Solving an integrated scheduling and routing problem with inventory, routing and penalty costs

Hugo Chevroton
Université de Tours
LIFAT, EA 6300, ERL CNRS ROOT 6305
Tours, France
hugo.chevroton@univ-tours.fr

Yannick Kergosien
Université de Tours
LIFAT, EA 6300, ERL CNRS ROOT 6305
Tours, France
yannick.kergosien@univ-tours.fr

Lotte Berghman
Toulouse Business School
Toulouse, France
l.berghman@tbs-education.fr

Jean-Charles Billaut
Université de Tours
LIFAT, EA 6300, ERL CNRS ROOT 6305
Tours, France
jean-charles.billaut@univ-tours.fr

July 31, 2020

Abstract

This paper considers an integrated routing and scheduling problem where the routing part takes into account routing costs and tardiness penalties and the scheduling part is modelled by a permutation flow shop with inventory costs. We assume that each batch is served by a dedicated vehicle, and that the number of batches and their compositions (the number of jobs and the parameters of those jobs) are known in advance. The problem is to determine the starting times of the jobs for each machine in the flow shop, the starting times of the batches and their delivery route, such that the total cost (sum of inventory, routing and penalty costs) is minimised. A two-step approach is proposed. In a first step, the optimal delivery routes for each batch and each possible departure date are calculated. This is possible as determining the min cost route for a particular delivery batch and a particular departure date is easy. As a result, we have a delivery cost function for each batch, depending on the departure date. In a second step, we use those functions to find a schedule that minimises the total cost. Computational experiments are performed on randomly generated instances.

Key Words: routing, scheduling, inventory costs, integrated problem, two-step approach

1 Introduction

Nowadays, storage is an expensive and complicated issue in a business environment. This is particularly true in production areas where products are perishable, as for food or medical products. In a hospital context, the effectiveness of a drug may decrease a few hours after manufacturing and degradation of the product is very costly. In the case of short life span, synchronisation between the production and the distribution is mandatory.

This paper considers an integrated scheduling and routing problem in a specific supply chain context. There is a single manufacturer. The production is presented by a permutation flow shop. All orders are known in advance and each job represents a customer order that has to be manufactured and delivered. Inventory costs are considered: any waiting time of a job between two consecutive machines or between the last machine and the departure time of the delivery vehicle generates a cost. Each job has to be delivered at a given due date, otherwise a tardiness penalty cost is generated. The delivery is carried out by a 3PL provider with a homogeneous fleet of vehicles, where each vehicle performs a single trip. We suppose that contracts were signed in advance: jobs are grouped in batches in order to be delivered together (*"delivery batches are fixed in advance"*). This assumption is also motivated by practical applications where several customers belong to the same group and work with the same 3PL provider, Deliveries may be grouped by geographic area, or for other logistical or political reasons. However, the route of each vehicle remains to be determined and a vehicle cannot leave the production site before all his jobs are completed.

The problem consists in fixing the production starting times of the jobs on the machines, finding the best departure date of each vehicle and determining the route of each vehicle, in order to minimise the sum of the inventory, the routing and the penalty costs.

Even if we consider only one vehicle and only the penalty costs, the problem is strongly NP-hard. Indeed, this problem can be reduced to a single-machine total weighted tardiness problem with sequence-dependent setup times, which is strongly NP-hard [Ruiz and Stützle, 2008].

The main contributions of the paper are threefold. We present a MILP program for the presented problem, we decompose the problem in two parts and we present different solution methods for both parts. More concretely, the paper is organised as follows. Section 2 gives an overview of the relevant literature on integrated scheduling and routing problems. Section 3 gives a formal description of the problem, a MILP model, a greedy algorithm and the outline of the two-step resolution method. The first step – that consists in determining the min-cost route per batch for each possible departure date – is detailed in Section 4 and Section 5 presents an approach to solve the integrated problem. Section 6 describes the benchmark instances used for the computational experiments and presents numerical results that compare the proposed methods. Finally, a conclusion and future research directions are given in Section 7.

2 Literature review

While there is a large amount of literature on scheduling and routing problems individually, the studies dealing with integrated problems are more recent but motivated by the possibility of making substantial savings by solving both problems simultaneously. In this paper, we study an integrated scheduling and routing problem where the composition of the batches is known. To the best of our knowledge, an integrated problem with a flow shop and inventory costs has never been studied in literature.

A classification of integrated problems is proposed in [Chen, 2010]. This paper provides a notation for integrated models based on different criteria such as the machines configuration, the delivery characteristics, the number of customers, the objective function etc. The author also proposes a classification of the problems in five sets, based on their delivery features. The problem studied in this paper belongs to the class *Models with batch delivery to multiple customers*. In more recent survey on integrated models can be found in [Moons et al., 2017], although there is a focus on studies where the delivery part is modelled as a VRP.

According to [Moons et al., 2017] and to the best of our knowledge, very few papers propose integrated problems with a production flow shop in the scheduling part and a VRP problem in the routing part. The most relevant paper is probably [Scholz-Reiter et al., 2010], where inventory costs are considered in the production flow shop and where routing and tardiness penalty costs are considered in the VRP. A mathematical model is presented and the performances of a commercial solver applied to small instances is evaluated.

The literature contains more papers were the production environment consist of a single machine or a set of parallel machines. For example, in [Amorim et al., 2013], the production and delivery of perishable food is studied. A set of parallel machines with setup costs is considered and a routing cost in calculated. A fleet of vehicles serves customers in different locations, taking hard time windows into account. A lot sizing formulation is used to show the interest of splitting orders between different machines. In [Wang et al., 2019b] the authors consider a 3-stage hybrid flow shop scheduling problem followed by a multi-trip vehicle routing problem. The authors want to minimize the maximum delivery time. Two methods are proposed to solve the problem: a variable neighborhood search-based procedure and a constructive heuristic combined with VNS.

Numerous scheduling models in literature take inventory costs into account. An integrated scheduling and routing model with inventory costs in studied in [Koç et al., 2017]. A set of jobs is transported from a warehouse to a workshop facility in order to be processed and they are transported back to the warehouse afterwards. A same set of vehicles is used for both transportation activities and the inventory holding costs are taken into account when evaluating a solution. A branch-and-bound approach with several lower bounds and different heuristics are presented. In [Bülbül et al., 2004], authors study a m-machine flow

shop earliness/tardiness scheduling problem with intermediate inventory holding costs and ready times. They propose a heuristic based on the LP relaxation and a Dantzig-Wolfe reformulation that is solved approximately by column generation. Even if this paper does not take a delivery part into account, it encompasses most of the production aspect of this paper.

Inventory costs are also important in lot sizing problems where the product demand is known over a certain number of periods and for each of those periods the quantities to produce and to store must be determined. In [Neves-Moreira et al., 2019] and [Absi et al., 2015], a production routing problem that combines a lot-sizing problem and a vehicle routing problem is studied. A meat producer problem where the production of several types of goods is planned on parallel machines over several periods is addressed in [Neves-Moreira et al., 2019]. For each period, customers are delivered by vehicles to satisfy daily demand. Delivery time windows need to be respected and both inventory cost on customer sites and routing costs need to be minimised. In [Absi et al., 2015], a two-step method is proposed to solve a problem with a single product. The first step determines the quantities produced, stored and delivered to each customer over the different periods and the second step determines the delivery routes. In [Absi et al., 2018], a comparison between a sequential and an integrated approach for the previous problem is presented. The integrated approach is solved with the state of the art heuristic, while the subproblems of the sequential approaches are solved optimally. The computational results show that the quality of the approaches strongly depends on the characteristics of the instance.

In integrated scheduling and routing problems, the customer satisfaction is often expressed with respect to due dates or time windows. For example, in [Park and Hong, 2009], a deadline must be respected and a soft due date is used to penalise solutions for which late deliveries occur. Setup costs and setup times are taken into account during the production as well as routing costs during the delivery. The integrated problem with a single machine and a set of vehicles to deliver items to customers is solved using a genetic algorithm. In [Yağmur and Kesen, 2020], the authors proposed a problem where a set of jobs must be scheduled on a permutation flow-shop and delivered to distant customers by a single vehicle. The objective is to minimise the travel time and the delivery tardiness. The integrated problem is solved using a genetic algorithm. In [Wang et al., 2019a], authors also use a set of parallel machines for production. Jobs are delivered by an homogeneous fleet of vehicles with limited capacity and within a predefined time window. An uncertain travel time is taken into account and the goal is to maximise the robustness of the solution. A genetic algorithm is proposed. In [Chang et al., 2014], a solution is evaluated by a fixed vehicle usage cost, the total travel costs and the delivered service to the customers. The later is defined as the weighted sum of the job delivery times. An ACO algorithm is proposed to solve this problem with unrelated parallel machines and a set of vehicles.

Some papers introduce the notion of short life-span constraints for jobs. In a chemotherapy production

context, like in [Kergosien et al., 2017], drugs have a short life-span between the production completion time and the intervention date for the patient. The objective function is to minimise the maximum delivery tardiness. Drugs are produced on parallel machines and delivered to different sites by a single vehicle. A Benders decomposition-based heuristic with several cuts, upper and lower bounds is proposed to solve this problem. Short life-span constraints also occur in home meal delivery services, like in [Dayarian and Desaulniers, 2019]. The model ensures freshness of delivered product, customers satisfaction using time window constraints and employees satisfaction. The meal production is modelled as a parallel machine scheduling problem and deliveries are carried out by a homogeneous fleet of vehicles with limited capacity. A branch-and-price algorithm and matheuristics are proposed as solution methods. In most of these articles, the delivery part could assimilate to a simple vehicle routing problem.

The problem treated in this paper is an extension of two studies: [Rohmer and Billaut, 2015] and [Chevroton et al., 2018]. In these two articles, the production part is a permutation flow shop, batches must be determined and the delivery is ensured by a fleet of vehicles. The objective is to minimize the sum of the inventory, routing and penalty costs. In [Rohmer and Billaut, 2015], only mixed integer linear programming models are presented. In [Chevroton et al., 2018], the manufacturer assumes that the third part logistic provider uses the earliest due date rule to serve his customers and schedules the production and decides the batching and the vehicle departure dates accordingly. Heuristics are proposed to solve this problem.

3 Problem description and outline of the resolution method

3.1 Problem statement and notations

In the integrated problem that we consider, the production workshop is a permutation flow shop: all the jobs are executed in the same order on the machines. Once the jobs are completed, they have to be delivered to customers in predefined batches. Each job is already assigned to a given batch that corresponds to a given vehicle and there is only one batch per vehicle. We assume that the fleet size is equal to the number of batches. Each vehicle starts its route at the production site as soon as the last job of its delivery batch is completed and comes back to the production site once the route is finished. The schedule of the jobs on the machines and the routing of the vehicles have to be determined.

The n jobs to schedule on m machines $M_1, M_2, ..., M_m$ are grouped into V batches. We denote by J^v the set of jobs in batch v and $n_v = |J^v|$, with $v \in \{1, \ldots, V\}$.

The processing time of job j on machine M_i is denoted by $p_{j,i}$. Furthermore, each job j is characterised by a due date d_j, a unitary tardiness delivery cost π_j and a delivery site s_j.

We consider inventory costs throughout the entire production process, from the arrival of raw materials until the departure of the final product. The unitary initial inventory (START) cost h_j^{START} represents the

storage cost of the raw materials of job j that are not processed yet, i.e. before the start of the processing on machine M_1. The unitary work-in-process inventory (WIP) cost $h_{j,i}^{WIP}$ represents the storage cost of job j between its completion time on machine M_i and its starting time on machine M_{i+1}. The final inventory (FIN) cost h_j^{FIN} represents the storage cost of the final product between its completion time and the departure date of its vehicle for delivery. Finally, we denote by $tt_{j,k}$ and $tc_{j,k}$ with $j,k \in \{0,\ldots,n\}^2$ the travel time and travel cost between site s_j and site s_k. Note that index 0 represents the production site.

