Master's degree in Automatic Control and Robotics

General Advanced Job Shop Scheduling Approach

Master Thesis

September 16, 2022

Author: Javier Pedrosa Alias

Director: Dr. Vicenç Puig Cayuela

Call: 09/2022

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Summary

The development of this thesis aims to design a new approach for solving production planning and scheduling in the process industries in such a way to be adaptable to any manufacturing plant, the description of which would have to be previously provided together with a series of ordered jobs.

The planning and scheduling solving is concerned with the allocation over time of scarce resources between competing activities to meet a given set of requirements with an efficient organization. But, things get complicated as larger the scale of the problem is, i.e. as more resources, activities and requirements are involved. That is why the orientation of the work is focused on an innovative method in the style of Artificial Intelligence, by means of an automated process seeking to converge to a predefined objective.

Although this research object has been studied since the middle of the last century, major breakthroughs were not achieved until the emergence of high-performance computing technologies; since by nature these are combinatorial problems which, the larger the scale, the more exploration they require to find some optimal. In addition, most of the last years related articles has been focused on solution approaches based on mathematical programming techniques, and it is important to note that there are other solution methods for dealing with this kind of problems. These methods can be used either as alternative methods, or as methods that can be combined with mathematical programming models, like the one proposed in this document.

Acknowledgments

I am pleased to offer my most sincere thanks to all the faculty and staff of the university who make advanced degrees such as this one possible. For the resources and all the material they made available to us; as well as for the patience and knowledge they passed on to us during this course. In particular, I would like to thank my thesis director who never ceased to give me all the support and help possible during the elaboration of the work. And also, I cannot forget to thank my family, especially my mother, who has always been and will always be by my side.

Contents

List of Figures

List of Tables

1 Introduction

This is an introductory chapter, aiming to present the project with a brief explanation about the reasons behind this work; its focus, the coverage achieved, and an outlining of the report content.

1.1 Motivation

Nowadays, to be competitive in the process industries, corporations must optimize the productivity by minimizing the operating inefficiencies. In manufacturing, productivity is inherently linked to how effectively the firm manage its available resources, i.e, how to allocate them over time between competing activities to meet the requirements in an efficient fashion. And, in large-scale production factories, finding the optimal planning to maximize productivity not only is not trivial, but it can be extremely complex according to the huge variability of resource allocation possibilities. That is why, the need to create tools to help calculate and optimise production plannings has existed for many years ago.

The first evidences of applications of the early planning methods were reported even before the first computers appeared, around the 1950s. They were based on the formulation and solution of Linear Programming (LP) models. But it was not until the 1980s, that production planning and scheduling for process industries received significant attention, due to the resurgence in interest in flexible processing either as a means of ensuring responsiveness or adapting to the trends in process industries toward lower volume, higher added-value materials in the developed economies. Also, the emerging computational technology facilitated the implementation of more complex methods of a combinatorial or heuristic nature, requiring a large amount of computation to obtain acceptable solutions from an optimisation point of view.

More recently, the topic has received a new impetus as enterprises attempt to optimize their overall Supply Chain (SC) in response to competitive pressures, or to take advantage of recent relaxations in restrictions on global trade; this implies a larger concerning with meeting fairly specific production requirements, such as customer orders, stock imperatives, limited capacities, etc. And this is translated to the need of a more complex modelling to be able to encapsulate all these constraints and to be solvable in an affordable timeframe. Fortunately, over the last few years, a powerful new technological current, Artificial Intelligence (AI), has been growing and gaining prominence in many areas. Surprisingly, with very diverse purposes, but following a line of use centred on complex, changing systems, and with wide domains of resolution. And this is what inspired this thesis, which is expected to end up araising as an industrial PhD afterwards; due to the intention of an industrial digitalisation company to expand its competences offering an optimal production planner service based on some innovative approach, expecting better results than the actual ones.

In addition, a common drawback of most current planning and scheduling methods for process industries is that they offer a very specific solution for particular plants or products, with very tailored algorithms, leading to a very tedious and even expensive design and implementation for each case. So, this is another crucial motivation for this project: to endow the method the ability to be adaptable to a wide range of production system.

1.2 Objective

This section presents the objectives of the thesis, both in general terms and by breaking it down into smaller, more specific objectives.

1.2.1 General Objective

The main goal of this thesis is to propose a general adaptable approach for dynamic production planning and scheduling in the process industries exploring innovative hybrid optimization methods, e.g combining some standard, classic mathematical frameworks with some AI, heuristics or metaheuristics.

Particularly, given a manufacturing plant, the objective is to determine which operations need to be completed on which machines, in what order and at what time while fulfilling a set of requested jobs. This kind of problem is usually called Job Shop Scheduling Problem (JSSP), so from now on, this term will be also addressed. However, JSSP has a large number of variations and extensions regarding the parameters and complexity level considered. For example, a simple model can be formulated such as a factory with linear processing, i.e. with one machine available for each operation; with infinite resources in terms of raw materials and workers; and with deterministic data. Otherwise, degrees of complexity can be added considering, for instance: multiple machines available per operation; machines' maintenance requirements; limited resources in time; limited capacities along the process (between machines, in the final storage, etc.); probabilistic data; or even the optimisation problem can be approached from different points of view: time, cost, delay, etc. also with the possibility of multi-objective optimisation problem.

Then, to explicitly narrow down the style of the manufacturing plants focused on this thesis, they are basically semi-continuous process multi-stage flexible JSSP with intermediate and final limited-buffers, in which parallel processing is possible; jobs arrive at the shop at different times; and machine breakdowns occur stochastically; with a multi-objective optimisation of: (i) the makespan, (ii) the total operation cost, and (iii) the storage usage.

In addition, the method is expected to be very easy to handle at user level, owing to the fact there will be a pre-processing for the input data, where they only have to define the setup of the factory in question, the orders, and a few related parameters; and automatically, after the complete computation and a post-processing stage to illustrate the outcome, the detailed production planning and scheduling will be obtained ready for execution.

1.2.2 Specific Objectives

In order to face the main objective in an easier and more arranged way, some specific sub-objectives have been identified:

- **Objective 1**: conduct an exhaustive research on the works and approaches developed so far in order to identify trends and lacks in the field.
- **Objective 2**: design the input data format to be able to compute the production planning and scheduling of any manufacturing plant configuration, to achieve the generalisation

and adaptability of the method.

- **Objective 3**: define an optimization modelling prototype to obtain a production planning and scheduling based on the input data.
- **Objective 4**: analyse and improve the modelling, by considering the implementation of several optimisation layers, and thinking about some AI, heuristics or metaheuristics approaches.
- **Objective 5**: check the results and return an outcome comprising the useful information extracted from the optimisation to execute, if feasible, the obtained production planning and scheduling.

1.3 Coverage

The aim of this section is to make the final coverage of this work explicit. From the specific objectives perspective, all of them have been accomplished in a certain level. Starting from **Objective 1**, which has been a major workload for the thesis, a lot of knowledge on the field has been gathered so as to steer the thesis in the right direction. The **Objective 2** has been faced by two JSON files containing, on one hand, the information about the set of orders/jobs to fulfill; and on the other hand, the details of the relevant factory where to realize the production (extended in Section 4.1). Being able of describing a wide range of semi-continuous process manufacturing typologies due to the fact that the basic components of these have been identified and modelled. However, it is important to bear in mind that there are some extensions that would need to be added for real implementations, such as maintenance requirements, which eventually would result in additional tasks to allocate; or the possibility of incompatibilities between products to be processed on the same lines, which would imply some extra constraints on the production model.

Regarding **Objective 3** and **Objective 4**, several approaches and modelling proposals have been contemplated during the development of the work. Starting from a very simple and single-layer optimization model that required very well defined tasks (with a fixed duration, a previously assigned operating device, etc.), to a more complex model capable of deciding between the use of parallel lines of operation; of minimising the timing between production and deliveries; and to detect and fix storage troubles with a complementary optimization layer. Concretely, the final proposed method consist of four processes: the first one corresponds to an heuristics that generates equivalent request cases with the purpose of comparing different production batches and dates, in avoid having to include this feature in the optimisation model of the next process. The second process is the core of the approach, which corresponds to a first optimisation layer in which each of the equivalent orders groups generated in the first process is constructed and solved as a JSSP with a Constraint Programming optimisation. The third one is a validation process for the different solutions obtained from the second, and consists of detecting storage mismatches by means of a simulation model of each solution. And lastly, there is a forth process which only intervenes if storage mismatches are detected in the third one, and consists of a second optimisation layer based on a Genetic Algorithm to fix them. This final process not always is required owing to the fact that some of the solutions from the equivalent orders may fit with the available storage; and it is referred as an attempt because is not always guaranteed

to find a solution when invoked, since it is essentially a metaheuristics of random nature.

Finally, **Objective 5** is a much more open question because it should be tailored to the needs of a real user or implementation. But, in this case, the production planning and scheduling is illustrated with a Gantt chart and with a written report. In addition, the monitoring of all inputted storage are shown with line graphs.

1.4 Report outline

The thesis is organised as follows:

- **Chapter 2. Background and State of the Art**: covering the State of the Art on production planning and scheduling approaches; the more relevant methods applied in the field; as well as the promising new techniques.
- **Chapter 3. Proposed Method**: presenting the theory behind the proposed method in detail.
- **Chapter 4. Implementation and Results**: explaining the implementation carried out to test and assess the proposed method.
- **Chapter 5. Budget**: suggesting the expenses related to this work.
- **Chapter 6. Social, Economic and Environmental Impact**: reviewing some consequences that may bring about this approach.
- **Chapter 7. Conclusions**: concluding the thesis with the key ideas and with the remaining open fronts for further investigation.

2 Background and State of the Art

This chapter is devoted to review the most outstanding achievements so far in the optimisation of production planning and scheduling in process industries. However, before going into the details of the resolution techniques, we should first talk about the major production process patterns found in the literature, which can be classified as *continuous*, *batch* and *semi-continuous* [Kopanos and Puigjaner, 2019].

2.1 Production process patterns

A continuous production process is characterized by yielding a constant product flow, with a continuous fed of the resources, and with an online interaction in terms of product modifications and lines maintenance. It is quite suitable in mass production for similar products, but it also requires a very high alertness and quick response times in breakdown situations.

A batch production process is characterized by discrete tasks with a specific sequence, implying a completion dependence between succeeding tasks. This makes the product modifications and lines maintenance easier to address, but a big inconvenience is the need to wait until the sequence is completely finished to get product, which compared to continuous processes, often involve longer times.

A semi-continuous production process is characterized by being a hybrid between continuous and batch, running continuously with periodic startups and shutdowns for product transitions. They appear to be the most optimal as they offers a more customized operation for highly dynamic and uncertain environments, owing to the fact that production is more flexible and equipment can be more efficiently utilized.

2.2 Planning and Scheduling Problems

Now then, this is an introductory discussion about the principal matter of the problems considered in the thesis. First of all, it is important to highlight that exists a disjuncture between the objectives of the planning and the scheduling problems. On one hand, the objective in production planning is to determine the production and inventory levels that will allow fulfilling given customer demand at the minimum cost (including processing, holding and backlog, and switchover expenses) subject to (typically aggregate) production capacity constraints. Thus, a production planning solution consists of production targets and inventory levels over a number of periods into which the planning horizon under consideration is partitioned. On the other hand, the objective in scheduling is the allocation of limited resources (e.g., equipment units, utilities, and manpower) to competing tasks and the sequencing and timing of tasks on units, given a set of production targets and subject to detailed production constraints. Clearly, the two problems are interdependent since the solution of production planning (production targets) is input to scheduling, and the production capacity constraints in production planning depend on the scheduling solution [Kopanos and Puigjaner, 2019]. Nevertheless, they often have to be incorporated into a single framework, share information, and interact extensively with one another [Kreipl and Pinedo, 2004].

The key components of the planning and/or scheduling problem are resources, tasks, and time. The resources need not be limited to processing equipment items (e.g., machines, lines), but

may include material storage equipment, transportation equipment, operators, utilities (e.g., steam, electricity, cooling water), auxiliary devices and so on. The tasks typically comprise processing operations (e.g., reaction, separation, blending, and packing) as well as other activities that change the nature of materials and other resources such as transportation, quality control, cleaning, changeovers, etc. There are both external and internal elements of the time component. The external element arises out of the need to coordinate manufacturing and inventory with expected product lifting's or demands, as well as scheduled raw material receipts and even service outages. The internal element relates to executing the tasks in an appropriate sequence and at right times, taking into account of the external time events and resource availabilities. Overall, this arrangement of tasks over time and the assignment of appropriate resources to the tasks in a resource-constrained framework must be performed in an efficient way, which implies the optimization, as far as possible, of some objective. Typical objectives include the minimization of cost or maximization of profit, maximization of customer satisfaction, minimization of deviation from target performance, etc.

Furthermore, most of the JSSP, except the most trivial ones, belong to the class of NP-hard problems, thus there are no known solution algorithms that are of polynomial complexity in the problem size. This challenged the research community since the 1950s, when the early planning/scheduling methods arose based on LP approaches [Kopanos and Puigjaner, 2019]. Mathematical programming is one of the most important techniques available for quantitative decision making. The use of this kind of standard models approaches entails the presence of: (i) a mathematical framework, and (ii) the employment of an optimisation algorithm. These will be the two main frames designed and combined in countless works about the topic.

2.3 Mathematical framework

To start with, a mathematical framework consist of mathematical expressions encapsulating the domain of a problem. These expressions involve parameters and decision variables. The parameters are input data, while the decision variables represent the optimization outcome [Cha, 2007]. Generally speaking, the nature of the main parameters and variables involved in the mathematical models employed for JSSP used to be: binary decision variables (e.g., for sequencing, resource selection, etc.), resources quantities (e.g., production rates, production batches, inventories, etc.), and tasks times (e.g., starting, ending, duration, etc.). Therefore, the conceptual constraints with which they are usually subjected to are the following ones:

- non-preemptive processing once started (i.e., processing activities must proceed until completion);
- resource constraints at anytime (i.e., the utilization of a resource must not exceed its availability);
- material balances;
- capacity constraints for processing and storage; and
- full demand satisfaction for orders by their due dates (if backlogs are not allowed).

Besides, there are many models according to the type of variables that are used to express the

problem, such as Linear Programming (LP), Quadratic Programming (QP), Non-Linear Programming (NLP), etc. and all of them with numerous variations and extensions (Integer Linear Programming (ILP), Binary Integer Programming (BIP), Mixed Integer Quadratically Constrained Programming (MIQCP), etc.). The trend when modeling JSSP is for Mixed Integer Programming (MIP) models.

Then, as explained in detail in Kopanos and Puigjaner [2019], the mathematical programming modeling of scheduling and planning in process industries is focused on four key elements: (i) the time representation, (ii) the event representation, (iii) the material balance approach, and (iv) the objective function.

In terms of time representation, there are three main approaches:

1. **Discrete-time**: the horizon is divided into a number of equally spaced intervals. As discussed in Méndez et al. [2006], the main advantage is that constraints have only to be monitored at certain known time points, reducing the problem complexity and making the model structure simpler and easier to solve. But there are also drawbacks to this simplification, such as the size of the mathematical model as well as its computational efficiency strongly depends on the number of time intervals postulated; and that suboptimal or even infeasible schedules may be generated because of the reduction of the domain of the timing decisions.

2. **Continuous-time**: the horizon is divided into fewer intervals. According to Méndez et al. [2006], this representation could overcome the previous limitations and generate data-independent models, since a variable time handling allows obtaining a significant reduction of the number of variables and at the same time, more flexible solutions.

3. **Mixed-time**: the time grid is fixed, but the duration of the tasks are variable. This have proven to be efficient, adaptable and convenient for some industrial applications, such in Maravelias [2005], where a mixed-time representation was proposed with a fixed time grid, and processing times were allowed to be variable and span an unknown number of time periods. This was able to handle batch and continuous processes, and optimized holding, backlog, and utility costs. And it also dealt with the release and due dates at no additional computational effort, and coped with variable processing times.

In terms of material balance, depending on the handling of batches and batch sizes, scheduling formulations can be broadly classified into:

1. **Network-based formulations**: it is addressed for general processes and refers to monolithic approaches, which simultaneously deal with the lot-sizing and scheduling problems. These methods are able to deal with arbitrary network processes involving complex product recipes. However, their generality usually implies large model sizes and consequently, their application is currently restricted to processes involving a small number of processing tasks and rather short scheduling horizons.

2. **Batch-based formulations**: it is addressed for sequential processes and are used for single-stage, multistage and multipurpose processes where batches are processed sequentially and where batch splitting/mixing are not allowed and there are no recycle streams. In

these approaches, the number and the size of batches are known in advance. In other words, the lot-sizing (or batching) problem has already been solved. The batching problem converts the demand of products into individual batches aiming at optimizing some criterion. Afterward, the available manufacturing resources are allocated to the batches over time. This approximate two-stage approach, widely used in industry, can address much larger practical problems than monolithic methods, especially those involving a quite large number of batch tasks related to different intermediates or final products [Méndez et al., 2006].

In terms of event representation, five basic event concepts are distinguished:

1. **Global time intervals**: fixed time grids are predefined and the tasks are forced to begin and finish exactly at a point of the grid.

2. **Global time points**: the timing of time intervals is treated as a new model variable. Thus, a common and a variable time grid are defined for all shared resources while the starting and the finishing times are linked to specific time points through key discrete variables.

