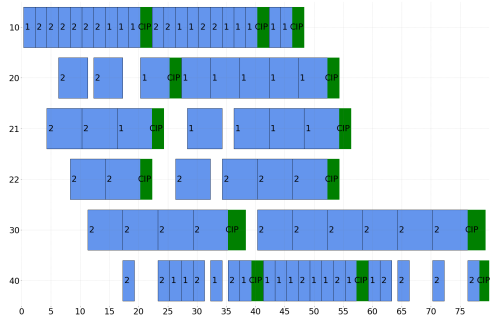
Now that the solution has been cleaned, there will be some idle time in the schedule due to the deletion of tasks and a new problem has to be solved because it will not be the best schedule possible. This new problem will have the same constraints as before using the same subsequences

and adding a new constraint, the constraint of precedence inside a machine introduced in Eq. (18).

In order to make the comparison between the two methods, two similar problems have been solved with this approach and the main scheduler. For further understanding of the two methods, Figure 4 and Figure 5 shows the first solution and the final solution. Keep in mind that these two problems solved are not entirely the same.

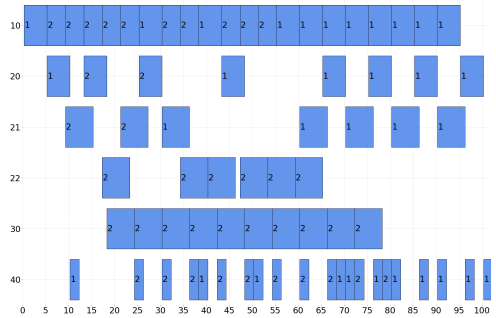


(a) First solution before cleaning not required cip tasks

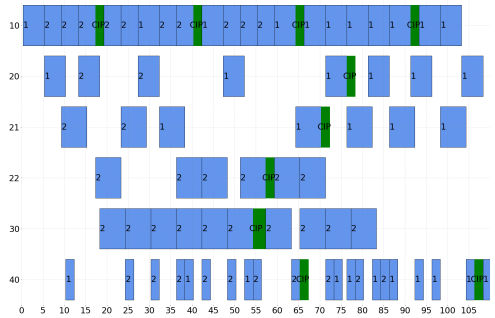


(b) Final Solution

Figure 4: Solution with the alternative method



(a) First solution without cip tasks



(b) Final Solution

Figure 5: Solution with the main schedule method

## 5.3 Benchmarking of the two approaches

In order to justify the benefits of the alternative method proposed for the cip tasks, a benchmark has been done.

### 5.3.1 Testing Scenario

Benchmarking has been made with two different problems, the first one consists on a simple to solve problem where the relations of the time delay of the formulas and the periodicity of the cip tasks are multiples, and on the second one, this relation is not satisfied. Both, cost and maximum output, are set to one for simplicity. The formulas are unidirectional meaning that the step is preceded only by the step before, there is no branching. The machines and the formula information for both problems are presented in Tables 5-9.

Simple Problem

| Machines | | | | Formulas | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ma_id | cl_delay | cl_cost | cl_every | fo_id | stp | pro_del | max_out | c | ma_id | pro_id |
| 10 | 2 | 3 | 20 | 111 | 1 | 2 | 1 | 1 | 10 | 11 |
| 20 | 2 | 2 | 25 | 112 | 2 | 5 | 1 | 1 | 20 | 11 |
| 21 | 2 | 2 | 24 | 113 | 2 | 6 | 1 | 1 | 21 | 11 |
| 22 | 2 | 2 | 24 | 114 | 3 | 2 | 1 | 1 | 40 | 11 |
| 30 | 4 | 5 | 40 | 121 | 1 | 2 | 1 | 1 | 10 | 12 |
| 40 | 2 | 4 | 20 | 122 | 2 | 5 | 1 | 1 | 20 | 12 |
| | | | | 123 | 2 | 6 | 1 | 1 | 21 | 12 |
| | | | | 124 | 2 | 6 | 1 | 1 | 22 | 12 |
| | | | | 125 | 3 | 5 | 1 | 1 | 30 | 12 |
| | | | | 126 | 4 | 2 | 1 | 1 | 40 | 12 |

Table 5: Machine information

Table 6: Formula information

Complex Problem

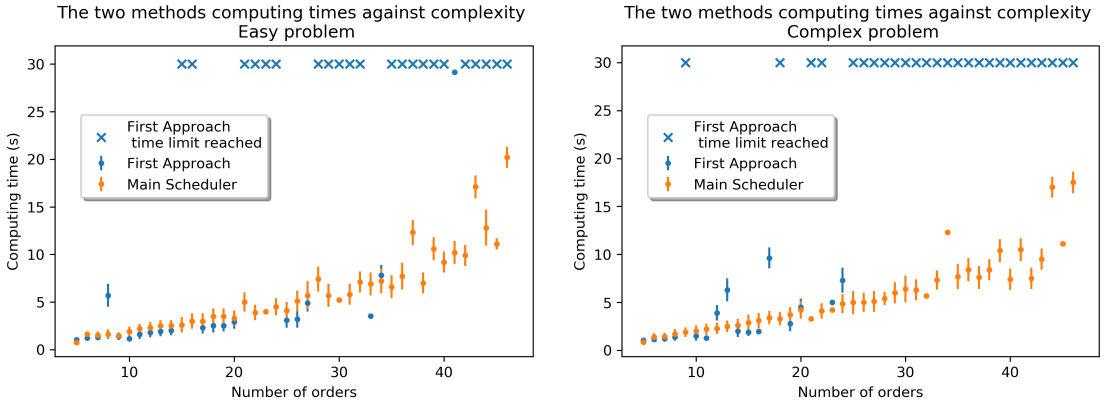
| Machines | | | | Formulas | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ma_id | cl_delay | cl_cost | cl_every | fo_id | stp | pro_del | max_out | c | ma_id | pro_id |
| 10 | 2 | 3 | 17 | 111 | 1 | 3 | 1 | 1 | 10 | 11 |
| 20 | 2 | 2 | 23 | 112 | 2 | 5 | 1 | 1 | 20 | 11 |
| 21 | 2 | 2 | 21 | 113 | 2 | 5 | 1 | 1 | 21 | 11 |
| 22 | 2 | 2 | 21 | 114 | 3 | 3 | 1 | 1 | 40 | 11 |
| 30 | 4 | 5 | 32 | 121 | 1 | 3 | 1 | 1 | 10 | 12 |
| 40 | 2 | 4 | 21 | 122 | 2 | 5 | 1 | 1 | 20 | 12 |
| | | | | 123 | 2 | 5 | 1 | 1 | 21 | 12 |
| | | | | 124 | 2 | 6 | 1 | 1 | 22 | 12 |
| | | | | 125 | 3 | 7 | 1 | 1 | 30 | 12 |
| | | | | 126 | 4 | 3 | 1 | 1 | 40 | 12 |