It is well known that finding the production sequence of jobs in a m-machine permutation flow shop minimizing the makespan (corresponding to the departure date of a vehicle delivering all jobs) is a strongly NP-hard problem [Garey et al., 1976]. Minimising the routing cost associated to one route is equivalent to solving a travelling salesman problem which is also known to be NP-hard [Karp, 1975]. Clearly, the problem that we consider is NP-hard.

3.2 Mixed Integer Linear Program

In this section, we propose an MILP formulation of the problem that we call $GLOBAL_{MILP}$. The decision variables are the following:

$$y_{j,k} = \begin{cases} 1 & \text{if job } j \text{ is scheduled just before job } k \\ 0 & \text{otherwise} \end{cases}$$

$\forall j,k \in \{0,\ldots,n\}^2$ and where job 0 is a dummy representing both the first and the last job.

$$x_{v,j,k} = \begin{cases} 1 & \text{if in the route of vehicle } v \text{ the site } s_j \text{ is visited just before the site } s_k \\ 0 & \text{otherwise} \end{cases}$$

$\forall v \in \{1,\ldots,V\}$ and $\forall j,k \in \{0\} \cup J^v$ and where index 0 represents the production site. $C_{j,i} \geq 0$ is the completion time of job j on machine M_i, $\forall j \in \{1,\ldots,n\}, \forall i \in \{1,\ldots,m\}$. $G_v \geq 0$ is the departure date of vehicle v, $\forall v \in \{1,\ldots,V\}$. $D_j \geq 0$ is the delivery date of job j, $\forall j \in \{1,\ldots,n\}$. $T_j \geq 0$ is the delivery tardiness of job j, $\forall j \in \{1,\ldots,n\}$.

To formulate the objective function we use the following intermediate variables: IC is the total inventory cost, IC^{START} is the total starting inventory cost, IC^{WIP} is the total work-in-process inventory cost and IC^{FIN} is the total final inventory cost. RC is the total routing cost and PC is the total penalty cost associated to the delivery tardiness where PC_v denotes the part related to batch v.

The expressions of these intermediate variables are the following (notice that some constant elements could be deleted).

$$IC = IC^{START} + IC^{WIP} + IC^{FIN} \tag{1}$$

$$IC = \sum_{j=1}^{n}(C_{j,1} - p_{j,1})h_j^{START} + \sum_{j=1}^{n}\sum_{i=1}^{m-1}(C_{j,i+1} - p_{j,i+1} - C_{j,i})h_{j,i}^{WIP} + \sum_{v=1}^{V}\sum_{j\in J^v}(G_v - C_{j,m})h_j^{FIN} \quad (2)$$

and

$$RC = \sum_{v=1}^{V} RC_v = \sum_{v=1}^{V}\sum_{j\in\{0\}\cup J^v}\sum_{k\in J^v} tc_{j,k}x_{v,j,k} \quad (3)$$

$$PC = \sum_{v=1}^{V} PC_v = \sum_{v=1}^{V}\sum_{j\in J^v}\pi_j T_j \quad (4)$$

The formulation $GLOBAL_{MILP}$ is the following.

$$\text{Minimise } IC + RC + PC \quad (5)$$

$$\sum_{k=1}^{n} y_{j,k} = 1 \qquad \forall j \in \{0,\ldots,n\} \quad (6)$$

$$\sum_{j=0}^{n} y_{j,k} = 1 \qquad \forall k \in \{0,\ldots,n\} \quad (7)$$

$$p_{j,1} \leq C_{j,1} \qquad \forall j \in \{1,\ldots,n\} \quad (8)$$

$$C_{j,i} + p_{k,i} - M(1 - y_{j,k}) \leq C_{k,i} \qquad \forall j,k \in \{1,\ldots,n\}^2, \forall i \in \{1,\ldots,m\} \quad (9)$$

$$C_{j,i} + p_{j,i+1} \leq C_{j,i+1} \qquad \forall j \in \{1,\ldots,n\}, \forall i \in \{1,\ldots,m-1\} \quad (10)$$

$$C_{j,m} \leq G_v \qquad \forall v \in \{1,\ldots,V\}, \forall j \in J^v \quad (11)$$

$$\sum_{k\in\{0\}\cup J^v\setminus\{j\}} x_{v,j,k} = 1 \qquad \forall v \in \{1,\ldots,V\}, \forall j \in \{0\}\cup J^v \quad (12)$$

$$\sum_{j\in\{0\}\cup J^v\setminus\{k\}} x_{v,j,k} = 1 \qquad \forall v \in \{1,\ldots,V\}, \forall k \in \{0\}\cup J^v \quad (13)$$

$$x_{v,k,j} = 0 \qquad \forall j,k \in \{J^v\}^2 : j < k, tt_{j,k} = 0, \forall v \in \{1,\ldots,V\} \quad (14)$$

$$G_v + tt_{0,j} \leq D_j \qquad \forall j \in J^v, \forall v \in \{1,\ldots,V\} \quad (15)$$

$$D_j + tt_{j,k} - M(1 - x_{v,j,k}) \leq D_k \qquad \forall j,k \in \{J^v\}^2, \forall v \in \{1,\ldots,V\} \quad (16)$$

$$D_j - d_j \leq T_j \qquad \forall j \in \{1,\ldots,n\} \quad (17)$$

with M an arbitrary high value.

Constraints (6) ensure that each job (including the dummy first job) has exactly one successor in the sequence (which might be the dummy last job). Constraints (7) ensure that each job (including the dummy last job) has exactly one predecessor in the sequence (which might be the dummy first job). Constraints (8), (9) and (10) give the values of the job completion times, respecting the processing order on the machines and the disjunctive capacity of the resources. Constraints (11) ensure that a vehicle cannot leave the production site before the completion of all the jobs in its batch. Constraints (12) ensure that each batch, each customer site and the production site all have exactly one successor in the delivery sequence (which

might be the production site as final point). Constraints (13) ensure that each batch, each customer site and the production site all have exactly one predecessor in the delivery sequence (which might be the production site as starting point). Constraints (14) are symmetry breaking constraints that impose that the jobs in a batch with the same location are delivered in increasing order of their index. Constraints (15) ensure that a customer cannot be served before the vehicle departure date plus the travel time from the production site to its location. Constraints (16) give the value to the job delivery dates respecting the travel times between the different sites. Finally, Constraints (17) determine the tardiness of jobs. We can note that the elimination of subtours is guarantied by the combination of Constraints (12), (13), (15) and (16).

This model contains $O(n^2 + Vn_k^2)$ binary variables, $O(nm)$ continuous variables and $O(n^2m + Vn_k^2)$ constraints (including $O(n^2m + Vn_k^2)$ constraints with an high value).

3.3 Greedy algorithm

In order to present the added value of our methods, a greedy algorithm (GR) is proposed. This algorithm builds a solution in a way that is close to what a human operator could do. This algorithm considers the different batches produced one by one as a whole, i.e. all the jobs of a batch are produced successively. The batches are produced in non-decreasing average delivery date. The jobs inside a batch are scheduled using the NEH heuristic [Nawaz et al., 1983] and the schedule is left shifted. Afterwards, the deliveries inside a batch are sequenced following the Nearest Neighbour rule.

3.4 Outline of the resolution method

Remember that two integrated problems have to be solved: the scheduling of the jobs on the permutation flow shop and the sequencing of the delivery sites for each batch. These problems are linked by the vehicle departure dates. Changing a departure date may change both the inventory costs and the penalty costs.

The resolution method is a two-step algorithm. In the first step, the min-cost delivery routes are found for each possible departure date of each batch. The results are aggregated into a set of piecewise linear cost functions, one for each batch, that we call "delivery cost functions" or DC functions. They are detailed in Section 4.1 and different methods (exact and heuristic) to build these functions are given in Section 4.2.

Then, Section 5 presents the second step. Both a MILP model (Section 5.1) and a CP model (Section 5.2) including the DC functions are presented. Assuming that the production sequence of jobs is known, an optimal evaluation function is given by a Linear Programming model presented in Section 5.3. A neighbourhood search method to determine the best possible sequence of jobs is proposed in Section 5.4. Each sequence is evaluated using the before mentioned function.

4 Preprocessing: Delivery cost (DC) functions

In this section we formally define the DC functions and propose some methods to build them.

4.1 Definition

For each batch, we have to evaluate the total cost for all possible departure dates comprised between 0 and an upperbound T^{prod}, the highest possible departure date of a delivery tour which is equal to the highest possible completion time of a job. The sum of the routing and the penalty costs ($RC + PC$) which we will call the *delivery cost*, is a function which increases linearly with the departure time.

We denote by DC_v the function associated to batch v which gives for each departure date $t \in \{0, \ldots, T^{prod}\}$ the minimal delivery cost. For a particular route, this function is non decreasing and piecewise linear. Increasing the departure date increases the penalty costs if there is at least one late delivery. From a certain departure date on, the optimal route always remains the same and the penalty cost for a job is a linear function of the tardiness. Considering another route and taking the minimum cost of both also leads to a piecewise linear function. So the DC_v function associated to batch v is also piecewise linear. Figure 1 presents an example of a function profile.

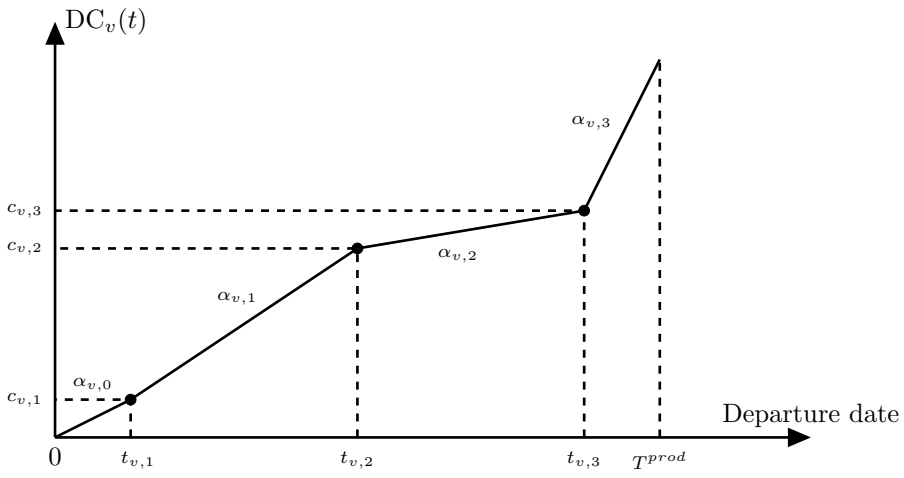

Figure 1: Routing and penalty cost of batch v

The notations are the following:

- S_v is the number of segments of the piecewise linear function associated to batch v,
- $t_{v,s}$ is the first departure date of batch v associated to segment s ($1 \leq s \leq S_v$),
- $c_{v,s}$ is the delivery cost of batch v for a departure date equal to $t_{v,s}$,

- $\alpha_{v,s}$ is the slope coefficient associated to segment s for batch v. Notice that this slope is equal to the sum of the unitary penalty costs of the late jobs.

At date $t_{v,s}$, two possible conditions might impose the creation of a new segment s. Either the same delivery route is the less costly but when the departure date is equal to $t_{v,s}$, there is at least one new tardy job. Or another delivery route becomes less costly for the departure date $t_{v,s}$. In the first case, the slope generally increases. In the second case, the slope generally decreases.