3. **Unit-specific time events**: a different variable time grid is defined for each shared resource, allowing different tasks to start at different moments for the same event point.

4. **Time slots**: a set of an appropriate number of predefined time intervals for each processing unit with unknown duration is firstly postulated in order to allocate them to the tasks.

5. **Precedence-based**: sequencing binary variables, through big-M constraints, enforcing the sequential use of shared resources are explicitly employed. As a result, sequence-dependent changeovers can be treated in a straightforward manner.

Finally, in terms of the objective function, different measures of the quality of the solution can be used for JSSP, and their selection will directly affect the solution as well as the model computational performance. Typical objective functions include the optimization of makespan, weighted lateness, production costs, inventory expenses, total cost, revenue, and profit. It should be noted that some objective functions can be very hard to implement for some event representations, requiring additional variables and complex constraints.

Concluding, the appropriate selection for each of the four key elements mainly depends on: the production process, the resource limitations, and the goals under consideration. In Kopanos and Puigjaner [2019], all the above information is extended and supported by many works. For example, one related with the focus of this thesis is Kondili et al. [1993], in which the STN process representation was introduced and a discrete-time MILP model was presented. This model was based on the definition of binary variables that indicate whether tasks start in specific units at the beginning of each time interval, together with associated continuous batch sizes. Other key variables were the amount of material in each state held in dedicated storage over each time interval, and the amount of each utility required for processing tasks over each time interval. The key constraints were related to resources (i.e., processing units and utilities), material balances and capacity. The use of a discrete-time grid captured all the plant resource utilization in a straightforward manner; discontinuities in these were forced to occur at the predefined interval boundaries. However, this approach was hindered in its ability to handle large prob-

lems by the weakness of the allocation constraints and the general limitations of discrete-time approaches such as the need for relatively large numbers of grid points to represent activities with significantly different durations.

Another noteworthy articles for the purpose are Yee and Shah [1997] and Yee and Shah [1998], which considered various manipulations to improve the performance of general discrete-time scheduling models. An important feature of their work was the variable elimination, since they recognized that in such models, only about 5–15% of the variables reflecting task-to-unit allocations were active at the integer solution, and it would be beneficial to identify as far as possible inactive variables prior to solution. For this reason, they proposed an LP-based heuristic, flexibility and sequence reduction technique, and a formal branch-and-price method. They also recognized that some problem instances resulted in poor relaxations and propose valid inequalities and a disaggregation procedure similar to that of Sahinidis and Grossmann [1991] for particular data instances.

And Méndez et al. [2000] is also interesting owing to the fact they developed a continuous-time precedence-based MIP model for a process with a single production stage with parallel units followed by a storage stage with multiple units, with restricted connectivity between the stages. Afterward, Méndez et al. [2001] further extended this work to the multistage case with discrete shared resources. In common with other models, there were no explicit time slots in the model, and the key variables were allocations of activities to units and the relative orderings of activities.

2.4 Optimization algorithm

Once the problem is correctly expressed in mathematical form, some tool is required to solve it. This tool corresponds to an optimization algorithm, which can be defined as a procedure that is run iteratively by comparing several solutions until an optimum or satisfactory solution is found. With the advent of computers, optimization has become a part of computer-aided design activities; and there are two distinct types of optimization algorithms widely used today: (i) deterministic and (ii) non-deterministic.

The deterministic optimization algorithms use specific rules for moving one solution to other. These algorithms are in use to suite some times and have been successfully applied for many engineering design problems. Concerning the software, there are usually two main frameworks implicated: (i) the problem definition, which is dealt by a special syntax called Algebraic Modeling Language (AML), and exists some general ones such as GAMS, Mosel, AIMMS or AMPL (with APIs for almost all the main programming languages like C++, Java, Python, R, MATLAB, NET, etc. and suitable for practically all the principal mathematical models); as well as some others for more specific environments, e.g. YALMIP for MATLAB, GEKKO for Python, or OPL for a exclusive IBM IDE; but with the same versatility in relation to mathematical modelling. And (ii) the solver, which not all are suitable for any mathematical model, depending above all on the type of variables with which the model is defined; and not all are free, existing some open-source ones (such as SCIP, GLPK, or Google's GLOP and CP-SAT) and some commercial ones (like Gurobi or IBM ILOG CPLEX).

Regarding the non-deterministic optimization algorithms, Kopanos and Puigjaner [2019] high-

lights the following alternative valued methods for solving JSSP: (i) heuristics, (ii) metaheuristics, (iii) Artificial Intelligence (AI), (iv) Constraint Programming (CP), and (v) hybrid methods. In addition to these methods, in the same book, two elaborate approaches developed by the Process Systems Engineering (PSE) research community are also presented to deal with scheduling and planning in process industries: the event operation network representation, and the S-graph representation.

Most scheduling heuristics, also called dispatching rules, are concerned with formulating rules for determining sequences of activities. They are therefore best suited to processes where the production of a product involves a pre-specified sequence of tasks with fixed batch sizes; i.e. variants of multiproduct processes. Despite of this, there is a lack of works that developed heuristics for process scheduling problems, since it is difficult to devise a series of rules to solve such complex processes.

The metaheuristics algorithms are often inspired by moves arising in natural phenomena, optimizing a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Very known examples are: genetic algorithms, graphs theory, simulated annealing, particle swarm, ant colony and tabu search. Unlike the previous alternative, they have been widely used in a variety of JSSP. These techniques have become popular, but they also have significant drawbacks such as that they do not guarantee the quality of the solution obtained, and it is often impossible to tell how far the current solution is from the optimal. Furthermore, these methods do not formulate the problem as a mathematical framework, since they involve procedural search techniques that in turn require some type of codification or graph representation, and the violation of constraints is handled through ad hoc penalty functions. For this reason, the use of metaheuristics might be problematic for problems involving general processes, complex inequality constraints and continuous decision variables.

The main existing AI techniques are rule-based methods, agent-based methods, and expert systems. All of them have a kernel based on a model underlying the desired behavior. Some takes advantage of a pre-training based on previous experiences and patterns, and apply that knowledge automatically to make decisions. Its application to JSSP is not trivial, nevertheless, Kopanos and Puigjaner [2019] contains some articles that implement it.

Moving to CP, it is a programming paradigm that was originally developed to solve feasibility problems, but later it has been extended to solve optimization problems, particularly JSSP. One of the most important attractions is the fact that it is very expressive since continuous, integer, and Boolean variables are permitted; moreover, variables can be indexed by other variables; and constraints need neither be linear nor convex. Their solution is based on performing constraint propagation at each node by reducing the domains of the variables. If an empty domain is found, the node is pruned. Branching is performed whenever a domain of an integer, binary or Boolean variable has more than one element, or when the bounds of the domain of a continuous variable do not lie within a tolerance. Then, whenever a solution is found, or a domain of a variable is reduced, new constraints are added. And the search terminates when no further nodes must be examined. Constraint programming methods have proved to be quite effective in solving certain JSSP, especially those that involve sequencing and resource constraints. However, they seem to lose effectiveness when trying to add resource assignments [Méndez et al., 2006].

Lastly, a highly valued and researched alternative for the community as well are the hybrid methods, i.e. the combination of some of the algorithms commented above. It can be either a deterministic algorithm with a non-deterministic algorithm, or a blend of deterministics or non-deterministics exclusevely. In Méndez et al. [2001], also referenced above, uses an heuristic model reduction method, which basically takes advantage from an empirical solution tactic or a particular problem feature and incorporate this knowledge into the mathematical problem representation; simplifying the model and obtaining good solutions with less time. Similarly, Roslöf et al. [2001] is centred on an improvement optimization-based technique, which relies on a rescheduling of an initial solution with a unique criterion; iteratively enhancing the existing solution in a systematic manner, with even different criterion.

All in all, although production planning and scheduling has become the subject of intensive research, new contributions are needed for a major impact in real-world applications. Mathematical models that may seem perfect and efficient in paper may not find application in the real scenario for several reasons. For example, some data required by the models may be hard to obtain or difficult to get good estimate values, and frequently the industrial practice requires the consideration of multiple qualitative rules or aspects that may be challenging to be quantified and integrated into the overall optimization framework in an effective manner. From the Kopanos and Puigjaner [2019] attentive review, they highlighted some specific issues of special interest for further investigation in the field:

- Further study and improvement of the mathematical-based solution techniques for hybrid methods. For instance, a MIP-based solution strategy combined with metaheuristics in an attempt to reduce the computational burden of large-scale optimization problems.

- Since process industries are dynamic in nature, the consideration of the uncertainty also arises as a challenging research task.

- More attention should be given to the multisite production problem. A major task should be the simultaneous optimization of production and logistics operations across multiple production facilities and distribution centers, in order to enhance the overall performance, responsiveness, and profitability of the enterprise. And appropriate modeling frameworks and solution strategies should be devised in order to tackle competently these highly complicated optimization problems.

- There are process industries that have received little attention regarding scheduling and/or planning research, such as the food or the ceramics and tiles process industries, in spite of the fact that they imply many optimization challenges. Also, the solution techniques and concepts developed for the planning and scheduling of the process industries can be implemented in other classes of planning or scheduling problems, so the exploration of new contexts is interesting.

3 Proposed Method

In this chapter, the proposed method is accurately described. The problem to be solved is the organisation of the realization of a specific set of orders given a manufacturing plant description. So, firstly, it is worth considering all the required input data for the purpose:

- Information about the orders, which basically are: request and delivery dates, products, and quantities.
- Information about the manufacturing plant setup, which basically corresponds to the features of the operating devices and the storage limitations.
- Information about the production recipes, or formulas, comprising the required processes and resources for the realization of each possible final product.

With this information ready to be managed by the proposed approach, the allocation of the operations related to the production in specific operating devices, as well as their concrete sequence in time is expected to get in return. Then, the method includes four distinctive processes, depicted and listed in the scheme of Figure 1.

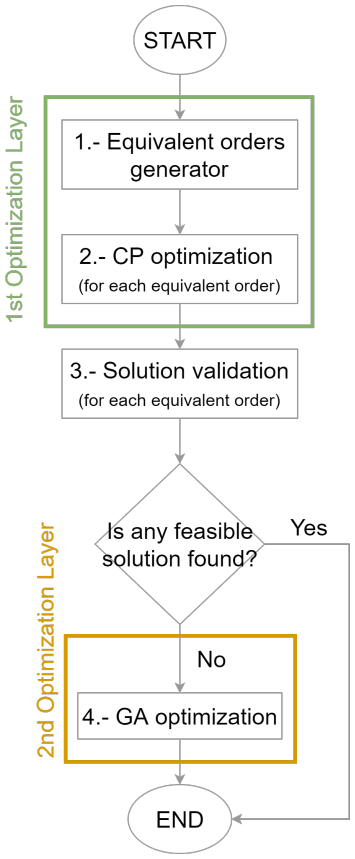


Figure 1: Proposed method Scheme.

With the "Start" is assumed that the input data is already available and prepared to be handled by the subsequent processes; and the "End" leads to a posterior treatment of the final solution to illustrate the result on production planning and scheduling. These pre- and post- processings are explained afterwards in Chapter 4, which also extends all the method processes by means of the implementation, and includes examples for clarification.

3.1 Equivalent orders generator

The first process is an heuristics designed to lighten the solver's workload of the following process, owing to the fact that it allows to evaluate a wide variety of task combinations by generating a set of orders groups derived from the provided one and meeting the original requirements.

It is considered as part of the first optimization layer, and the procedure consists on two stages: the first one generates new orders by reading all of them and joining them according to a delivery date margin (in days, hours or whatever unit of time). I.e., given a boundary, it sums the quantities which delivery dates are lower than this boundary, and assigns it the earliest delivery date to meet the deadlines. This is done for all the possible boundaries between 0 and the margin specified in the corresponding input parameter. While, the second stage takes each of these new orders and splits them in smaller batches with the same original delivery date, to create more but shorter tasks achieving more orders combinations to check.

3.2 CP Optimization

The second process is the core both of the method and the first optimization layer, as it is where all the JSSP based on each of the equivalent orders group are solved. It is responsible for translating the relevant orders, and the operation capabilities of the corresponding factory into a mathematical modelling that includes all the system constraints and variables of interest for the production planning and scheduling.

3.2.1 Mathematical modelling

This mathematical model is based on a CP modelling, defined in terms of the items and notations included in the following Tables 1, 2 and 3.

Table 1: Indices and parameters

O	a set of orders (one of the equivalent orders group).
n_O	number of orders in the O set.
o	order index, $o \in \{1, ..., n_O\}$.
O_o	$o\text{-}th$ order, $O_o \in O$.
T_{r_o}	time of request associated to the $o\text{-}th$ order.
T_{d_o}	time of delivery associated to the $o\text{-}th$ order.
Q_o	quantity associated to the $o\text{-}th$ order.
P	a set of processes that can be performed in the corresponding factory, $P \subset \mathbb{Z}^{1 \times n_P}$.
n_P	number of processes in the P set.
p	process index, $p \in \{1, ..., n_P\}$.
ρ_p	$p\text{-}th$ process, $\rho_p \in P$.
T_o	a set of tasks sets associated to the $o\text{-}th$ order, $T_o \subset \mathbb{Z}^{1 \times n_{T_o}}$.
n_{T_o}	number of tasks sets in the T_o set.
T_{op}	a set of tasks associated to the $o\text{-}th$ order and to the $p\text{-}th$ process, $T_{op} \in T_o \wedge T_{op} \subset \mathbb{Z}^{1 \times n_{T_{op}}}$.
$n_{T_{op}}$	number of tasks in the T_{op} set.
I_{op}	set of tasks indices for the T_{op} set, $I_{op} \subset \mathbb{Z}^{1 \times n_{T_{op}}}$.
i	task index for the T_{op} set, $i \in I_{op} = \{1, ..., n_{T_{op}}\}$.
T_{opi}	$i\text{-}th$ task associated to the $p\text{-}th$ process of the $o\text{-}th$ order, $T_{opi} \in T_{op}$.
M	a set of operating devices available in the corresponding factory, $M \subset \mathbb{Z}^{1 \times n_M}$.
n_M	number of operating devices in the M set.
J	set of all the possible operating device indices, $J \subset \mathbb{Z}^{1 \times n_M}$.
j	operating device index, $j \in \{1, ..., n_M\}$.
M_j	$j\text{-}th$ operating device, $M_j \in M$.
s_j	number of units processed in the $j\text{-}th$ operating device per unit time (speed).
c_j	number of monetary units corresponding to the operation cost of the $j\text{-}th$ operating device per unit time.
P_p	a set of operating devices performing the ρ_p process, $M_j \in P_p : j \in J_p \subset \mathbb{Z}^{1 \times n_{P_p}} \wedge P_p \subset \mathbb{Z}^{1 \times n_{P_p}}$.
n_{P_p}	number of operating devices in the P_p set.
J_p	a set of $j\text{-}th$ indices corresponding to the operating devices performing the ρ_p process, $J_p \subseteq J$.
X_{js}	a set comprising all the starting times ($t_{s_{opi}}$) of the tasks associated to M_j.
X_{je}	a set comprising all the ending times ($t_{e_{opi}}$) of the tasks associated to M_j.
n_{X_j}	number of timing variables both in X_{js} and in X_{je}.
K_{js}	a set comprising all possible permutations of the starting times in X_{js}.
K_{je}	a set comprising all possible permutations of the ending times in X_{je}, in correlation with K_{js}.
k	permutation index, $k \in \{1, ..., n_{X_j}!\}$.

Table 2: Variables

$t_{s_{opi}}$	starting time of the T_{opi} task.
$t_{e_{opi}}$	ending time of the T_{opi} task.
q_{opi}	quantity of the T_{opi} task.
β_{j_k}	boolean variable associated to the selection of each permutation in K_{js}, correlated with K_{je}.
t_{y_s}	makespan starting time.
t_{y_e}	makespan ending time.
y	makespan.
C	total production cost.
d	accumulation of delays with respect to delivery times.

Table 3: Notation

$\overline{x}$	Given a decimal number denoted as x with a line above is equivalent to the rounding up of the same.
$\underline{x}$	Given a decimal number denoted as x with an underline is equivalent to the rounding down of the same.
$\min_{\forall i}(x_i)$	It is equivalent to the lowest number among all the corresponding x-sub-i values.
$\max_{\forall i}(x_i)$	It is equivalent to the highest number among all the corresponding x-sub-i values.

With the following assumptions under consideration:

1. All model parameters are deterministic.

2. A set of orders O will stipulate the main requirements of the model, each one with an associated product and a requested quantity (Q_o) of this.

3. Each of the ordered products has a related formula which describes the required processes for its realization, from a set P that includes all ones that can be performed in the corresponding factory.

4. Each p-th process in P has a related set (P_p) of operating devices (M_j) to perform it.

5. One task (T_{opi}) is assigned for each i-th available operating device of each p-th process required in the realization of the o-th order, such that each task T_{opi} has a unique related operating device M_j from the corresponding P_p.

6. A known and finite planning horizon for each order will be determined by their times of request and delivery, in such a way all tasks necessary for the realization of the o-th order will have to be comprehended within this time horizon. Leading to:

$$t_{s_{opi}} \in [T_{r_o}, T_{d_o}] \text{ and} \tag{1}$$

$$t_{e_{opi}} \in [T_{r_o}, T_{d_o}] \quad \forall p \text{ of the associated sets } T_{op} \in T_o \text{ and } \forall i \text{ of the associated tasks } T_{opi} \in T_{op}. \tag{2}$$

7. Once the processing of a task is started, it should be carried out until completion without interruption (i.e., non-preemptive principle).