Table 8: Machine information

Table 9: Formula information

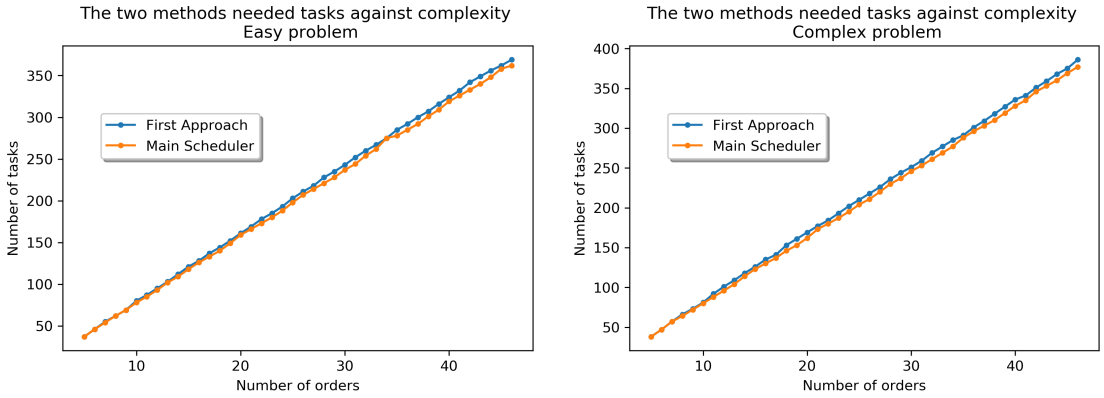
### 5.3.2 Results

To compare both methods, two performance indices have been analyzed: computing time and tasks needed to fulfill the schedule. Both indices are calculated with different number of orders. Precisely, one order means one order per each product, 11 and 12. For the computing time, for each number of orders, the problem has been solved ten times. The results are shown in Fig. 6 and Fig. 7. The vertical lines on Fig. 6 are the standard deviation and the data represented is the mean of these ten runs of the scheduler.




(a) Computing times for the simple problem. (b) Computing times for the complex problem.

Figure 6: Computing times for both methods and problems.




(a) Tasks needed for the simple problem. (b) Tasks needed for the complex problem.

Figure 7: Tasks needed for both methods and problems.

## 5.4 Conclusion

As shown in the obtained results, the main scheduler approach is more robust than the alternative approach in Fig.6. Even though the alternative method sometimes presents lower computing times, it is not able to ensure an optimal solution and presents inconsistencies. This means that this method needs a time bound and it would make all the implementation fully dependent on this bounding. When analyzing Fig.7, the results are the expected ones. For construction, the main scheduler present a solution with less tasks, as the cip tasks are scheduled just when needed and not before, as can be the case of the alternative approach.

# 6 Improving the scheduler with reinforcement learning

## 6.1 Why reinforcement learning?

So far, the scheduler only tries to minimize the time makespan of the schedule while the goal is much more ambitious. The schedule has to be able to optimize other objectives as previously discussed. Including this more complex optimization in the main scheduler will be useless and that is why the simple objective function (makespan minimization only) is used. In the constructive step, the tasks that will fulfill the requirements are decided using a simple heuristic and maybe, this heuristic is not leading to the best solution. This heuristic cannot adapt to the particularities of each production process and this shows the necessity of another layer. The total cost will be the sum of all the tasks costs passed to the solver. The due dates requirements are set as a constraint, and the machine usage is not an aspect that the solver can control either. These aspects of the solution are decided on the pre-processing and the core solver cannot do anything to optimize them.

The strategy followed to address these problems is an optimization algorithm with momentum and memory, based on reinforced learning. On this step, the objective function to minimize is the most complete one: Eq.(10) along with other criteria in order to improve the algorithm.

It is important to note that this algorithm has been built from scratch as this problem has its own particularities that with an open source tool will be difficult to encapsulate.

## 6.2 Reinforcement learning overview

Reinforcement learning is an AI technique based on deciding an action maximizing a reward that is updated after each iteration. The algorithm explores different actions to a particular situation, updating the rewards of each action based on the observed outcome. Normally this update is done after some explorations (called batch). For example, in exploratory robotics, a robot is able to get out of a maze with this kind of algorithm. The current situation may be the current position and the rewards are based on the past outcomes when turning left, right or going straight (actions). This example illustrates that the rewards are not calculated and updated with the direct outcome, they are affected by future outcomes too. At the end, the robot will know which action has to take on each situation in order to get out of the maze.

There are multiple different reinforcement learning algorithms as Monte Carlo, Markov Decision problems or Temporal Difference methods among others, and each one of them with different design choices. For this work, it has been chosen to pick the core idea of reinforcement learning with two other concepts: momentum and memory. The concept of memory is very straightforward to explain: the actions made on a current situation are stored in order to not explore them again. And the momentum is a concept that comes from optimization algorithms. Momentum is a way to avoid local minima. Maybe there are no better solutions encountered with the current situations but after some actions, a better solution is found.

### 6.3 Implementation of reinforcement learning on the scheduling problem

First, let resume the state of the problem. A feasible solution is already known. This solution has the multitask decision linked (construction step), but this solution has been elaborated based on a very specific heuristic (Section 4.3) that cannot include the particularities of each different problem and, as a consequence, the solution cannot be ensured optimal. In order to assess this fact and get closer to the optimal solution, a reinforcement learning method is applied.

The main strategy is the following: Change one selection done on the constructive step and solve the new problem with the new different set of tasks. Here, the current state is the actual best solution found, the actions are the changes applied to the current solution. And the rewards are linked to the $fo\_id$ that can be changed.

As it is an optimization problem, each outcome (solution) encountered has to be evaluated. The evaluation is directly the objective function Eq.(10) and, after some changes to the initial solution (batch), the best solution is picked and the rewards are updated. These rewards are used to choose the changes applied in order to morph the solution on the next iteration and are linked to the $fo\_id$ of the task changed.

Each solution found is evaluated using the multi-objective function

$$Score = w_1 * \frac{new\_horizon}{old\_horizon} + w_2 * \frac{new\_cost}{old\_cost} + w_3 * \frac{new\_numma}{old\_numma} + w_4 * \frac{new\_hoursdelayed}{old\_hoursdelayed} \quad (21)$$

This way of evaluating each sequence normalizes to one the score, referenced with the current best solution. So, a better solution is found when this score (21) is less than one. This score is directly linked to the objective function (10).

The rewards are linked to the formula_id. If the sequence obtained has a score less than 1, it means that the new formula that has been selected has a higher probability to give a better schedule in the next iteration, so this change has to be prioritized. The rewards of each formula_id are defined as (1-score) and they are accumulative. So, after a refresh of the rewards, those $fo\_id$ that have positive rewards are prioritized when choosing the changes to the current solution to test.

