Combined planning and scheduling in a divergent production system with co-production

Gaudreault, J., Frayret, J.-M., Rousseau, A., D'Amours, S.,

May 2008 (revised)

Working Paper DT-2006-JMF-1

Research Consortium in e-Business in the Forest Products Industry (FOR@C)

Interuniversity research Center on Enterprise Networks, Logistics and transportation (CIRRELT),

Université Laval, Québec, Canada

FOR@C
DE LA FORÊT AU CLIENT

CIRRELT

Combined planning and scheduling in a divergent production system with co-production

Jonathan Gaudreault[a*], Jean-Marc Frayret[a,b,c], Alain Rousseau[d], Sophie D'Amours[a,b,c]

[a] *FORAC Research Consortium, Pavillon Adrien-Pouliot, Université Laval, Quebec (QC), Canada, G1K 7P4*

[b] *CIRRELT, Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport*

[c] *CRB, Centre de Recherche sur le Bois, Université Laval*

[d] *Synaptas inc., 2740 rue Einstein, Quebec (QC), Canada, G1P 4S4*

**corresponding author*

Abstract

Many research initiatives carried out in production management consider process planning and operations scheduling as two separate and sequential functions. However, there are some contexts where the two functions must be better integrated. As is the case in divergent production systems with co-production (i.e. producing different products at the same time from a single product input) when alternative production processes are available. This paper studies such a context.

The studied application concerns drying and finishing operations in a softwood lumber facility. This situation is addressed using a single model that simultaneously performs process planning and scheduling. We propose two alternative formulations. The first one is based on mixed integer programming (MIP) and the second on constraint programming (CP). The two approaches are compared according to their capacity to be used in a real-time agent-based supply chain planning system.

Keywords: Integrated Planning and Scheduling; Mathematical Programming; Constraint Programming, Wood Drying and Finishing; Anytime algorithm

1. Introduction

This paper addresses a real industrial process planning and operations scheduling problem from the softwood lumber industry. More specifically, it deals with the planning and scheduling of drying and finishing operations. This production system is characterized as being (1) a divergent system with co-production (i.e., it produces different products at the same time from a single product at input), (2) with alternative production processes.

This section first introduces the nature of these processes and explains the complexity erasing from the intertwined problems of deciding what processes to implement, when to schedule them and on what kiln dryers. This section also introduces the objective and the experimentation approach used to address this planning and scheduling problem.

1.1 Wood drying and finishing

Softwood drying is a transformation operation which aims at decreasing the lumber moisture content in order to meet customer requirements. These requirements are usually specified by industry standards, although some customers may require specific levels of moisture content. Softwood lumber drying is a rather complex process to carry out. It takes days and is done in batches in large kiln dryers. Under certain circumstance, special sections of the wood yard may be used to perform air drying. Air drying, which precedes kiln drying, may take several weeks but allows for the reduction of the drying time in the kiln. The same principle applies to the time spent in the wood yard after kiln drying. It is called air equalizing. Air drying and equalizing also play a role in increasing the overall quality of the finished products.

Once dry, lumber is planed, sorted and trimmed. These operations are referred to in this paper as the finishing process. After being planed, each piece of lumber is sorted according to its grade (i.e., quality) with respect to its moisture content and its physical defects (most defects are not perceivable before planing). A piece of lumber may be trimmed in order to produce a shorter lumber of a higher grade. The process is usually optimized to produce the products with the greatest market value, with no consideration for actual customer demand. This causes the finishing process to produce multiple product types at the same time (co-production) from a single product type at input (divergence). It is important to note that co-production cannot be avoided. It is embedded within the process of transformation. For a given batch of lumber to be dried and finished, the resulting mix of products (each product type is defined by a combination of length and grade) is a function of the way both drying and finishing processes were carried out.

In brief, lumber drying and finishing can be described as a four stage transformation process which includes air drying, kiln drying, air equalizing and finishing. For a given batch, there are different possible operations at each stage (see example in Fig. 1). For air drying and equalizing, the different possible operations are mostly differentiated according to their durations. For kiln operations, they differ from one another according to air temperature and humidity parameters. These parameters form a "kiln program". Each of those kiln programs may be considered as a distinct operation.

Each allowed combination of operations (i.e. each path in Fig. 1) defines an alternative process that produces a specific mix of output products and has a different usage rate of resources.

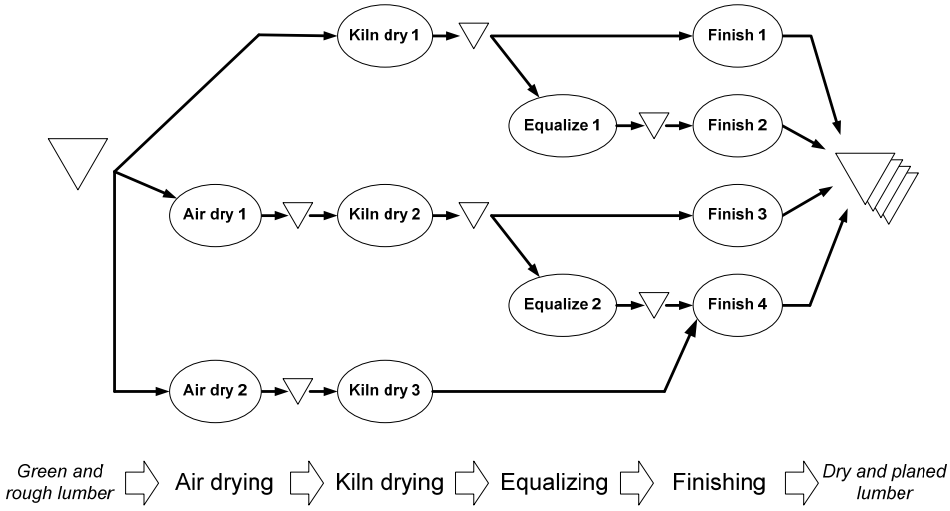

Fig. 1. Alternative processes for a specific lot of lumber

A graph similar to Fig. 1 must be defined for each product type that can be processed by the drying and finishing facility. Most sawmills allow the processing of batches that contain different but compatible product types. In industry, these allowed combinations are called the "kiln loading patterns". Again, alternative processes must be defined for each.

In order to design a planning system for drying and finishing operations, some characteristics of the problem need to be taken into account:

- Due to co-production, one batch of raw materials will contribute to fulfilling many customer orders for different product types, even if the batch contains only one product type at the beginning of the process;
- As the volume of each customer order for a specific product is usually larger than the amount that can be produced with one single batch, many batches are usually needed to fulfill any particular need;
- The capacity of a process to satisfy customer orders is not only linked to the co-product output mix associated with the process, but also to the moment the process can be done (scheduling). With this kind of problem, process planning and scheduling must be tightly integrated.

1.2 Objective and research methodology

The models and experimentation results provided in this paper aim to address this planning situation, and consequently to simultaneously perform process planning and scheduling. This is sometimes referred to as *combined planning and scheduling* [1]. In order to do so, we propose two alternative models. The first one is based on mixed integer programming (MIP). The other one is based on constraint programming (CP), for which, we developed a specific search procedure. The two approaches are then compared using real industrial data from a lumber producer in eastern Canada. In particular, we compare their capacity to provide good quality solutions in time. These models have been implemented within an agent-based experimentation platform described in [2], with which these experimentation have been carried out.

This paper is organized as follows. Section 2 presents a review of the literature. Section 3 then elaborates a mathematical and a constraint programming model to solve the discussed problem. Section 4 discusses the quantitative evaluation and comparison of these two approaches, according to the real-time expectations presented in section 1.1 and section 5 presents our conclusions.

2. Literature revue

This review of the literature includes three sub-sections in order to address the main features of the proposed production planning problem: divergence and co-production (2.1), integration of planning and scheduling (2.2), and the specific applications of wood drying operations planning (2.3).

2.1 Divergence and co-production

In the manufacturing problem previously introduced, lumber of the same type are processed and transformed into a number of distinct finished product types. Such a divergent process refers to a type of plant that is known as the V-Plant type. This subject is studied by Umble [3] who presents different types of plants: V, A and T plants and gives some examples of industries for each one. The V plant is characterized by the existence of divergent points in the manufacturing process. In this way, the number of finished goods is large compared with the number of raw materials or component parts. V-Plants are characterized with (1) a number of end items that is large compared to the number of raw materials; (2) all end items sold by the plant are processed in essentially the same way; and (3) the equipment is generally capital intensive and highly specialized.

Umble points out that a focus on efficiency may cause material to be processed through V-plants in large batch size resulting in excess work-in-process and finished good inventories and inflated lead times, a situation which is common in the softwood lumber industry.

In many V-Plants, the flow is divergent but the plant operates as if there is a switch that allows operators to choose which product type to produce. In the context of drying and finishing, we face both divergence and co-production, because we produce many product types at the same time from a given product. There are some (but few) industries facing such problems, for example some chemical processes, meat cutlers or petroleum refineries. Vila *et al.* [4] propose mathematical formulation for divergent processes in the specific application of the softwood lumber sector.

2.2 Integration of process planning and operations scheduling

Many research initiatives carried out in production management consider process planning (i.e., a set of sequentially interdependent operations to be carried out) and operations scheduling (i.e., a set of machine/operation allocations in time) as two separate functions.

In a context with no co-production, it is possible to specify a distinct process plan to fulfill each job. To simplify the overall planning process, a plan is usually specified independently before operations are scheduled. In practice, this is usually handled through what Myers *et al.* [5] refer to as an *iterative waterfall* model (Fig. 2), where (1) process planning and operations scheduling are carried out sequentially and independently, and (2) a new process plan is generated when a problem is encountered during scheduling. However, process planning and scheduling are interdependent operations. Indeed, on the one hand, the performance of a manufacturing system to carry out a process plan depends on machine capabilities and availabilities. On the other hand, the performance of a

manufacturing system to carry out a schedule depends on the ability of the process plans to avoid creating bottleneck machines. Therefore, process plans should be generated taking into account machine capabilities and their available capacity in order to identify which machine should carry out which operations to more efficiently use available capacity.