3.2.2 CP model constraints

The set of constraints comprised in the model to assure the logic of the production are:

1. The ending time of each task must be greater than its starting time:

$$t_{s_{opi}} \leq t_{e_{opi}} \tag{3}$$

2. The quantity handled in each task must be must be coherent with its duration, and as the model only works with integers, an upward margin of error is set in case the corresponding $j\text{-}th$ operating device speed is not integer. Leading to:

$$q_{opi} \leq (t_{e_{opi}} - t_{s_{opi}}) \cdot \underline{s_j} \leq q_{opi} + \underline{s_j} \tag{4}$$

3. The quantity handled with all the operating devices available for each $p\text{-}th$ process of the $o\text{-}th$ order must be equal to or greater than the ordered quantity, to fulfill the request:

$$Q_o \leq \sum_{i=1}^{n_{T_{op}}} q_{opi} \tag{5}$$

4. Given that an operating device may be required several times in the same order production, or even in several orders' productions; it must be ensured that the associated tasks do not overlap. For this reason, the model must include all the possible tasks combinations for each operating device M_j, and also must select only one of them for each M_j by associating a group of constraints for each combination to a boolean (β_{j_k}) with the following principle:

$$t_{e_{opi}} \leq t_{s_{opi'}} \tag{6}$$

such that i' stands for the succeeding task of the $i\text{-}th$ task in a specific $k\text{-}th$ combination, so there will be $n_{X_j} - 1$ of these constraints, which is the number of succeeding pairs for the tasks associated to M_j, for each of the $n_{X_j}!$ combinations; and only one boolean β_{j_k} for each operating device must be active.

5. In order to impose succeeding between the processes of the corresponding order production sequence, this last is represented as a graph; and a group of constraints is defined for each minimum path obtained from each parent node to each ending node, with the following principle:

$$max_{\{o\text{-}p\text{-}i\text{-}th \text{ and } o\text{-}p'\text{-}i'\text{-}th \text{ indices corresponding to the associated tasks both in } X_{js} \text{ and } X_{je}\}}(t_{s_{opi}} + \overline{\sigma}) \leq t_{s_{op'i'}} \tag{7}$$

$max_{\{o\text{-}p\text{-}i\text{-}th \text{ and } o\text{-}p'\text{-}i'\text{-}th \text{ indices corresponding to the associated tasks both in } X_{js} \text{ and } X_{je}\}}(t_{e_{opi}} + \overline{\sigma}) \leq t_{e_{op'i'}}$ (8)

being $\sigma = \frac{s'_j}{s_j}$. Then, there are one pair of these constraints for each available operating device associated to each process/node of each minimum path that is not a parent one. Or what would be the same, there are one pair of these constraints for each i'-th associated task to each p'-th process of each o-th order that is not a parent one, since the o-p-i-th and o-p'-i'-th indices corresponding to the associated tasks both in X_{js} and X_{je} correspond to the tasks T_{opi}-th associated to the previous process in the relevant sequence, complying that p-th process is previous to p'-th one. Besides, s_j is the speed of the operating device associated to T_{opi}, while s'_j is the speed of the operating device associated to $T_{op'i'}$.

3.2.3 Objective function

From the objective function point of view, denoted as J, the following terms are involved:

1. **Time**: this objective term stands for minimising the makespan of the whole production planning and scheduling, formulated as

$$J_y = y \cdot W_y \tag{9}$$

where $y = t_{y_e} - t_{y_s}$ such that $t_{y_e} = \max_{\forall o,\forall p,\forall i}(t_{e_{opi}})$ and $t_{y_s} = \min_{\forall o,\forall p,\forall i}(t_{s_{opi}})$. And W_y is a weight used to prioritize the term in the objective function for a possible solution tuning, there being an homologous one for each objective term.

2. **Cost**: this objective term stands for minimising the total cost of the production planning and scheduling, formulated as

$$J_C = C \cdot W_C \tag{10}$$

where $C = \sum_{o=1}^{n_O} \sum_{p=1}^{n_{T_o}} \sum_{i=1}^{n_{T_{op}}} c_j \cdot (t_{e_{opi}} - t_{s_{opi}})$, given that each task T_{opi} is associated to a j-th operating device. And W_c the corresponding prioritising weight.

3. **Freshness**: this objective term stands for minimising the elapsed time between production and delivery, and it is closely related to minimise the storage usage; formulated as

$$J_d = d \cdot W_d \tag{11}$$

where $d = \sum_{o=1}^{n_O} \sum_{p=1}^{n_{T_o}} \sum_{i=1}^{n_{T_{op}}} T_{d_o} - t_{e_{opi}}$ and W_d the corresponding prioritising weight.

Leading to

$$J(y, C, d) = J_y + J_C + J_d \tag{12}$$

And the complete formulation would correspond as follows

$$\min_{y,C,d} J(y, C, d) \tag{13}$$

subject to: Eq. 1, 2, 3, 4, 5, 6, 7 and 8.

3.2.4 Solution

Finally, if the CP optimization model is properly defined a priori, a solver should try to find an optimal solution, which consists of the values for each variable presented in Table 2. However, there is talk of trying to because this is not ensured, since the computation time to obtain this solution depends directly on several factors: (i) the level of complexity and dimensions of the model, which would increase it as higher they are; (ii) the optimization algorithm behind the solver, which has an intrinsic *modus operandi* which would directly affect it as well; and (iii) the computing performance of the machine on which it is executed, which is also directly proportional. And, if the factors are adverse, it could be so computationally demanding that it would be unfeasible by means of time. That is why, although it is a Section 4.2.2 issue, solvers often have a parameter relevant to the timeout, offering the chance of limiting it so as not to waste too much time looking for a solution.

Anyway, when the timeout is over, the solver may not have found all possible solutions, and therefore optimality cannot be assured, but it may still have been enough time to collect some feasible solutions.

3.3 Solution validation

The third process corresponds to a validation of the real feasibility in terms of resources management, of the solutions obtained from the previous process; understanding resources as the raw materials, the by-products from production, and the final products. This is needed owing to the fact that the CP optimization model is based on an ideal scenario considering unlimited storage, i.e. no intermediate neither final buffers are included. Then, an exclusive monitoring for each resource involved in the production is performed, on the basis of a concrete solution with its defined tasks set; to detect incompatible amounts with the capacities of the corresponding storage (extended in Section 4.3).

3.4 GA Optimization

Last but not least, the forth process is particular because it is not always invoked, as it depends on whether a mismatch has been detected in the third process. It corresponds to the second optimization layer, and it is based on a Genetic Algorithm to try to fix the mismatches by, basically, testing random combinations of tasks among the different solutions corresponding to the equivalent order groups (extended in Section 4.4). This, being in essence a metaheuristic, does not guarantee optimality, since the implicit algorithm is non-deterministic. But also does not guarantee a proper convergence to the expected solution, due to its random behaviour, reaching very different outputs between executions. Nevertheless, it is an efficient way of exploring a wider domain of production planning and scheduling solutions that may eventually fit in with the corresponding storage restrictions.

Surely, the solution that will end up meeting the storage restrictions will have a higher throughput than exactly what is requested, since at best, the precise one is the one with the detected mismatches; and lower cannot be because then it would not meet the throughput requirements of the corresponding orders group.

4 Implementation and Results

In order to test the method, a program was implemented in a Python project based both on standard libraries and more advanced open-source packages (detailed in Section 4.6). The scheme of the whole program is depicted in the Figure 2, and it is correlated with the flow diagram of the Figure 3, which provides more detailed steps of the proceeding.

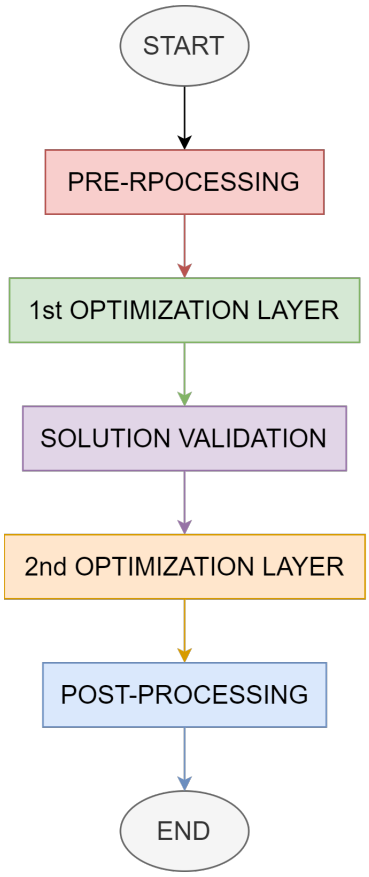


Figure 2: Implementation scheme.

As it can be seen, the program scheme implicitly contains the processes of the proposed method, wrapped by a previous and a posterior processing to convey the information properly.

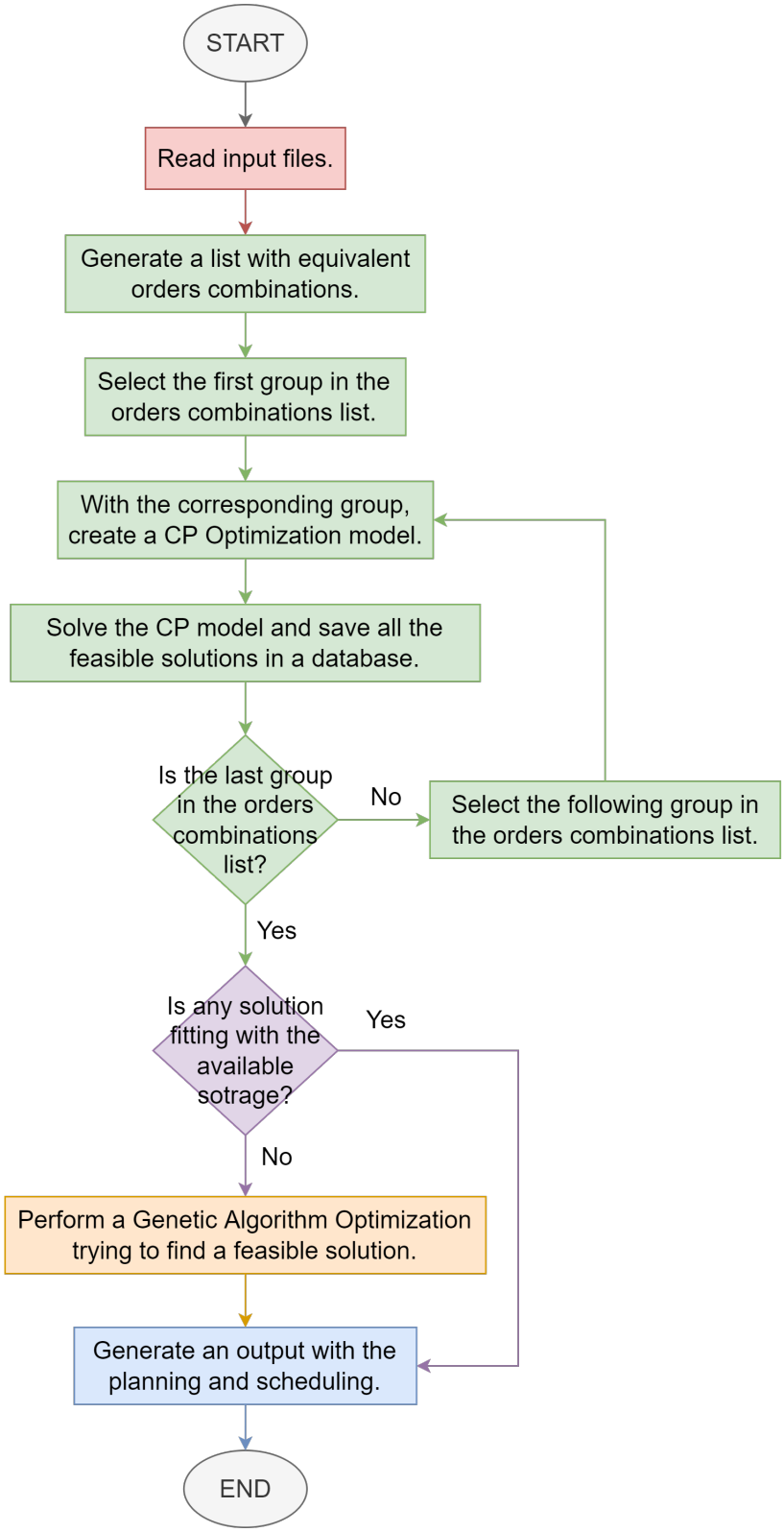


Figure 3: General implementation flow diagram.

Moreover, a simple example, hereafter referred as clarifying example, will be addressed among this chapter, which was designed to make the functionality of some processes more clear, and to show the results of an execution case. Attached in the , there are the corresponding input files.

4.1 Pre-processing

So, the pre-processing is the first stage of the program, featuring the red blocks in Figure 2 and 3. It is crucial because it is the part responsible for adaptability, due to the fact that it translates the input files into a Python class including all the necessary data for the creation and solution of the subsequent optimization model.

There are two input files involved, both are JSON files:

1. one contains information about the orders (jobs); comprising two objects:
   (a) *Factory*: a single-valued object indicating the factory name, in order to access it from an external database with all the factories description files.
   (b) *Orders*: an array of nested objects corresponding to the solicited orders; each one with four key-value pairs:
      - *Product*: identifier of the ordered product.
      - *Request*: date of the order request.
      - *Delivery*: date of the order delivery.
      - *Quantity*: amount of ordered product.
2. and the other contains information about the factory; comprising five objects:
   (a) *Settings*: object with some parameters required to set some configuration options for the modelling and the solver, specifically:
      - *multiple_solutions*: integer used as flag to indicate that only the optimal solution found by the solver is considered (=0); or that all feasible solutions are considered (=1).
      - *multiple_sol_verbose*: integer used as flag to indicate whether (=1) or not (=0) to print some information related to the solver outcome.
      - *show_verbose*: integer used as flag to indicate whether (=1) or not (=0) to print some other information related to the solver outcome.
      - *main_weights*: nested object containing the weights for the objective function terms, with the key-value pairs named as the corresponding terms (presented in Section 3.2.3).
      - *delivery_margin*: integer indicating the time units limit for combining nearby or-

ders' delivery dates.

- *max_orders_division*: integer indicating the times limit for batching the orders' quantities.

(b) *Lines*: object with the information about the characteristics and dependences of each operating device (i.e. machines, process-lines, etc.):

- *id*: integer used as unique identifier for the corresponding device.
- *type*: string indicating the function performed by the corresponding device.
- *speed*: number indicating the amount of units processed by the corresponding device per unit time.
- *cost*: number indicating the amount of monetary units corresponding to the operating cost per unit time.
- *available*: integer used as flag to indicate if the device is operational (=1) or in failure (=0).
- *in_storages*: array of integers indicating the unique identifiers of the storing devices that provide resources to the operating device.
- *out_storages*: array of integers indicating the unique identifiers of the storing devices that keep the output resource from the operating device.

(c) *Storages*: object with the information about the characteristics and dependences of the storing devices (i.e. limited-buffers between operating devices, stock warehouses, etc.):

- *id*: integer used as unique identifier for the corresponding device.
- *resources*: array of integers indicating the unique identifiers of the resources that the corresponding device stores.
- *curr_capacities*: array of integers, in correlation with the previous one, indicating the actual amount of units for each resource.
- *min_capacities*: array of integers, in correlation with the previous one, indicating the maximum amount of units available for each resource.

(d) *Formulas*: object with nested objects including the information about the final products formulas. These nested objects are named as the same final products, and they contain the description of the graph of the relevant formula. This description is based on key-value pairs, the keys of which are the involved processes; and the value for each one is a list containing the consecutive processes. Below, there is an example for clarification, where Figure 5 corresponds to the description of the Figure 4 graph.

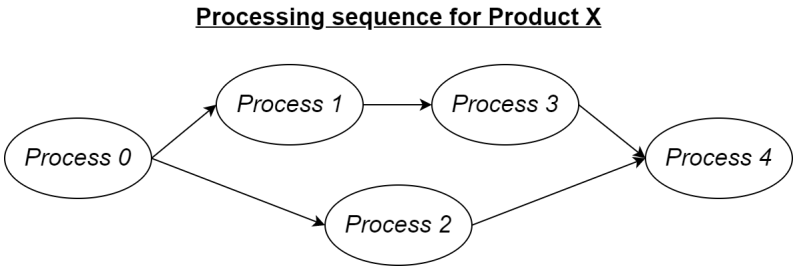


Figure 4: Process sequence graph example.

```
"X":  {  "Process 0": [ "Process 1", "Process 2" ],
         "Process 1": [ "Process 3" ],
         "Process 2": [ "Process 4" ],
         "Process 3": [ "Process 4" ],
         "Process 4": [ ]  }
```

Figure 5: Description of the process sequence graph example.

(e) *Processes*: object with nested objects including the information about the production processes. These nested objects are named as the same processes, and they contain the following key-value pairs:

- *in_resources*: array of integers indicating the unique identifiers of the input resources for the corresponding process.
- *in_ratio*: array of numbers, in correlation with the previous one, indicating the required proportion of each input resource with respect the corresponding process' output resource.
- *out_resource*: integer indicating the identifier of the output resource for the corresponding process.