We define the function $\mathrm{DC}_{v,r}$ as the delivery cost function associated to a route r that delivers all the jobs of batch v. This piecewise linear function can be obtained by identifying iteratively the inflexion points of the function and by computing the delivery cost of r for each of these dates. In this way, $\mathrm{DC}_{v,r}$ is computed in $O(n\log(n))$ with $n = |J^v|$. Afterwards, the DC_v functions are determined by a minimum set of routes $\mathcal{R}_v$ defined by: $\forall t$, $\mathrm{DC}_v(t) = \min_{r\in\mathcal{R}_v} \mathrm{DC}_{v,r}(t)$ and $\forall r \in \mathcal{R}_v, \exists t \in \{0,\ldots,T^{prod}\}$: $\mathrm{DC}_{v,r}(t) = \mathrm{DC}_v(t)$.

Example: A batch is composed of three jobs a, b and c. Each job has a due date ($d_a = 16$, $d_b = 15$ and $d_c = 22$) and a tardiness penalty cost ($\pi_a = 1, \pi_b = 3, \pi_c = 3$). The production site is denoted by PS. The travel times are presented in Figure 2. We assume that the travel time and the travel cost are equal. Two routes are considered: r and r' with $r = (b, a, c)$ and $r' = (a, c, b)$. Figure 3 gives the delivery dates of jobs for each route according to the departure date $t = 0$.

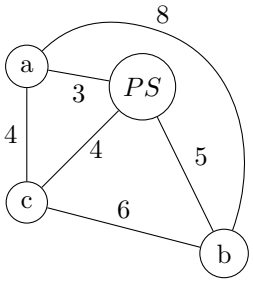

Figure 2: Distance matrix

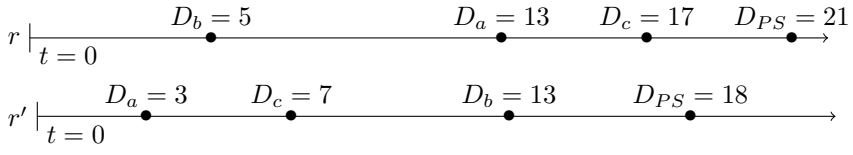

Figure 3: Travel time and routing costs

Figure 4 and Figure 5 represent $\mathrm{DC}_{v,r}$ and $\mathrm{DC}_{v,r'}$ functions for $t \in [0, 16]$. For example, for $t = 8$ the routing and penalty cost of route r is $\mathrm{DC}_{v,r}(8) = 35$ (tardiness of jobs a and c plus routing cost). Figure 6 represents the result $f(t) = \min\{\mathrm{DC}_{v,r}(t), \mathrm{DC}_{v,r'}(t)\}$: for intervals $[0, 3]$ and $[9, 16]$, the route r' dominates the route r and for interval $[3, 9]$ the route r dominates the route r'.

4.2 Methods to build DC functions

Several methods can be used to build the DC function for a batch. We start with a mathematical programming model. Afterwards we present the heuristic that we will use as an upper bound in our second exact

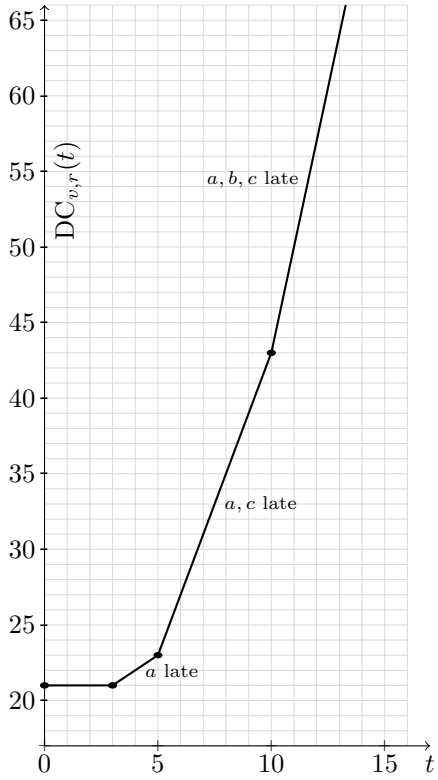

Figure 4: $\mathrm{DC}_{v,r}(t)$

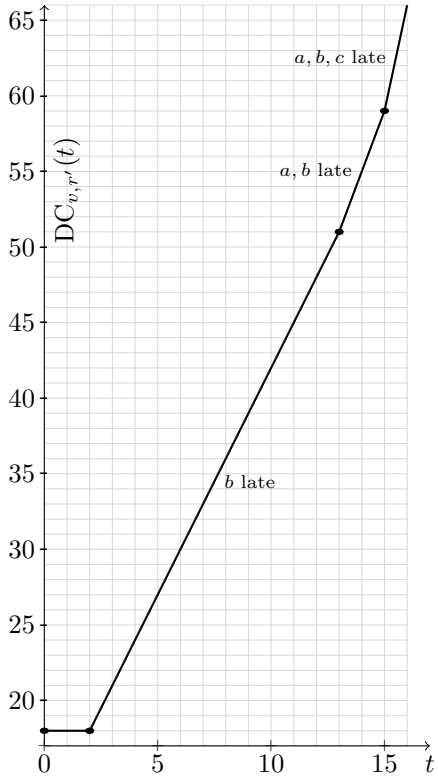

Figure 5: $\mathrm{DC}_{v,r'}(t)$

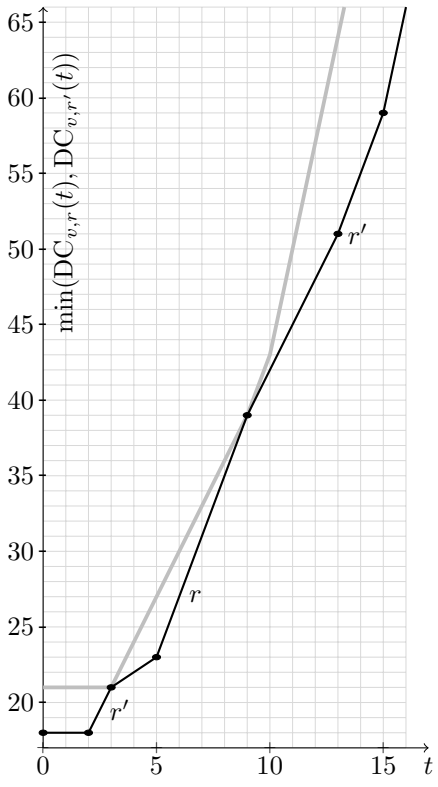

Figure 6: $\min(\mathrm{DC}_{v,r}(t), \mathrm{DC}_{v,r'}(t))$

algorithm that is detailed at the end of this section and that is inspired by the Branch-and-Bound method.

4.2.1 MILP model to build DC functions

To determine the lowest delivery cost for all possible starting dates $t \in \{0, \ldots, T^{prod}\}$ of batch v, we solve the following MILP model. This model uses some constraints already presented in Section 3 that are limited to batch v we add Constraints (18) to ensure that batch v is delivered with a departure date equal to t.

$$\text{Minimise } RC_v + PC_v$$

$$(12), (13), (14), (16), (17)$$

$$D_j \geq t + tt_{0,j} \qquad \forall j \in J^v \tag{18}$$

In order to generate the $DC_v(t)$ functions, this model is solved for each departure date t in $\{0, \ldots, T^{prod}\}$ and for each batch v.

4.2.2 Heuristic method

Because the DC_v functions must be calculated for all batches, we propose a heuristic method that approximates the DC_v function within a short computation time. Algorithm 1 describes the main steps of the heuristic. We denote by DC_v^{app} the approximation of the DC_v function and by $\mathcal{R}_v^{app}$ the corresponding set of routes, such that: $\mathrm{DC}_v^{app}(t) = \min_{r \in \mathcal{R}_v^{app}} \mathrm{DC}_{v,r}(t)$ and $\forall r \in \mathcal{R}_v^{app}, \exists t \in \{0, \ldots, T^{prod}\}$: $\mathrm{DC}_{v,r}(t) = \mathrm{DC}_v^{app}(t)$.

Algorithm 1

1: $\mathcal{R}_v^{app} \leftarrow Initial\ route\ computation()$
2: $Tabu \leftarrow \emptyset$
3: **while** $(\mathcal{R}_v^{app} \backslash Tabu) \neq \emptyset$ **do**
4: $\quad (t, r) \leftarrow Route\ Selection(\mathcal{R}_v^{app} \backslash Tabu)$
5: $\quad \{r_{best}, \mathcal{R}_v^{app}\} \leftarrow\ Local\ Search(t, r, \mathcal{R}_v^{app})$
6: $\quad Tabu \leftarrow Tabu \cup r_{best}$
7: **end while**
8: **return** $\mathcal{R}_v^{app}$

The first step of Algorithm 1 consists in initializing the set $\mathcal{R}_v^{app}$ of routes using the procedure *Initial route computation*(). $\mathcal{R}_v^{app}$ is initially composed of a single route, obtained by one among the eight following strategies:

- NN: the Nearest Neighbour first rule (NN rule) [Cover and Hart, 1967].
- NN^{rev}: the reverse sequence of the NN rule.
- $NN_{2,1}$: the second and the first half of the NN sequence are concatenated.
- $NN_{2,1}^{rev}$: the reverse sequence of $NN_{2,1}$.
- EDD: the Earliest Due Date first rule.
- EDD^{rev}: the reverse sequence of the EDD rule.
- $EDD_{2,1}$: the second and the first half of the EDD sequence are concatenated.
- $EDD_{2,1}^{rev}$: the reverse sequence of $EDD_{2,1}$.

Several possibilities are considered. In the 1-start method only the NN strategy is used, in the 2-start method, both the NN and the EDD strategy are used and in the 8-start method, all eight strategies are used. Remark that when we have more strategies, the final solution is obtained by the union of all $\mathcal{R}_v^{app}$ results determined by each strategy.

Before improving the routes in $\mathcal{R}_v^{app}$ by a local search, an empty tabu list is created. This tabu list will contain all routes that cannot be improved by the local search. At each iteration of the main loop (lines 3-7), a random route $r \in \mathcal{R}_v^{app}$ that is not in the tabu list yet is selected and a date t is determined such that t is the earliest departure from which the associated route r becomes the best route among all routes in $\mathcal{R}_v^{app}$ (i.e. $\min_v \mathrm{DC}_v^{app}(t) = \mathrm{DC}_{v,r}(t)$). For route r, date t and set $\mathrm{DC}_v^{app}(t)$, a local search is launched in line 5. Building the neighbourhood of a current route with a fixed departure date t consists in moving one job to another place in the delivery sequence. We move each job to all possible places to obtain the complete neighbourhood and the route that minimises the delivery cost for departure date t becomes the next current route. The local search stops when the current route cannot be improved anymore using the presented neighbourhood. Then, the final route is noted r_{best} and is added to the tabu list (line 6).

Whenever a neighbour route r' can improve the current approximation function DC_v^{app}, i.e. $\exists t \in \{0, \ldots, T^{prod}\}$: $\mathrm{DC}_v^{app}(t) > \mathrm{DC}_{v,r'}(t)$, the route r' is added to $\mathcal{R}_v^{app}$. After adding a route in $\mathcal{R}_v^{app}$, a route r'' is removed from $\mathcal{R}_v^{app}$ if $\forall t \in \{0, \ldots, T^{prod}\}$ $\mathrm{DC}_v^{app}(t) < \mathrm{DC}_{v,r''}(t)$.