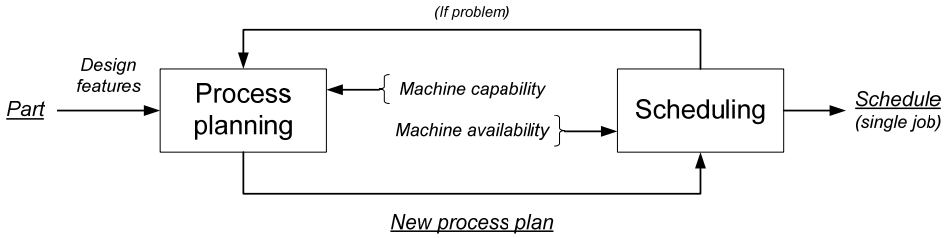

Fig. 2. Iterative waterfall process planning

A common approach in the field of operations management to solve such a problem, referred to as *non-linear process planning* (Fig. 3) in [6], is to first identify alternative process plans based on the design features of the product to manufacture and the capabilities of the machines available in the shop. When production must take place, each process plan is evaluated taking into account the individual schedule of each machine. A particular feasible plan and its corresponding schedule are then selected either automatically or interactively by the user who will carry out the plan. Many authors ([7-10], among others) exploit this general approach to iteratively schedule jobs, one after the other. Because process plan alternatives are designed without any references to real time shop status, the main drawback of this general approach is related to the possible large number of process plan alternatives needed to produce efficient operations schedules. Furthermore, some of these plans may be infeasible with regard to the real time shop status.

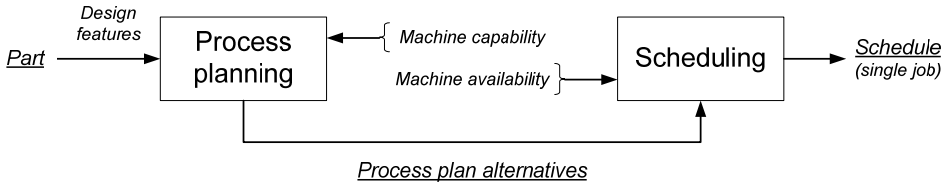

Fig. 3. Non-linear process planning

Another generic approach, referred to as *closed-loop process planning* (Fig. 4) in [6], is to generate a feasible process plan using the real-time status of the machines in the shop. Khoshnevis and Chen [11] propose a solution based on this generic approach. For each part that can be manufactured, two lists are dynamically maintained. The first list contains the machines which are available to process the part, and the second list contains the part design features that can be processed on each machine. When a part is needed, a heuristic algorithm generates a process plan by matching elements of the two lists to produce all the design features of that part. In this way, the process plan is generated using the real time status of the machines in the shop. However, production still needs to be scheduled. As with the non-linear process planning approach, in these approaches machine utilization is not taken into account to generate a process plan. Consequently, as a bottleneck machine may appear during scheduling, the resulting schedule may be inefficient or even infeasible.

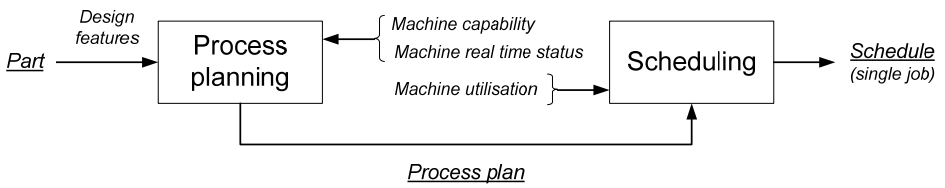

Fig. 4. Closed-loop process planning

In order to reduce this drawback, another generic approach referred to as *distributed process planning* in [6], consists in simultaneously generating the process plan of a job along with the scheduling of the identified operations. Similarly, processing requirements are identified during a preplanning phase by recognizing design features and analyzing the machine processing potential based on their capabilities. However, here operation/machine allocation alternatives are investigated during a pairing planning phase using the actual or estimated availability of the machines. For instance, Huang *et al.* [12] propose a progressive approach of integrated process planning and scheduling (Fig. 5). First, process plan alternatives are generated off-line. Next, for each part to be manufactured a process plan is selected based on the shortest manufacturing anticipated lead time by tentatively allocating operations to machine groups. To do so, operations processing and waiting times are roughly estimated for groups of similar machines. Then, at a detailed level several parts are simultaneously scheduled for each group of machines. Pushing the idea further (Fig. 6), McDonnell *et al.* [13] propose a cascaded auction protocol in a distributed decision making context where machine availability guides the simultaneous specification of the operations of a single job to be carried out, the machines to carry them out and the time when they will be processed. In this approach, the design specificities of the final product are once again exploited in order to identify the alternative operations that could be realized to contribute to produce it. Here, the process plan is built along with the scheduling of its operations.

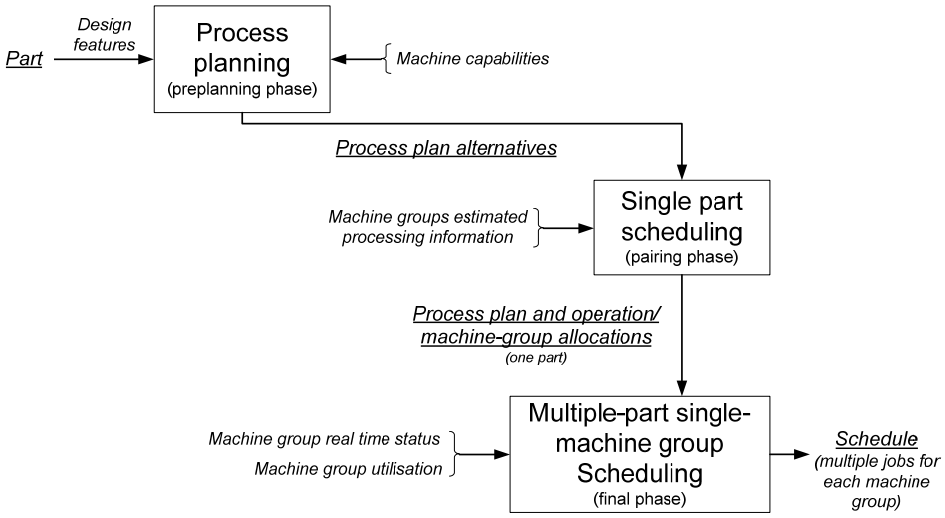

Fig. 5. Distributed process planning (Huang *et al.* [12])

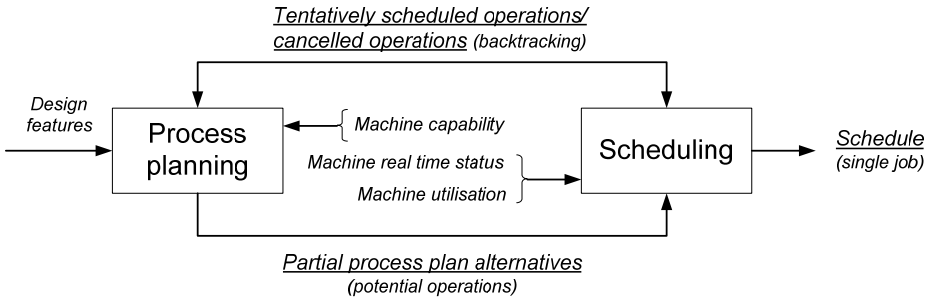

Fig. 6. Distributed process planning (McDonnell *et al.* [13])

Other contributions, such as [14] and [15] consider the scheduling of many jobs simultaneously, each having many alternative process plans. Here the emphasis is generally put on the scheduling process rather than on the generation of alternative process plans, which are considered as parameters of the scheduling problem.

Another approach to address process planning and scheduling is referred to as *combined process planning and scheduling* in [1]. It is also referred to as *integrated planning and scheduling* or *centralized optimization approaches* [16]. The main feature of this approach, which is adopted in the solution proposed in this paper, is to optimize a single model in order to decide what to do, when to do it, and with which resources. Some authors have proposed

specific algorithms to solve such combined problems. Several methodological approaches have been investigated, including mathematical programming ([17]), simulated annealing ([18]), genetic and evolutionary algorithms ([19-21]), and finally constraint programming ([22]). The reader is referred to [23] for a review of these approaches. Similarly, agent-based technology has been applied to solve the integration of process planning and scheduling. The reader is referred to [16] for a review. Here, software agents can either be used as a technology to integrate and coordinate high level planning and scheduling functions, or they can be used to model granular elements of the problem, such as resources and jobs, within a distributed heuristic-based optimization approach (see [13] for example).

2.3 Wood drying and finishing operations planning

First, one must distinguish between the drying of appearance wood and softwood lumber. For appearance wood (e.g., hardwood, as maple or oak), the wood is sold just after being dried. Each piece of wood is graded before drying. Due to the fact that drying is performed so as to minimize defects, the grade usually remains the same for most pieces. This way, divergence is marginal; we can assume that for a green product at input, we will obtain the same product (but dry) as output.

For softwood lumber, which is the case studied in this paper, finishing operations must be performed on the lumber before the product can be sold. As seen previously, the mix of co-products is revealed only after finishing operations. In this context with co-production, drying operations planning consists in determining which batch of wood will be dried, at what time, in which kiln and following which process alternative, taking into account demand, available quantities of green lumber and resource availability.

Different authors propose partial solutions to these problems. One of the first attempts to address the problem of drying applied to hardwood comes from Gascon *et al.* [24], who propose an approach that deals with the scheduling of a set of operations to be carried out in various kiln dryers located in different facilities so as to meet the demand generated by a floor factory. Here, no divergence is explicitly considered. The authors develop a heuristic which aims to schedule drying activities in a multi-item, multi-machine and multi-site production environment. The objective function consists in keeping inventories as low as possible while meeting demand and avoiding stock outs.

Later, Yaghubian *et al.* [25] model the problem as a special case of scheduling multiple independent jobs with no alternative process plan to multiple non-identical parallel machines, with the objective to minimize the maximum tardiness of orders. A heuristic is proposed.

Joines and Culbreth [26] propose an approach that takes into account job sequencing as well as inventory control in a wood furniture manufacturing context. Here, production capacity consists in a series of parallel processors, which includes both in-house kiln dryers and external capacity from service providers. The objective of the approach is to minimize the drying costs plus any inventory and carrying costs. In this approach, all due-dates must be met. Due to the complexity of the problem, the latter is decomposed into two sub problems: (1) the scheduling of drying operations, in other words the sequencing of product types (i.e., lumber) to be processed on the machines (i.e., kilns); and (2) the allocation of inventory to orders so as to meet demand. A hybrid heuristic is proposed to solve these problems. First, a genetic algorithm determines the sequence of product types to produce and an embedded linear program determines the optimal allocation of inventory and quantity of outsourced lumber that minimizes total cost. This hybrid heuristic is shown to be effective in this context.