For generating the new nodes to explore (batch), the possible changes are sorted according to its reward (highest first) and chosen following a half normal distribution. This has been made in order to not explore always the best rewards of the list, as this can yield to a bias algorithm or unexplored possibilities. The changes with less reward have a lower probability to being picked, but the probability is not zero.

The search is discrete and it has a finite but large pool of possible set of tasks to choose from. The nodes explored are stored inside a **memory**. This memory ensure that no node is solved twice as the solving of a node can spend some time depending on the dimensionality of the problem.

Another key aspect is **momentum**. It can happen that inside the batch generated, no better solution is found, but the current solution is not the optimal one (local minima). For that, the algorithm keeps searching from the new alternatives found even though they are not better. It is very frequent that more than one permutation is needed in order to see an improvement on the schedule. The wholes procedure is summarized in Algorithm 7.

**Algorithm 7** Reinforcement learning step

```
Initialize rewards to 0
while Not time exceeded do
  Update new reference points for the score with current solution
  Select nodes to evaluate(maxnodes)
  for all nodes do
    Solve node
    Evaluate node
    Store Score
    Store node to memory
  end for
  if All Scores >=1 then
    if no solution found count < momentum then
      Store best node as next starting point
    end if
    if no solution found count >= momentum then
      Reset past best solution found as next starting point
    end if
  end if
  if Any Score < 1 then
    Store best node as best solution
    Store the sequence found with this node
  end if
  Update rewards
end while
return sequence
```

The hyperparameters of this reinforcement learning algorithm are the size of the batch (maxnodes), the momentum and the time constraint. As there is no method known, apart from exploring all solutions, to ensure if the best solution found is the best one, the algorithm have to be restraint by time as generally there is too many permutations for explore them all.

The implementation of the reinforcement learning phase ensure an efficient tool to explore better solutions.

# 7 Program interface: Scheduler as a service

The intention of this work is that the scheduler should be the most accessible and easy to use as possible. First, the architecture of the input/output of the program will be described. Then, how the program works as a standalone application and as a service will be presented.

## 7.1 Comunication with the program

The input/output communication with the scheduler is through JSON files. The scheduler is able to pick all the information of a single JSON file, solve it and store all the sequence in an other one, and then send it to the user. This scheme makes the program very versatile.

Inside the JSON file there are five objects: the formulas, the machines, the orders, the machine states and the problem info. The first three are lists with the items following the problem encoding explained in Section 3 that will be illustrated later on. The machine states is an object with an entry for each machine identifier. This entries are 0 or 1 meaning if they are available or not. The last object is the problem information and refers on how the user wants the scheduler to behave. The entries of this info object are:

- cip tasks: if the user wants that the scheduler takes into account the cip tasks: yes (one) or no (zero).
- weights: these are the weights described in Section 6 that can be selected by the user.
- maximum time: maximum time permitted to the solver to search the best solution.
- maximum time for the solver: here it is determined the maximum time that the solver dedicates to each solution.
- explorations: solutions explored until a new branch is selected and the rewards are updated.

Next, an example of a JSON file in the input format with only one entry and element of each list is shown:

```
{     "problem_info": {
        "cipp_tasks": 1, "weight_time": 0.5, "weight_cost": 0.5, "weight_machines" : 0.5,
        "max_total_time": 60, "max_time_solver": 15, "explorations": 10 },
    "formulas": [ {
        "formula_id": 1, "product_id": 1, "step": 1, "stp_pre": [0], "machine_id": 1, "type": 1,
        "max_output": 50, "processing_delay": 4, "cost": 2 } ]
   "machines": [ {
        "machine_id":1, "clean_delay":2, "clean_cost":3.0, "clean_every":8 } ],
    "machines_states": {
        "available":{ "1":1 } },
    "orders": [ {
        "order_id":1, "delivery_date":"2020-01-20", "quantity":400, "product_id":1 } ]          }
```

The output of the scheduler is obtained as another JSON file. This output file has all the tasks that produce the best schedule found with all the associated information. The solution is a list. An example of an output file with three task is shown next:

[ {"tid":0,"cip":0,"oid":1,"foid":1,"mid":1,"q":50,"c":2,"stp":1,"st":0,"end":4 },
{"tid":1,"cip":0,"oid":1,"foid":1,"mid":1,"q":50,"c":2,"stp":1,"st":4,"end":8 },
{"tid":92,"cip":1,"oid":92,"foid":0,"mid":1,"q":1,"c":3,"stp":1,"st":8,"end":10 } ]

## 7.2 Standalone application

The scheduler can work as a standalone application in Python. All the program has been developed under an object design point of view. The scheduler is an object with different calls that the user can use. The principal ones are three. First, the scheduler itself. This call takes as an input the JSON file and resolves the problem under the conditions specified.

The other two calls are output methods. One method shows the schedule as a graph. This graph has been developed with matplotlib and it is the graph shown through this thesis. The other call gives the option to print the solution as a JSON file object as explained in Section 7.1.

## 7.3 Service

This project has been developed within a bigger project as said, with the purpose to develop industry 4.0 tools. These tools will be all inside the same platform. The platform will run inside a server and the user will make requests to these services. For this case, one service is the scheduler. For the platform, all the JSON files will be obtained from a unified database. The scheme of the platform and how it interacts with the scheduler is shown in Fig.8.

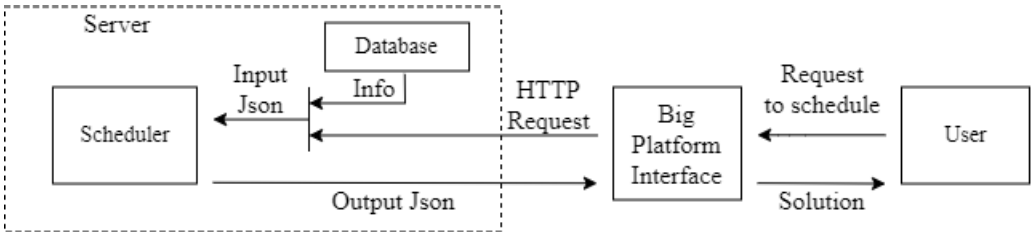



Figure 8: Scheme of the bigger platform.

The scheduler has been implemented as an API using the Flask library of Python. This approach makes the program run with a simple HTTP call with a single JSON file as input and output. For now, the HTTP request contains the JSON itself. The bigger platform has to be able to transform this request with the information stored on a database into the JSON file. This implementation permits the scheduler be part of a bigger platform.

# 8 Real Application

The scheduler has been tested on a real case study. First, a detailed explanation on how the production works in the real case is provided. Then, the results are shown along with a test of the useful functionalities that are available.

## 8.1 Description

The testing plant is a food chain plant related with milk type products. The plant is able to produce three types of milk powders, yogurt and two types of quark.

This particular plant takes milk concentrate and through evaporation and drying, it obtains the desired products. Yogurt and quark have to pass through a process of pasteurizing and fermentation before the drying process can be applied. The different steps required along with its associated machines for the different products can be seen in Figs.9 and 10.

