Second paper

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

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

September 19, 2018

Abstract

Keys Words :

1 Introduction and literature review

1.1 Problem complexity *a mettre dans la bib???*

If we considers the routing problem of one batch and minimise the routing cost only, it's possible to reduce this problem to a single-machine total weighted tardiness problem with sequence-dependent setup times, which can be express as $1/s_{ij}/\sum w_i T_i$ using the Graham notation. And this problem is strongly NP-hard. **REF**

2 Problem description and mathematical model

2.1 Problem statement

We study a supply chain composed of a production and delivery part. The production part is modelled as a permutation flow-shop scheduling problem: all the tasks of each job must be executed on the machines in a specific order that is the same for all the jobs and all jobs must be scheduled on the same order on each machine. Once jobs are completed, they have to be delivered to customers. Each job is associated to a batch that represents a full load of vehicles. A homogeneous fleet of vehicles is available to deliver the jobs without limitation on the capacity or the number of vehicles. Each vehicle starts as soon as the last of job of its batch is produced. The route of vehicle must be determined and each and each vehicle is associated to only one batch.

The production problem is composed of m machines. We suppose the scheduling order of the jobs on the different machines is already fixed. We consider n jobs and position in the production schedule. We also suppose that the batch composition are already determined. The jobs are grouped in V batches. We note n_k, the number of jobs in batch k, J^k the set of jobs in batch k and J_l^k the index of the l^{th} job in batch k.

About the scheduling part, let p_{ij} the processing time of job j on machine M_i $\forall i \in \{1, \dots, m\}$. We considers two kinds of inventory cost. The work in process inventory (WIP) represents the storage of the job between its completion on a machine i and the start of the processing on the next machine $i+1$. $h_{i,j}^{WIP}$ denotes the unitary storage cost of the job j for the i^{th} machine to the $(i+1)^{th}$ $\forall i \in \{1, \dots, m-1\}$. The final inventory cost is from the completion of job on the last machine until the departure of its vehicle associated, let h_j^{FIN} the unitary cost associated for job j. At the end, let d_j the due date associated to job J_j and π_j the unitary cost associated to its delivery tardiness $\forall j \in \{1, \dots, n\}$.

About the delivery part, we note t_{j_1,j_2} and $c_{j_1,j_2} \forall j_1, j_2 \in \{0, \dots, n+1\}$ the travel time and cost between the destination delivery of job j_1 and the destination of job j_2. Note that the index 0 and $n+1$ represent the depot from which the vehicles start their routes.

2.2 Mixed Linear Programming model

We define the variable $C_{i,j}$ which represent the completion time of job j $\forall j \in \{1, \dots, n\}$ on machine M_i, $\forall i \in \{1, \dots, m\}$. x_{k,j_1,j_2} is a binary variable equals to 1 if the site associated to job j_2 is visited just after the site associated to job j_1. Let D_j be the delivery date and T_j the delivery tardiness associated to job j $\forall j \in \{1, \dots, n\}$. Variable D_j, T_j, $C_{i,j}$ are positive $\forall j \in \{1, \dots, n\}$, $\forall i \in \{1, \dots, m\}$.

We define the following expression: IC^{WIP} represent the total work in process inventory cost. IC^{FIN} represent the total final inventory cost. Let IC be the total inventory cost. RC represent the routing cost associated to delivery (the routing after the last client service is not take in account) and PC represent the total penalty cost associated to tardiness on delivery.

$$IC^{WIP} = \sum_{j=1}^{n} \sum_{i=1}^{m-1} (C_{i+1,j} - p_{i+1,j} - C_{i,j}) h_j^{WIP}$$

$$IC^{FIN} = \sum_{k=1}^{V} \sum_{j \in J^k} (C_{m,j_k} - p_{m,j^k_{n_k}} - C_{m,j}) h_j^{FIN}$$

$$IC = IC^{WIP} + IC^{FIN}$$

$$RC = \sum_{k=1}^{V} \sum_{j_1 \in \{0 \cup J^k\}} \sum_{j_2 \in J^k} c_{j_1,j_2} x_{k,j_1,j_2}$$

$$PC = \sum_{j=1}^{n} \pi_j T_j$$

$$\min \; = IC + RC + PC$$

$$s.t.\; C_{1,1} = p_{1,1} \qquad (1)$$

$$C_{i,j+1} \geq C_{j,i} + p_{i,j+1} \qquad \forall j \in \{1, \ldots, n-1\}, \forall i \in \{1, \ldots, m\} \qquad (2)$$