As commented above, an instance of input files is included in the , corresponding to the clarifying example.

4.2 1st Optimization Layer

The 1st optimization layer is the second stage of the program, featuring the green blocks in Figure 2 and 3, and it is already part of the proposed method. As set out above, it is composed by two processes, the implementation of which is discussed in the next subsections.

4.2.1 Equivalent orders generator

Once both the orders and the factory description are already read, the next step is to modify the original orders list to generate equivalent orders groups. This is done by testing different combinations of batches and dates to reach the optimal one, always complying with the original specifications, as introduced in Section 3.1.

The concrete implementation corresponds to:

1. Firstly, in a for loop, a variable is iterated, going from 1 to the *delivery_margin* indicated in the *settings* object from the factory input file, one by one. Then, with a second for loop iterating the order entries from the original list related with a specific product, it checks and joins the quantities corresponding to those order entries expected to be delivered with a difference of days less than the current iterating variable value; and this new order is associated to the earliest delivery date among those joined orders. Obtaining all the possible merges groups according to proximate dates for each requested product.

2. Secondly, each of these merges groups for each requested product are iterated dividing their quantities into sub-batches, with a second for loop that iterates a variable from 1 to the *max_orders_division* indicated in the *settings* object from the factory input file, one by one, and representing the number of sub-batches with which the relevant entry is partitioned.

3. Thirdly, all the possible combinations between the requested products with each of these new orders partitions are saved, creating the final orders groups set.

As larger the original orders list, the *delivery_margin* and the *max_orders_division* are, more orders combinations are obtained.

For clarification, the Table 4 includes the original orders list corresponding to the clarifying example.

Table 4: Original orders list

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	100
1	9	21/04/2022	02/06/2022	100
2	5	21/04/2022	03/06/2022	50

And, by setting $delivery\_margin = 2\ days$ and $max\_orders\_division = 2$, the program generates 11 equivalent orders groups, stated in Tables 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15.

Table 5: Equivalent orders group 1

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	100
1	9	21/04/2022	02/06/2022	50
2	9	21/04/2022	02/06/2022	50
3	5	21/04/2022	03/06/2022	50

Table 6: Equivalent orders group 2

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	50
1	5	21/04/2022	01/06/2022	50
2	9	21/04/2022	02/06/2022	100
3	5	21/04/2022	03/06/2022	50

Table 7: Equivalent orders group 3

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	50
1	5	21/04/2022	01/06/2022	50
2	9	21/04/2022	02/06/2022	50
3	9	21/04/2022	02/06/2022	50
4	5	21/04/2022	03/06/2022	50

Table 8: Equivalent orders group 4

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	100
1	9	21/04/2022	02/06/2022	100
2	5	21/04/2022	03/06/2022	25
3	5	21/04/2022	03/06/2022	25

Table 9: Equivalent orders group 5

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	100
1	9	21/04/2022	02/06/2022	50
2	9	21/04/2022	02/06/2022	50
3	5	21/04/2022	03/06/2022	25
4	5	21/04/2022	03/06/2022	25

Table 10: Equivalent orders group 6

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	50
1	5	21/04/2022	01/06/2022	50
2	9	21/04/2022	02/06/2022	100
3	5	21/04/2022	03/06/2022	25
4	5	21/04/2022	03/06/2022	25

Table 11: Equivalent orders group 7

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	50
1	5	21/04/2022	01/06/2022	50
2	9	21/04/2022	02/06/2022	50
3	9	21/04/2022	02/06/2022	50
4	5	21/04/2022	03/06/2022	25
5	5	21/04/2022	03/06/2022	25

Table 12: Equivalent orders group 8

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	150
1	9	21/04/2022	02/06/2022	100

Table 13: Equivalent orders group 9

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	150
1	9	21/04/2022	02/06/2022	50
2	9	21/04/2022	02/06/2022	50

Table 14: Equivalent orders group 10

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	75
1	5	21/04/2022	01/06/2022	75
2	9	21/04/2022	02/06/2022	100

Table 15: Equivalent orders group 11

Order id	Product	Request date	Delivery date	Quantity
0	5	21/04/2022	01/06/2022	75
1	5	21/04/2022	01/06/2022	75
2	9	21/04/2022	02/06/2022	50
3	9	21/04/2022	02/06/2022	50

4.2.2 CP Optimization

With the equivalent orders groups already generated, it is time to model and solve each of them. For this purpose, an open-source software for combinatorial optimization developed by Google, called OR-Tools, was selected. Apart from being free, it has the advantage of integrating, in a unique package, the two implicated frameworks mentioned in Section 2.4: the problem definition (with the package's own AML), and the solver, specifically the CP-SAT solver.

The implementation of this stage is reduced to the coding of the model, described in Section 3.2, with the appropriate syntax; and to a post-treatment of the returned data to be interpreted in the subsequent phases. Nevertheless, this returned data depends on the *multiple_solutions* parameter from the *settings* object of the factory input file, which is a flag that conditions the solution to be single, expecting the optimal one, or to be multiple, expecting all the feasible ones, by means of each group of equivalent orders. This is useful to assess more situations and to achieve a larger population for the GA optimization if desired.

Figures 6 and 7 shows, respectively, the Gantt chart of both the optimal and the worst solution, in terms of the makespan length, obtained in this stage for the clarifying example case; from among all the solutions of all the equivalent orders groups. That is why the solutions are based on a different number of orders, concretely: the optimal solution corresponds to the group of equivalent orders in Table 12; while the worst solution corresponds to one of the feasible solutions corresponding to the group of equivalent orders in Table 9.

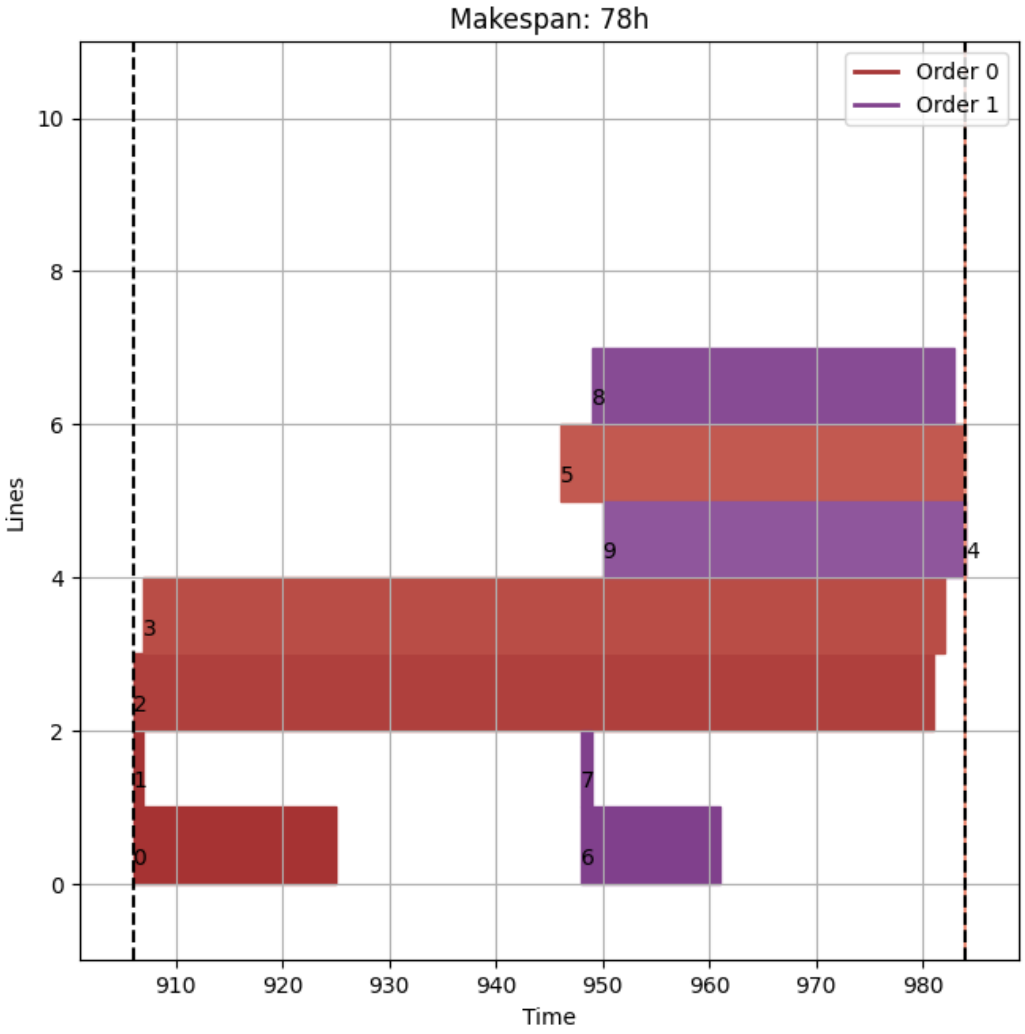


Figure 6: Gantt chart of the optimal solution for the clarifying example.

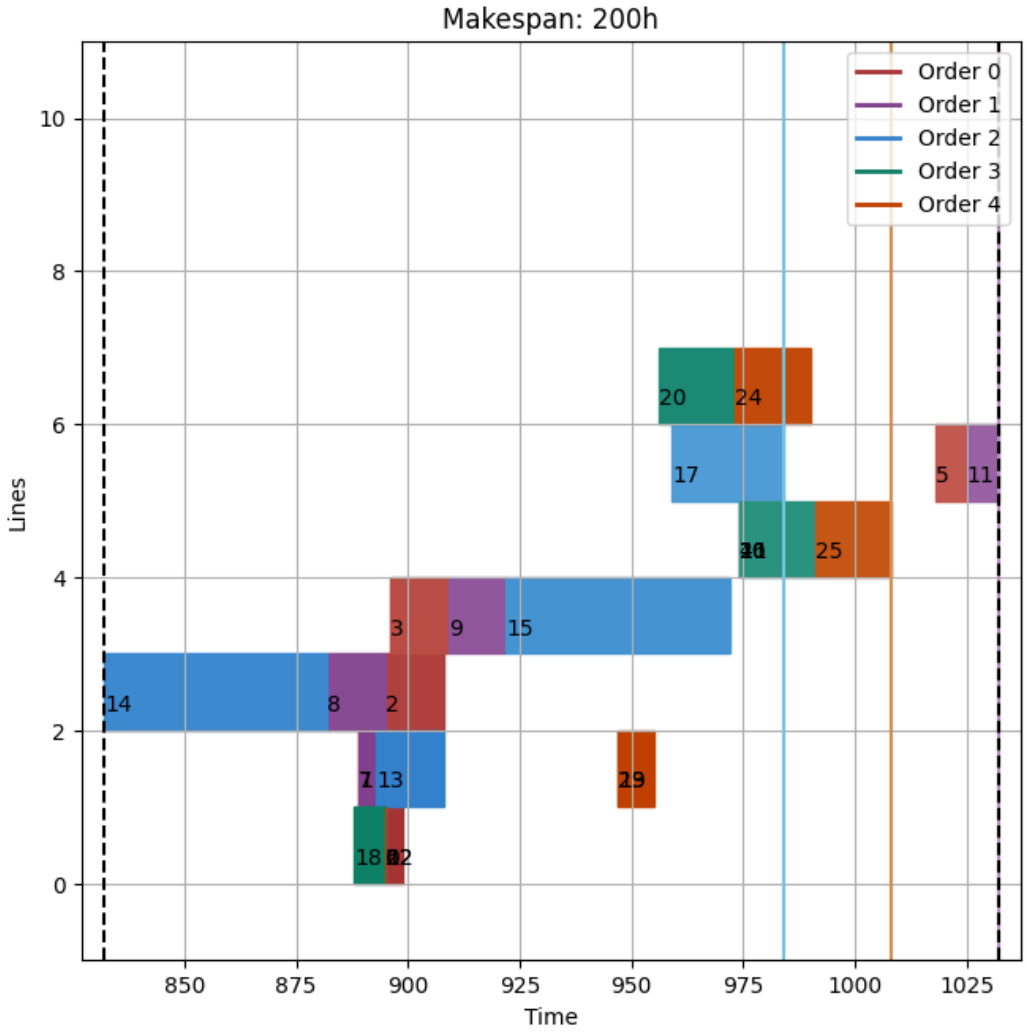


Figure 7: Gantt chart of the worst solution for the clarifying example.

The details about the Gantt chart are set out later in Section 4.5.

4.3 Solution validation

At this stage, featuring the purple blocks in Figure 2 and 3, and for every solution of each group of equivalent orders, the proceeding consist on simulating all resources flow in all the storage described in the input data. Then, given a production planning and scheduling solution, an array with a length equal to the corresponding makespan ending time (t_{y_e}) is defined for each resource of each storage, in order to represent the monitoring of each time unit of the complete production scheduling. Thus, through a first processing that checks all the tasks defined in the solution schedule, each of which has an associated operating device that in turn has its input and output storage and the related resources defined in the input data; it accumulates or subtracts

the amount of resource dealt with in the corresponding array, for each unit of time that exists a task associated with the same. And there is a second processing that reviews all the arrays associated to each resource with the corresponding storage bounds, performing a report of the storage mismatches by means of each solution.

To illustrate this, the Figures 8, 9 and 10 shows the resources flow in every storage according to the optimal production planning and scheduling solution, with no storage mismatches; and the Figures 11, 12 and 13 shows the same for the worst solution, in this case, with a mismatch in the Storage 3 for the resoruce 2 ("R 2" in the legend), that is exhausted. Both of them are obtained from the execution of the first optimization layer keeping with the clarifying example, the corresponding Gantt charts of which are shown above in Figure 6 and 7.

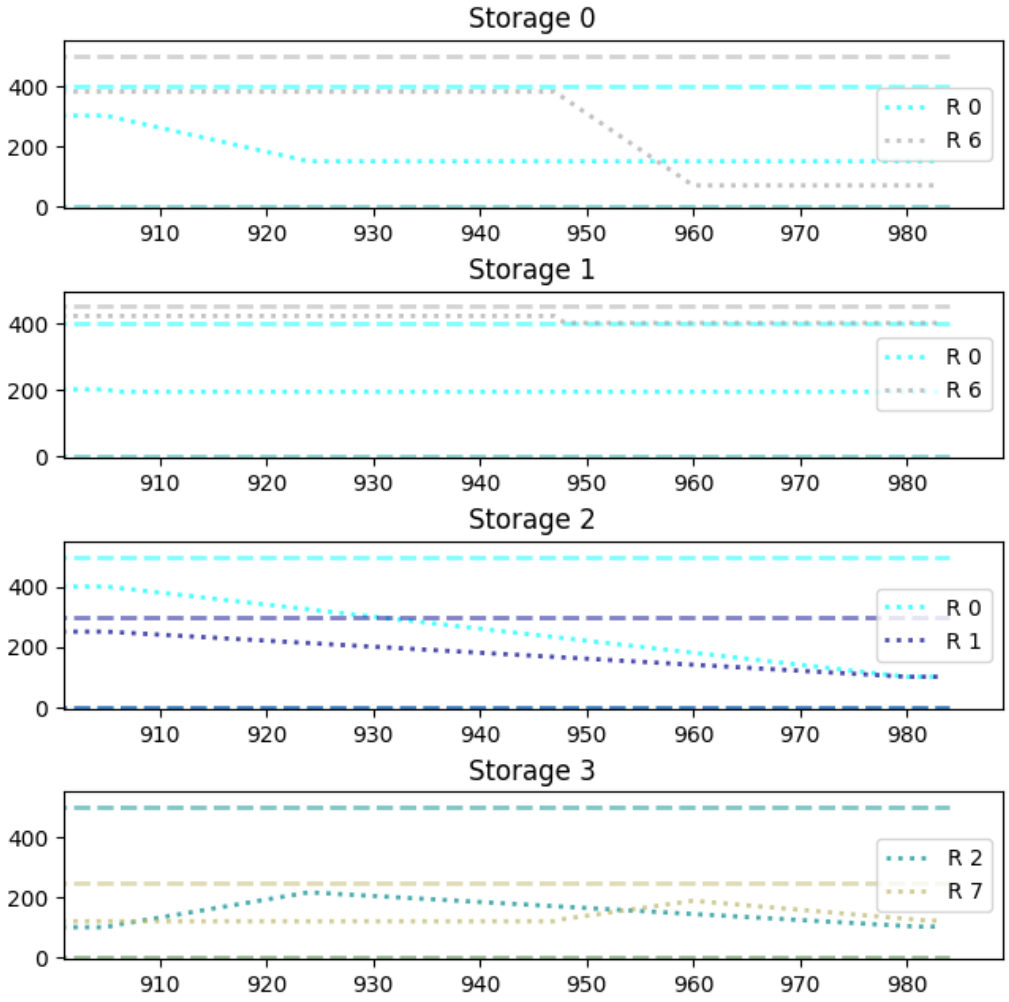


Figure 8: Monitoring of the first set of storages of the optimal solution for the the clarifying example.