The problems addressed in the reviewed literature are different from the problem addressed in this paper because they do not explicitly consider diverging flow (finishing operations are not considered, so there is no co-production). Furthermore, they address the scheduling of appearance wood drying jobs that usually takes more time, which tends to reduce the combinatorial complexity of the problem (i.e., for the same planning horizon length, there are less jobs to schedule). Finally, they do not consider drying process alternatives as none of the authors considers process planning as being part of the problem to solve.

3. Problem formulations

This section outlines the proposed models for combined planning and scheduling. Two alternative models based respectively on mixed integer programming and constraint programming are proposed. It will be shown that these two formulations allow for the implementation of radically opposed resolution strategies and response time.

These models were developed for our drying and finishing planning problem but can also address other problems with alternative processes. The models support co-production in divergent processes.

We chose not to explicitly model the different process alternatives; instead, we modeled the individual activities that can be combined in different ways to give different process alternatives. Fig. 7 presents the main idea involved in these models. We have different activity types $a \in \mathbf{A}$. Each can be executed on any machine $m \in \mathbf{M}_a \subseteq \mathbf{M}$, and has a specific duration δ_a. The parameters $q_{a,p}^{consume}$ and $q_{a,p}^{produce}$ specifies the consumption and production of products $p \in \mathbf{P}$ for the activity type a.

In this context, building a plan can be seen as deciding which activities to perform, when to perform them and using which machines. A solution can be represented by a Gantt chart of activities. Note that each type of activity can be inserted as many times as needed in the plan. Inserted activities have an impact on product inventories $\left(\mathrm{I}_{p,t}\right)$ by increasing or decreasing it since they produce and consume different products. Customer demand $\left(d_{p,t}\right)$ also influences product inventories.

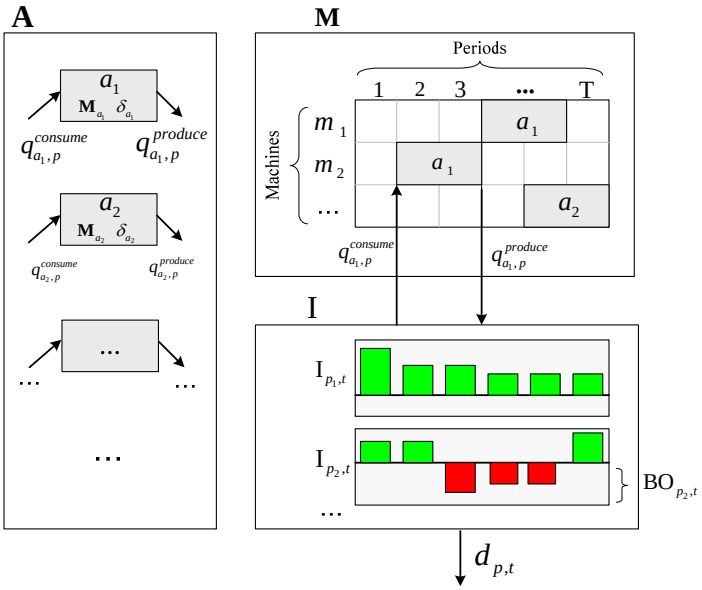

Fig. 7. Intuitive idea supporting the models

This kind of model is referred to as a *timetable* model or *time-line* model by Bartak [1,27]. According to Bartak, the main limitations of these kinds of models are the following: (1) each resource can perform only one activity per period, (2) the duration of the periods must be a common divisor of the activities' duration and (3) it is difficult to model complex dependencies between activities. For (1) and (2), this is not a problem as 12 or 24 hours is commonly used as a time unit in this industry. Concerning (3), this was similarly not an issue because all constraints regarding dependencies between activities can be enforced by making sure that the consumed products are in stock at the right time.

Objective

Within our real world application, it is impractical to consider demand as a hard constraint. For all the datasets we had access to, late deliveries were inevitable. Therefore, we chose to minimize tardiness multiplied by quantity as the objective. We propose the following method to easily compute it in a time-line model. First, we allow the inventory variables in the model to take negative values. For each product/period, a negative inventory corresponds to a backordered quantity, and a positive one corresponds to a volume physically in stock. The sum of all daily backorder quantities is equal to the tardiness. For example, a backlog of 10 units for 5 days (10x5=50) is equivalent to a tardiness of 5 days of 10 units (5x10=50). Of course, for this approach to be mathematically valid, negative inventories must be restricted in order to prevent the model from consuming unavailable products.

The next section describes the variables, parameters and objective common to our MIP and CP formulations. The two models are then introduced.

3.1 Sets, variables, parameters, constraints and objective common to both MIP and CP models

Sets

$\mathbf{M}$	set of machines *m;*
$\mathbf{P}$	set of products *p*;
$\mathbf{P}^{demanded}$	subset of $\mathbf{P}$ containing products with demand. $\mathbf{P}^{demanded} \subseteq \mathbf{P}$;
$\mathbf{A}$	set of all types of activities *a*;
$\mathbf{A}_p^{consume}$	subset of activities which consume the product *p*. $\mathbf{A}_p^{consume} \subseteq \mathbf{A}$;
$\mathbf{A}_p^{produce}$	subset of activities which produce the product *p*. $\mathbf{A}_p^{produce} \subseteq \mathbf{A}$;
$\mathbf{A}_m$	subset containing all types of activities that can be processed on machine *m*. $\mathbf{A}_m \subseteq \mathbf{A}$;
$\mathbf{M}_a$	set of machines that can carry out activity *a*. $\mathbf{M}_a = \{m \in \mathbf{M} \mid a \in \mathbf{A}_m\}$.

Parameters

T	number of periods in the planning horizon
$i_{p,0}$	quantity of product *p* in stock at the beginning of the planning horizon;
$s_{p,t}$	quantity of product *p* supplied at the beginning of period *t*;
$d_{p,t}$	demand for product *p* at period *t*. Must be delivered at the end of period t;
δ_a	number of consecutive periods needed to perform activity *a*;
$q_{a,p}^{consume}$	quantity of product *p* consumed by activity *a*;
$q_{a,p}^{produce}$	quantity of product *p* produced by activity *a*;
$c_{m,t}$	$\begin{cases} 1, \text{ if machine } m \text{ is available during period } t, \\ 0, \text{otherwise.} \end{cases}$

Variables

$\mathrm{TC}_{p,t}$	total consumption of product *p* at period *t*;

$\mathrm{TP}_{p,t}$ total production of product *p* at period *t*;

$\mathrm{I}_{p,t}$ volume of product *p* that would be in stock if the cumulated demand were satisfied. This variable can take negative values; therefore, the "real" volume physically in stock is equal to $\max\left(0, I_{p,t}\right)$;

$\mathrm{BO}_{p,t}$ backorder of product *p* at the end of period *t*. Defined only for $p \in \mathbf{P}^{demanded}$.

In addition to these variables, each of the proposed models (MIP and constraint programming) will define its own decision variables (see sections 3.2 and 2.3).

Constraints

The flow conservation constraints (1.1) and (1.2) balance equations linking inventory, supply, consumption, production and demand variables. The particularity of these equations is that a product *p* can be both consumed and sold.

$$\mathrm{I}_{p,1} = i_{p,0} + s_{p,1} - \mathrm{TC}_{p,1} - d_{p,1} \qquad \forall p \in \mathbf{P} \tag{1.1}$$

$$\mathrm{I}_{p,t} = \mathrm{I}_{p,t-1} + s_{p,t} - \mathrm{TC}_{p,t} + \mathrm{TP}_{p,t-1} - d_{p,t} \quad \forall p \in \mathbf{P},\ t = 2,...,\mathrm{T} \tag{1.2}$$

Constraint (1.3) defines an upper bound for negative inventories. It is limited to be the part of the cumulated demand that is not satisfied by starting inventory plus supply. This prevents us from consuming a product that is not in stock.

$$\mathrm{I}_{p,t} \geq \min\left(0,\ i_{p,0} + \sum_{\tau=1}^{t}\left(s_{p,\tau} - d_{p,\tau}\right)\right) \qquad \forall p \in \mathbf{P},\ t = 1,...,\mathrm{T} \tag{1.3}$$

Equation (1.4) expresses that the consumption variables for a product are set to 0 if there is no activity consuming that product. Similarly, equation (1.5) set that all production is set to 0 if there is no activity producing that product.

$$\mathrm{TC}_{p,t} = 0 \qquad \forall p \mid \mathbf{A}_p^{consume} = \varnothing,\ t = 1,...,\mathrm{T} \tag{1.4}$$

$$\mathrm{TP}_{p,t} = 0 \qquad \forall p \mid \mathbf{A}_p^{produce} = \varnothing,\ t = 1,...,\mathrm{T} \tag{1.5}$$

Objective function

The primary goal of the objective function (1.6) is to minimize tardiness of the quantities ordered by customers, measured as the sum of backorder quantities.

$$\mathrm{Min} \sum_{p \in \mathbf{P}^{demanded}} \sum_{t=1}^{\mathrm{T}} \mathrm{BO}_{p,t} \tag{1.6}$$

3.2 Mixed Integer Programming model

This model includes the following decision variables:

$\mathrm{S}_{(a,m),t}$ Binary decision variable. It takes the value 1 if an activity of type *a* starts on machine *m* at period *t*, 0 otherwise. It is defined for each $(a,m) \mid a \in \mathbf{A}_m$.

Additionally, the variables introduced in section 3.1 are present in this model as continuous variables.

Constraints

The following variables are restricted to positive values: $\mathrm{TC}_{p,t}$, $\mathrm{TP}_{p,t}$ and $\mathrm{BO}_{p,t}$.