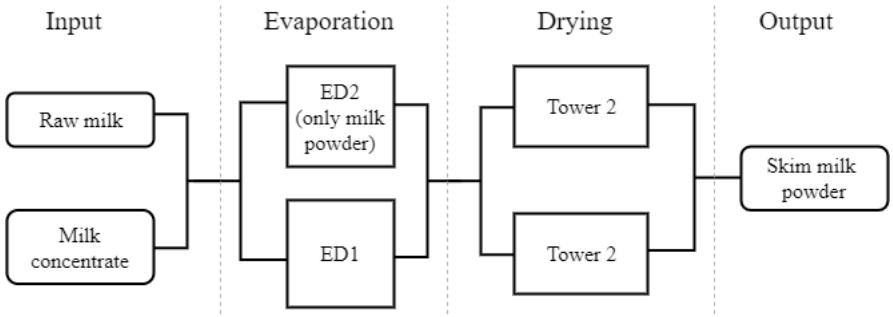



Figure 9: Production sequence for milk powder.

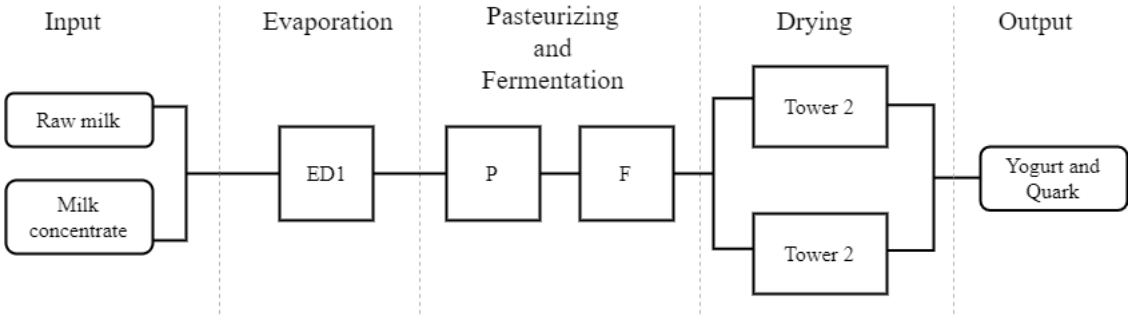



Figure 10: Production sequence for yogurt and quark.

For the evaporation, two evaporators are available: ED1 and ED2, each of them with different throughput capacity and ED2 can only be used to produce milk powder. Pasteurizing and fermentation processes are simply time steps. The product is placed into a sterile place with optimal temperature and humidity for a determined period of time. Finally, all the products pass through a drying process which is made at two towers. The difference of these two towers is the capacity. Even though the two towers can make any kind of product, Tower 2 is kept specially for powder products due to his high capacity. Moreover, there is a type of quark that does not need to be evaporated because the input resources are already suited.

## 8.2 Codification

The milk powder production process has been codified as explained in Section 3 in Tables 12, 14 and 15, with the Ma, Fo and Or lists respectively. Next, it is explained in detail how to codify it with real data.

The machine costs provided are only for ED1 and ED2, this is because the other machines does not have a significant cost compared with this two machines. The production costs are associated to electrical, thermal energy, and water consumption in Table 10.

| | Electrical energy | | Thermal energy | | Water consumption |
|---|---|---|---|---|---|
| | Prod. [kWh] | CIP [kWh] | Prod. [kWh] | CIP [kWh] | CIP [to/h] |
| ED1 | 91.2 | 106.7 | 345.6 | 551.5 | 4.2 |
| ED2 | 72.7 | 81.5 | 2063.2 | 1274.0 | 8.9 |

Table 10: Consumption of the machine ED1 and ED2

The throughput are ratios that vary for each machine and product, this means that all the formulas will be type three (see Section 3) as they are expressed as ratios. For the fermentation and pasteurizing steps no throughput is given. So, to codify these steps, a large enough number is chosen for the $max\_output$ field. This number has to be at least the maximum quantity of an order. For this case 20000 is set. As the output does not affect time, the type is set to two. This will help the scheduler to put all this step for a given order on a single task. If the scheduler puts two tasks because the step does not have enough throughput, the time required will be doubled due to its type and affect the schedule unrealistically.

For the codification of $step$ and $stp\_pre$ Figures 9 and 10 are very useful. As this production line does not have any branches, all the lists of $stp\_pre$ contain only one element. Remember that all the multistep phases (Tower 1 and Tower 2 for example) must have the same number.

The costs are not that straightforward. Here a problem arises, how each type of energy should be treated. For this case, the mean economic cost of each energy is taken, but it can be done very differently. The mean costs have been calculated for Spain for the electrical and water consumption, and for the case of thermal energy it has been calculated the ratio between the natural gas quantity to energy and then changed to euros based on the mean value of the last two years. The costs in euros with these procedures are presented in Table 11.

| | Electrical energy | | Thermal energy | | Water consumption |
|---|---|---|---|---|---|
| | Prod. [€/h] | CIP [€] | Prod. [€/h] | CIP [€] | CIP [€] |
| ED1 | 7.27 | 34.22 | 2.36 | 15.16 | 30.3 |
| ED2 | 5.82 | 26.07 | 14.17 | 35.01 | 64.1 |

Table 11: Consumption of the machine ED1 and ED2 in €.

| Ma | | | | |
|---|---|---|---|---|
| name | ma_id | cl_delay | cl_cost | cl_every |
| ED1 | 1 | 4 | 79.68 | 24 |
| ED2 | 2 | 4 | 125.18 | 24 |
| TW1 | 3 | 4 | 1 | 120 |
| TW2 | 4 | 4 | 1 | 120 |
| Past | 5 | 4 | 1 | 120 |
| Ferm | 6 | 4 | 1 | 120 |

Table 12: Ma list for the real case problem.

| Product names | |
|---|---|
| name | prod_id |
| Skim milk powder | 1 |
| Whole milk powder | 2 |
| Cream powder | 3 |
| Yoghurt | 4 |
| Quark | 5 |
| UF-Quark | 6 |

Table 13: Product names with their ids.