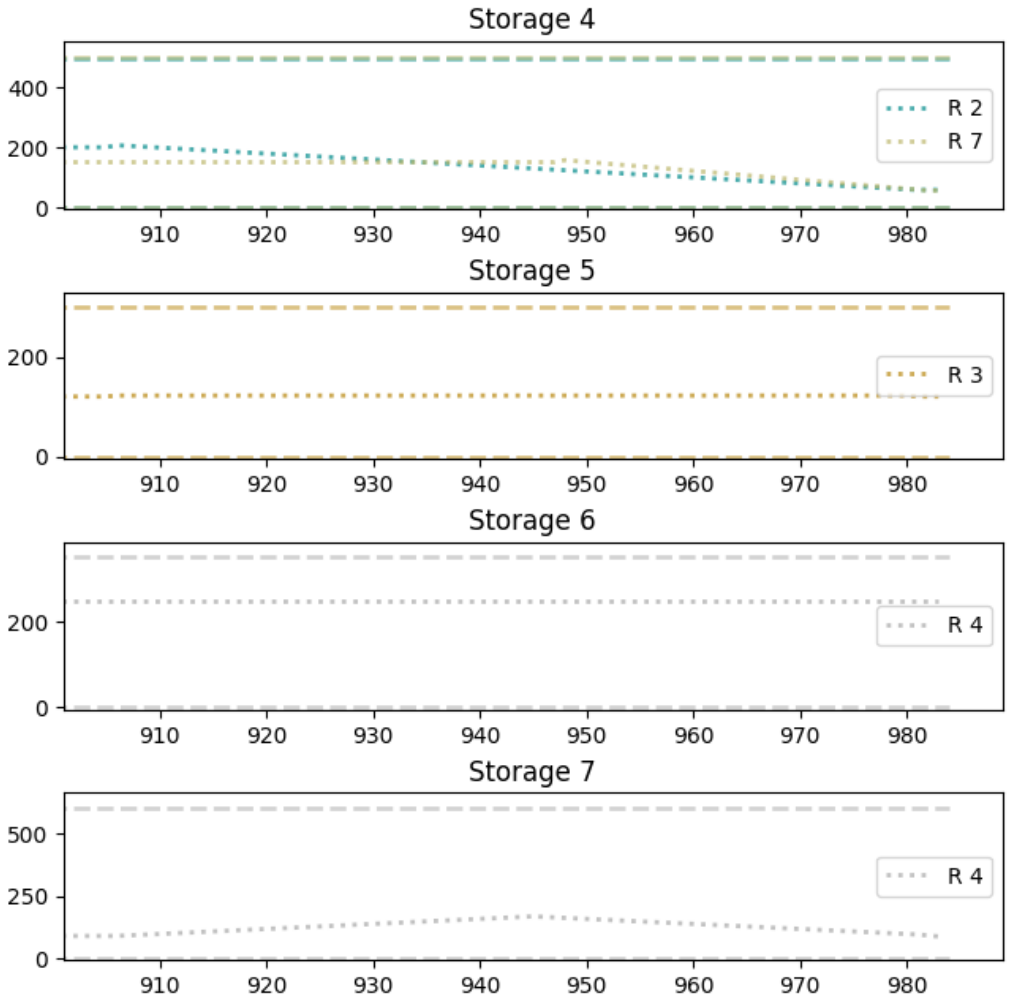


Figure 9: Monitoring of the second set of storages of the optimal solution for the the clarifying example.

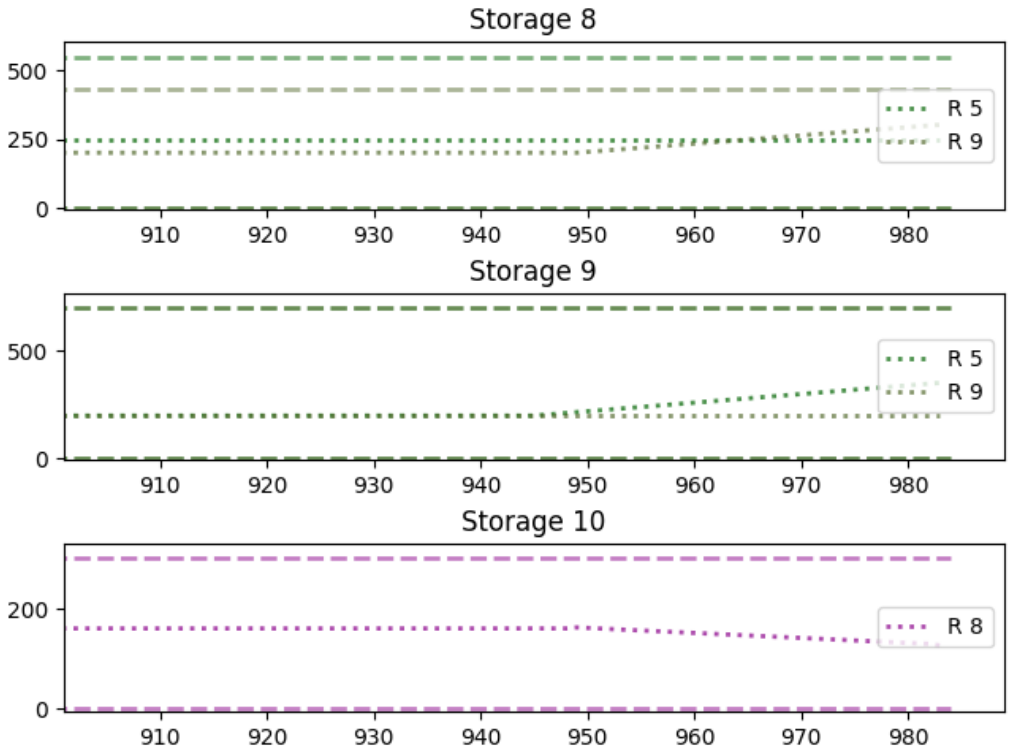


Figure 10: Monitoring of the third set of storages of the optimal solution for the the clarifying example.

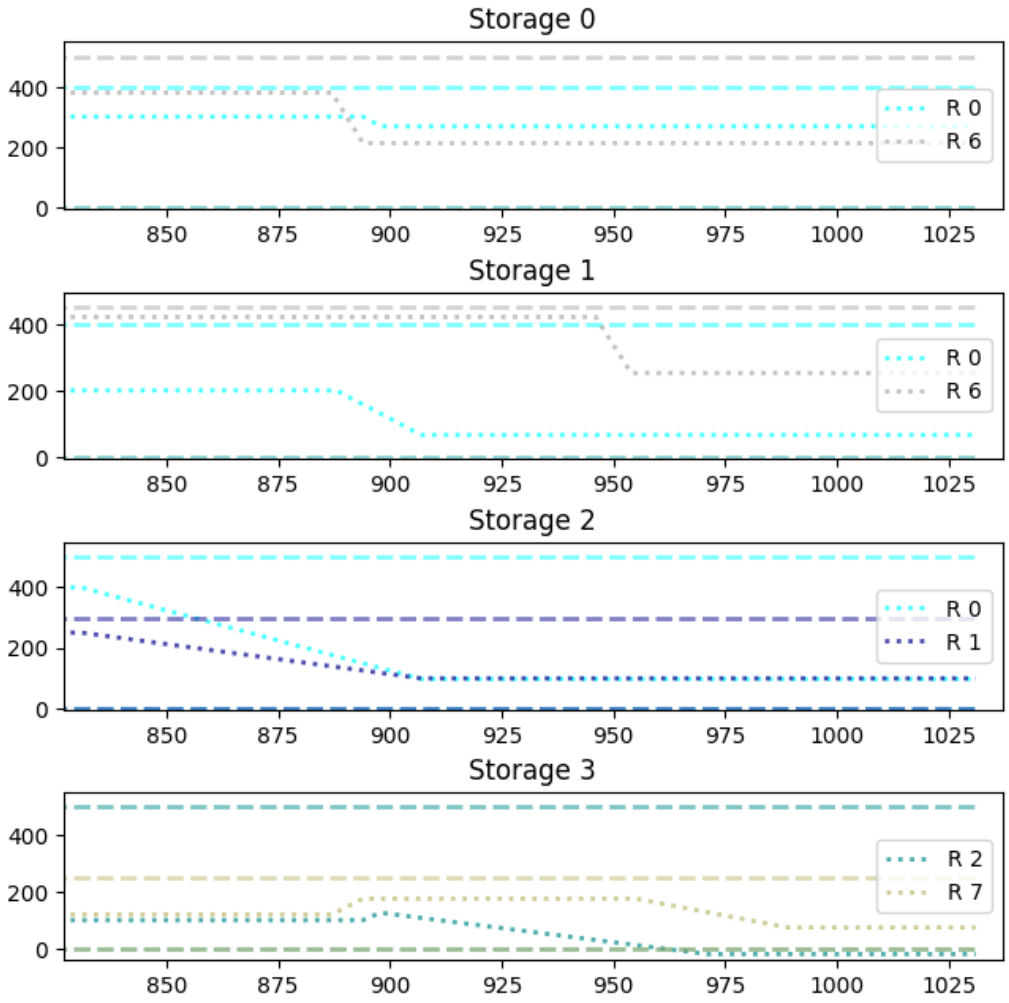


Figure 11: Monitoring of the first set of storages of the worst solution for the the clarifying example.

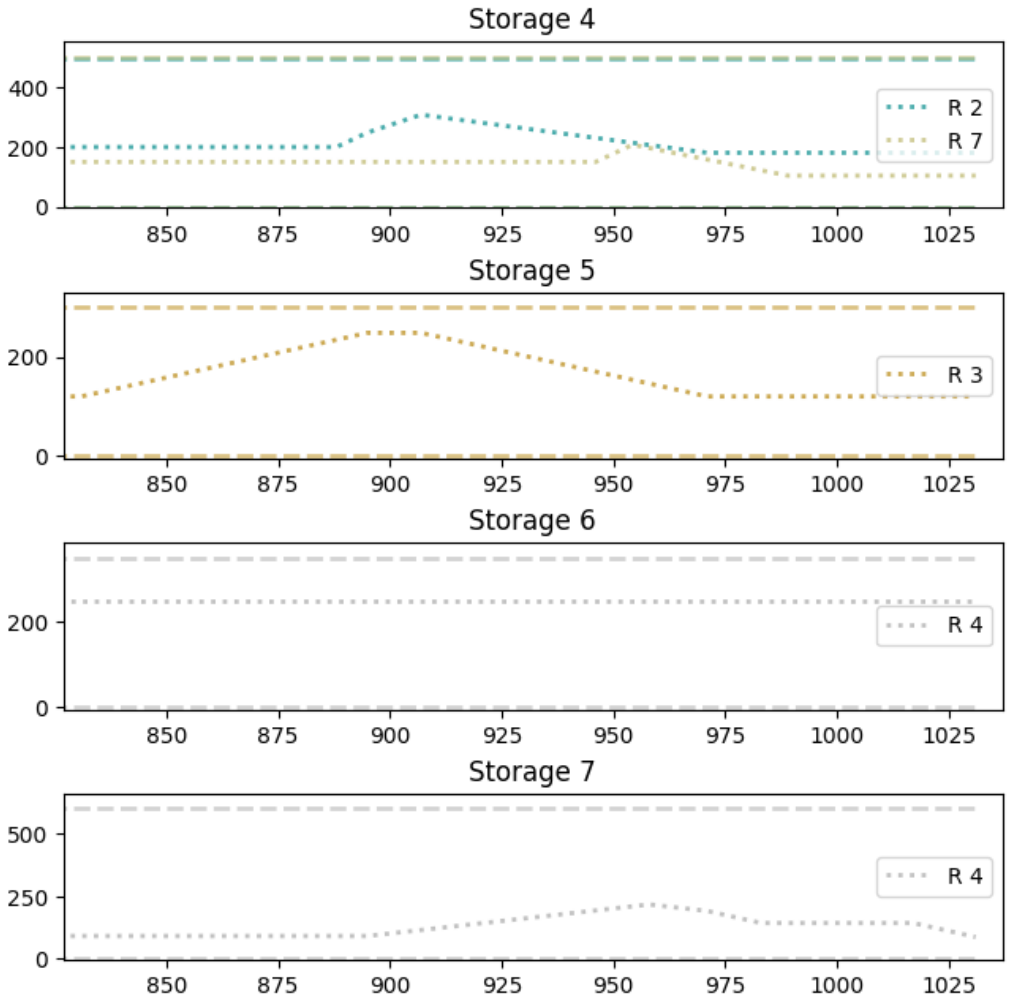


Figure 12: Monitoring of the second set of storages of the worst solution for the the clarifying example.

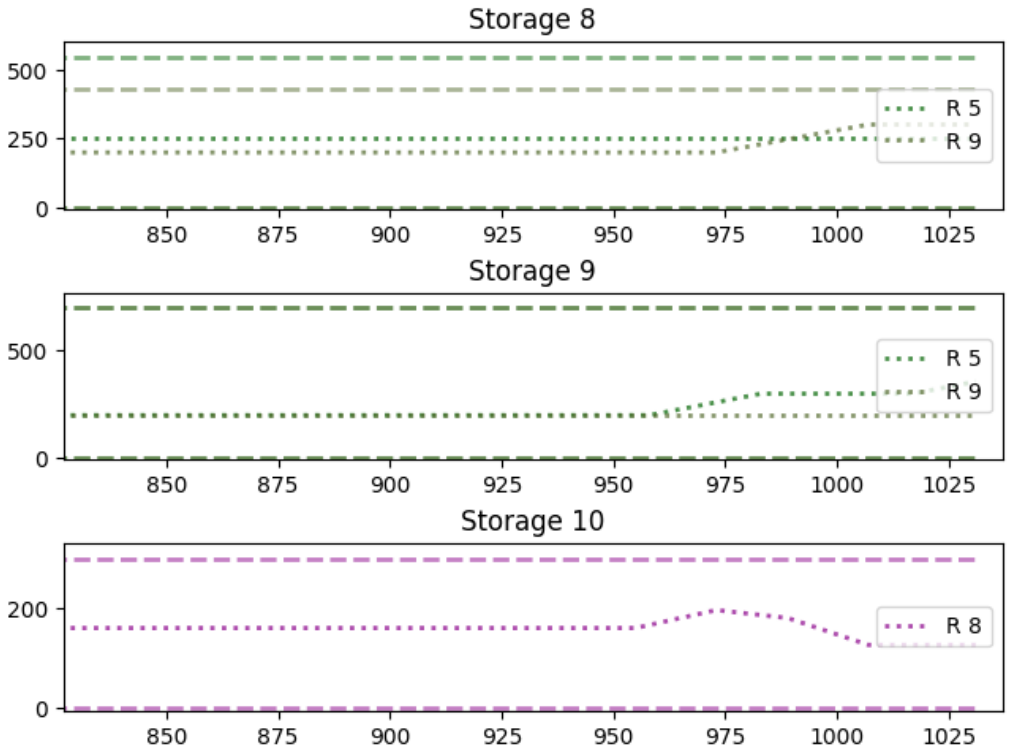


Figure 13: Monitoring of the third set of storages of the worst solution for the the clarifying example.

Each figure contains the plots of 4 storages at most, automatically managed; their vertical axis stands for the amount of resource, while their horizontal axis stands for the timeline, both in the relevant units of the input data; and each storage plot has 3 lines for each resource in the legend, with the corresponding color: one dotted line representing the amount flow, and two dashed ones indicating the related storage bounds. All the lower bounds in the clarifying example are set to 0, but they could be set greater pretending some kind of safety level. Moreover, in some plots, the bounds of two or more resources coincide and the colors are superposed (especially for the lower bounds).

4.4 2nd Optimization Layer

In the fourth place, featuring the orange blocks in Figure 2 and 3, there is an optional second optimization for those cases in which storage mismatches has been detected. Unfortunately, this approach was one of the last enhancements of the method during the development of the thesis, and owing to the fact that it is a complex algorithm with many tuning possibilities, the available time and knowledge was not enough to reach expertise on the subject. Consequently, the implementation of this last process has not been as polished and well implemented as desired. The lack of familiarity with the topic forced us to choose a software as easy as possible

to test some simple cases, but it looks like the convergence of the implemented algorithm is not getting good performance results. Despite of this, it has been possible to work with it, having found some solution for not too difficult situations such as the clarifying example. Next, there is the explanation about two implementation proposals developed in this thesis for a GA, though only one of them (the *2:1* one) ended up solving some mismatch situation.

4.4.1 GA implementation proposals

A GA belongs to the larger class of Evolutionary Algorithms (EA) and is based on a codification called genome, which is characteristic for each individual of a population, and it repeatedly modifies this population over successive generations by (i) altering some element, or gene, of the genome; and also (ii) by mixing random segments between individuals. Eventually, the population "evolves" toward an optimal solution, since the individuals are discriminated in each generation by what is known as (iii) a fitness function. These "evolution" relies on biologically inspired operators such as mutation (i), crossover (ii) and selection (iii).

Here are the two GA implementations proposed in this thesis. As their codification is based on the tasks, they are named according the number of elements related per task:

- ***2:1***: among the variables of a production planning and scheduling solution from the first optimization layer, there are the starting and ending times of each task, which are strongly related to the quantity produced by the same; due to the fact that each task also has an associated operating device with a default operating speed. Then, a genome g can be codified with the starting and ending times of all tasks like

$$g = \{t_{s_{111}}, t_{e_{111}}, ..., t_{s_{n_O n_{T_O} n_{T_{Op}}}}, t_{e_{n_O n_{T_O} n_{T_{Op}}}}\} \tag{14}$$

  This way, each task has two dedicated genes; and through crosses and mutations, these times, and consequently, the corresponding production quantities, will slightly vary. The fitness function for the selection between generations is founded on the respective makespan of each genome, aiming to converge in the minimum one. But feasibility is essential for the final solution, so that the solution validation in the previous process of the method is reused for the ultimate selection of the final production planning and scheduling solution. Moreover, the population of the new generations is no longer drawn from the CP optimisation model, so the production logics is no longer guaranteed either. This implies an extra validation of all the constraints included in Section 3.2.2 for the ultimate selection of the final solution.