The main loop (line 3 to 7) stops when all routes in $\mathcal{R}_v^{app}$ are included in the Tabu list (i.e. no route in $\mathcal{R}_v^{app}$ can be improved anymore by the local search).

4.2.3 Branch-and-Bound approach

In the Branch-and-Bound approach, a node of the search tree represents a partial delivery sequence σ of jobs, associated to a set τ of possible departure dates. For each partial sequence σ, A_σ is the set of remaining jobs. In the root node $\sigma = \emptyset$ and $\tau = \{0, \ldots, T^{prod}\}$. The initial upper bound is given by the heuristic algorithm presented in Section 4.2.2 which returns a value $UB(t)$ for any $t \in \{0, \ldots, T^{prod}\}$ and the associated sequences for each t.

The branching consists in selecting the next job to deliver. A child node σ' is composed of the sequence of parent node σ and one additional job from A_σ. To determine the associated departure dates τ' of σ', a lower bound $lb_{\sigma'}(t)$ on the delivery cost is computed for any $t \in \tau$ and $\tau' = \{t \in \tau : lb_{\sigma'}(t) < UB(t)\}$. Whenever τ' is empty, the node is pruned, because the corresponding partial sequence will never lead to an improvement of $UB(t)$. A leaf of the tree may be a minimum cost sequence for some $t \in \{0, \ldots, T\}$ and in that case $UB(t)$ is updated. The procedure stops when all the remaining nodes are leaves and $UB(t)$ is the mincost DC function.

Lower bound computation The lower bound $lb_\sigma(t)$ on the delivery cost for a departure date t is decomposed in three parts: $lb_\sigma(t) = RPC_\sigma(t) + lb_\sigma^R + lb_\sigma^P(t)$ with:

- $RPC_\sigma(t)$ the delivery costs of the partial sequence σ with departure date t,
- lb_σ^R a lower bound on the routing cost to serve all jobs in A_σ,

- $lb_\sigma^P(t)$ a lower bound on the penalty cost to serve all jobs in A_σ, when departing at date t.

lb_σ^R is computed by solving a minimum spanning tree problem on the complete graph G^R containing $|A_\sigma|+2$ vertices: one vertex per remaining job to deliver, a vertex for the last job in sequence σ and one for the depot. The values on the edges are equal to the travel costs. Remark that lb_σ^R does not depend on the departure date t.

$lb_\sigma^P(t)$ is computed using the complete graph G^P composed of $|A_\sigma|+1$ vertices: one vertex per remaining job to deliver and a vertex for the last job in sequence σ. The values on the edges are equal to the travel times. First, the earliest possible delivery dates are computed based on the minimum spanning tree of graph G^P and on Theorem 1. Afterwards, a minimum cost assignment of the jobs to the dates is determined.

Theorem 1. *For any path of size s in a graph G, the length of this path is higher than or equal to the sum of the s smallest edges of the minimum spanning tree of G.*

Proof. We define μ as a path of size s in G and A as the set of nodes of μ so $|A| = s+1$. We denote by T^* the minimum spanning tree of G and by T_A^* the minimum spanning tree of the subgraph of G induced by A. The total length of μ cannot be smaller than the total length of T_A^*, because μ is a particular tree of A: it is connected and without cycles, as T_A^*. Furthermore, the length of T_A^* cannot be smaller than the total length of the s smallest edges of T^*, because a tree in A contains at most s edges, which proofs the theorem. □

According to Theorem 1, the i^{th} earliest possible delivery date is higher than or equal to the delivery date of the last job in sequence σ plus the sum of the i smallest travel times in the minimum spanning tree of G^P (noted $D_i^{min}(t)$).

The lower bound $lb_\sigma^P(t)$ is given by the optimal objective value of a linear assignment problem with a $|A_\sigma| \times |A_\sigma|$ cost matrix $U(t)$ where

$$u_{j,i}(t) = \max\{0, \pi_j(D_i^{min}(t) - d_j)\}$$

with $j \in A_\sigma$ and $i \in \{1, \ldots, |A_\sigma|\}$.

Theorem 2. *$lb_\sigma^P(t)$ is a lower bound on the penalty cost to serve all jobs in A_σ.*

The proof of Theorem 2 can be found in Section 8 (Appendix).

The Kuhn-Munkres algorithm [Kuhn, 1955, Munkres, 1957] can be used to solve this assignment problem in polynomial time. Remark that between the computing of the lower bound $lb_\sigma(t)$ and $lb_\sigma(t+1)$, only the cost matrix $U(t)$ of the assignment problem changes, $D_i^{\min}(t+1) = D_i^{\min}(t) + 1$. The minimal routing cost lb_σ^R and the minimum spanning tree are unchanged.

An example of the lower bound computation is given in Figures 7 and 8. The example is composed of 3 remaining jobs k, l and m with: $d_k = 5$, $\pi_k = 1$, $d_l = 9$, $\pi_l = 2$, $d_m = 11$, $\pi_m = 3$ and a minimum spanning

tree as given in Figure 7. The minimal routing cost lb_σ^R is equal to 7. Supposing that the delivery date t of the last job in sequence σ is equal to 8, we can calculate $D_1^{\min}(8) = 8+1 = 9$, $D_2^{\min}(8) = 8+1+2 = 11$ and $D_3^{\min}(8) = 8+1+2+2 = 13$. Moreover, we can calculate all values for $u_{j,i}(8)$ using the above formula. For example for job k and $i = 2$, we can calculate $u_{k,2}(8) = \max\{0, \pi_k(D_2^{min}(8) - d_k)\} = \max\{0, 1(11-5)\}$. All possible values are represented in Figure 8. To solve the assignment problem, the Kuhn-Munkres algorithm (see [Kuhn, 1955] and [Munkres, 1957]) applied on the matrix of Figure 8, assigns task k to the third departure date, task l to the first date and task m to the second and gives a lower bound on the penalty cost equal to $8 + 0 + 0 = 8$.

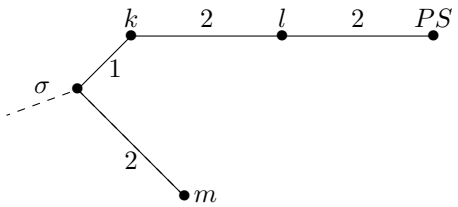

Figure 7: Minimum spanning tree

$u_{j,i}(8)$	$i=1$	$i=2$	$i=3$
$j=k$	4	6	8
$j=l$	0	4	8
$j=m$	0	0	6

Figure 8: Assignment matrix

5 Scheduling resolution methods

When the preprocessing step has been performed, the DC functions are known for each batch. The remaining problem consists in determining the sequence of the jobs. For a given job sequence, we can easily calculate the job completion times, the inventory cost and the earliest departure time for each batch. And once we know these departure times, the corresponding delivery costs can be read in the DC functions.

In this section, we first present a new MILP model and a constraint programming model, both using the DC functions. Then, we present an optimal timing procedure to find the job completion times and the vehicle departure dates for a fixed sequence. Finally, a neighbourhood search method is proposed to find the best possible sequence using this optimal timing procedure for evaluation.

5.1 MILP model including DC functions

DC_{MILP} is a new MILP model that is based on $GLOBAL_{MILP}$ of Section 3.2 and the DC functions. Let us remember that a DC_v function is defined by S_v segments where $t_{v,s}$ is the first departure date associated to segment s, $c_{k,s}$ is the penalty and routing costs for a departure date equal to $t_{v,s}$ and $\alpha_{v,s}$ is the slope coefficient associated to segment s. T^{prod} is the upper bound on the production time.

The decision variables of the DC_{MILP} model are the following: as in $GLOBAL_{MILP}$: $y_{j,k}$ binary variables $\forall j,k \in \{0,\ldots,n\}^2$, $C_{j,i}$ positive continuous variables $\forall j \in \{1,\ldots,n\}, i \in \{1,\ldots,m\}$, G_v positive

continuous variables $\forall v \in \{1, \ldots, V\}$.

$$z_{v,s} = \begin{cases} 1 & \text{if the departure date } G_v \text{ of batch } v \text{ belongs to the } s^{th} \text{ segment of the DC}_v \text{ function} \\ 0 & \text{otherwise} \end{cases}$$

$\forall v \in \{1, \ldots, V\}, \forall s \in \{1, \ldots, S_v\}$. $u_{v,s} \geq 0$, such that $u_{v,s} + t_{v,s}$ is equal to the departure date of the vehicle associated to batch v if $z_{v,s} = 1$ (otherwise, $u_{v,s} = 0$), $\forall v \in \{1, \ldots, V\}$, $\forall s \in \{1, \ldots, S_v\}$.

The model is the following.

$$\text{Minimize } IC + \sum_{v=1}^{V} \sum_{s=1}^{S_v} (z_{v,s} c_{v,s} + u_{v,s} \alpha_{v,s}) \tag{19}$$

$$s.t.\ (2), (6), (7), (8), (9), (10), (11)$$

$$G_v = \sum_{s=1}^{S_v} (z_{v,s} t_{v,s} + u_{v,s}) \qquad \forall v \in \{1, \ldots, V\} \tag{20}$$

$$\sum_{s=1}^{S_v} z_{v,s} = 1 \qquad \forall v \in \{1, \ldots, V\} \tag{21}$$

$$u_{v,s} \leq z_{v,s}(t_{v,s+1} - t_{v,s} - 1) \qquad \forall v \in \{1, \ldots, V\}, \forall s \in \{1, \ldots, S_v - 1\} \tag{22}$$

$$u_{v,S_v} \leq z_{v,S_v}(T^{prod} - t_{v,S_v}) \qquad \forall v \in \{1, \ldots, V\} \tag{23}$$

$$u_{v,s} \geq 0 \qquad \forall v \in \{1, \ldots, V\}, \forall s \in \{1, \ldots, S_v\} \tag{24}$$

$$z_{v,s} \in [0, 1] \qquad \forall v \in \{1, \ldots, V\}, \forall s \in \{1, \ldots, S_v\} \tag{25}$$

The objective function (19) remains the same: minimize the sum of the inventory, the routing and the penalty costs. Constraints (6) to (11) express the scheduling part of the problem. Constraints (20) define the departure date of batch v, using the information of the DC_v function and Constraints (21) to (23) model the DC_v functions in order to obtain the delivery costs for the different departure dates.

This model contains $O(n^2 + VS_v)$ binary variables, $O(nm + VS_v)$ continuous variables, $O(nm + VS_v)$ constraints without big-M and $O(n^2 m)$ big-M constraints.

5.2 Constraint Programming model using DC functions

The proposed model uses interval variables, interval sequence variables and global constraints. Interval variables are commonly used to model scheduling problems as they allow to define the start time, the end time and the processing time of a job in a single decision variable. An interval variable is associated to several attributes like "size", "start" and "end" which represent the size, the start time and the completion time of a job, respectively. An interval sequence variable on a set of interval variables aims to model a total ordering inside this set of interval variables. This type of variable can be used in global constraints like "noOverlap"

or "sameSequence" allowing to have a chain of non-overlapping intervals and to impose identical sequences, respectively. We refer the reader to [Laborie et al., 2018] for more details.

The decision variables are the following : The interval $\mathcal{I}_{j,i}$ models the processing duration of a job j on a machine M_i, $\forall j \in \{1,\ldots,n\}$, $\forall i \in \{1,\ldots,m\}$. The sequence Ψ_i models the sequence of all intervals $\mathcal{I}_{j,i}$ on machine M_i, $\forall i \in \{1,\ldots,m\}$: $\Psi_i = Sequence(\bigcup_{j\in\{1,\ldots,n\}} \mathcal{I}_{j,i})$.