| Fo | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| fo_id | prod_id | step | stp_pre | ma_id | type | max_output | pro_delay | cost |
| 1 | 1 | 1 | [0] | 1 | 3 | 2040 | 1 | 9.63 |
| 2 | 1 | 1 | [0] | 2 | 3 | 1980 | 1 | 20 |
| 3 | 1 | 2 | [1] | 4 | 3 | 1780 | 1 | 9.63 |
| 4 | 2 | 1 | [0] | 1 | 3 | 1740 | 1 | 20 |
| 5 | 2 | 1 | [0] | 2 | 3 | 1710 | 1 | 1 |
| 6 | 2 | 2 | [1] | 4 | 3 | 1760 | 1 | 1 |
| 7 | 3 | 1 | [0] | 1 | 3 | 990 | 1 | 9.63 |
| 8 | 3 | 1 | [0] | 2 | 3 | 1440 | 1 | 20 |
| 9 | 3 | 2 | [1] | 4 | 3 | 1760 | 1 | 1 |
| 10 | 4 | 1 | [0] | 1 | 3 | 990 | 1 | 9.63 |
| 11 | 4 | 2 | [1] | 5 | 3 | 1680 | 1 | 1 |
| 12 | 4 | 3 | [2] | 6 | 2 | 20000 | 5 | 1 |
| 13 | 4 | 4 | [3] | 4 | 3 | 715 | 1 | 1 |
| 14 | 4 | 4 | [3] | 3 | 3 | 105 | 1 | 1 |
| 15 | 5 | 1 | [0] | 1 | 3 | 990 | 1 | 9.63 |
| 16 | 5 | 2 | [1] | 5 | 3 | 1680 | 1 | 1 |
| 17 | 5 | 3 | [2] | 6 | 2 | 20000 | 15 | 1 |
| 18 | 5 | 4 | [3] | 4 | 3 | 714 | 1 | 1 |
| 19 | 5 | 4 | [3] | 3 | 3 | 160 | 1 | 1 |
| 20 | 6 | 1 | [0] | 5 | 3 | 1680 | 1 | 1 |
| 21 | 6 | 2 | [1] | 6 | 2 | 20000 | 18 | 1 |
| 22 | 6 | 3 | [2] | 4 | 3 | 715 | 1 | 1 |
| 23 | 6 | 3 | [2] | 3 | 3 | 160 | 1 | 1 |

Table 14: Fo list for the real case problem.

The orders are given in Table 15 and are representative of an average working month.

Or

| or_id | prod_id | q | due date | or_id | prod_id | q | due date |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 12000 | 2020-01-05 | 12 | 3 | 19500 | 2020-01-29 |
| 2 | 2 | 22000 | 2020-01-23 | 13 | 6 | 8000 | 2020-01-26 |
| 3 | 3 | 15000 | 2020-01-08 | 14 | 6 | 15000 | 2020-02-15 |
| 4 | 4 | 18000 | 2020-01-12 | 15 | 2 | 18500 | 2020-02-10 |
| 5 | 5 | 6000 | 2020-01-12 | 16 | 4 | 26700 | 2020-02-10 |
| 6 | 6 | 8200 | 2020-01-16 | 17 | 3 | 21300 | 2020-02-14 |
| 7 | 1 | 18000 | 2020-01-23 | 18 | 1 | 11300 | 2020-02-06 |
| 8 | 1 | 22000 | 2020-01-25 | 19 | 5 | 6300 | 2020-02-15 |
| 9 | 3 | 18000 | 2020-01-18 | 20 | 6 | 6300 | 2020-02-21 |
| 10 | 5 | 17500 | 2020-01-22 | 21 | 2 | 18300 | 2020-02-18 |
| 11 | 4 | 12300 | 2020-01-19 | 22 | 4 | 16500 | 2020-02-21 |

Table 15: Or list for the real case problem.

### 8.3 Results

The schedule obtained in the first iteration and the final solution after an hour of computation time is presented in Figs.11 and 12. The final solution is computed with different weights in order to show the optimization differences. The best schedule found when the weights are set to prioritize the cost efficiency of the schedule is presented in Fig.13. In Fig.16, the weights are set to prioritize time.

In all figures, in blue, the regular tasks are presented with the order_id that fulfill attached to them, and in green, it is shown the cleaning/maintenance (cip) tasks.