$$C_{i+1,j} \geq C_{i,j} + p_{i+1,j} \qquad \forall j \in \{1, \ldots, n\}, \forall i \in \{1, \ldots, m-1\} \qquad (3)$$

$$\sum_{j_2 \in J^k \setminus j_1} x_{k,j_1,j_2} = 1 \qquad \forall j_1 \in \{0 \cup J^k\}, \forall k \in \{1, \ldots, V\} \qquad (4)$$

$$\sum_{j_1 \in J^k \setminus j_2} x_{k,j_1,j_2} = 1 \qquad \forall j_2 \in \{J^k \cup n_k + 1\}, \forall k \in \{1, \ldots, V\} \qquad (5)$$

$$D_j \geq C_{J^k_{n_k},m} + t_{0,j} \qquad \forall j \in J^k, \forall k \in \{1, \ldots, V\} \qquad (6)$$

$$D_{j_2} \geq D_{j_1} + t_{j_1,j2} - HV(1 - x_{j_1,j_2}) \qquad \forall (j_1, j_2) \in \{J^k\}^2, \forall k \in \{1, \ldots, V\} \qquad (7)$$

$$T_j \geq D_j - d_j \qquad \forall j \in \{1, \ldots, n\} \qquad (8)$$

Constraints (1) fixed the first task of the first job as soon as possible. The resource constraints on machine are guaranty by constraint (2). Constraints (3) ensure that a job is performed by only one machine at a time. Constraints (4) guaranty that during a batch delivery, the initial deposit and each client in the the batch have only one successors among other client and the final deposit. Constraint (5) guaranty that during a batch delivery, the final depot and each client in the the route have only one predecessors among other client and the initial depot. The relation between the completion of the last job of a batch and the departure of this batch is imposed with constraint (6). The constraints (7) guaranty the delivery date of the different client. HV is a arbitrary high value. At the end, the constraints (8) ensure the correct tardiness. We can note that the sub tours elimination constraint is guaranty by combination of constraint (4), (5) and (7).

In case where two places j1 and j2 have the same locationv($t_{j_1,j_2} = 0$), sub tour must be broken by hand by adding the following constraint:

$$x_{j_2,j_1} = 0 \qquad \forall (j_1, j_2) \in \{J^k\}^2, j_1 < j_2, t_{j_1,j_2} = 0, \forall k \in \{1, \dots, V\} \tag{9}$$

3 Matheuristics - Protocol

As it say in the previous section (1.1), each batch delivery represented an NP-hard problem. Otherwise, the departure of such batch is not fixed. Indeed, two aspects oppose each other in the scheduling. The minimization of the penalty cost PC (due to late delivery on customers) lead to plan the vehicle departure date as soon as possible, whereas minimizing the inventory could lead to delay this departure date. Let's see the small following example with two batches B_1 and B_2. The first scheduling (I) minimizes the departure time of the vehicle whereas the second scheduling (II) minimizes the inventory cost IC, we could notice the difference of departure date for batch B_2 between the both example:

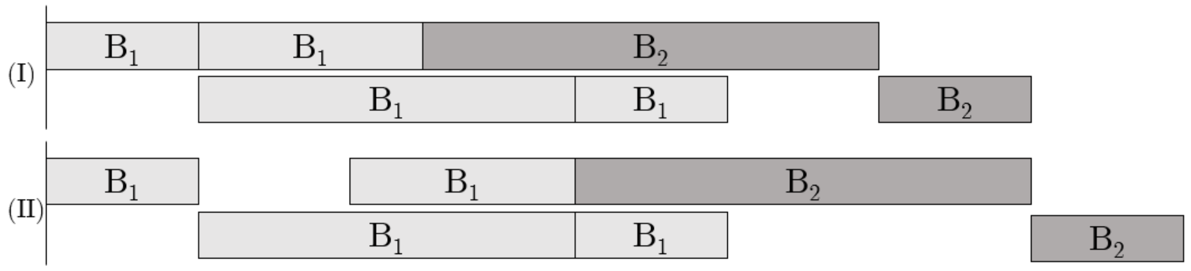

Table 1: Minimize IC versus departure date

But, solve the scheduling problem is quite simple if we manage only the completion time of jobs (means the departure date of the vehicles) and the inventory cost without consideration for the delivery aspect. The global idea of the presented matheuristic is to isolate the different delivery routing problems and solve it separately using heuristics for a certain time windows of departure dates and include this result in a global model which stay easy to solve.