The constraint programming framework allows us to define directly the piecewize linear DC_v function (called "piecewise" function in CSP), based on the set of dates $t_{v,s}$ and the set of slopes $\alpha_{v,s}$, $\forall v \in \{1,\ldots,V\}$ and $\forall s \in \{1,\ldots,S_v\}$. The objective function becomes:

$$\begin{aligned}\min RC + PC + IC = \min \sum_{v=1}^{V} \mathrm{DC}_v(G_v) + \sum_{j=1}^{n} start(\mathcal{I}_{j,1}) h_j^{START} \\ + \sum_{j=1}^{n}\sum_{i=1}^{m-1} (start(\mathcal{I}_{j,i+1}) - p_{j,i} - start(\mathcal{I}_{j,i})) h_{j,i}^{WIP} \\ + \sum_{v=1}^{V}\sum_{j\in J^v} (G_v - p_{j,m} - start(\mathcal{I}_{j,m})) h_j^{FIN}\end{aligned}$$

And the constraints are as follows:

$$size(\mathcal{I}_{j,i}) = p_{j,i} \qquad \forall j \in \{1,\ldots,J\}, \forall i \in \{1,\ldots,I\} \tag{26}$$

$$end(\mathcal{I}_{j,i}) \leq start(\mathcal{I}_{j,i+1}) \qquad \forall j \in \{1,\ldots,J\}, \forall i \in \{1,\ldots,I-1\} \tag{27}$$

$$0 \leq start(\mathcal{I}_{j,1}) \qquad \forall j \in \{1,\ldots,J\} \tag{28}$$

$$noOverlap(\Psi_i) \qquad \forall i \in \{1,\ldots,I\} \tag{29}$$

$$sameSequence(\Psi_0 1, \Psi_i) \qquad \forall i \in \{1,\ldots,I\} \tag{30}$$

$$end(\mathcal{I}_{j,m}) \leq G_v \qquad \forall v \in \{1,\ldots,V\}, \forall j \in J^v \tag{31}$$

Constraints (26) equals the interval size of each job on each machine to the correct processing time. Constraints (27) ensure the production order of the jobs on the different machines and Constraints (28) ensure positive starting times. Constraints (29) guaranty the capacity on the different machines. Constraints (30) ensure that the processing order is the same on all machines. Finally, constraints (31) limit the vehicle departure date.

5.3 Optimal timing procedure

Whenever the sequence of the jobs is given, (i.e. the $y_{j,k}$ variables of the previous model are fixed), we still need to compute the optimal completion times of all jobs on each machine and the departure dates of all

vehicles. These values can be obtained by solving the following linear programming model where the positive continuous variables $C_{j,i}$ are equal to the completion time of the j^{th} job on machine M_i, $\forall j \in \{1,\ldots,n\}$, $\forall i \in \{1,\ldots,m\}$. Without loss of generality, we assume that the jobs in the sequence are numbered from 1 to n.

Minimize $IC + RC + PC$

s.t. $(2), (3), (4), (21), (22), (23), (24), (25)$

$$C_{1,1} = p_{1,1} \tag{32}$$

$$C_{j+1,i} \geq C_{j,i} + p_{j+1,i} \qquad \forall j \in \{1,\ldots,n-1\}, \forall i \in \{1,\ldots,m\} \tag{33}$$

$$C_{j,i+1} \geq C_{j,i} + p_{j,i+1} \qquad \forall j \in \{1,\ldots,n\}, \forall i \in \{1,\ldots,m-1\} \tag{34}$$

$$C_{j,m} \leq \sum_{s=1}^{S_v} (z_{v,s} t_{v,s} + u_{v,s}) \qquad \forall v \in \{1,\ldots,V\}, \forall j \in J^v \tag{35}$$

This model is a simplification of the previous model where Constraints (6) to (11) and Constraints (20) are replaced by Constraints (32) to (35). Constraints (32), (33) and (34) ensure that the sequencing and the machine capacity is respected. Constraints (35) impose that the departure date of vehicle v is greater than or equal to the completion times of the jobs in batch v.

In practice, this model can be solved quickly by a solver due to the low number of variables. For this reason, this model is used to evaluate the schedules found by the neighbourhood search method described in the next section.

5.4 Neighbourhood search (NS) method

In this section, we propose a neighbourhood search method NS that consists in applying consecutively two local search methods. The first one works at a "batch-level": a greedy heuristic determines the order of the jobs inside each batch. Afterwards, the second local search method acts at "job-level" and improves the solution by modifying the job sequences inside the different batches. The batch sequence is not changed anymore at this stage. Both local searches use the optimal timing procedure of Section 5.3 for evaluating solutions.

5.4.1 First local search

In the first local search method, a solution is represented by a sequence of batches and the jobs inside each batch are scheduled using the NEH algorithm of [Nawaz et al., 1983] that is summarised in Algorithm 2.

As an initial solution, the batches are sequenced by non-decreasing order of average due date ("ADD" order) over all jobs inside the batch. The neighbourhood operator consists in moving a batch v from a

Algorithm 2 Overview of the NEH algorithm

For each job j, calculate $P_j = \sum_{i=1}^{m} p_{j,i}$, the sum of processing times.
Create the list L_{NEH} where the jobs are sorted in non-increasing order of P_j
$S \longleftarrow \{\emptyset\}$
For $i \in \{1, \ldots, n\}$ **do :**
    For $j \in \{1, \ldots, |S|\}$ **do :**
        Evaluate the insertion cost of the i^{th} element of L_{NEH} at the j^{th} position in S
    Insert the i^{th} element of L_{NEH} at the best position in S
return S

position a to a position b in the sequence. To limit the neighbourhood, each batch is moved from its position a to a position $b \in \{a - \lambda_{batch}, \ldots, a + \lambda_{batch}\}$.

Whenever a better solution is found by moving a batch from a position a to a position b, it becomes the current solution and the exploration continues according to one of the two following strategies:

- "1" strategy: The exploration continues with current position a and test position b.
- "P" strategy: The exploration continues with current position $a = \min\{a, b\}$ and test position $b = \min\{a, b\} - \lambda_{batch}$ to intensify the exploration around each new current solution.

The exploration stops when $a = n$ and $b = n - 1$.

5.4.2 Second local search

In the second local search method, a solution is represented by a sequence of jobs. As an initial solution, we take the solution found by the first local search. The neighbourhood operator consists in moving a job from a position a to a position b in the sequence. To limit the neighbourhood, each job is moved from its position a to a test position $b \in \{a - \lambda_{job}, \ldots, a + \lambda_{job}\}$. Whenever a better solution is found by moving a job from a position a to a position b, it becomes the current solution and the exploration continues according to one of the exploration strategies previously described.

6 Computational experiments

Before presenting the computational experiments that evaluate the performance of the proposed methods, we describe in Section 6.1 how the random instances are created. The comparison parameters are explained and motivated in Section 6.2. In Section 6.3, the efficiency of the preprocessing methods proposed to determine the DC functions is evaluated and in Section 6.4, the results of the NS method to solve the integrated problem are presented and discussed.

All experiments are conducted on an Intel Core i7-7820HQ - CPU 2.90GHz and 16.0 GB of RAM. The algorithms run on a single thread on a windows system. IBM Ilog Cplex 12.7 is used as a solver and the algorithms are developed in Visual C++.

6.1 Instance generation

The difficulty of solving an instance is mainly related to the balance between the production part and the delivery part of the problem. If one part largely dominates the other in terms of costs, focusing the resolution of the whole problem to this part may be sufficient (even if already NP-hard). We consider these cases as "easy" in the sense that the resolution of one of the two parts is more important than the other. In order to deal with "difficult" instances, we define random instances for which the impact of both parts are in the same order of magnitude.

Three types of instances are created: *1-batch instances (1I)*, *small instances (SI)* and *large instances (LI)*. 1-batch instances are only used to test the methods that determine the DC functions in the preprocessing phase and therefore do not contain any scheduling data. Small and large instances are used to test the scheduling resolution methods. There are 16, 12 and 6 sets of instances of type 1I, SI and LI respectively. Each set contains 20 instances. The number of jobs and the batch composition are different for each set of instances. 1I is generated for a number of jobs $n \in \{5, ..., 20\}$, SI is generated for $n \in \{5, ..., 10\}$ and LI is generated for $n \in \{20, 50, 100\}$. We consider a 2-machine flow shop problem for set SI and a 5-machine flow shop problem for set LI. For each job j (and each machine M_i), the tardiness delivery cost π_j is generated following a normal distribution with a mean equal to 5 and a standard deviation equal to 2. The inventory holding costs of each job j are generated so that they increase by 1 or 2 units from one machine to another: h_j^{START} are uniformly random in the range $[1, 2]$, $h_{j,1}^{WIP}$ in range $h_j^{START} + [1, 2]$, $h_{j,i}^{WIP}$ in range $h_{j,i-1}^{WIP} + [1, 2]$ $\forall i \in \{1, \ldots, m-1\}$ and h_j^{FIN} in range $h_{j,m-1}^{WIP} + [1, 2]$. The processing time $p_{i,j}$ is randomly chosen between 1 and 10. The different delivery locations are randomly chosen on a square of size $SQ \times SQ$ with $SQ = 10$ and the distance $tt_{j,k}$ between delivery points j and k is determined using the classical Euclidean distance. For the sake of clarity, we consider the travel cost $tc_{j,k}$ between delivery points j and k equal to the travel time $tt_{j,k}$ between those points.

The due dates of jobs are randomly chosen between 0 and an upper bound on the last delivery time, equal to $T^{prod} + T^{tour}$. $T^{prod} = (n + m - 1) \times 10$, with 10 the maximum processing time of a task and $T^{tour} = (n/|K|) \times SQ\sqrt{2}$, with $n/|K|$ the average number of jobs per batch and with $SQ\sqrt{2}$ the maximum distance of an edge inside a SQ $\times$ SQ square. For all instances of type SI, the jobs are assigned randomly to one of the two batches. For all instances of type LI, the number of jobs per batch is randomly chosen in $\{3, \ldots, 7\}$, with exception of the last batch for which the number of jobs is calculated such that total number of jobs is correct. Afterwards, jobs are assigned to batches using one of the two following methods:

- *Unsorted*: all jobs are randomly assigned to a batch.

- *Sorted*: the jobs are sorted according to the earliest due date rule (EDD) and afterwards the batches are filled one by one corresponding to that sorted list.

Instances created with the *Sorted* (or *Unsorted*) method are called sorted instances (unsorted instances).

6.2 Comparison indicators

Before presenting the computational results of the preprocessing methods to determine the DC functions, we introduce different indicators denoted $MI(X)$, $AI(X)$ and $BI(X)$. The goal of the indicators is to fairly compare the results of exact and heuristic methods. Each method returns a set $\mathcal{R}_v^{app}$ of routes that describes the DC_v function associated to a batch v. We evaluate the different methods on the interval $[T^{\min}, T^{\max}]$ where $T^{\min}$ represents the latest departure date for which all the jobs are delivered on time regardless of the route and $T^{\max}$ corresponds to the earliest departure date for which all the jobs are delivered late regardless of the route. Notice that, unlike the interval $[0, T^{\text{prod}}]$, this interval is independent of the production data. To have a better estimation of the interval during the experiments, we set $T^{\min} = \min_{j\in\{1,\dots,n\}}\{d_j\} - n \times SQ\sqrt{2}$ and $T^{\max} = \max_{j\in\{1,\dots,n\}}\{d_j\}$.