Constraint (2.1), together with the minimization of the objective function, defines backorder as the minimum positive quantity that covers negative inventories up to 0.

$$\mathrm{I}_{p,t} + \mathrm{BO}_{p,t} \geq 0 \qquad \forall p \in \mathbf{P}^{demanded}, t = 1,...,\mathrm{T} \tag{2.1}$$

The consumption constraint (2.2) defines $\mathrm{TC}_{p,t}$ as the total consumption of activities starting during period *t* and consuming product *p*. The sum is computed only for the pairs of activities *a* and machines *m* for which *a* consumes *p* and *m* can carry out *a*. The production constraint (2.3) is the counterpart of the previous constraint. It sets that total production is the sum of what is produced by activities *a* ending during period *t* (i.e. those starting at period $t - \delta_a + 1$) and producing product *p*.

$$\mathrm{TC}_{p,t} = \sum_{(a,m) \left| \begin{array}{l} a \in \mathbf{A}_m \\ a \in \mathbf{A}_p^{consume} \end{array} \right.} \mathrm{S}_{(a,m),t} \times q_{a,p}^{consume} \qquad \forall p \in \mathbf{P},\ t = 1,...,\mathrm{T} \tag{2.2}$$

$$\mathrm{TP}_{p,t} = \sum_{(a,m) \left| \begin{array}{l} a \in \mathbf{A}_m \\ a \in \mathbf{A}_p^{produce} \\ t - \delta_a + 1 \geq 1 \end{array} \right.} \mathrm{S}_{(a,m),t-\delta_a+1} \times q_{a,p}^{produce} \qquad \forall p \in \mathbf{P},\ t = 1,...,\mathrm{T} \tag{2.3}$$

The capacity constraint (2.4) sets that the number of activities running on a machine *m* at period *t* must be smaller than or equal to 1. For each type of activity *a*, there is an instance running at period *t* if and only if one started during the interval $[t - \delta_a + 1,...,t]$.

$$\sum_{a \in \mathbf{A}_m} \sum_{\tau = \max[t - \delta_a + 1,\ 1]}^{t} \mathrm{S}_{(a,m),\tau} \leq c_{m,t} \qquad \forall m \in \mathrm{M},\ t = 1,...,\mathrm{T} \tag{2.4}$$

3.2.1 MIP model implementation and resolution

The implementation of this model was made with OPL Studio from ILOG. The model was then solved using ILOG CPLEX 9.1 (results are presented in section 4.1). Because the results did not meet our expectations regarding response time (see section 1.1), we also developed a constraint programming (CP) model introduced in the next section.

3.3 Constraint Programming model

This section proposes an alternative formulation of the previous planning and scheduling problem based on constraint programming, and proposes a search procedure for its resolution.

Constraint programming (CP) [28] is rooted in the field of artificial intelligence (AI). CP proposes logic-based methods for optimization [29] that search through a tree representation of the solution space by iteratively and tentatively fixing variables and by propagating constraints in order to reduce the domain (i.e., possible values) of the remaining variables so as to find feasible solutions. From the perspective of the OR community, CP provides an alternative paradigm, complementary to mathematical programming (MP), for modeling and solving problems. Modeling in CP is easier than modeling in MP for problems where constraints are expressed in terms of logic-based discrete choices such as *If A = 3 and B < 5 then either C = 4 or D > 10*, which is rather complex to model using MP. Although they are different and lead to radically different kinds of models, mathematical and constraint programming are sometimes used in conjunction to solve large scale problems [30]. Being able to combine the strengths of both even appears as a challenge that is gaining more and more importance in the OR community.

Variables

In CP, the value of a variable is not necessarily restricted to numerical values. For example, a variable named "color" could take any value from a set "{blue, white, red}". The set of the allowed values is called the *domain* of the variable. This situation would be modeled with mathematical programming by defining three binary variables and a constraint stating that the sum of the three must be equal to 1, while no constraint is necessary in CP. The model proposed here uses this type of variable. The CP decision variables $S_{m,t}$ specify the type of activity starting on processor *m* at period *t*. It can take the value "∅" (meaning that no activity is starting) or any "compatible activity type" (a set of objects defined in the same way as the set of colors presented previously):

$S_{m,t}$ type of the activity starting on machine *m* at the beginning of period *t*, $S_{m,t} \in \mathbf{A}_m \cup \{\varnothing\}$;

In this CP formulation, the numerical variables from section 3.1 are defined as integer variables (while they are continuous in the MIP formulation) in order to take advantage of fast integer computation of computer processors:

$TC_{p,t}$, $TP_{p,t}$ and $BO_{p,t} \in \mathbb{N}$

$I_{p,t} \in \mathbb{Z}$

Particular notations

Most notations used in our CP model are similar to those of the MIP model. Other constraints are modeled using Boolean expressions. Table 1 introduces the Boolean operators used.

Table 1
Boolean operators used in the CP model

Operator	Name	Example
$\vee$	Disjunction ("or")	The expression "A $\vee$ B" is true if at least one of the expressions "A" or "B" is true.
$\wedge$	Conjunction ("and")	"A $\wedge$ B" defines a new expression that is true if and only if expression "A" and expression "B" are true. To describe a series of conjunctions (e.g. $A_1 \wedge A_2 \wedge A_3$) we use the operator $\forall$ as in: $\bigvee_{i=1}^{3}(A_i)$.
$\Rightarrow$	Implication	"A $\Rightarrow$ B" is true according to this: When expression "A" is true, expression "B" must also be true. When expression "A" is false, "B" may take any value.

Constraints

Constraint (3.1) defines the backorder as being either 0, if the inventory is positive, or the opposite value of the inventory if the latter is negative (i.e., the absolute value of a negative inventory).

$$BO_{p,t} = \max(0, -I_{p,t}) \qquad \forall p \in \mathbf{P}^{demanded},\ t = 1,...,T \tag{3.1}$$

Consumption constraint (3.2) states that each time an activity of type *a* that consumes product *p* is started during period *t*, then the consumption variable $TC_{p,t}$ is increased linearly by the consumption factor $q_{a,p}^{consume}$. Equation (3.3) defines $NAS_{a,t}$ as the number of times an activity of type *a* is started at period *t*. Constraint (3.4) is the counterpart of constraint (3.2) for production.

$$\mathrm{TC}_{p,t} = \sum_{a \in \mathbf{A}_p^{consume}} \mathrm{NAS}_{a,t} \times q_{a,p}^{consume} \qquad \forall p \in \mathbf{P},\ t = 1,...,\mathrm{T} \tag{3.2}$$

$$\text{with } \mathrm{NAS}_{a,t} = \sum_{m \in \mathrm{M}_a} \left(\mathrm{S}_{m,t} = a \right) \tag{3.3}$$

$$\mathrm{TP}_{p,t} = \sum_{a \in \mathbf{A}_p^{produce} \mid (t - \delta_a + 1) > 1} \mathrm{NAS}_{a,t-\delta_a+1} \times q_{a,p}^{produce} \qquad \forall p \in \mathbf{P},\ t = 1,...,\mathrm{T} \tag{3.4}$$

The capacity constraint (3.5) states that for all machines *m* unavailable during period *t*, the corresponding type of activity starting on this machine is set to $\varnothing$.

$$\mathrm{S}_{m,t} = \varnothing \qquad \forall m \in \mathbf{M} \mid c_{m,t} = 0,\ t = 1,...,\mathrm{T} \tag{3.5}$$

Next, constraint (3.6) states that if an activity $a \in \mathbf{A}_m$ starts during period *t*, then no other activity can start on the same machine before the end of this activity. Similarly, constraint (3.7) indicates that if an activity starts during period *t*, or if no activity can be carried out on machine *m* because it is not available, than all preceding activities on *m* must be completed.

$$\left(\mathrm{S}_{m,t} \neq \varnothing \right) \Rightarrow \underset{\left(\tau = 1,...,T \mid t+1 \leq \tau \leq t + \delta_{(\mathrm{S}_{m,t})} - 1 \right)}{\forall} \left(\mathrm{S}_{m,\tau} = \varnothing \right) \qquad \forall m \in \mathrm{M},\ t = 1,...,\mathbf{T} \tag{3.6}$$

$$\left(\mathrm{S}_{m,t} \neq \varnothing \right) \vee \left(c_{m,t} = 0 \right) \Rightarrow \underset{\left(\begin{array}{l} a \in \mathbf{A}_m \\ \tau = 1,...,T \end{array} \middle| \, t - \delta_{(\mathrm{S}_{m,t})} + 1 \leq \tau \leq t - 1 \right)}{\forall} \left(\mathrm{S}_{m,\tau} \neq a \right) \qquad \forall m \in \mathrm{M},\ t = 1,...,\mathbf{T} \tag{3.7}$$

3.3.1 CP model implementation and resolution

The implementation of this model was made using C++ and the ILOG SOLVER 6.0 library. In order to solve a problem in CP different algorithms are involved to search throughout the solution space [31]. The first algorithm deals with the building of a search tree. It is referred to as the *search procedure*. The second algorithm concerns how to explore the search tree in order to find feasible solutions. This is referred to as the *search strategy*. Both types of algorithms are presented hereafter along with the development made specifically for our problem.

Search procedure, general constraint programming concepts

The *search procedure* aims at defining a search tree for a problem. Fig. 8 shows an example of a search tree for a simple problem. The tree can be described as follow: Each *node* of the tree represents the status of the search for a solution at a given time. This status is defined by the domain of each variable (i.e., the remaining potential values of a variable). Each *arc* descending from a node represents an alternative way to reduce the domain of one or many variables. These alternatives are specified by the search procedure. A node for which all variables have one and only one value in their domain (those variables are said to be valued) corresponds to a solution. It is important to note that different search trees (that is, different search procedures) could be defined for the same problem.

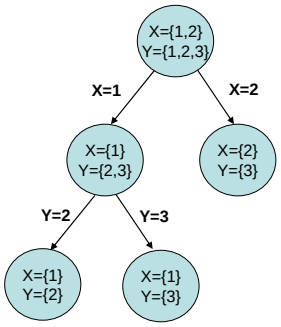

Fig. 8. Search tree for a simple problem $\left(\mathrm{X} \in \{1,2\}; \mathrm{Y} \in \{1,2,3\} \text{ s.c. } \mathrm{Y} > \mathrm{X}\right)$.

A search procedure generates the tree during the search; it is not generated beforehand. Each time a node is investigated, constraints are propagated to reduce the domain of the variables (and consequently, the number of arcs descending from that node). This is illustrated in Fig. 8. This propagation mechanism applies with our actual model. Constraints (3.6) and (3.7) have the following effect. Each time a value is assigned to a variable $\left(\text{e.g. } \mathrm{S}_{m,t} = a \neq \varnothing\right)$ the domains of the other variables related to this machine $\left(\text{e.g. } \mathrm{S}_{m,\tau},\ \forall \tau \neq t\right)$ are updated to prevent any double-use of this machine. Similarly, the propagation of constraint (3.2) together with flow conservation constraints (1.1)(1.2)(1.3) has the following effect. The activity type *a* consuming product *p* is withdrawn from the domain $\mathrm{S}_{m,t}$ if the upper bound of inventory $\mathrm{I}_{p,t}$ is such that the insertion of *a* would cause a negative inventory level that could not be compensated by any other activity producing *p* and carried out before (the next section explains how this propagation is carried out in more details).

The propagation of constraints during the investigation of the solution space contributes to decrease the size of the tree. However, in general constraint propagation is not sufficient to prevent the investigation of arcs leading to sub-trees with no solution at all. This search process can be very costly.

In the next section, we describe an efficient but incomplete search procedure (it does not explore the entire solution space). This search procedure has the advantage of preventing inefficient backtracking to produce good solutions in a short amount of time. It was developed as the completed search procedures were unable to find feasible solutions, even for large computation times (> 12 hour).