  However, there are two prominent drawbacks related to this genome codification:

  1. To preserve the task-operating device assignments, the length of the genomes must be kept constant, because the unique correspondence is their position in each individual. This is limiting the initial population to the solutions obtained from the CP optimisation of a single orders group, which might be few. They are referred to in plural assuming that the solver not only returns the optimal solution, but some more feasible ones that may have found for each of the equivalent orders groups.

  2. According to 1., the length of the genomes is consistent, which leads to a number of

tasks also steady. This is limiting the probability of finding a solution that meets the storage restrictions.

- ***1:1***: trying to solve the drawbacks of the *2:1* codification, this one is based on strings, such that a unique gene is related to each task consisting of a predetermined chain of numbers and characters. This numbers and characters chain includes the minimum but essential information for each task, which corresponds to:
  - The starting time ($t_{s_{opi}}$).
  - The ending time ($t_{e_{opi}}$).
  - The associated operating device (j).
  - The order index to which it originally belongs (o), to relate it with the order product.

  And a genome g can be codified as follows

$$g = \{\text{“}t_{s_{opi}} \# t_{e_{opi}} \# j \# o\text{”}\} \quad \forall o, \forall p, \forall i \tag{15}$$

  where the # character is used as separator to be able to split each gene variables, for example, in a validation purpose.

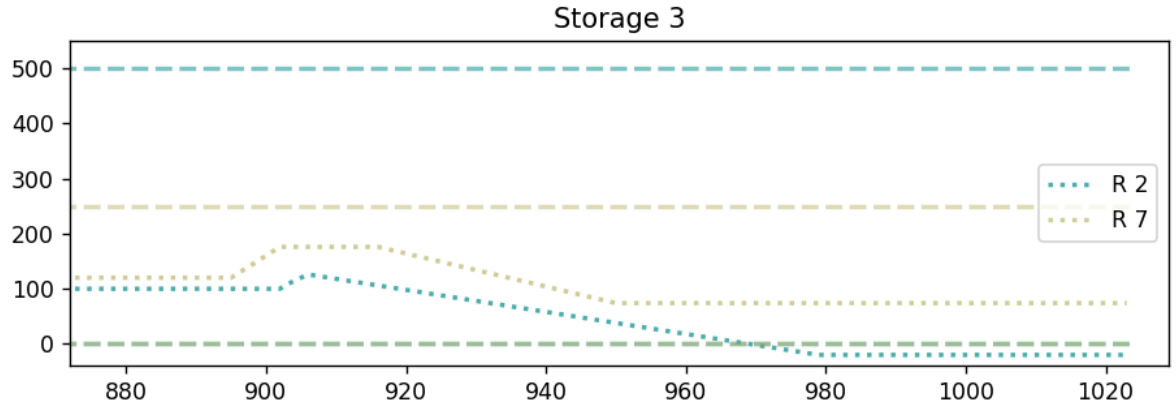
  In this case, the fitness function for the selection between generations is founded again on the respective makespan of each genome, aiming to converge in the minimum one. The feasibility checking, both for storage restrictions and production logic, is likewise engaged. As a contribution, the initial population limitation is minimized because every solution of every group of equivalent orders can be now considered to evolve, also avoiding a constant length of the genomes. And consequently, the number of tasks becomes variable. However, crossing and mutating the different individuals, many combinations of much more diverse orders jumble are obtained, becoming an important drawback when evaluating the probability of the resulting tasks arrangement of complying with the production formulas, or what is the same, satisfying the production logic validation.

4.4.2 GA implementation result

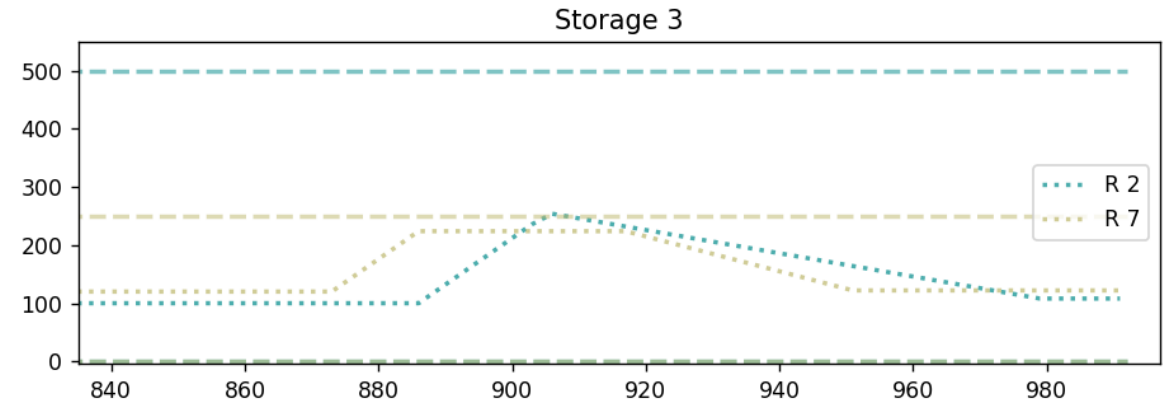
To verify the yield of the proposed implementations, a controlled case has been employed, i.e. a mismatch situation has been forced, to see if the process is able to face the problematic. It has been done by slightly moving the limits of the storages in the input data. And only the group of the equivalent orders corresponding to Table 9 was selected, because only the *2:1* codification is able to return a feasible solution for the controlled case, and its population must belong to a single orders configuration as explained. While the *1:1* one tends not to converge. This demonstrates that, with an optimal implementation, it could prove to be a successful approach.

All in all, after several simulation executions and adjustments, the *2:1* implementation with a few generations usually finds a feasible solution. So, taking a random execution in which the second optimisation layer is successful, Figure 14 shows the zoomed-in comparison of the resource flow in the concrete storage where the mismatch occurs (the resource 2, "R 2" in the

legend, in the Storage 3), before and after the GA optimization. Additionally, Figure 15 shows the Gantt chart of the production planning and scheduling solution where the mismatch was detected; Figures 17, 18 and 19 show their corresponding complete storages monitoring; Figure 16 shows the Gantt chart of the production planning and scheduling solution where the mismatch was already fixed by the GA; while Figures 20, 21 and 22 show their corresponding complete storages monitoring.



(a) Monitoring of the Storage 3 corresponding to the solution where the mismatch occurred.



(b) Monitoring of the Storage 3 corresponding to the new solution from the GA, where the mismatch not no longer occurs.

Figure 14: Storage 3 zoomed-in comparison.

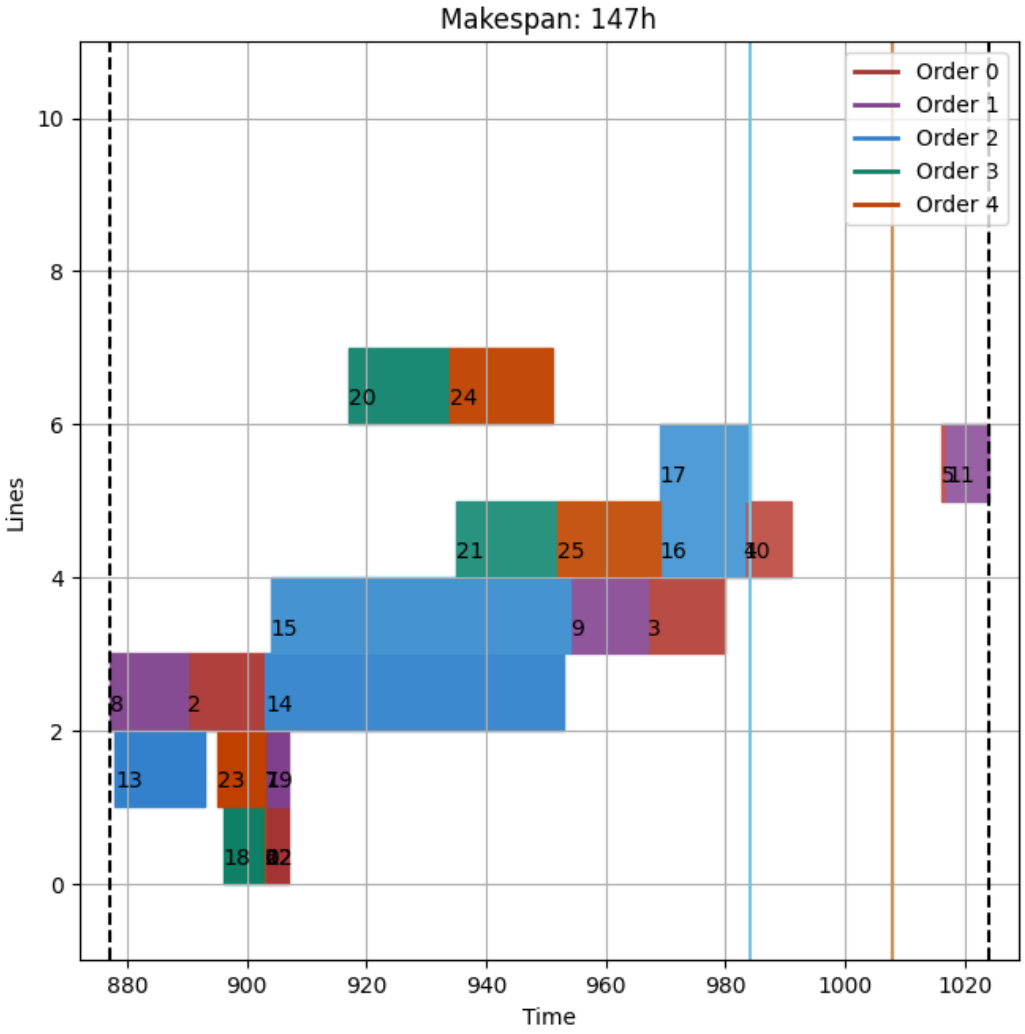


Figure 15: Gantt chart of the solution where the mismatch occurred.

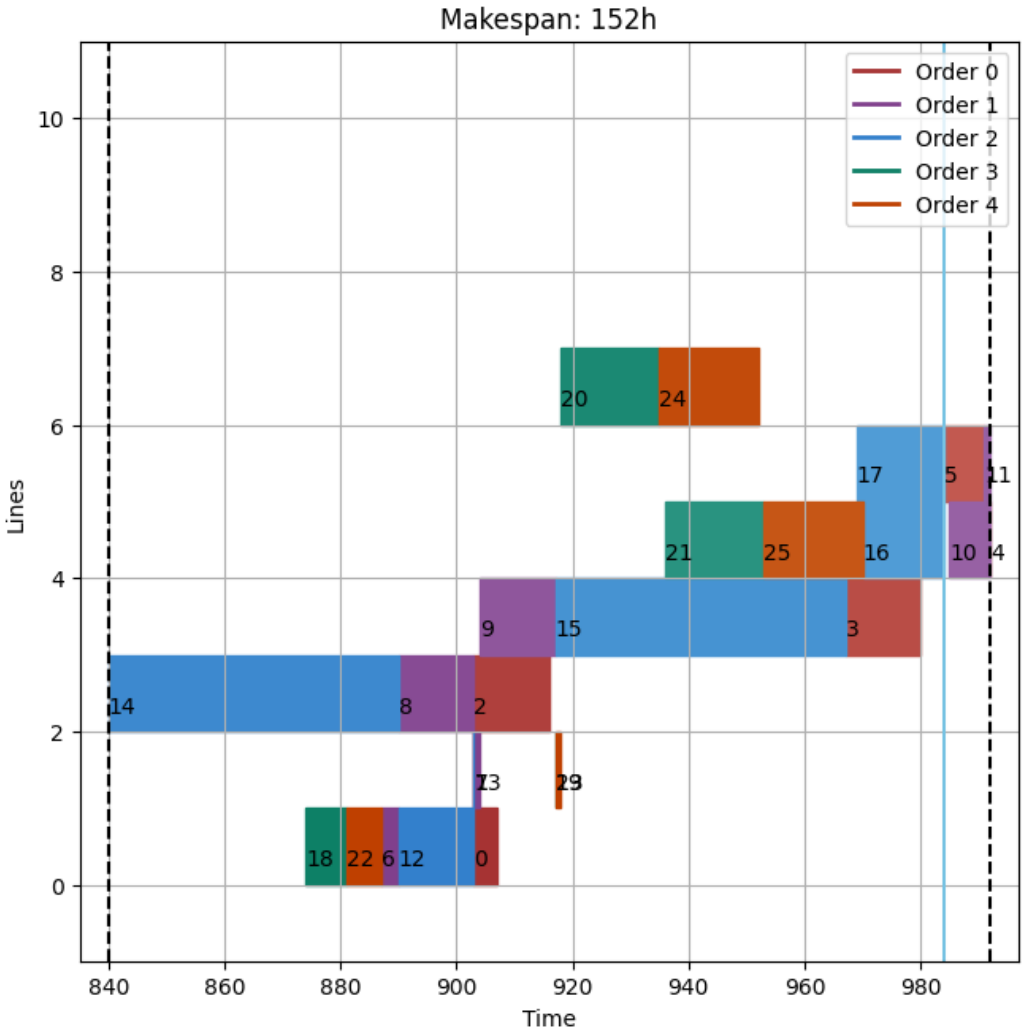


Figure 16: Gantt chart of the new solution from the GA, with the mismatch already fixed.

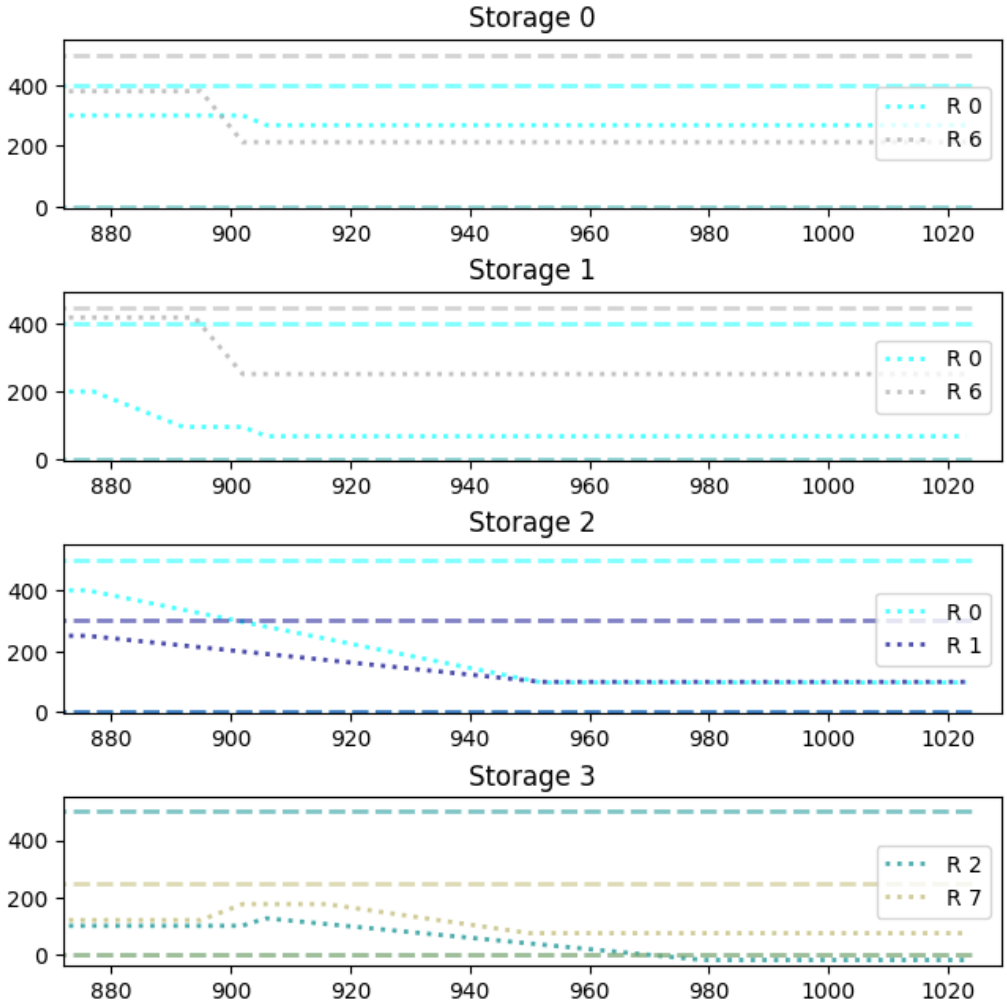


Figure 17: Monitoring of the first set of storages for the solution where the mismatch occurred.