The indicators aim to evaluate a given method X by comparing the DC^X function and the DC^* function given by the exact Branch-and-Bound algorithm. More precisely, the indicators have to reflect the gap between the two functions on interval $[T^{\min}, T^{\max}]$ that is divided into three sub-intervals: $[T^{\min}, A_X]$, $[A_X, B_X]$ and $[B_X, T^{\max}]$ with A_X the latest date for which the gap is constant on interval $[T^{\min}, A_X]$ and B_X the first date for which the gap is constant on interval $[B_X, T^{\max}]$. Let also define:

- a_X: the date of the first change of slope of the DC^X function,

- a^*: the date of the first change of slope of the DC^* function,

- b_X: the date of the last change of slope of the DC^X function,

- b^*: the date of the last change of slope of the DC^* function.

Therefore we have $A_X = \min\{a_X, a^*\}$ and $B_X = \max\{b_X, b^*\}$. An illustration of a DC^* function and a DC^X function on the interval $[T^{\min}, T^{\max}]$ is presented in Figure 9.

The main indicator $MI(X)$ measures the difference of the area below the DC^X function obtained by method X and the area below the optimal DC^* function obtained by the Branch-and-Bound algorithm, expressed as a percentage. $MI(X)$ is only defined on interval $[A_X, B_X]$ and can be seen as a relative deviation between the two curves: A ratio of 0% indicates that the curves are similar and the higher the

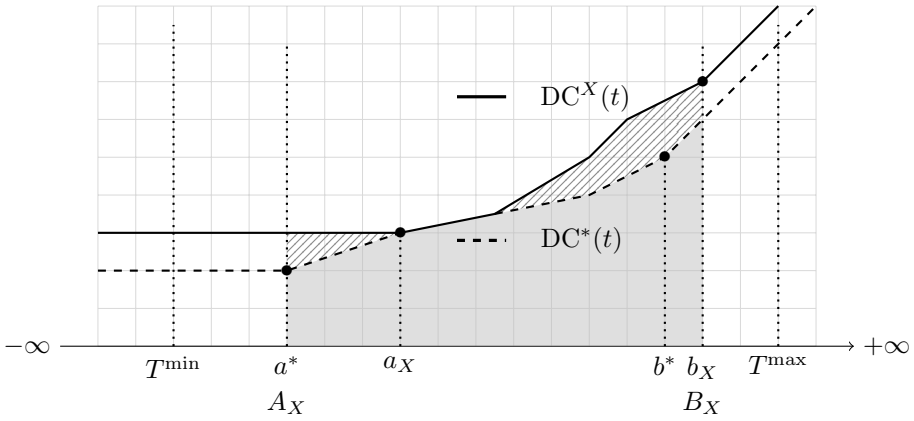

Figure 9: Comparison indicators

ratio the more distant the curves.

$$MI(X) = \frac{\int_{A_X}^{B_X} [\mathrm{DC}^X(t) - \mathrm{DC}^*(t)]\mathrm{d}t}{\int_{A_X}^{B_X} [\mathrm{DC}^*(t)]\mathrm{d}t}$$

The other two indicators, $AI(X)$ and $BI(X)$ calculate the ratio between the DC^X function and the optimal DC^* function at departure date A_X and B_X respectively. Note that the gaps are constant in the intervals $[-\infty, A_X]$ and $[B_X, \infty]$.

$$AI(X) = \frac{\mathrm{DC}^X(A_X)}{\mathrm{DC}^*(A_X)} \quad \text{and} \quad BI(X) = \frac{\mathrm{DC}^X(B_X)}{\mathrm{DC}^*(B_X)}$$

6.3 Efficiency of the preprocessing methods to determine the DC functions

First we compare the results obtained by the MILP model and the Branch-and-Bound algorithm on the set of instances 1I. The DC functions are computed for each departure date belonging to the time interval $[T^{\min}, T^{\max}]$. Both algorithms are interrupted after 3600 seconds. The 8-start strategy is used in the heuristic algorithm that gives the initial upper bound for the Branch-and-Bound algorithm.

Figure 10 shows the average computation times of the exact methods according to the number of jobs per batch. The Branch-and-Bound algorithm finds the optimal solution for all instances up to 17 jobs per batch within the time limit and the instances with up to 13 jobs per batch are solved within one minute. The MILP resolution with CPLEX can only solve instances with up to 9 jobs per batch within the time limit and only 85% of the instances with 10 jobs per batch are optimally solved. These results clearly show that the Branch-and-Bound algorithm outperforms the MIP solver of CPLEX.

Now, we compare the three versions (1-start, 2-start and 8-start strategy) of the heuristic presented in Section 4.2.2. Figure 11 presents the average computation times for the three versions according to the

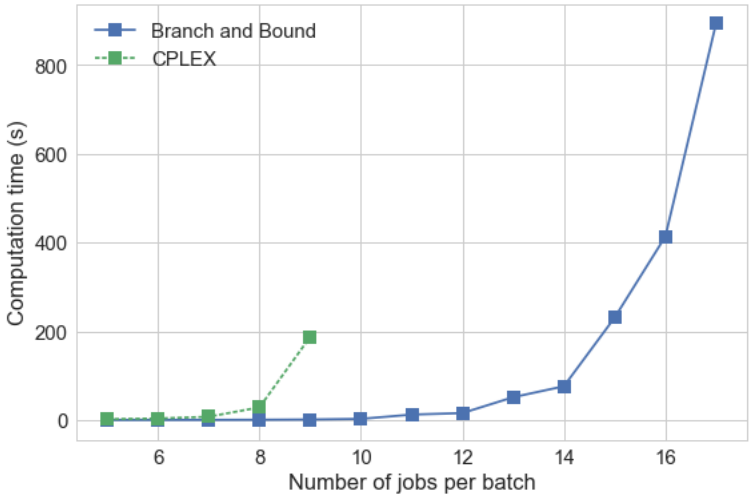

Figure 10: Average computation times of exact methods

number of jobs per batch. Figure 12, 13 and 14 show the average $MI(X)$, $AI(X)$ and $BI(X)$ indicators of each version.

In all cases, the quality of the 8-starts method is better than the 2-starts method that is better than the 1-start method. The multi-start strategy allows to obtain better approximated DC functions to the detriment of the computation time, which is directly proportional to the number of restarts. The heuristics are clearly faster than the Branch-and-Bound algorithm for large batch sizes and allow to obtain good approximations of the DC functions. Indeed, the average value of the $AI(X)$ indicator of the 8-starts method remains below 3% whereas the average values of $MI(X)$ and $BI(X)$ are less than 0.4%. We also observe that the values of $AI(X)$ are greater than the values of $BI(X)$ whatever the heuristic and the number of jobs per batch. The main reason is that the values of $\text{DC}^*(A_X)$ are smaller than the values of $\text{DC}^*(B_X)$. Indeed, the $\text{DC}^*(A_X)$ is only due to the routing costs to deliver all jobs on time, whereas the $\text{DC}^*(B_X)$ depends on both (higher) routing costs and penalty costs. Therefore, the ratio between $\text{DC}^*(A_X)$ and $\text{DC}^*(B_X)$ is 20 for the small instances and more than 100 for large instances. So, a small deviation from the optimal solution has a higher impact on $AI(X)$ than on $BI(X)$.

To conclude, the Branch-and-Bound algorithm is efficient to solve instances with up to 12 jobs per batch. For instances with a larger number of jobs, the 8-starts heuristic is the most effective, but with a higher computation time.

6.4 Efficiency of the neighbourhood search method

This section presents the parametrisation and the efficiency of the NS method to solve the integrated scheduling and routing problem. The method has three parameters: λ_{batch}, λ_{job} (the neighbourhood size limit of

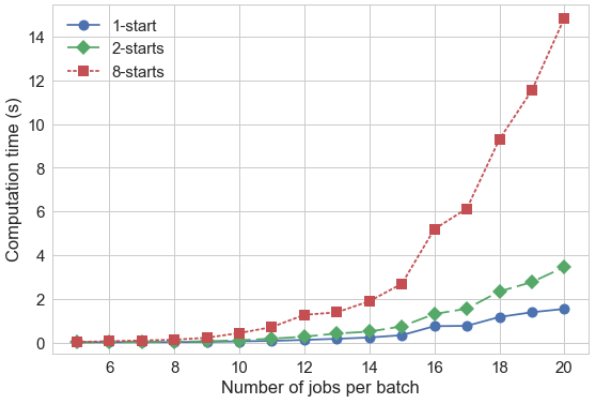

Figure 11: Computation times

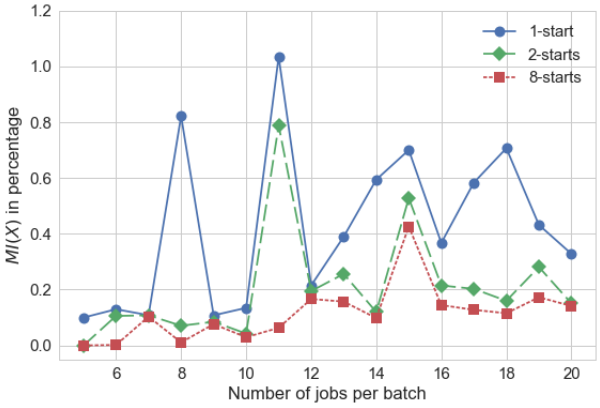

Figure 12: $MI(X)$ indicator results

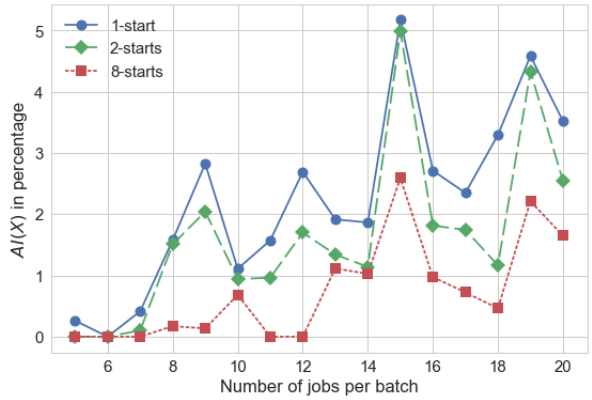

Figure 13: $AI(X)$ indicator results

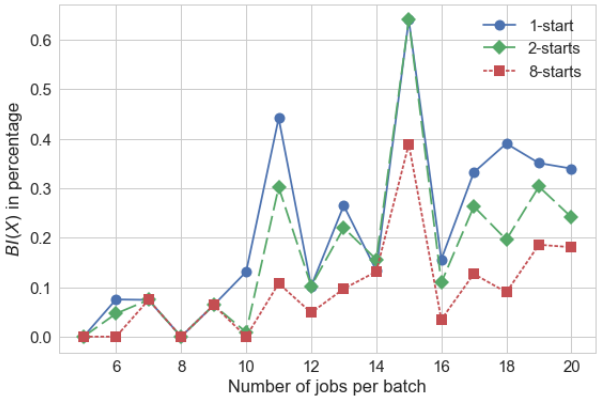

Figure 14: $BI(X)$ indicator results

the first and second local search presented in Section 5.4.1 and 5.4.2 respectively) and the exploration strategy noted "1" or "P". The first results, given in Section 6.4.1, aim to find the optimal parameters for the instances of type SI and to compare the solution found by the NS method to the optimal solution. The next results, presented in Section 6.4.2, aim to find the optimal parameters for the instances of type LI. Because there are up to 8 jobs per batch, we use the Branch-and-Bound method presented in Section 4.2.3 to compute the DC functions needed for the optimal timing procedure.