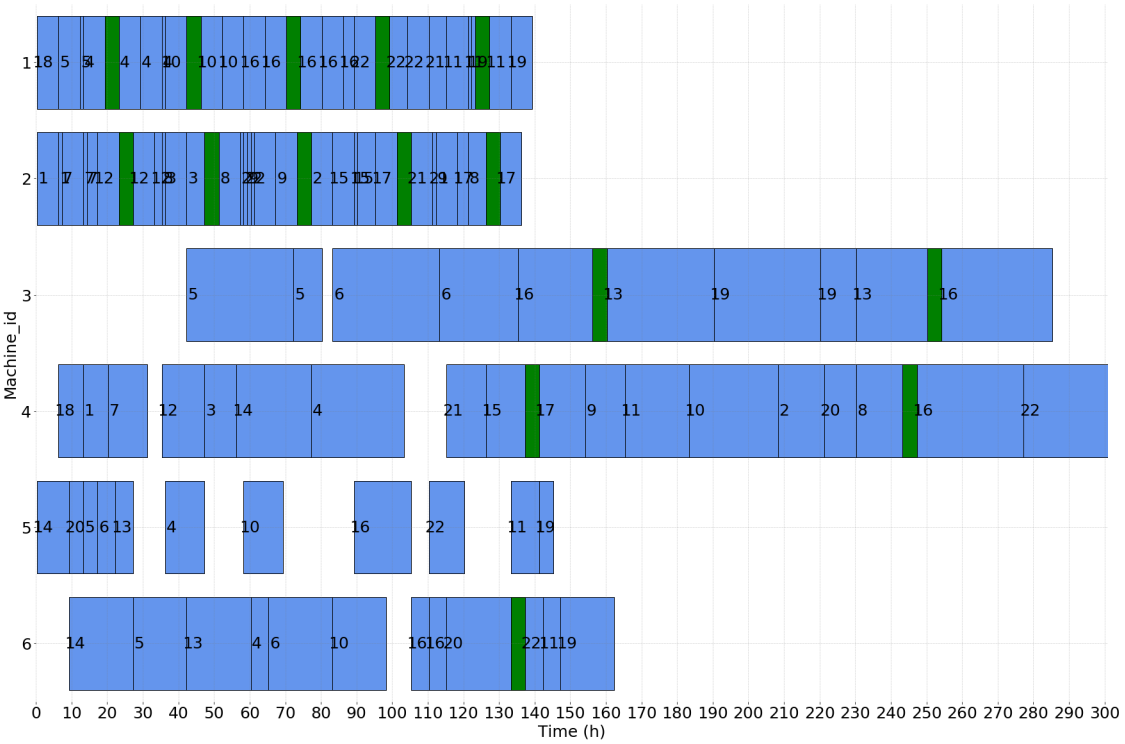



Figure 11: First solution obtained by the scheduler

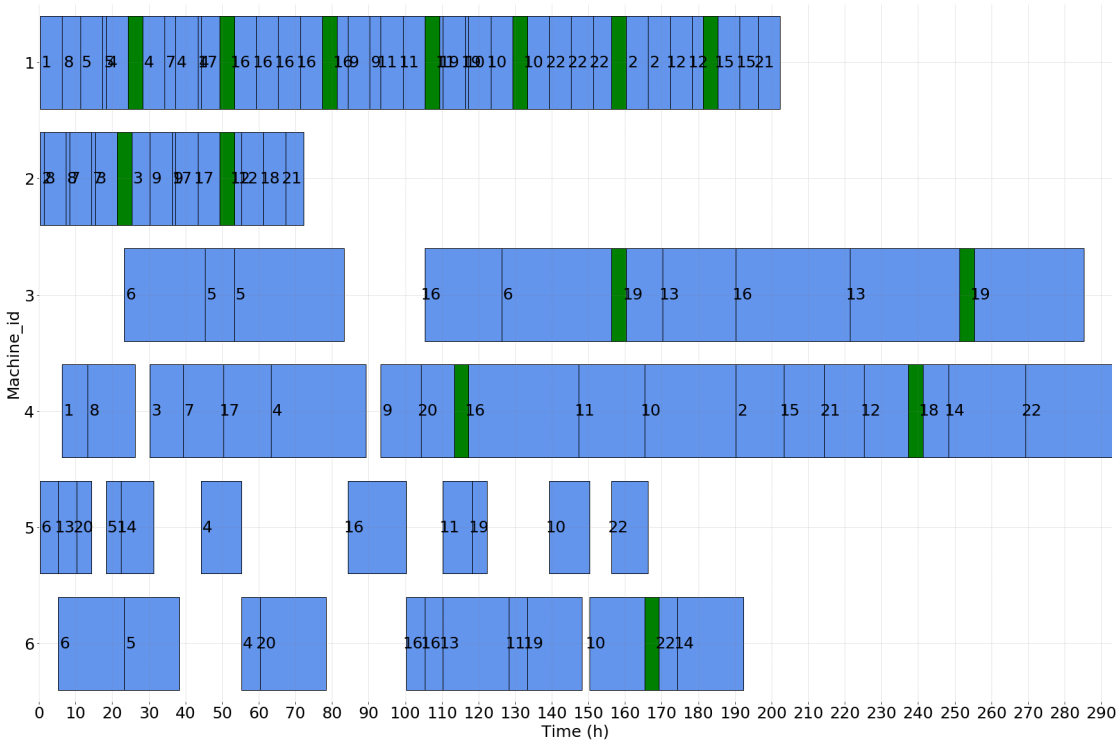



Figure 12: Best solution after 1 hour of reinforcement learning with standard weights.

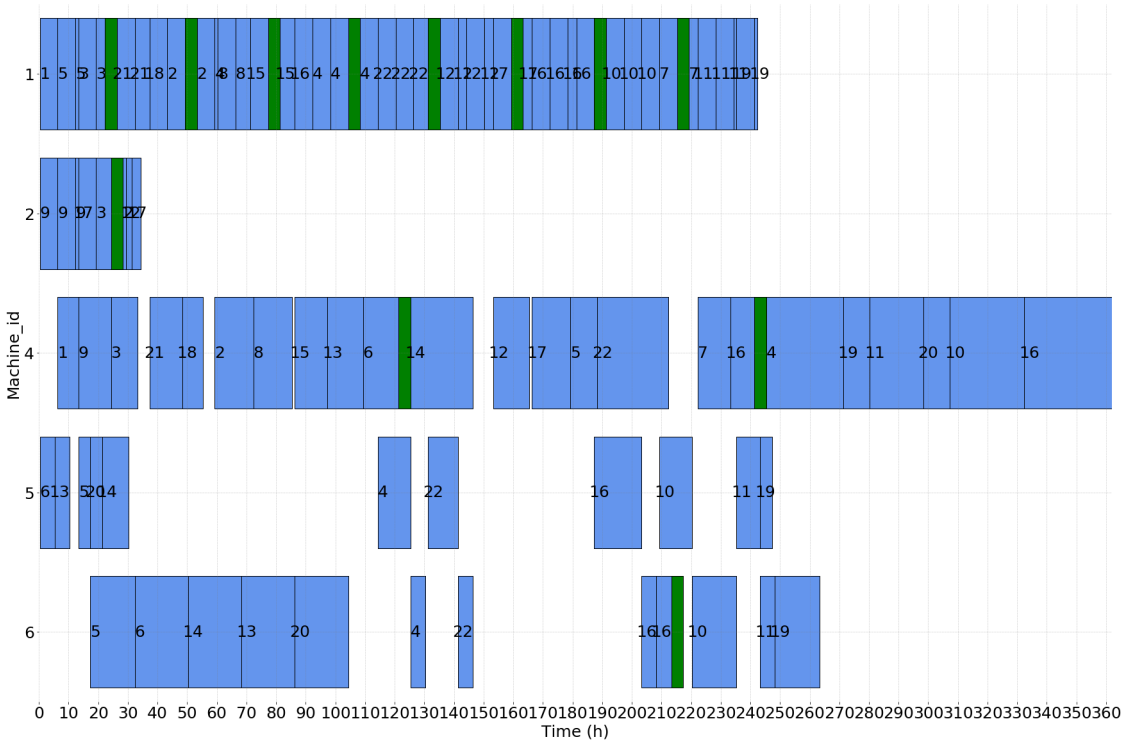



Figure 13: Best solution after 1 hour of reinforcement learning with weights prioritizing costs.