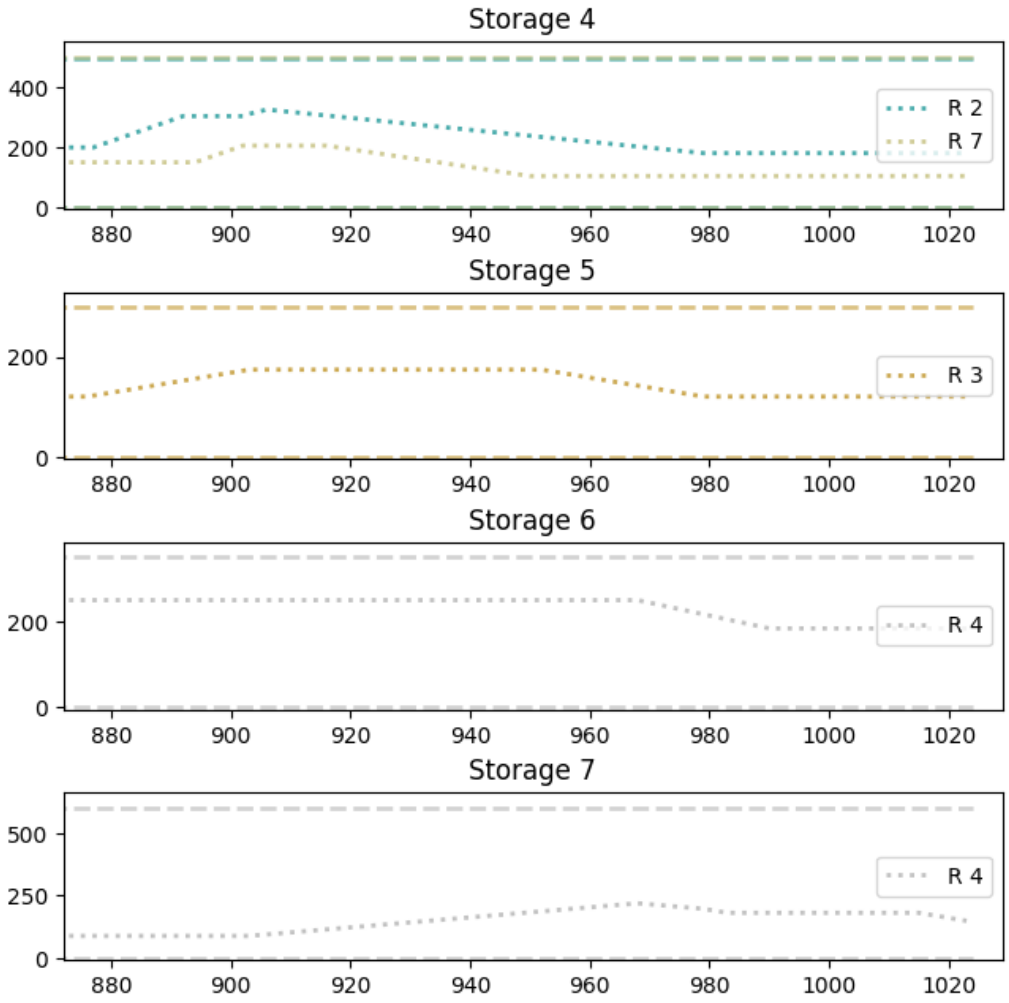


Figure 18: Monitoring of the second set of storages for the solution where the mismatch occurred.

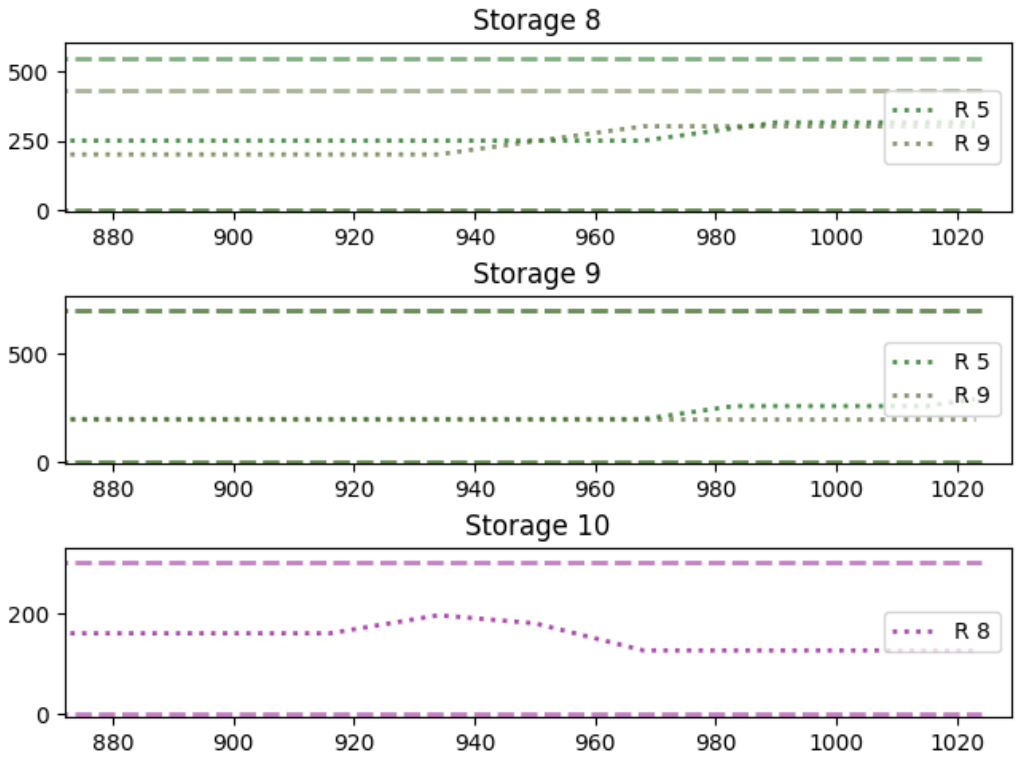


Figure 19: Monitoring of the third set of storages for the solution where the mismatch occurred.

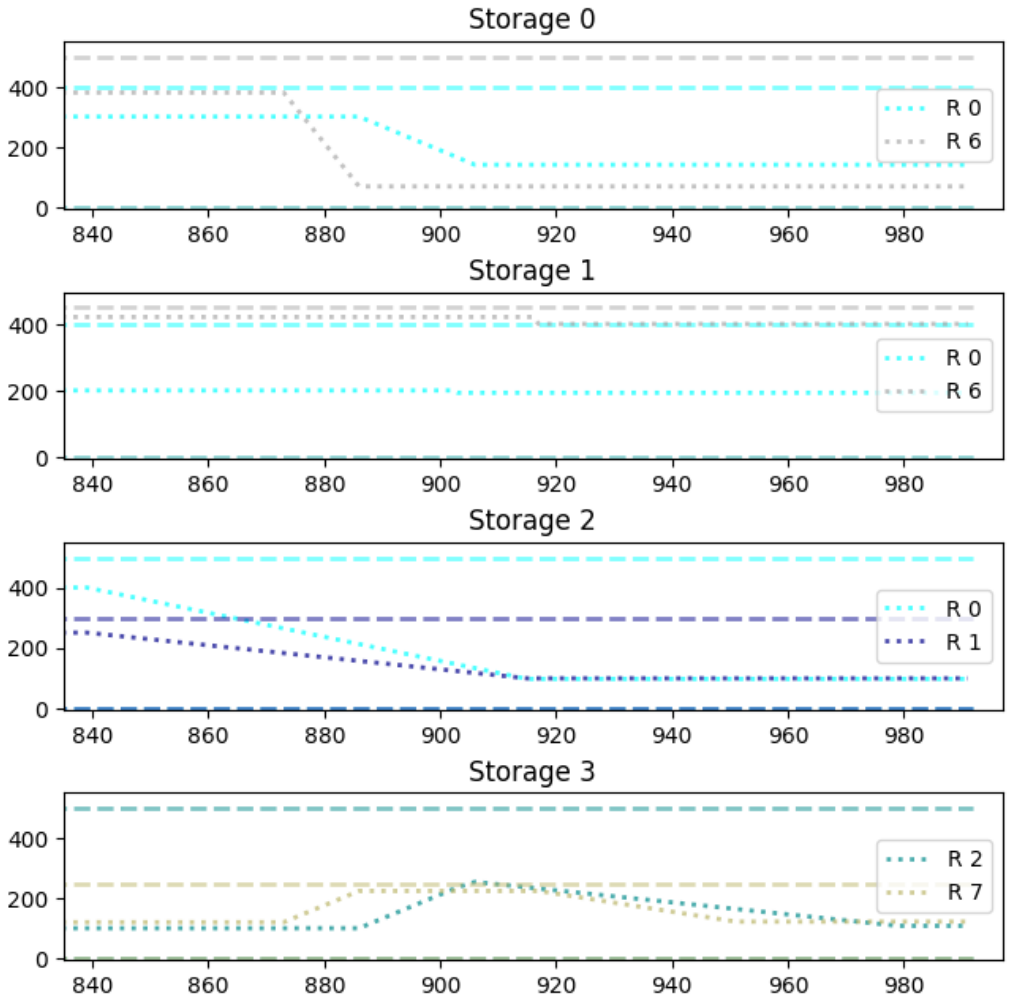


Figure 20: Monitoring of the first set of storages for the new solution from the GA, where the mismatch not no longer occurs.

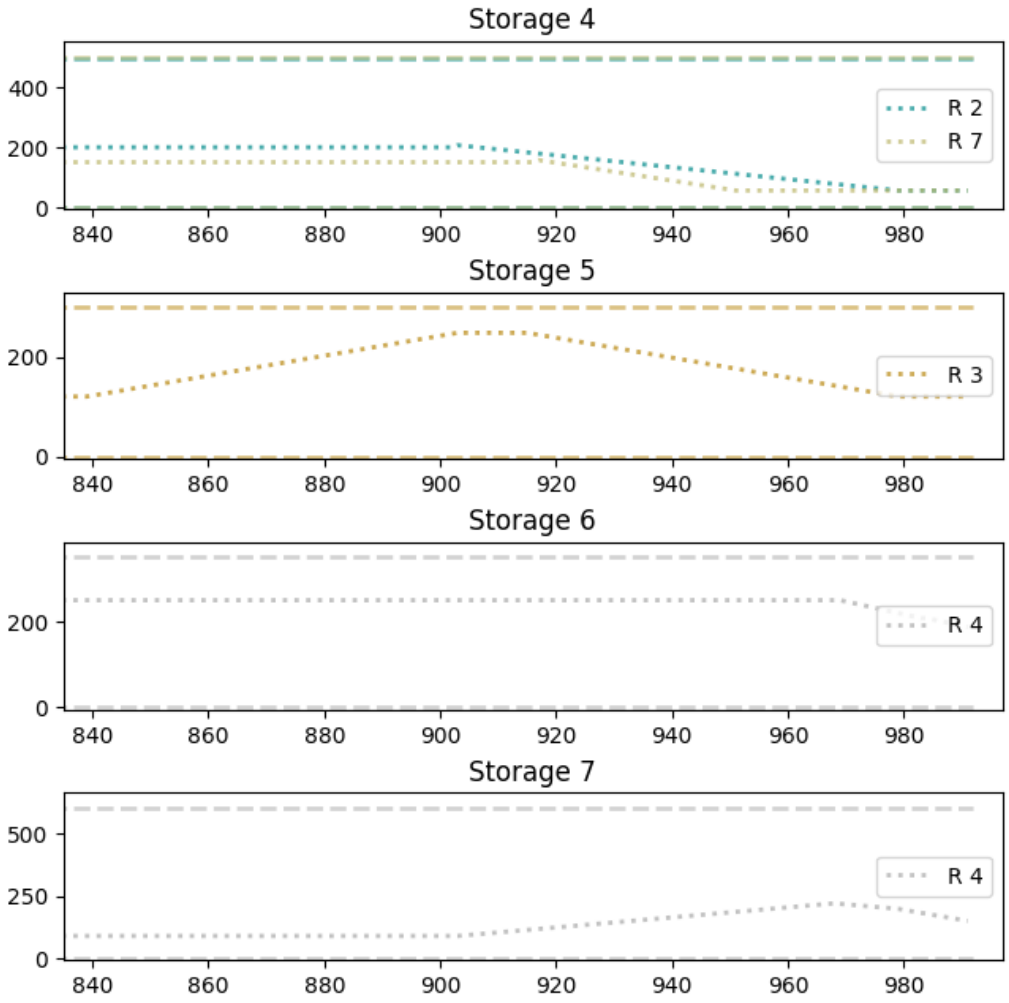


Figure 21: Monitoring of the second set of storages for the new solution from the GA, where the mismatch not no longer occurs.

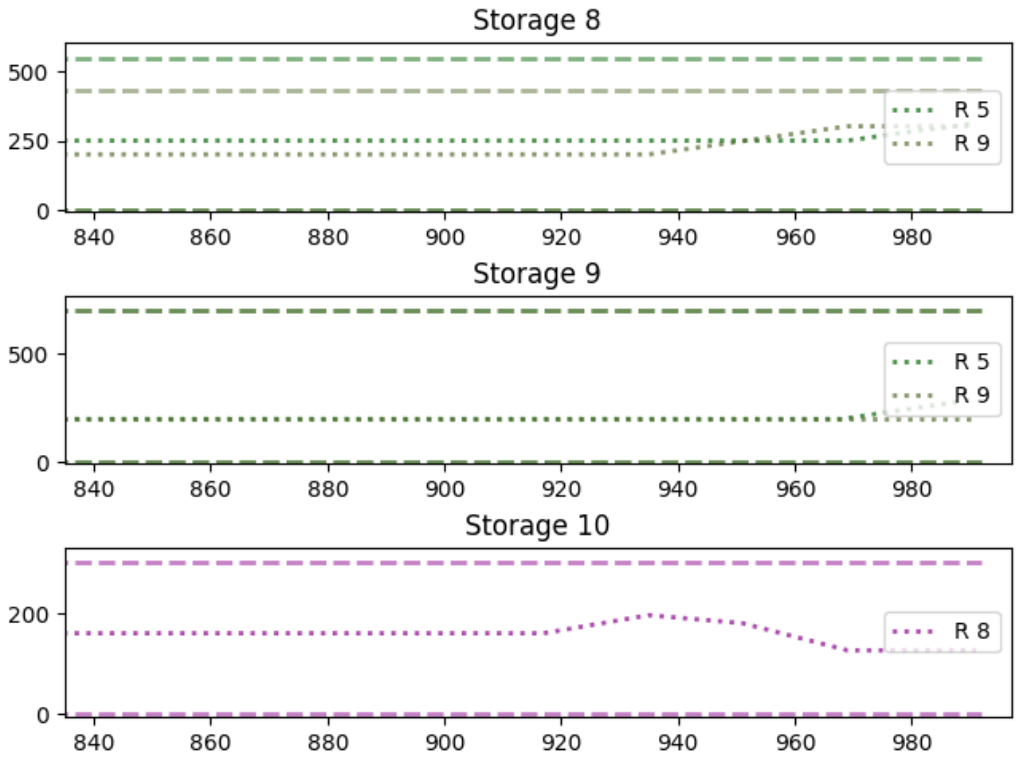


Figure 22: Monitoring of the third set of storages for the new solution from the GA, where the mismatch not no longer occurs.

4.5 Post-processing

This stage is the final one, featuring the blue blocks in Figure 2 and 3. Its function is as simple as needed, being the one in charge to translate the solutions obtained during the optimizations into something understandable and applicable by the user.

In this implementation, two modes of representing a solution have been opted for:

1. **Gantt chart**: which is a visual way to easily interpret the different orders realizations, distinguished by colors. For instance, the one of Figure 23.

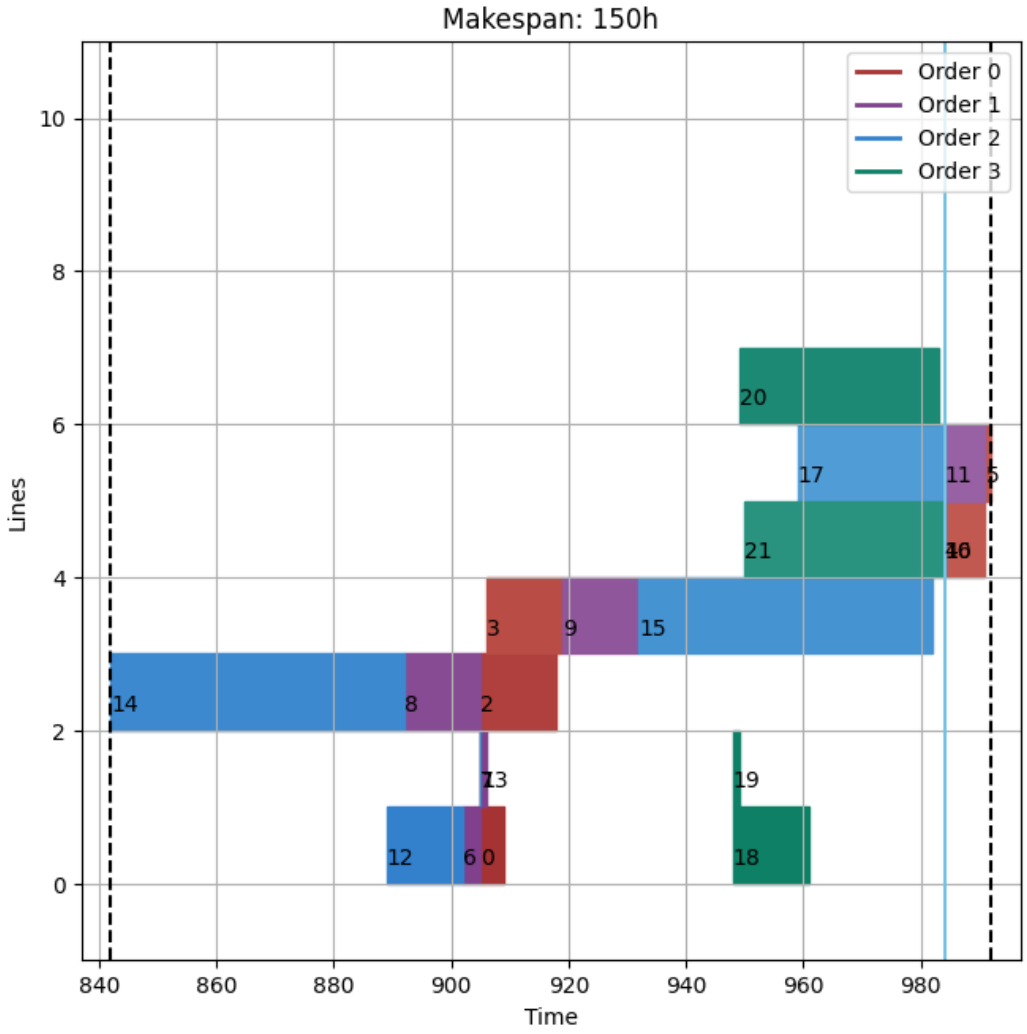


Figure 23: Gantt chart of a random solution for the clarifying example.