6.4.1 Small instances and solution quality

For the small instances, the parameters λ_{batch} and λ_{job} of NS do not have a great impact. In order to have a method that explores the whole neighbourhood, we set $\lambda_{batch} = 2$ and $\lambda_{job} = 10$. Both strategies of the local search ("1" and "P") are tested. The left part of Table 1 summarises the results. The first two columns indicate the types of instances, each line corresponds to a set of 20 instances. The "Gap NS (%)" column gives for each strategy the average Gap between the solutions found by the NS method and the

optimal solutions: $Gap = \frac{Z(NS)-Z^*}{Z^*}$. The "#OPT NS" column gives for each search strategy the number of instances where the NS method returns an optimal solution.

Whatever the strategy, the NS method finds optimal solutions for a large number of the smaller instances and near-optimal solutions for the larger instances of SI (average gap around 1%) in short computation times. The "P" strategy is the most effective but requires a higher computation time in comparison to strategy "1".

The efficiency of the NS method on small instances is evaluated by comparing the solutions found by the NS method to the solutions found by the GR algorithm (Section 3.3) on the one hand, and to the optimal solutions returned by the models DC_{MILP} (Section 5.1) and DC_{CP} (Section 5.2) on the other hand. The right part of Table 1 summarises the results. The "Gap GR (%)" column gives the average Gap between solutions found by the GR algorithm and the optimal solutions: $Gap = \frac{Z(GR)-Z^*}{Z^*}$. The "CPU MIP" and "CPU CP" columns indicate the average running time of the DC_{MILP} and the DC_{CP} method, respectively.

The CP model found an optimal solution within 1 hour. However, the MILP model does not find optimal solutions on instances with more than 1 jobs. The greedy algorithm GR finds feasible solutions for all instances almost instantaneously, but the average Gap of each set is more than 13.2% and up to 20.0%.

6.4.2 Large instances

This section presents the results of the GR and the NS method on the large-size instances. The exploration strategies of the NS method are tested for $\lambda_{batch} \in \{3, \ldots, 5\}$ and $\lambda_{job} \in \{5, \ldots, 10\}$. All possible combinations are tested on the 6 types of 20 instances. However, for 20-job instances, preliminary experiments have shown that the λ_{batch} parameter has a minor impact because of the low number of batches. Therefore, only $\lambda_{batch} = 3$ is used for instances with 20 jobs.

Figures 15 to 20 present the results of each combination of parameters of the NS method, with one figure per type of instance. Each figure represents a Cartesian coordinate system where each point corresponds to a combination of parameters. The coordinates of a point are given by the average computation time and the average Gap between the solution found the particular combination of parameters and the best solution found among the 12 combinations.

The computation times clearly increase with the number of jobs. The computation times are between 2 and 9 seconds for instances with 20 jobs whereas they are between 50 and 500 seconds for instances with 100 jobs.

The Gaps also increase with the number of jobs. For example, the GAP for $\lambda_{batch} = 3, \lambda_{job} = 5$ and "1" search strategy is 1.5% for instances with 20 jobs and up to 3% (8%) for sorted (unsorted) instances with 100 jobs. The Gap and the computation time widely depend on the local search strategy used. The "1" strategy allows to obtain shorter computation times but gives most of the time the worst solution while the "P" strategy always gives the best results but it requires the highest computation time. For sorted and

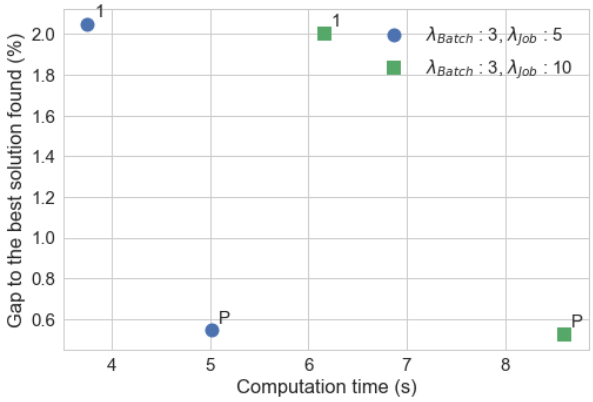

Figure 15: 20 jobs and unsorted instances

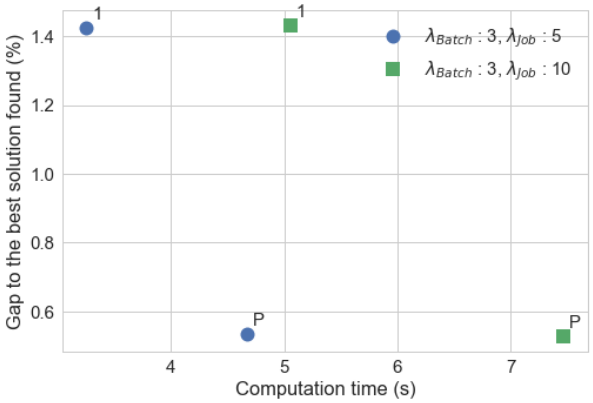

Figure 16: 20 jobs and sorted instances

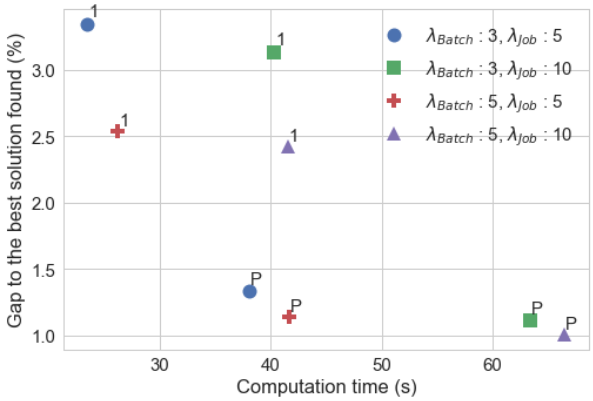

Figure 17: 50 jobs and unsorted instances

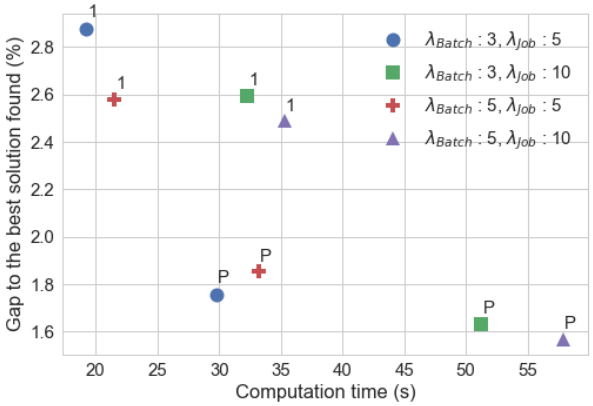

Figure 18: 50 jobs and sorted instances

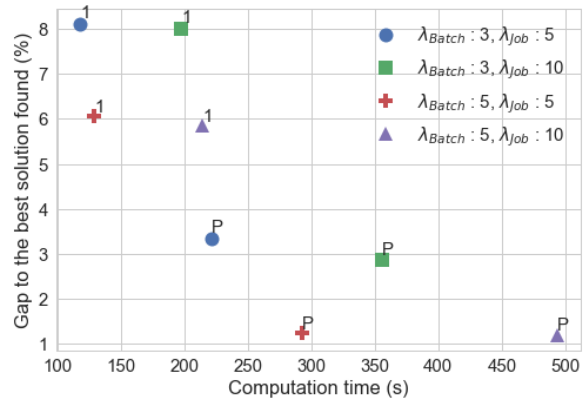

Figure 19: 100 jobs and unsorted instances

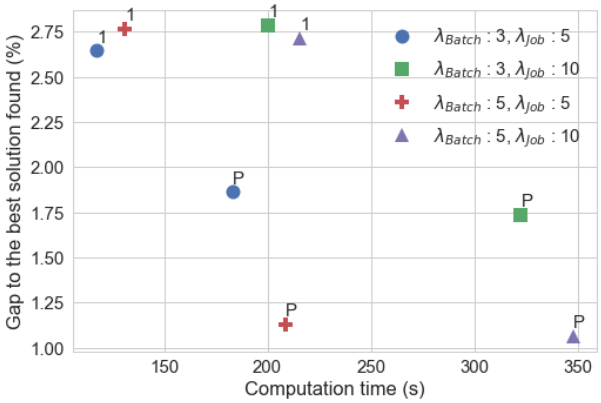

Figure 20: 100 jobs and sorted instances

Instances		Gap NS (%)		#OPT NS		CPU time (sec)		Gap GR	CPU MIP	CPU CP
n	type	1	P	1	P	1	P	(%)	(sec)	(sec)
5	Unsorted	**0.0**	**0.0**	**20**	**20**	**0.4**	0.4	13.4	0.2	0.2
	Sorted	**0.0**	**0.0**	**18**	**18**	**0.4**	0.4	13.6	0.1	0.2
6	Unsorted	0.2	**0.0**	15	**18**	**0.5**	0.7	13.2	0.3	0.4
	Sorted	0.4	**0.0**	15	**19**	**0.5**	0.7	14.5	0.1	0.3
7	Unsorted	0.4	**0.3**	13	**17**	**0.7**	1.0	16.9	1.3	0.9
	Sorted	0.2	**0.0**	17	**19**	**0.8**	0.9	12.0	0.6	0.9
8	Unsorted	0.5	**0.3**	11	**17**	**1.1**	1.3	19.0	3.3	0.8
	Sorted	0.3	**0.1**	10	**17**	**1.0**	1.3	15.4	4.0	0.8
9	Unsorted	0.8	**0.3**	8	**17**	**1.3**	1.8	14.9	18.5	1.2
	Sorted	0.6	**0.0**	7	**18**	**1.2**	1.9	14.8	33.0	1.2
10	Unsorted	0.8	**0.3**	4	**12**	**1.6**	2.6	17.1	241.4	2.1
	Sorted	1.1	**0.1**	5	**16**	**1.6**	2.7	14.2	430.6	1.9
11	Unsorted	0.6	**0.0**	6	**15**	**2.1**	3.5	19.6	-	4.7
	Sorted	1.9	**0.9**	2	**14**	**2.0**	4.0	16.6	-	5.2
12	Unsorted	0.8	**0.2**	2	**13**	**2.4**	5.1	18.5	-	8.4
	Sorted	1.4	**0.1**	5	**16**	**2.4**	4.8	16.2	-	5.7
13	Unsorted	1.3	**0.3**	0	**12**	**2.9**	6.4	16.2	-	14.5
	Sorted	1.0	**0.2**	2	**14**	**2.8**	5.7	14.6	-	12.1
14	Unsorted	0.9	**0.2**	2	**11**	**3.3**	6.7	14.1	-	31.2
	Sorted	1.2	**0.3**	0	**12**	**3.2**	7.2	13.8	-	34.8
15	Unsorted	1.1	**0.2**	1	**12**	**4.2**	9.8	20.0	-	112.2
	Sorted	1.5	**0.2**	0	**8**	**4.2**	10.5	18.6	-	165.9
16	Unsorted	1.2	**0.2**	0	**9**	**4.8**	12.0	13.7	-	348.4
	Sorted	1.0	**0.3**	0	**6**	**4.7**	11.9	13.2	-	274.7

Table 1: Results on small instances

unsorted instances of 20 jobs, the local search strategy has a bigger impact on the quality of the solution than the values of λ_{batch} and λ_{job}. The main reason is the structure of these instances (due date uniform in each batch and few jobs) which is conducive to attractive local optima and implies that a specific local search always leads to the same results and the other parameters only influence the computation times. For all instance sizes and all strategies, the computation time of the NS method on unsorted instances is always higher than the time required to solve sorted instances. The Gaps for unsorted instances are also higher on average.