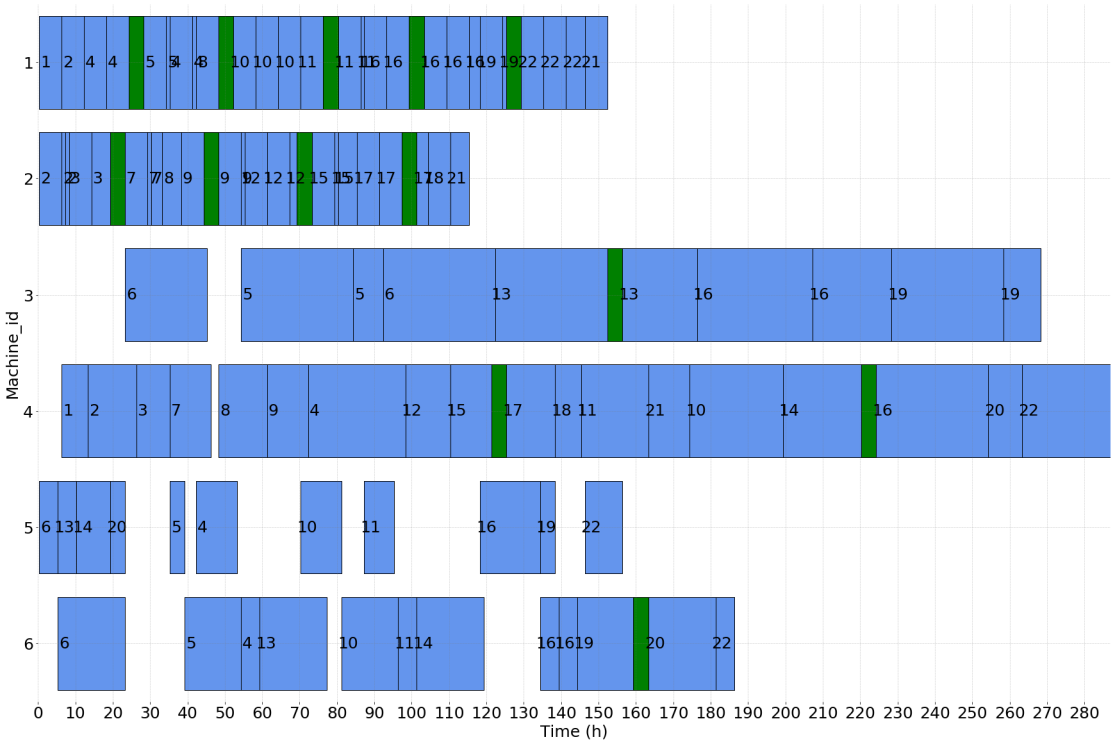



Figure 14: Best solution after 1 hour of reinforcement learning with weights prioritizing time.

The horizon and the costs of the presented schedules obtained with its associated normalized weights are presented in Table 16

| | Weights | | | | Results | | |
|---|---|---|---|---|---|---|---|
| | time | cost | machines | due dates | Horizon(h) | Total Cost | Due dates |
| First Solution | | | | | 302 | 4842.43 | Yes |
| Standard | 0.333 | 0.333 | 0.333 | 0.001 | 289 | 4669.81 | Yes |
| Cost efficiency | 0.05 | 0.475 | 0.475 | 0 | 362 | 3652.67 | No |
| Time efficiency | 0.45 | 0.05 | 0.05 | 0.45 | 287 | 4764.79 | Yes |

Table 16: Weight tuning and costs obtained

Note that the scheduler has been able to achieve a better solution than the first solution after the application of the optimization process via reinforcement learning. The total cost is substantially less when the priority was to optimize the cost and the due dates were not important. The makespan is less when the time is set as the priority to optimize, making the schedule a little bit more expensive.

## 8.4 Different cost scenarios

Now, to show how the schedule performs in different costs scenarios, two new problems are described. First, the machine ED1 cost($c_{ED1}$) is much higher than the cost of ED2($c_{ED2}$). The other scenario is the other way around. These assumptions are not purely imaginative, as the electrical energy consumption is bigger in ED1 but the thermal energy consumption is bigger in ED2. The variations of prices of these costs can vary, making one machine more expensive than the other. Weights are set as the standard case example on the last section, not prioritizing any machine according to the cost in particular.

### 8.4.1 First scenario $c_{ED1} > c_{ED2}$

The costs are set to $c_{ED1} = 50$ and $c_{ED2} = 1$ and for the cip tasks $c^{cip}_{ED1} = 100$ and $c^{cip}_{ED2} = 50$.

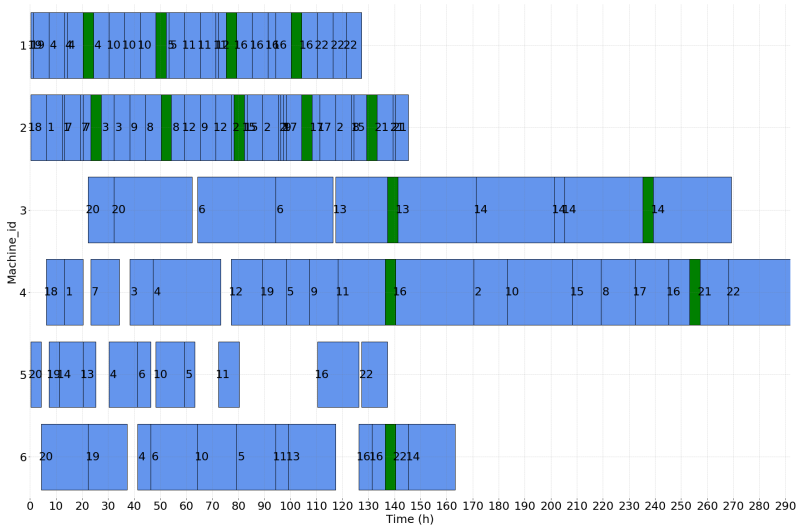



Figure 15: Best solution found on the first scenario with normal weights

Note that in this case the machine 2 (ED2) is prioritized and is more busy than machine 1 (ED1). Not every task is scheduled on (ED2) as other requirements have to be fulfilled.

### 8.4.2 Second scenario $c_{ED1} < c_{ED2}$

The costs are set to $c_{ED2} = 50$ and $c_{ED1} = 1$ and for the cip tasks $c^{cip}_{ED1} = 50$ and $c^{cip}_{ED2} = 100$.

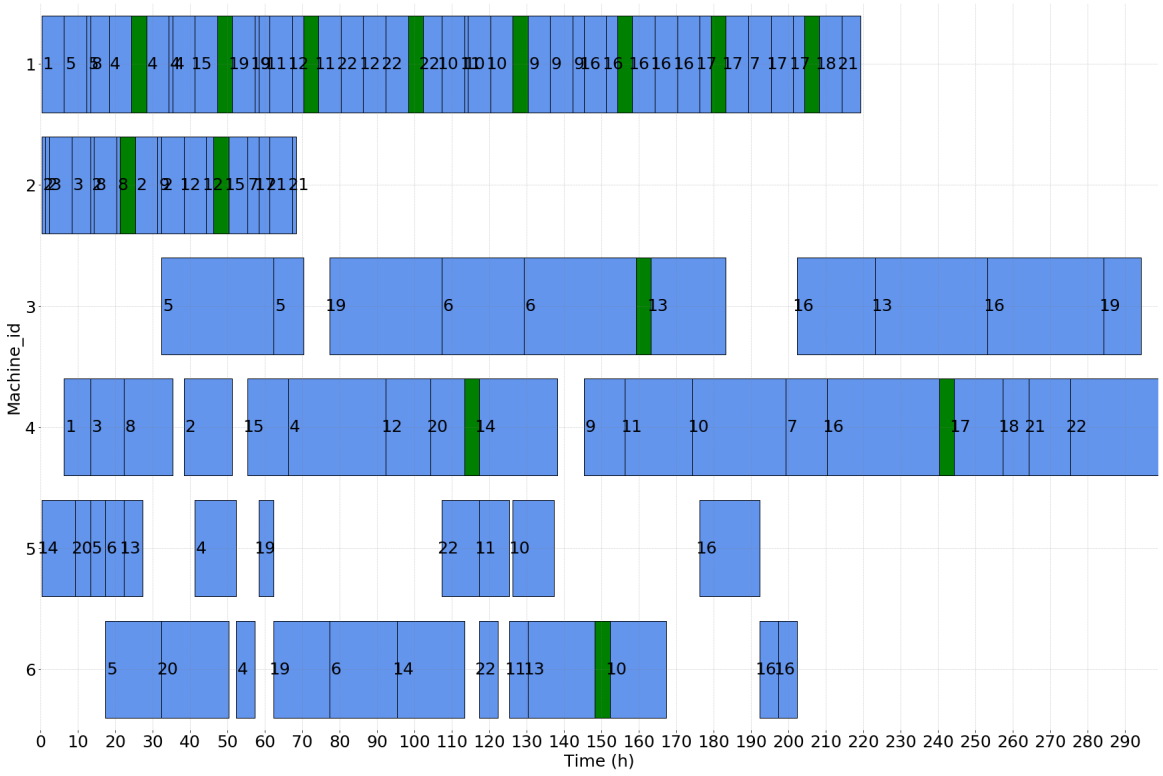



Figure 16: Best solution found on the second scenario with normal weights

Again, it can be seen that the machine 1 (ED1) is prioritized over the machine 2 (ED2) affecting all the schedule, but it still is able to fulfill the same due dates.