The horizontal axis of the chart, labeled with "Time", corresponds to the timeline in the relevant time units. Whereas the vertical axis, labeled with "Lines", corresponds to the unique identifiers of the operating devices, such that each entire row corresponding to an integer includes all the tasks performed in the relevant device.

A task is represented by a rectangle, the bottom-left corner of which is aligned with the corresponding operating device identifier in the vertical axis, and with the corresponding starting time in the horizontal axis; with a height equal to 1 (to only occupy the relevant row); a width equal to the task duration; and a number at the beginning of the rectangle indicating the unique identifier of itself.

In addition, there are some vertical lines to highlight some important moments, such as

the black-dashed ones, which emphasises the beginning (left one) and the ending (right one) of the makespan; and the solid-colored ones, which in correlation with the orders, emphasises their request times and delivery deadlines. Not all the orders deadlines are visible because the plot is adapted to display the smallest window containing the whole production planning and scheduling.

Sometimes, there are possible tasks included in the model that are discarded for the solver, by assigning them the same starting and ending time. Then, they are represented by a small vertical line somewhere. That is why could be some "ghost" task that overwrites some other task identifier, or there may be a lost line with a task identifier on some of the included Gantt charts.

2. **Print verbose**: which is a written way including much more details, but more tedious to interpret, since all the information related to each task of each order is printed out by the terminal. Figure 24 includes a capture corresponding to a representative piece of the printing style, which has the shown structure based on an indentation hierarchy according to the data type.

```
- Order: 3 (product: 9)
    Process 0: p5
        Task: 18
            Start: 888
            End: 895
            Duration: 7
            Quantity: 50
            Line: 0
        Task: 19
            Start: 947
            End: 947
            Duration: 0
            Quantity: 0
            Line: 1
    Process 1: p6
        Task: 20
            Start: 956
            End: 973
            Duration: 17
            Quantity: 50
            Line: 6
    Process 2: p7
        Task: 21
            Start: 974
            End: 991
            Duration: 17
            Quantity: 50
            Line: 4
```

Figure 24: Capture of a piece of the verbose printed out by the terminal.

4.6 Requirements

This section is devoted to summarize the used software. As introduced above, the program has been developed in Python, concretely 3.9, so to implement this tool, it is required to have a python interpreter of the corresponding version installed. It was executed in a Windows 10 Pro OS. In addition, the program involves the following packages:

- pillow 9.2.0
- absl-py 1.2.0
- cycler 0.11.0
- deap 1.3.1
- et-xmlfile 1.1.0
- fonttools 4.34.4
- kiwisolver 1.4.4
- matplotlib 3.5.2
- networkx 2.8.5
- numpy 1.23.1
- openpyxl 3.0.10
- ortools 9.3.10497
- packaging 21.3
- pandas 1.4.3
- pip 21.2.3
- protobuf 4.21.4
- pyparsing 3.0.9
- python-dateutil 2.8.2
- pytz 2022.1
- scipy 1.9.0
- setuptools 57.4.0
- six 1.16.0

5 Budget

This chapter aims to account the cost incurred in the completion of this thesis. According to the fact that it is purely theoretical research, and its application has been carried out programmatically in simulation, the budget does not become very complicated; including only the following aspects:

- **Cost related to necessary equipment**: the equipment that this work needs for the implementation is reduced to a computer, complying with the minimum specifications required to work with the software listed in the Section 4.6. Then, the associated cost to this aspect would be the corresponding amortization (C_A) of the same, which corresponds to the acquisition price multiplied by the percentage of usage time with respect its expected life span. In this case, the acquisition price of the laptop was 1700, the expected life span is $6\ \ years$, and considering $6\ \ months$ of time devoted to the thesis, $C_A = 1700 \cdot \frac{0.5}{6} = 121,43\ €$

- **Cost related to programming development**: the development budget would include both the cost related to software licences and the cost associated to the programmer's work, proportional to the developing time invested. In this case, all the implemented software is open-source, i.e. free; and the wage for the programmer was considered to be $17\ €/h$, which is a balanced salary for a worker who should be considered well-qualified. So, considering an investment of around $900\ h$ of developing time, the total amount of money regarding the development is $15300\ €$.

- **Implicit expenses**: obviously, there were some other costs related to energy utilisation, in terms of the working accommodation (e.g. lighting, air-conditioning, etc.), as well as some expenses of the space where the thesis was carried out. However, this calculation is much more complex because the development has been conducted from many different places (from shared to private), and under different external conditions; and it is not really going to make a big difference to the total budget either, so it is neglected.

Table 16 summarizes the resulting total cost.

Table 16: Budget

Concept	**Cost [€]**
Equipment	121,43
Development	15300,00
Total	15421,43

6 Social, Economic and Environmental Impact

The project sustainability is discussed in this chapter. Although it is just as subjective as the budget, we can identify some evident impacts relative to the realisation and contribution of this thesis:

- A notable positive impact for the industry related to this work is the reduction of so many factors such as: transport, factory-related consumption like both storage and production cost and pollution, and even embodied energy of the resources that will be spared; on the basis of the resources optimisation that achieves the proposed approach. However, it is very difficult to quantify since it is a general method that encompasses many types, dimensions and configurations of manufacturing plants, leading to a countless number of variables to assess impact.

- Not standing out that much, but equally obvious is the impact, now rather negative, concerning the work development. Basically, we should contemplate the carbon footprint of the utilized assets, such from the laptop used to the working accommodation relative supplies (e.g. lighting, air-conditioning, etc.). With the same problem as in the previous chapter, this calculation is complex because the different places and conditions under which the development has been conducted; and again it is not really critical from this point of view.

To sum up, the real impact of the work is minimum so far, owing to the fact that the thesis is nothing more than research and simulated testing. In order to be able to evaluate the impact more accurately, it should be studied from an implementation point of view in a real application.

7 Conclusions

The proposed approach is promising because from scratch was designed based on the real needs of the current industry in the area, mainly dedicated to the food industry. More specifically, the influence to this work comes from flour factories to any semi-continuous process manufacturing, such as dairy production and the like. The access to this information was possible thanks to a digitisation and industrial monitoring company which supports the project, because it considers the need for a tool capable of offering an optimal production planner for production systems; and the present ones are not sufficiently matured.

The interaction with the program is one of the features that has received the most attention during the development of this thesis, since as mentioned at the very beginning, it is an important part for the adaptability and usability of the method. This is an interesting property that adds a lot of value to the same, since a typical weakness that has been recognized of most current planning and scheduling approaches for process industries is that their design and implementation for real applications is conditioned to very specific solutions and very tailored algorithms, which ends up being to much tedious and expensive.

Besides, the Constraint Programming modelling has been proven to be particularly suited to this problem, due to the fact that it provides a lot of expressive potential, accepting the coexistence of variables and constraints of different natures, i.e. both mathematical relations and logical operations. However, up to now, it has not been possible to integrate the question of buffers and storage management into the mathematical framework, although it is one of the most open matters in general terms. That is why, this in particular, is one of the issues with which we will continue working on. Consequently, this fact forced us to look for an alternative to the problem for the proper deployment of this method, which corresponded to the use of a Genetic Algorithm metaheuristics. Despite the fact that the second optimisation layer still needs significant refinements, the approach is very encouraging if it is studied in depth.

8 Future work

Following, to point the path forward for further research, a collection of proposals is suggested:

- To deepen Genetic Algorithms in an attempt of achieving a proper algorithm for JSSPs with optimal performance.
- To incorporate the resource limitation to the optimization model aiming to achieve a more accurate and robust solution.
- To implement and test the prototypes in a real manufacturing plant to assess their performance as well as their impact.
- To extend the production planning and scheduling modelling to an upper layer of logistics, trying to reach a high level of coordination between production and pick-up schedules for distribution.

To conclude, the work with this method is intended to be continued with the development of a PhD thesis, together with the aforementioned company supporting the project.

References

Mathematical Programming Approaches, pages 183–206. Springer US, Boston, MA, 2007. ISBN 978-0-387-68155-9. doi: 10.1007/978-0-387-68155-9_8. URL https://doi.org/10.1007/978-0-387-68155-9_8.

E. Kondili, C.C. Pantelides, and R.W.H. Sargent. A general algorithm for short-term scheduling of batch operations—i. milp formulation. *Computers & Chemical Engineering*, 17(2):211–227, 1993. ISSN 0098-1354. doi: https://doi.org/10.1016/0098-1354(93)80015-F. URL https://www.sciencedirect.com/science/article/pii/009813549380015F. An International Journal of Computer Applications in Chemical Enginnering.

Georgios M. Kopanos and Luis Puigjaner. *Solving Large-Scale Production Scheduling and Planning in the Process Industries*. 2019.

Stephan Kreipl and Michael Pinedo. Planning and scheduling in supply chains: An overview of issues in practice. *Production and Operations Management*, 13(1):77–92, 2004. doi: https://doi.org/10.1111/j.1937-5956.2004.tb00146.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1937-5956.2004.tb00146.x.

Christos T. Maravelias. Mixed-time representation for state-task network models. *Industrial & Engineering Chemistry Research*, 44(24):9129–9145, 2005. doi: 10.1021/ie0500117. URL https://doi.org/10.1021/ie0500117.

C.A Méndez, G.P Henning, and J Cerdá. Optimal scheduling of batch plants satisfying multiple product orders with different due-dates. *Computers & Chemical Engineering*, 24(9):2223–2245, 2000. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(00)00584-6. URL https://www.sciencedirect.com/science/article/pii/S0098135400005846.

C.A. Méndez, G.P. Henning, and J. Cerdá. An milp continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Computers & Chemical Engineering*, 25(4):701–711, 2001. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(01)00671-8. URL https://www.sciencedirect.com/science/article/pii/S0098135401006718.

Carlos A. Méndez, Jaime Cerdá, Ignacio E. Grossmann, Iiro Harjunkoski, and Marco Fahl. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, 30(6):913–946, 2006. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2006.02.008. URL https://www.sciencedirect.com/science/article/pii/S0098135406000287.

J. Roslöf, I. Harjunkoski, J. Björkqvist, S. Karlsson, and T. Westerlund. An milp-based reordering algorithm for complex industrial scheduling and rescheduling. *Computers & Chemical Engineering*, 25(4):821–828, 2001. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(01)00674-3. URL https://www.sciencedirect.com/science/article/pii/S0098135401006743.

N.V. Sahinidis and I.E. Grossmann. Minlp model for cyclic multiproduct scheduling on contin-

uous parallel lines. *Computers & Chemical Engineering*, 15(2):85–103, 1991. ISSN 0098-1354. doi: https://doi.org/10.1016/0098-1354(91)87008-W. URL https://www.sciencedirect.com/science/article/pii/009813549187008W.

K.L. Yee and N. Shah. Scheduling of multistage fast-moving consumer goods plants. *Journal of the Operational Research Society*, 48(12):1201–1214, 1997. ISSN 0098-1354. doi: https://doi.org/10.1057/palgrave.jors.2600482. URL https://doi.org/10.1057/palgrave.jors.2600482. An International Journal of Computer Applications in Chemical Enginnering.

K.L. Yee and N. Shah. Improving the efficiency of discrete time scheduling formulation. *Computers & Chemical Engineering*, 22:S403–S410, 1998. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(98)00081-7. URL https://www.sciencedirect.com/science/article/pii/S0098135498000817. European Symposium on Computer Aided Process Engineering-8.

Appendix

The following listing corresponds to the JSON input file of the orders for the clarifying example addressed in the Implementation and Results section.

```
{ "factory": "FacNumNone",

  "orders": [
    { "product": 5,
      "request_date": "21/04/2022",
      "delivery_date": "01/06/2022",
      "quantity": 100
    },
    { "product": 9,
      "request_date": "21/04/2022",
      "delivery_date": "02/06/2022",
      "quantity": 100
    },
    { "product": 5,
      "request_date": "21/04/2022",
      "delivery_date": "03/06/2022",
      "quantity": 50
    }
  ]
}
```

Whereas the following one corresponds to the JSON input file corresponding to the factory description for the clarifying example addressed in the Implementation and Results section.

```
{ "settings":
  { "multiple_solutions": 1,
    "multiple_sol_verbose": 0,
    "show_verbose": 0,
    "main_weights": {"time": 10, "cost": 1, "freshness": 1},
    "delivery_margin": 49,
    "max_orders_division": 2,
    "solver_timeout_s": 30
  },

  "lines": [
    { "id": 0,
      "type": ["p1", "p5"],
      "speed": 8,
      "cost": 2,
      "available": 1,
      "maintenance": 2,
      "in_storages": [0],
```

```
      "out_storages": [3]
    },
    { "id": 1,
      "type": ["p1", "p5"],
      "speed": 7,
      "cost": 3,
      "available": 1,
      "maintenance": 2,
      "in_storages": [1],
      "out_storages": [4]
    },
    { "id": 2,
      "type": ["p2"],
      "speed": 2,
      "cost": 4,
      "available": 1,
      "maintenance": 2,
      "in_storages": [2],
      "out_storages": [5]
    },
    { "id": 3,
      "type": ["p3"],
      "speed": 2,
      "cost": 4,
      "available": 1,
      "maintenance": 2,
      "in_storages": [3,4,5],
      "out_storages": [6,7]
    },
    { "id": 4,
      "type": ["p4", "p7"],
      "speed": 3,
      "cost": 4,
      "available": 1,
      "maintenance": 2,
      "in_storages": [6,10],
      "out_storages": [8]
    },
    { "id": 5,
      "type": ["p4"],
      "speed": 4,
      "cost": 4,
      "available": 1,
      "maintenance": 2,
      "in_storages": [7],
      "out_storages": [9]
```

```
    },
    { "id": 6,
      "type": ["p6"],
      "speed": 3,
      "cost": 8,
      "available": 1,
      "maintenance": 2,
      "in_storages": [3,4],
      "out_storages": [10]
    }
  ],

  "storages": [
    { "id": 0,
      "resources": [0,6],
      "curr_capacities": [300,380],
      "max_capacities": [400,500]
    },
    { "id": 1,
      "resources": [0,6],
      "curr_capacities": [200,420],
      "max_capacities": [400,450]
    },
    { "id": 2,
      "resources": [0,1],
      "curr_capacities": [400,250],
      "max_capacities": [500,300]
    },
    { "id": 3,
      "resources": [2,7],
      "curr_capacities": [100,120],
      "max_capacities": [500,250]
    },
    { "id": 4,
      "resources": [2,7],
      "curr_capacities": [200,150],
      "max_capacities": [500,505]
    },
    { "id": 5,
      "resources": [3],
      "curr_capacities": [120],
      "max_capacities": [300]
    },
    { "id": 6,
      "resources": [4],
      "curr_capacities": [250],
```

```
      "max_capacities": [350]
    },
    { "id": 7,
      "resources": [4],
      "curr_capacities": [90],
      "max_capacities": [600]
    },
    { "id": 8,
      "resources": [5,9],
      "curr_capacities": [250,200],
      "max_capacities": [550,430]
    },
    { "id": 9,
      "resources": [5,9],
      "curr_capacities": [200,200],
      "max_capacities": [700,700]
    },
    { "id": 10,
      "resources": [8],
      "curr_capacities": [160],
      "max_capacities": [300]
    }
  ],

  "formulas":{
    "5": { "p1": ["p3"],
           "p2": ["p3"],
           "p3": ["p4"],
           "p4": []
    },
    "9": { "p5": ["p6"],
           "p6": ["p7"],
           "p7": []
    }
  },

  "processes": {
    "p1": {
      "in_resources": [0],
      "in_ratio": [1],
      "out_resource": 2
    },
    "p2": {
      "in_resources": [0,1],
      "in_ratio": [2,1],
      "out_resource": 3
```

```
    },
    "p3": {
      "in_resources": [2,3],
      "in_ratio": [1,1],
      "out_resource": 4
    },
    "p4": {
      "in_resources": [4],
      "in_ratio": [1],
      "out_resource": 5
    },
    "p5": {
      "in_resources": [6],
      "in_ratio": [3],
      "out_resource": 7
    },
    "p6": {
      "in_resources": [7],
      "in_ratio": [1],
      "out_resource": 8
    },
    "p7": {
      "in_resources": [8],
      "in_ratio": [1],
      "out_resource": 9
    }
  }
}
```