For each local search strategy, we can observe that an increase of λ_{batch} from 3 to 5 allows to obtain a better gain in terms of ratio between the Gap and the computation time compared to an increase of λ_{job} from 5 to 10. However, as we expected, the combination of $\lambda_{batch} = 5$ and $\lambda_{job} = 10$ obtains the lowest Gaps.

Finally we can conclude that regarding the quality of the solution, the best combination of parameters is the P strategy, $\lambda_{batch} = 5$ and $\lambda_{job} = 10$, but it requires a very large computation time. We can also observe that the best compromise between computation time and solution quality among the combinations on all types of instances is the P strategy, with $\lambda_{batch} = 5$ and $\lambda_{job} = 5$.

Table 2 shows the Gap between the solution found by the GR method and the best solution found by the NS method with the parameters of the previous best compromise. We can observe that the NS method outperforms the GR method with a gap of at least 23% and up to 30%.

Number of jobs	20		50		100	
type	Unsorted	Sorted	Unsorted	Sorted	Unsorted	Sorted
GR vs NS with (*;$\lambda_{batch} = 5$;$\lambda_{job} = 10$)	-18.7%	-15.2%	-17.5%	-16.2%	-17.0%	-14.8%
GR vs NS with (P;$\lambda_{batch} = 5$;$\lambda_{job} = 5$)	-30.0%	-23.8%	-26.4%	-25.8%	-29.8%	-26.8%

Table 2: Gap between GR and NS methods on LI instances

7 Conclusion

In this paper we present an integrated routing and scheduling problem in a specific supply chain context. The production part is modelled by a permutation flow shop with inventory costs. We assume that the number of delivery batches and their compositions are known in advance and that each batch is served by a dedicated vehicle. The delivery part considers routing costs and tardiness penalty costs. The problem is to find a schedule for the different jobs on the different machines (starting times), and a delivery route for each vehicle (departure date and sequence) such that the total cost (inventory, routing and penalty costs) is minimized.

We propose to solve the problem in two steps. The first step is dedicated to the resolution of the routing part. For each batch and each departure date, the delivery route that minimizes the routing and penalty cost is determined. The results are aggregated into a set of piecewise linear cost functions, one for each batch. To generate these functions, we developed both a Branch-and-Bound and a heuristic method. The second step is dedicated to the resolution of the scheduling part. Two exact methods and one heuristic method are proposed. The first exact method is based on a MILP model and the second one is based on a CP model. The heuristic method is based on a neighbourhood search method and uses an optimal timing procedure to find the optimal starting times for the jobs and the optimal departure dates for the vehicles, given the job sequence and the piecewise linear cost functions. The computational experiments show that the CP model provides significantly better results than the MILP model. Once the size of the instances reaches 1 jobs, the exact methods need too much computation time and the heuristic method is preferred, as it outperforms the

greedy algorithm to find good solution in short computation times.

Several research perspectives can be considered. A first interesting extension is to explore the resolution of a more general problem where the number of delivery batches and their composition is unknown. The case where the batch composition is known in advance but the number of vehicles is limited can be another interesting generalisation. Finally, this study deals with an integrated problem where a producer and a 3PL provider take decisions together. In the future, some other scenarios could be modelled and tested, for example, the scenario where the producer or the 3PL provider takes upstream decisions alone.

References

[Absi et al., 2015] Absi, N., Archetti, C., Dauzère-Pérès, S., and Feillet, D. (2015). A Two-Phase Iterative Heuristic Approach for the Production Routing Problem. *Transportation Science*, 49(4):784–795.

[Absi et al., 2018] Absi, N., Archetti, C., Dauzère-Pérès, S., Feillet, D., and Speranza, M. G. (2018). Comparing sequential and integrated approaches for the production routing problem. *European Journal of Operational Research*, 269(2):633 – 646.

[Amorim et al., 2013] Amorim, P., Belo-Filho, M. A., Toledo, F. M., Almeder, C., and Almada-Lobo, B. (2013). Lot sizing versus batching in the production and distribution planning of perishable goods. *International Journal of Production Economics*, 146(1):208–218.

[Bülbül et al., 2004] Bülbül, K., Kaminsky, P., and Yano, C. (2004). Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. *Naval Research Logistics (NRL)*, 51(3):407–445.

[Chang et al., 2014] Chang, Y.-C., Li, V. C., and Chiang, C.-J. (2014). An ant colony optimization heuristic for an integrated production and distribution scheduling problem. *Engineering Optimization*, 46(4):503–520.

[Chen, 2010] Chen, Z.-l. (2010). Integrated Production and Outbound Distribution Scheduling: Review and Extensions. *Operations Research*, 58(1):130–148.

[Chevroton et al., 2018] Chevroton, H., Billaut, J., and Rohmer, S. (2018). A framework for production and outbound distribution: manufacturer dominates. *International Journal of Computational Engineering Science (in revision)*.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[Dayarian and Desaulniers, 2019] Dayarian, I. and Desaulniers, G. (2019). A branch-price-and-cut algorithm for a production-routing problem with short-life-span products. *Transportation Science*, 53(3):829 – 849.

[Garey et al., 1976] Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129.

[Karp, 1975] Karp, R. M. (1975). On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68.

[Kergosien et al., 2017] Kergosien, Y., Gendreau, M., and Billaut, J.-C. (2017). Benders decomposition based heuristic for a production and outbound distribution scheduling problem with strict delivery constraints. *European Journal of Operational Research*, 262(1):287–298.

[Koç et al., 2017] Koç, U., Toptal, A., and Sabuncuoglu, I. (2017). Coordination of inbound and outbound transportation schedules with the production schedule. *Computers & Industrial Engineering*, 103:178–192.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

[Laborie et al., 2018] Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

[Moons et al., 2017] Moons, S., Ramaekers, K., Caris, A., and Arda, Y. (2017). Integrating production scheduling and vehicle routing decisions at the operational decision level: A review and discussion. *Computers & Industrial Engineering*, 104:224–245.

[Munkres, 1957] Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

[Nawaz et al., 1983] Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

[Neves-Moreira et al., 2019] Neves-Moreira, F., Almada-Lobo, B., Cordeau, J.-F., Guimarães, L., and Jans, R. (2019). Solving a large multi-product production-routing problem with delivery time windows. *Omega*, 86:154–172.

[Park and Hong, 2009] Park, Y. B. and Hong, S. C. (2009). Integrated production and distribution planning for single-period inventory products. *International Journal of Computer Integrated Manufacturing*, 22(5):443–457.

[Rohmer and Billaut, 2015] Rohmer, S. and Billaut, J.-C. (2015). Production and outbound distribution scheduling: a two-agent approach. In *2015 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 135–144. IEEE.

[Ruiz and Stützle, 2008] Ruiz, R. and Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143 – 1159.

[Scholz-Reiter et al., 2010] Scholz-Reiter, B., Frazzon, E. M., and Makuschewitz, T. (2010). Integrating manufacturing and logistic systems along global supply chains. *CIRP Journal of Manufacturing Science and Technology*, 2(3):216–223.

[Wang et al., 2019a] Wang, D., Zhu, J., Wei, X., Cheng, T., Yin, Y., and Wang, Y. (2019a). Integrated production and multiple trips vehicle routing with time windows and uncertain travel times. *Computers & Operations Research*, 103:1–12.

[Wang et al., 2019b] Wang, S., Wu, R., Chu, F., and Yu, J. (2019b). Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution. *Soft Computing*, 24.

[Yağmur and Kesen, 2020] Yağmur, E. and Kesen, S. E. (2020). A memetic algorithm for joint production and distribution scheduling with due dates. *Computers & Industrial Engineering*, 142.

8 Appendix

Proof of Theorem 2

We remember the notations:

- A_σ: the set of index of remaining jobs to deliver after the sequence σ,
- j_σ: the last job of the sequence σ,
- π_j and d_j are respectively the unitary penalty cost and the due date of job j,
- $D_k^{min}(t)$: the k^{th} earliest possible delivery date after delivering all jobs of sequence σ and from a departure date t.

We denote by $\tau^{OPT}(t)$ the delivery sequence for all jobs $j \in A_\sigma$ delivered after j_σ, from a departure date t, that minimizes the total penalty cost (and $\tau_k^{OPT}(t)$ represents the job index in A_σ of the k^{th} element of this sequence). Let $z_{A_\sigma}^{OPT}(t)$ be the optimal penalty cost that can be obtained from the jobs of A_σ, for departure date t. We have:

$$z_{A_\sigma}^{OPT}(t) = \sum_{k=1}^{|A_\sigma|} \max\left(0, \pi_j(D_k^{OPT}(t) - d_j)\right) \text{ with } j = \tau_k^{OPT}(t)$$

and with $D_k^{OPT}(t)$ the delivery date of the k^{th} job in sequence $\tau^{OPT}(t)$.

$lb_\sigma^P(t)$ is equal to the optimal solution value of a min-cost assignment problem AP with a $|A_\sigma| \times |A_\sigma|$ cost matrix $U(t) = (u_{j,k}(t))_{j,k}$ where $u_{j,k}(t) = \max(0, \pi_j(D_k^{min}(t) - d_j))$, $\forall j \in A_\sigma, \forall k \in \{1, \ldots, |A_\sigma|\}$.

We want to prove that $lb_\sigma^P(t)$ is a lower bound of $z_{A_\sigma}^{OPT}(t)$:

Theorem 2.

$$lb_\sigma^P(t) \leq z_{A_\sigma}^{OPT}(t), \forall \sigma, \forall t \in \{0, \ldots, T\}$$

Proof. Let us define a second min-cost assignment problem $AP^{OPT}(t)$ with a $|A_\sigma| \times |A_\sigma|$ cost matrix $U^{OPT}(t)$ where $u_{j,k}^{OPT}(t) = \max(0, \pi_j(D_k^{OPT}(t) - d_j))$, $\forall j \in A_\sigma, \forall k \in \{1, \ldots, |A_\sigma|\}$.

A solution X of $AP^{OPT}(t)$ represents an assignment of the remaining jobs to the delivery dates $D_k^{OPT}(t)$. We denote by $f_t^{OPT}(X)$ the objective function value of a solution X to the assignment problem $AP^{OPT}(t)$.

The solution X^{OPT} given by the assignment of the optimal solution τ^{OPT} is a feasible solution of the problem $AP^{OPT}(t)$ and:

$$z_{A_\sigma}^{OPT}(t) = f_t^{OPT}(X^{OPT}) = \sum_{(j,k) \in X^{OPT}} u_{j,k}^{OPT}(t) \tag{36}$$

Because $D_k^{min}(t) \leq D_k^{OPT}(t)$ we have $u_{j,k}(t) \leq u_{j,k}^{OPT}(t)$, $\forall k \in \{1, \ldots, |A_\sigma|\}$ and $\forall j \in \tau^{OPT}(t)$. Since any feasible solution X of AP is a feasible solution of AP^{OPT}, thus :

$$\sum_{(i,k)\in X} u_{i,k}(t) \leq \sum_{(i,k)\in X} u_{i,k}^{OPT}(t)$$

$$f_t(X) \leq f_t^{OPT}(X) \qquad \text{where } f_t(X) \text{ is the objective function value of AP} \qquad (37)$$

$$lb_\sigma^P(t) \leq f_t(X) \qquad \text{since } lb_\sigma^P(t) \text{ is the optimal solution value of AP}$$

$$lb_\sigma^P(t) \leq f_t^{OPT}(X) \qquad \text{according to (37)}$$

$$lb_\sigma^P(t) \leq f_t^{OPT}(X^{OPT}) = z_{A_\sigma}^{OPT}(t) \qquad \text{according to (36)}$$

Theorem 2 is proved. □