Proposed search procedure

As the studied case leads to huge combinatorial problems, a search procedure that does not systematically investigate all possible solutions has been developed. This search procedure focuses the search for a solution in a small part of the solution space to accelerate the search for a good solution.

Our search procedure defines a tree that can be interpreted in the following way. Each node of the tree can be considered as a partial plan, defined by the variables $\mathrm{S}_{m,t}$ already valued. The root node corresponds to an empty plan. From a given node, different activities could be added to the partial plan. The arcs represent potential branching decisions, each one leading to a new node/partial plan. Fig. 9 shows an intuitive representation of this mechanism.

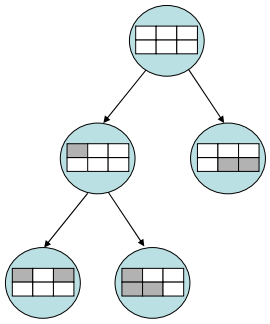

Fig. 9. Intuitive representation of the tree; each node corresponds to a partial plan.

We propose to allow branching on *processes* instead of on individual activities. We define a *process* as any sequences of activities that produce at least one product for which there is a demand $\left(\mathbf{P}^{demanded}\right)$. Consequently, a finishing activity alone is considered a process. A finishing activity preceded by a drying activity is also considered a process. However, a drying activity cannot be considered a process, unless there is a demand for unfinished products. The identification of these processes is carried out during a pre-treatment before the search. This provides the following set of data:

$\mathbf{W}$ set of processes $w \in \mathbf{W}$. For each process $w \in \mathbf{W}$, $\text{card}(w)$ designates the number of activities involved in *w*, and a_i^w designates the activity $a \in \mathbf{A}$ associated with the i^{th} step of *w*, for $i = 1,...,\text{card}(w)$. The same activity *a* can be part of more than one process.

Each time we branch on a specific process, its activities must be added to the current partial plan. In order to do so, we must schedule each of these activities (i.e., select a machine and a period). This scheduling is performed using the following set of rules:

- Using the current partial plan, compute the earliest date for which one of the products produced by the selected process is back-ordered;
- By considering this as the due date, backward schedule the activities of the process in just-in-time (machines have no priority).

Consequently, for a given node (partial plan) and a given arc (a process to add), there is only one schedule possible for each activity of the process (i.e., there is only one possible assignment for variables $S_{m,t}$). The main consequence is that the solution space of the problem is not entirely represented by this tree. However, even if the optimal solution was missed by this search, the speed of the search for a good solution is increased.

Two rules have been introduced to reduce the number of branching alternatives (but without additional restriction of the solution space explored). First (R1), the inclusion of a process must always lead to a plan that is better than the current plan; and (R2) the inclusion of a process must always lead to a feasible partial plan.

Rule R1 ensures that branching always contributes to improve the solution of the current node with regards to the objective function. Due to the fact that a process is defined as a sequence of activities that produces demanded products, it is useless to branch on a process that does not contribute to the objective in the current node: this process will not help in any of the nodes deriving from the current node.

Rule R2 ensures that the partial plans of every node in the search tree is a feasible plan. This rule makes sense in the context where we branch on processes rather than on activities: it removes unproductive sections from the tree that would lead to no feasible solutions. Another advantage is that one can stop the search for a solution at any time as the algorithm always gives a feasible solution. This is called an *anytime algorithm* (see [32]).

From a given node, it is easy to deterministically compute the contribution of each branching alternatives before branching (actually, we need to do this to enforce rule R1). It is therefore possible to sort the branching alternatives

according to this contribution level (this will be our rule R3). By doing so, the search process will perform a *gradient descent* (*hill climbing* in a minimization context).

Enforcing rule R2

To implement rule R2 (the inclusion of a process must always lead to a feasible partial plan), we introduce a new variable in the model. However, the value of this variable is not part of the solution (we do not want this variable to take a value). It is only meant to guide the search procedure; the domain of this variable will help identify the processes that could be inserted into a partial plan.

S_i^w the domain of this variable specifies the periods where it is possible to start the i^{th} activity of process *w*. Originally (before the search), the domain of the variable is $\{\varnothing, 1...\mathrm{T}\}$. In a given node, the domain of S_i^w specifies where the activity a_i^w of process *w* could be inserted into the current partial plan. If the domain is reduced to $\{\varnothing\}$, then the process *w* cannot be inserted into the plan. Therefore, the domain of S_i^w defines the branching alternatives available from a node.

It is important not to interpret the variable S_i^w as the '*start date of the activity* a_i^w*, taking a value when we insert this activity into the plan*' (as we would do in a scheduling problem). Indeed, when building the plan the same process *w* can be inserted several times. Each time we do so, the domains of variables S_i^w are updated to show which activities/processes can be further inserted from the new partial plan.

Constraints (3.8) and (3.9) are added to the model to ensure the consistency of these domains. Constraint (3.8) models the precedence between the activities of a process. Constraint (3.9) specifies that an activity can be inserted at period *t* only if a machine is available, and if the new plan would be feasible in terms of consumed product inventory.

$$\mathrm{S}_i^w > \mathrm{S}_{i-1}^w + \delta_{\left(a_i^w\right)} - 1 \qquad \forall (w,i) \mid \left(w \in \mathbf{W}\right) \wedge i = 2,...,\mathrm{card}\left(w\right) \tag{3.8}$$

$$\mathrm{dom}\left(\mathrm{S}_i^w\right) = \{\varnothing\} \cup \left\{ t \in 1,...,\mathbf{T} \;\middle|\; \begin{array}{l} \exists_{m \in \mathbf{M}_{\left(a_i^w\right)}} \left[a_i^w \in \mathrm{dom}\left(S_{m,t}\right) \wedge \neg\, \mathrm{bound}\left(S_{m,t}\right) \right] \\ \wedge \underset{\left(\begin{array}{l} p \in P : q^{consumed}_{\left(a_i^w\right),p} > 0 \\ \tau \in t..T \end{array}\right)}{\forall} \left[\mathrm{current}\left(p,\tau\right) \geq -q^{net}_{w,p,i} \right] \end{array} \right\} \tag{3.9}$$

$$\forall (w,i) \mid \left(w \in \mathbf{W}\right) \wedge i = 1,...,\mathrm{card}\left(w\right)$$

where :

$\mathrm{dom}\left(\mathrm{X}\right)$ represents the set of all possible values in the domain of variable X;

$\mathrm{bound}\left(\mathrm{X}\right)$ is a Boolean function that indicates if variable X is valued, i.e. if its domain contains only one value. In other words, $\mathrm{bound}\left(\mathrm{X}\right) := \left(\mathrm{card}\left(\mathrm{dom}\left(\mathrm{X}\right)\right) = 1\right)$;

$\mathrm{current}\left(p,t\right)$ is the function that indicates the inventory level of product *p* at period *t* that would result if no other activity were inserted in the current plan. In other words, $\mathrm{current}\left(p,t\right) := i_{p,0} + \sum_{\tau=1}^{t} s_{p,\tau} + \mathrm{dmin}\left(\mathrm{TP}_{p,\tau}\right) - d_{p,\tau} - \mathrm{dmin}\left(\mathrm{TC}_{p,\tau}\right)$;

$\mathrm{dmin}\left(\mathrm{X}\right)$ is the function that returns the smallest value of the domain of variable X. Formally, $\mathrm{dmin}\left(\mathrm{X}\right) := \min_{v \in \mathrm{dom}\left(X\right)}\left(v\right)$. In the present context, $\mathrm{dmin}\left(\mathrm{TP}_{p,t}\right)$ represents the quantity of

product p that is produced during period t given the current partial plan;

$q^{net}_{w,p}$ is a new parameter (computed in pre-treatment) that represents the production of product p by a process w, minus its consumption of the same product. In other words, $q^{net}_{w,p} := \sum_{i=1}^{\text{card}(w)} q^{produce}_{\left(a_i^w\right),p} - q^{consume}_{\left(a_i^w\right),p}$;

$q^{net}_{w,p,i}$ is a parameter that represents the production of product p by a process w minus its consumption, when the i^{th} activity of process w starts. In other words, $q^{net}_{w,p,i} := \sum_{j=1}^{i-1} q^{produce}_{\left(a_j^w\right),p} - \sum_{j=1}^{i} q^{consume}_{\left(a_j^w\right),p}$.

Enforcing rule R1 and R3

To implement rules R1 (branch only on processes with positive contribution) and R3 (branch first on the process with the greater contribution), we must be able to compute the contribution that would be gained from the insertion of process w in the current plan. To do so, we define the following functions that will be used in the search procedure.

First, the contribution of a process w (3.10) is measured as the reduction of the objective function it allowed (the reduction of backorders for the products with demand it produces). The contribution for a specific product p (3.11) can be computed by comparing the current inventory curve with the one obtained when taking the consumptions and productions of the process into account. To compute this we must know when the activities of the process would be carried out. Functions (3.12) and (3.13) compute this by analyzing the domain of S_i^w .

$$\text{contribution}(w) := \sum_{p \in \mathbf{P}^{demanded} \mid q^{net}_{w,p} > 0} \text{contribution}(w, p) \tag{3.10}$$

$$\begin{aligned} \text{contribution}(w, p) := & \sum_{\tau=\text{end}\left(w,\text{card}(w)\right)+1}^{\mathrm{T}} \left[\min\left(0, \quad \text{current}\left(p,\tau\right) + q^{net}_{w,p}\right) - \min\left(0, \quad \text{current}\left(p,\tau\right)\right)\right] \\ & + \sum_{i=1}^{\text{card}(w)} \sum_{\tau=\text{start}\left(w,i\right)}^{\text{end}\left(w,i\right)} \left[\min\left(0, \quad \text{current}\left(p,\tau\right) + q^{net}_{w,p,i}\right) - \min\left(0, \quad \text{current}\left(p,\tau\right)\right)\right] \end{aligned} \tag{3.11}$$

$$\text{start}\left(w,i\right) := \text{dmin}\left(\mathrm{S}_i^w\right) \tag{3.12}$$

$$\text{end}\left(w,i\right) := \text{start}\left(w,i\right) + \delta_{\left(a_i^w\right)} - 1 \tag{3.13}$$

Putting it all together

Fig. 10 presents the pseudocode for the whole search procedure. It is presented in a style that recalls *Optimization Programming Language* (OPL) [31].