## 8.5 Conclusions on the real case study

It has been shown that the scheduler is able to adapt to the different user preferences and different scenarios that this plant can suffer, demonstrating the usefulness of the scheduler developed. Tuning the weights is an incredibly useful tool. Moreover, keep in mind that the scheduler is able to provide a solution when one of the multistage machines are not available.

# 9 Socio-economic and environmental impact and budget

## 9.1 Socio-economic impact

The fact that the scheduler is developed with open source tools indicates that this program could reach more people. Focusing on small/medium business, the impact can be huge. It is proven that good planning inside a production plant not only can yield to efficiency, it also contributes to a good relation to the costumer.

It has been surprising that some companies focusing on health solutions have shown interest on this project. If the machines are surgical rooms, the scheduler is a perfect quick tool to provide an initial feasible and optimal schedule for an hospital. There has been other interest from social home care services. This fact illustrates the usefulness and adaptability of this tool. It can be applied to many scenarios if the abstraction of the concept of orders and machines is done correctly.

Overall, this program may seem a tiny piece inside a production planning phase, but this tiny piece is the core of it. And it can effect hugely to the productivity of many different business and institutions.

## 9.2 Environmental impact

The tool developed may help to reduce water and electricity consumption on the plant. The program can be used as an optimization tool to prioritize those machines that consume less.

Another interesting aspect is that the cost of each step/machine can be related to the pollution produced, using the program to minimize the environmental impact. Another interesting contribution is the ability to analyse the behaviour of the scheduler when the cost is prioritized. If a machine is mostly avoided on the plant when prioritizing (cost/pollution), it may yield to the conclusion that the machine avoided is very contaminant, and changing or improving the machine may have a huge environmental impact.

The fact that the scheduler can be used to reduce emissions/polution with the feature of the cost, makes the program an interesting eco-friendly tool.

## 9.3 Budget

The budget is very straight forward, as the scheduler does not use paid libraries. All of the tools used are open-source and they does not add any additional cost. The only cost is the work done by the programmer.

# 10 Conclusions and Future Work

## 10.1 Conclusions

It has been proven the utility of the scheduler developed through this thesis along with the feasibility to approach the scheduling problem with a constraint satisfaction solver. This approach not only is more elegant, it has also given the opportunity to express more difficult constraints in a concise manner. This fact, with pre- and post- processing makes the new approach very robust and efficient. It has also been proven the feasibility of introducing reinforcement learning techniques into this field. Artificial intelligence techniques are not yet studied on this context and more projects like this may open a new door to further studies and efforts. Moreover, the program developed may be of high interest to any kind of manufacturer for being a very efficient and adaptable tool. For this reason, more work will be added in order to improve the current state of the scheduler as this project has given very promising results.

## 10.2 Future work

The program is not perfect and some improvements can be done. Here is a list of improvements that, due to the complexity and time consuming needs of each of them, are not present on the scheduler presented on this thesis manuscript.

- **Optimize the reinforcement learning step further:** On each iteration, the scheduler has to solve a new problem. This can be improved making the solver only solve a partial problem taking information of previous solutions. This improvement will make the scheduler more efficient computationally.
- **Buffers:** The scheduler is not able to know any in between step restrictions that may occur. For example, maybe it is impossible to store the product between steps on two different machines because there is no a physical space or there are a maximum limit on this parameter.
- **Pre-order processing:** It can be very useful to have an extra step where the orders are processed and sorted to make the calculations of the solver easier. This pre-order processing can encapsulate the time of arrival of raw materials to produce the task, making all the implementation more complete.
- **Resource management:** Initially, resource management was part of the project but the implementation resulted too complex. As the scheduler is a constraint satisfaction solver is better to assess resource management as a pre- and post-process. The formulation of resource allocation with a constraint approach yields to a lot of extra difficult constraints. If the resource availability and management is complex, as usually it is, another solver have to be called before or after the scheduler. So, this aspect was taken out of the project as it falls outside the main scope of the scheduler.
- **Shifts:** Every plant has shifts and the actual scheduler does not take this into account. It schedules all the tasks without stops. This may seem easy to do at first, but maybe there are stages or machines that cannot be interrupted between shifts making the solution more complex.

There are a lot of more aspects that can be improved because every production plant will have its particularities.

## Acknowledgments

I would like to enormously thank my advisor Dr. Vicenç Puig for his help on the development of this project, his tracing on my work and his patience. I would like to thank my family and friends for the immense support in this important stage in my personal and academic path.

## References

[1] "G.Kopanos and L. Puigjaner". *Solving Large-Scale Production Scheduling and Planning in the Process Industries*. Springer, 2019.

[2] P. Castro, I. Harjunkoski, and I. Grossmann. "Optimal Short-Term Scheduling of Large-Scale Multistage Batch Plants". In: *Industrial Engineering Chemistry Research* 48 (2009), pp. 11002–11016.

[3] G. Da Col and C. Teppan. "Industrial Size Job Shop Scheduling Tackled by Present Day CP Solvers". In: *Lecture Notes in Computer Science* 11802 (2019), pp. 144–160.

[4] T. Escobet, V. Puig, J. Quevedo, Palá-Schönwälder, J. Romera, and W. Adelman. "Optimal batch scheduling of a multiproduct dairy process using a combined optimization/constraint programming approach". In: *Computer and Chemical Engineering* 124 (2019), pp. 228–237.

[5] *Google OR-Tools*. URL: https://developers.google.com/optimization.

[6] Jeffrey W. Herrmann. *Handbook of Production Scheduling*. Springer's International Series. Springer, 2006.

[7] G. Kopanos, C. Méndez, and L. Puigjaner. "MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry". In: *European Journal of Operational Research* 207 (2010), pp. 644–655.

[8] C. Méndez, G. Henning, and J. Cerdá. "Optimal scheduling of batch plants satisfying multiple product orders with different due-dates". In: *Computers and Chemichal Engineering* 24 (2000), pp. 2223–2245.