1) while $\left(\underset{w \in \mathbf{W}}{\exists} \middle| \text{possible}(w) \right)$

2) {

3) $\text{tryall}\left(w \in \mathbf{W} \middle| \text{possible}(w) \text{ ordered by increasing contribution}(w) \right)$

4) {

5) $i = \text{card}(w)$

6) $ideal = \max\left(0, \quad \text{ideal}(w) - \delta_{(a_i^w)} \right)$

7) while $(i \geq 1)$

8) {

9) $\tau_i = \min\left(\max\left(t \in 1..T \middle| t \leq ideal \right), \ \min\left(t \in 1..T \middle| t > ideal \right) \right)$

10) $i = i - 1$

11) if $(i \geq 1)$ $ideal = \max\left(0, \quad \tau_{i+1} - \delta_{(a_i^w)} \right)$

12) }

13) $\text{forall}\left(i \in 1..\text{card}(w) \right)$

14) {

15) $t = \tau_i$

16) $m = \text{first}\left(m \in \mathbf{M}_{(a_i^w)} \middle| a_i^w \in \mathrm{S}_{m,t} \wedge \neg \text{bound}\left(\mathrm{S}_{m,t} \right) \right)$

17) $\mathrm{S}_{m,t} = a_i^w$

18) }

19) }

20) }

Fig. 10. Proposed search procedure

Line 1 specifies that some processes must be inserted into the current partial plan as long as it is possible and useful to do so. It uses the Boolean function **Erreur ! Source du renvoi introuvable.** to check if the contribution of a process *w* is positive and if each activity of the process can be inserted into the plan.

$$\text{possible}(w) := \left(\text{contribution}(w) > 0 \right) \wedge \left[\underset{i=1}{\overset{\text{card}(w)}{\forall}} \left(\mathrm{S}_i^w \neq \varnothing \right) \right] \tag{3.14}$$

For a given node (i.e. partial plan), line 3 tries the available processes ordered by increasing contribution. Line 5 and 6 compute the ideal start date of the last activity of the process *w*, given the remaining unsatisfied demand. It uses the function **Erreur ! Source du renvoi introuvable.** that looks for the first period where a product produced by the process is expected to be backordered.

$$\text{ideal}(w) := \min\left(t \in 1,...,\mathbf{T} \middle| \left(\underset{p \in \mathbf{P}^{demanded}}{\exists} \middle| \text{current}(p,t) < 0 < q_{w,p}^{net} \right) \right) \tag{3.15}$$

Lines 7 to 12 compute the start dates (τ_i) for the activities a_i^w of process *w*. It applies just-in-time backward scheduling. Then, lines 13 to 18 choose a machine to execute each activity and insert them into the plan. For each activity, the first available machine is selected.

Search strategy

If, for a given node there is no branching alternative, then it is not possible to improve the plan without redrawing or moving activities already in the plan. It is therefore possible to generate alternative solutions by backtracking in the tree to explore other branching alternatives. Many backtracking strategies (or *search strategies*) are already available in the ILOG SOLVER library. During our computational study, we tested Depth-First Search (DFS) and Depth-Bounded Discrepancy search (DDS).

DFS is the most simple and natural way to explore a tree. Once a solution is found, the search backtracks to the last visited node for which there is at least one unexplored alternative. Then, branching is done with the next alternative. This is also termed *chronological backtracking.*

DDS, introduced by Walsh in [33], exploits the idea that "mistakes are more likely to be made near the top of the search tree rather than further down." Consequently, DDS uses the concept of discrepancy, which is defined as: "A discrepancy is any decision point in a search tree where we go against the heuristic" and "The depth of a discrepancy is the level in a binary tree at which we would see the discrepancy." The search is done using successive iterations. At the first iteration, no discrepancy is allowed. For the second iteration, discrepancy is allowed (and mandatory) at the root of the tree. In further iterations (e.g. iteration **i=3**) discrepancies are allowed in the first **i-2** levels of the tree and mandatory at level **i-1**. Fig. 11, inspired from [33], shows how a binary tree would be explored using DDS.

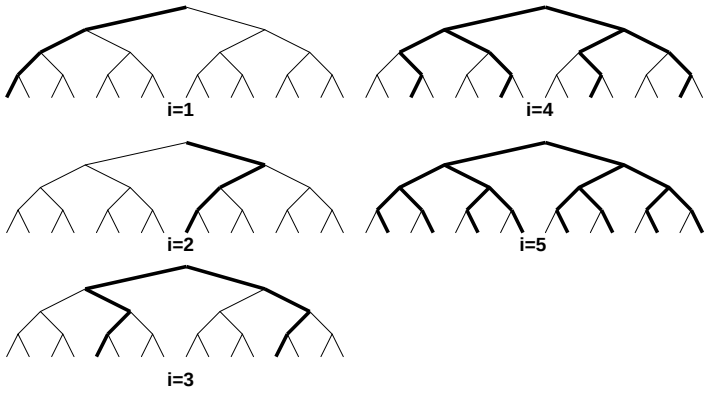

Fig. 11. Exploration of a binary tree using DDS

For a tree that is not binary, ILOG SOLVER transforms it automatically. A node that has *n* children is replaced by a binary sub-tree where each node represents the alternative between branching on a child or on one of its right neighbors.

Generally, implementations of DDS are slower than DFS for trivial reasons. However, if the search procedure exploits a heuristic that is "good", then DDS can give better solutions than DFS for the same computation time.

4. Quantitative evaluation

4.1 Data collection and solving

The two proposed models were tested with real dataset from a representative mid-sized lumber sawmill from the province of Quebec (Canada). Data on products, machines and activities were identified with the manager responsible for the production planning of the entire company. These data define 166 product types, 12 machines of 3 different types and 525 different types of activities. Next, data concerning initial stocks, supply and demand were exported from the enterprise database at four different times in the year (with 2 to 4 weeks between each export). In the models, we used 24 hours periods, the same granularity as used by the company. The planning horizon is 60 days. In the MIP formulation this results in 79 200 binary variables. Table 2 summarizes the four cases.

Table 2
Summary of the datasets shows total demand (in millions of units) and its relative distribution between product families.

	Case #1	Case #2	Case #3	Case #4
2x3 3-4	1.8%	14.3%	4.0%	2.7%
2x4 1-2	39.3%	29.1%	34.0%	59.6%
2x4 3-4	11.1%	31.4%	10.4%	16.8%
2x6 1-2	30.2%	11.1%	23.6%	4.4%
2x6 3-4	7.7%	8.0%	6.1%	1.3%
Total	13.0	15.5	14.0	11.6

The current planning procedure of the company, which we used as a benchmark, is done manually using a spreadsheet. It takes half a day for the expert to build a plan for one mill. Only the first week of each plan is executed as planned. The remaining of the plan is thus subject to change during the next planning phase (the planning is reviewed once a week).

In accordance with section 1.1, the performances of both approaches (MIP and CP) were evaluated with regard to the solution quality according to computation time. The MIP model was solved using ILOG CPLEX 9.1 with an emphasis on the generation of feasible solutions. Computations were stopped after 11 hours. The CP model was solved for the same computation time using the proposed search procedure (with DFS and DDS strategies). The first solution for DFS and DDS are the same (for trivial reasons). After that, DDS quickly improves solution quality. As for DFS, solution quality does not improve during the rest of the 11 hours. Therefore, DFS results were removed from the charts with no loss of information.

4.2 Results

Fig. 12 to Fig. 15 show results for the four cases. The upper bound (UB) is obtained using the manual planning procedure. The lower bound (LB) is the one provided by CPLEX at the end of 11 hours. The gap between these bounds represents the maximum reduction of backorders theoretically achievable by the company.

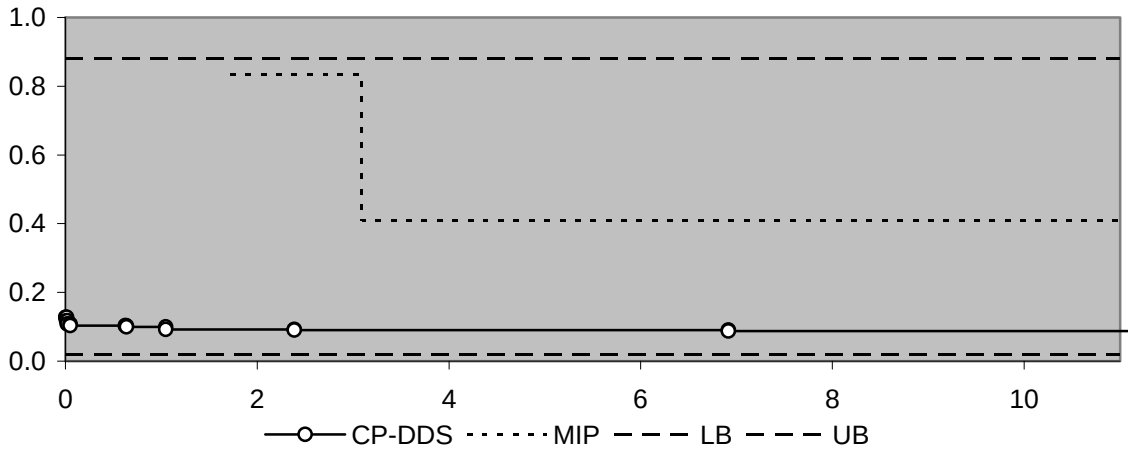

Fig. 12. Case #1, Backorders (millions of units) vs. computation time (hours)

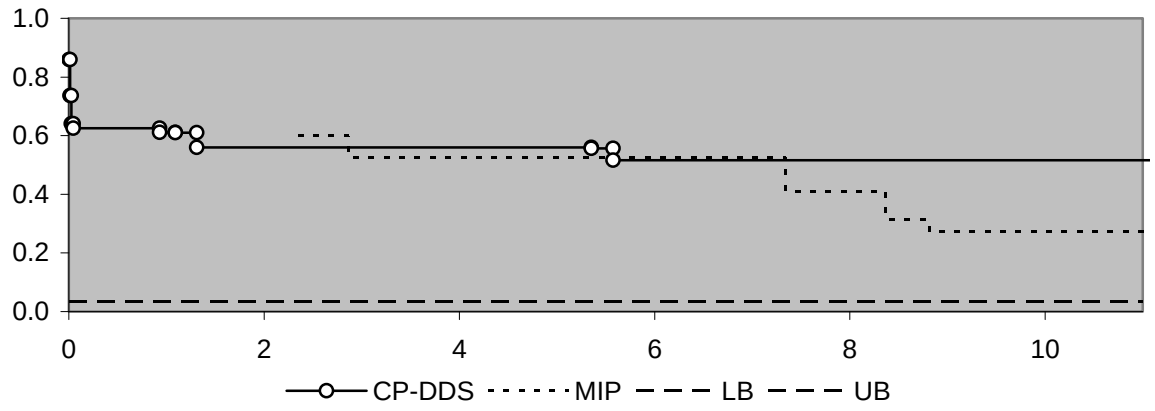

Fig. 13. Case #2, Backorders (millions of units) vs. computation time (hours).

Upper bound at 15.1 (not shown)

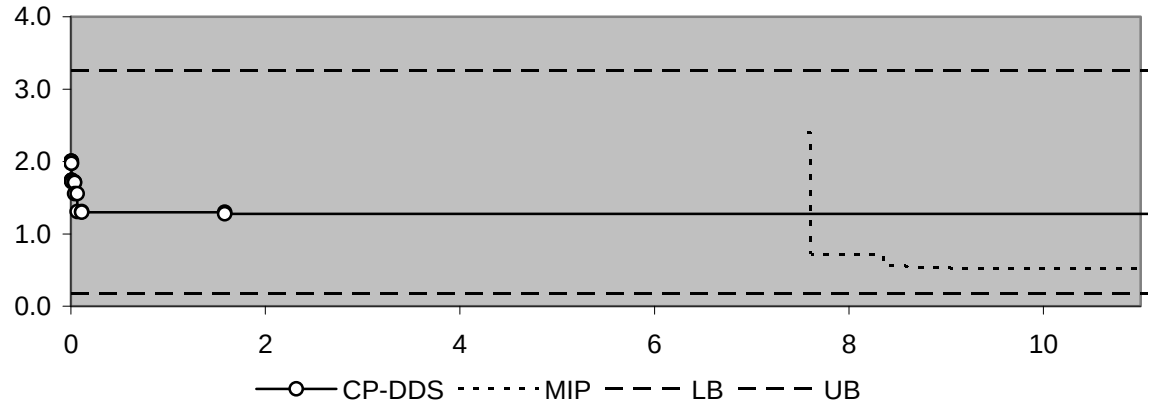

Fig. 14. Case #3, Backorders (millions of units) VS computation time (hours)

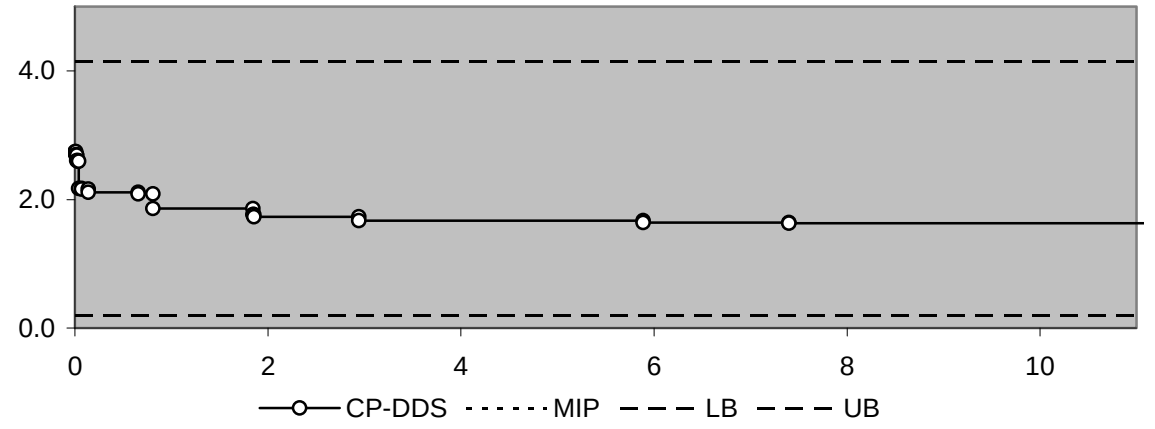

Fig. 15. Case #4, Backorders (millions of units) VS computation time (hours). No integer solution for the MIP

Considering the size of our test problems, neither the MIP nor the CP approach completed their search within 11 hours. The optimal solution is thus not obtained for any of the problems.

For all datasets, the CP approach gives a first solution within a few seconds (12.8; 11.6; 11.9; 12.3). The use of the DDS strategy provides good improvements of solution quality at the beginning of the search. After this it tends to an asymptote. On average, the CP approach provides 75% of the total improvement within five minutes. This result is rather usual with DDS when the branching heuristic used is "informed", therefore making more mistakes close to the root node. DDS first focuses on branching alternatives at the top of the tree, which results in the quick identification of good alternative solutions. Then, DDS focuses on branching alternatives closer to the bottom of the tree, which tends to produce solutions similar to the previous ones [33].

Concerning the MIP, results are very different for each problem instance. In general, it takes a long time to obtain a first solution and it is very poor when compared to CP solutions. For the first case, the best MIP solution after 11 hours is worse than the first solution of the CP. However, for cases 2 and 3 the MIP is better then the CP, but only after a few hours. For the last case, there is no feasible solution with the MIP after 11 hours.

MIP with starting values

In order to improve the performance of the MIP approach, we tested a mixed approach where the MIP is initialized with the solution provided by the CP approach (after five minutes of search). This mixed approach obtains a good solution quickly (the CP one), while continuing the search with branch-and-bound. This approach was tested with the same four datasets (Fig. 16 to Fig. 18).

In the first case (Fig. 16), the MIP with starting value (MIP-SV) is better than the original MIP and is better than the CP for a computation time greater than 3 hours. Furthermore, the quality of the solutions obtained with this approach also increased. For the second case (Fig. 17), the mixed approach is better than the CP approach for computation times greater than 8 hours. However, the solution is always worst than the non-initialized MIP. For the third case (Fig. 18), the mixed approach is better than the CP approach after 2.5 hours, but the non-initialized MIP found a better solution after 8 hours. For the fourth case, the approach did not improve the initial solution.

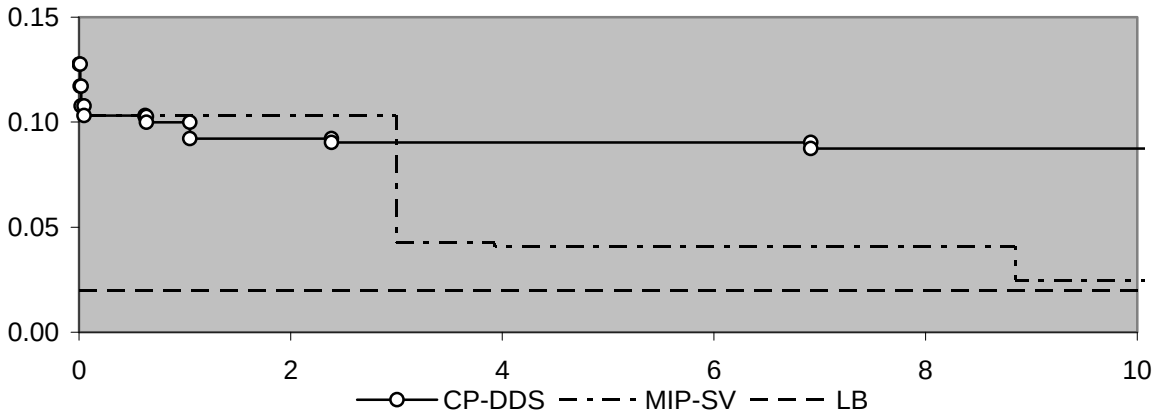

Fig. 16. Case #1, MIP with starting values. MIP and upper bound not shown (see Fig. 12)

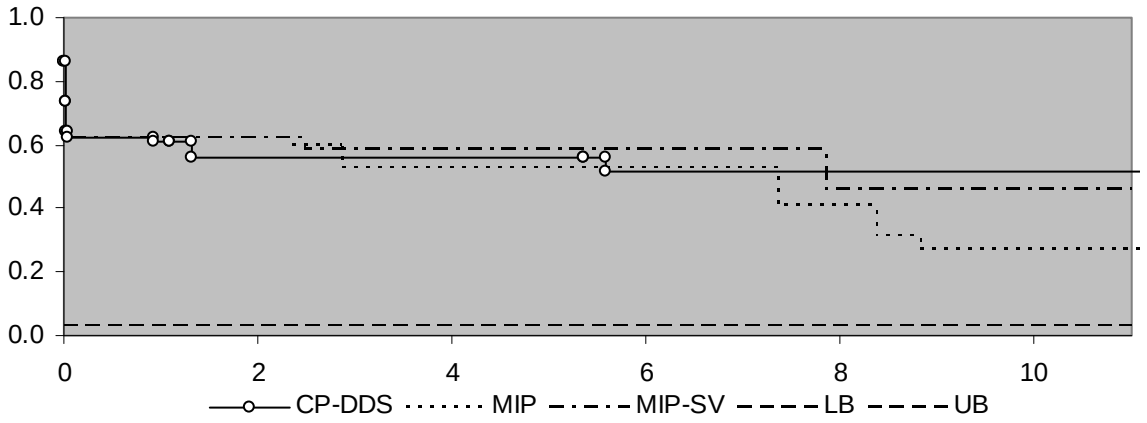

Fig. 17. Case #2, MIP with starting values. Upper bound at 15.1 not shown

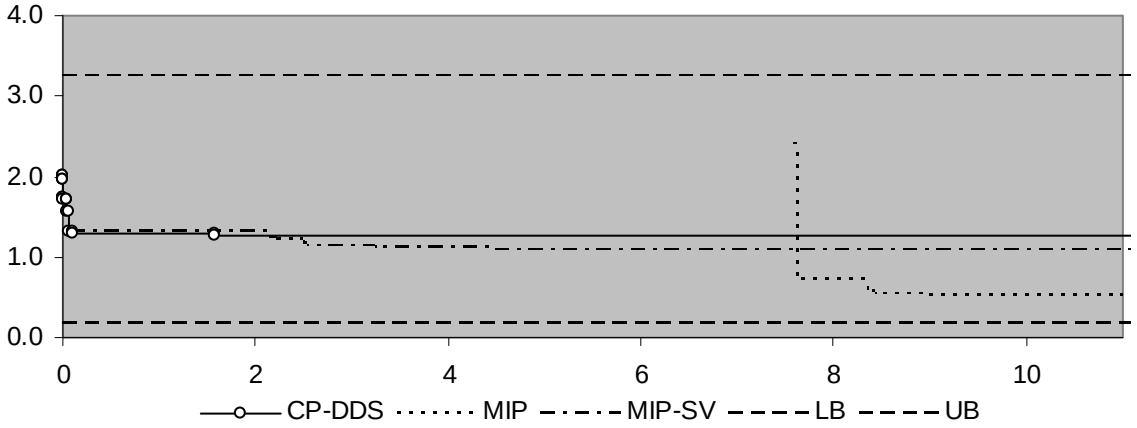

Fig. 18. Case #3, MIP with starting values

In brief, whether or not the MIP is initialized with starting values, solution quality is rather unequal. Due to the large amount of binary variables, the branch-and-bound algorithm is only able to search a small part of the tree.

Quality of the solutions

Table 3 summarizes solution quality provided by the different tested approaches at different moments of the search. The performance indicators (Fig. 19) used to assess this quality are: (%Gap) the relative gap of the solution ([BEST-LB] / BEST); (%Red) the relative reduction of backorders in comparison to the manual solution ([UB-BEST] / UB); and (%MaxRed) the relative reduction of backorders in comparison to the maximum achievable backorder reduction ([UB-BEST] / [UB-LB]). LB is the lower bound computed by CPLEX after 11 hours when solving the MIP.

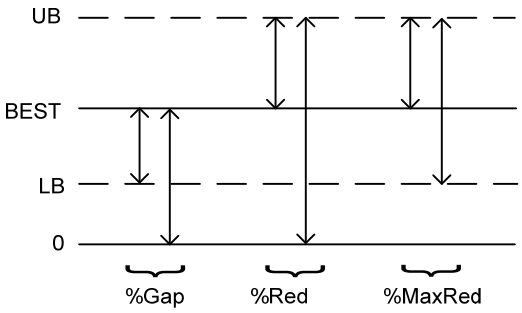

Fig. 19 Definition of the performance indicators

The second indicator (%Red) evaluates the progression of the algorithm over time on a linear scale which is not the case with the relative gap (%Gap), as the divisor changes over time. Furthermore, this measure can be interpreted as the percentage of cost reduction, which is common in the industry. The third indicator (%MaxRed) expresses the same information as (%Red) but the value is normalized to allow the comparison of performance from one dataset to another.

Table 3
Quality of solution VS computation time (summary)

		Case #1			Case #2			Case #3			Case #4		
		CP-DDS	MIP	MIP-SV	CP-DDS	MIP	MIP-SV	CP-DDS	MIP	MIP-SV	CP-DDS	MIP	MIP-SV
1 minute	%Gap	84.6	-	-	95.4	-	-	89.9	-	-	92.7	-	-
	%Red	85.5	-	-	95.1	-	-	47.3	-	-	35.1	-	-
	%MaxRed	87.5	-	-	95.3	-	-	49.9	-	-	36.8	-	-
5 minutes	%Gap	81.0	-	-	94.6	-	-	86.7	-	-	91.0	-	-
	%Red	88.3	-	-	95.9	-	-	59.9	-	-	47.9	-	-
	%MaxRed	90.3	-	-	96.1	-	-	63.2	-	-	50.2	-	-
1 hour	%Gap	80.4	-	-	94.5	-	-	86.6	-	-	89.5	-	-
	%Red	88.6	-	-	96.0	-	-	60.1	-	-	55.2	-	-
	%MaxRed	90.7	-	-	96.2	-	-	63.5	-	-	58.0	-	-
5 hours	%Gap	78.3	95.2	51.8	94.0	93.6	94.3	86.3	-	84.2	88.3	-	-
	%Red	89.7	53.6	95.4	96.3	96.5	96.1	60.8	-	66.1	59.8	-	-
	%MaxRed	91.8	54.9	97.6	96.5	96.7	96.3	64.3	-	69.8	62.8	-	-
11 hours	%Gap	77.6	95.2	20.4	93.5	87.8	92.7	86.3	66.3	84.0	88.0	-	-
	%Red	90.1	53.6	97.2	96.6	98.2	96.9	60.8	84.2	66.5	60.7	-	-
	%MaxRed	92.1	54.9	99.4	96.8	98.4	97.2	64.3	88.9	70.3	63.7	-	-

The gap (%Gap) is rather poor. The best value obtained is for the first dataset with the mixed MIP-CP approach (gap of 20.4%), which is rather modest. However, it is difficult to know if it is the solutions that are poor or the lower bound itself.

Overall, the quality of the CP approach obtains between 50.2% and 96.1% of the theoretical maximum improvement in five minutes, and between 63.7% and 96.8% in 11 hours.

5. Conclusion

In this work, we compared CP and MIP approaches for combined planning and scheduling in the context of huge industrial problems (divergent production system in presence of co-production with alternative processes). We introduced a simple and efficient way to model order lateness within a time-line model. For the CP model, we also proposed a search procedure aiming to produce good solutions in a short amount of time.

Some conclusions can be drawn from the industrial study. First, for short computation time, the CP-DDS approach provided the best solutions of all tested approaches. This CP approach best met the expectations for the real-time agent-based supply chain planning system. However, after a certain time, the CP-DDS approach does not improve significantly. If a few more hours are available, the MIP approach may provide a better solution but nothing is guaranteed with this approach.

References

[1] Bartak R. Conceptual Models for Combined Planning and Scheduling. Proceedings of CP99 Workshop on Large Scale Combinatorial Optimisation and Constraints. 1999, p. 2-14.

[2] Frayret JM, D'Amours S, Rousseau A, Harvey S, Gaudreault J. Agent-based Supply Chain Planning in the Forest Products Industry. International Journal of Flexible Manufacturing Systems 2008 (In press).

[3] Umble MM. Analyzing Manufacturing Problems Using V-A-T Analysis. Production and Inventory Management Journal 1992;33:2:55-60.

[4] Vila D, Martel A, Beauregard R. Designing logistics networks in divergent process industries: A methodology and its application to the lumber industry. International Journal of Production Economics 2006;102:2:358-78.

[5] Myers KL, Smith SF. Issues in the Integration of Planning and Scheduling for Enterprise Control. Proceedings of the DARPA Symposium on Advances in Enterprise Control. San Diego; 1999.

[6] Larsen NE, Alting L. Simultaneous engineering within process and production planning. Proceedings of the Pacific Conference on Manufacturing. Sydney; 1990, p. 1024-31.

[7] Sormaz DN, Khoshnevis B. Generation of alternative process plans in integrated manufacturing system. Journal of Intelligent Manufacturing 2003;14:6:509-26 .

[8] Yang YN, Parsaei HR, Leep HR. A prototype of a feature-based multiple-alternative process planning system with scheduling verification. Computer & Industrial Engineering 2001;39:1-2:109-24.

[9] Husbands P, McIlhagga M, Ives R. Experiments with an ecosystems model for integrated production planning. In: Back T, Fogel DB, Michalewicz Z, editors. Handbook of Evolutionary Computation. Oxford: Oxford University Press; 1995.

[10] Numao M. Integrated scheduling/planning environment for petrochemical production processes. Expert Systems with Applications 1995;8:2:263-73.

[11] Khoshnevis B, Qingmei Chen. Integration of process planning and scheduling functions. Journal of Intelligent Manufacturing 1991;2:3:165-76.

[12] Huang SH, Zhang HC, Smith ML. A progressive approach for the integration of process planning and scheduling. IIE Transactions 1995;27:4

[13] McDonnell P , Smith G, Joshi SJ, Kumara SRT. Cascading auction protocol as a framework for integrating process planning and heterarchical shop floor control. International Journal of Flexible Manufacturing Systems 1999;11:1:37-62.

[14] Lee YH, Jeong CS, Moon C. Advanced planning and scheduling with outsourcing in manufacturing supply chain. Computers & Industrial Engineering 2002;43:1-2:351-74.

[15] Weintraub A , Cormier D, Hodgson T, King R, Wilson J, Zozom A. Scheduling with alternatives: a link between process planning and scheduling. IIE Transactions 1999;31:11:1093-102.

[16] Shen W, Wang L, Hao Q. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. IEEE Transactions on Systems, Man and Cybernetics, Part C 2006/07/;36 :4:563-77.

[17] Wei Tan, Khoshnevis B. A linearized polynomial mixed integer programming model for the integration of process planning and scheduling. Journal of Intelligent Manufacturing 2004/10/;15:5:593-605.

[18] Li WD, McMahon CA. A simulated annealing-based optimization approach for integrated process planning and scheduling. International Journal of Computer Integrated Manufacturing 2007/01/;20:1:80-95.

[19] Lee H, Kim S-S. Integration of process planning and scheduling using simulation based genetic algorithms. International

Journal of Advanced Manufacturing Technology 2001;18:8:586-90.

[20] Yeo Keun Kim, Kitae Park, Jesuk Ko. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. Computers & Operations Research 2003;30:8:1151-71.

[21] Chiung Moon , Yoonho Seo. Evolutionary algorithm for advanced process planning and scheduling in a multi-plant. Computers & Industrial Engineering 2005;48:2:311-25.

[22] Bartak R. Visopt ShopFloor: on the edge of planning and scheduling. Principles and Practice of Constraint Programming - CP 2002. 8th International Conference, CP 2002. Proceedings, 9-13 Sept. 2002. 2002, p. 587-602.

[23] Wei Tan, Khoshnevis B. Integration of process planning and scheduling - a review. Journal of Intelligent Manufacturing 2000/03/;11:1:51-63.

[24] Gascon A, Lefrancois P, Cloutier L. Computer-assisted multi-item, multi-machine and multi-site scheduling in a hardwood flooring factory. Computers in Industry 1998;36:3:231-44.

[25] Yaghubian AR, Hodgson TJ, Joines JA. Dry-or-buy decision support for dry kiln scheduling in furniture production. IIE Transactions 2001;33:2:131-6.

[26] Joines JA, Culbreth CT. Job sequencing and inventory control for a parallel machine problem: a hybrid-GA approach. Proceedings of the 1999 Congress on Evolutionary Computation. 1999, p. 1130-7.

[27] Bartak R. On the Boundary of Planning and Scheduling: A Study. Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG). Manchester, UK; 1999, p. 28-39.

[28] Bartak R. Constraint Programming - What is behind? Proceedings of Constraint Programming for Decision Control Workshop. Gliwice; 1999.

[29] Hooker JN . Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. New York: John Wiley & Sons; 2000.

[30] Milano M.. Constraint and Integer Programming Toward a Unified Methodology. Berlin: Springer; 2004.

[31] Van Hentenryck P, Perron L, Puget JF. Search and strategies in OPL. ACM Transactions on Computational Logic 2000;1:2:285-320.

[32] Beaumont P. Multi-Platform Coordination and Resource Management in Command and Control. M.Sc. thesis, Université Laval; 2004.

[33] Walsh T. Depth-bounded discrepancy search. International Joint Conference on Artificial Intelligence. 1997, p. 1388-93.