We could describe our protocol in four steps :

1. For batch B_k $\forall k \in \{1, \dots, V\}$, let a_k the minimal departure date of batch B_k, $\forall k \in \{1, \dots, V\}$ by shifting jobs to the left. This is also the departure dates when we minimize the sum of the departure date.

2. For all batches B_k, get b_k the maximal departure date of batch B_k, $\forall k \in \{1, \dots, V\}$ by solving a problem minimizing the inventory (IC) cost only.

3. Compute an approximation of the penalty cost for all batches B_k in function of its departure date on the interval $[a_k, b_k]$, $\forall k \in \{1, \dots, V\}$.

4. Include this approximation in a global model and solve.

3.1 Point 1 et 2 : get a_k et b_k

Let introduce Σ_C, equals to the sum of the departure date of all batches.

$$\Sigma_C = \sum_{k=1}^{V} C_{m,j_k}$$

In order to get a_k $\forall k \in \{1,\dots,V\}$, the following model (A) is constructed including only the $C_{i,j}$ variables:

$$(A) \min \Sigma_C$$
$$s.t\ (1),(2),(3)$$

This model could be solve by scheduling all task as soon as possible. From the solution of (A), $a_k = C_{m,j_{Bk}}$, $\forall k \in \{1,\dots,V\}$.

In order to get b_k $\forall k \in \{1,\dots,V\}$, the same model is reused with a modification of the objective function, let it's model be (B):

$$(B) \min IC + \frac{\Sigma_C}{M}$$
$$s.t\ (1),(2),(3)$$

The objective function aim to minimize IC and Σ_C in lexicographic order. By experimentation, M could be a small multiple of the Σ_C found after the resolution of (A), for example $5\Sigma_C$. From the solution of (B), $b_k = C_{m,j_{Bk}}$, $\forall k \in \{1,\dots,V\}$.

3.2 Point 3 : Get an used an approximation

The objective of this section is to present a function for each vehicle where the image of the departure date of this vehicle by this function is associated to a routing and tardiness cost. The path used for the delivery at this departure date doesn't appeared on the function, but it exists and it's feasible. Formally, if RC^k_t and PC^k_t are respectively the routing and the tardiness cost of the batch B_k which leaves the deposit at date t, $\forall t \in \{a_k, b_k\}$, $\forall k \in \{1,\dots,V\}$, we want to build a function F_k such:

$$F_k(C_{m,j_k}) = RC^k_{C_{m,j_k}} + PC^k_{C_{m,j_k}}$$

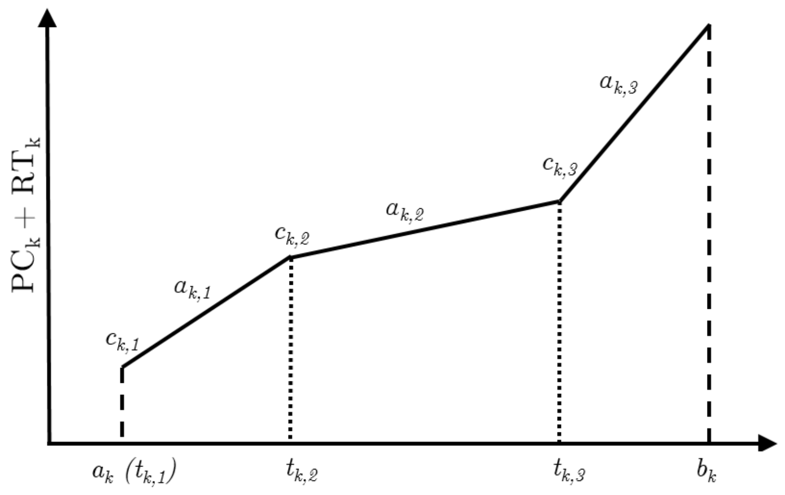

Figure 1: Profil of the approximation of the batch B_k

The profile of such function look like to the following one.

Each point of such function is associated to a specific path for delivered batch B_k with a specific departure date. This kind of function is linear piece by piece. So it is described segment by segment. Each segment is characterized by a triplet (t, c, α). t represents the departure date when the segment begins, c represents $F_k(t)$, the cost associated to the batch B_k for the departure time t. α represents the director coefficient of this segment (which correspond to the sum of unitary tardiness cost for jobs which are late for this path at the departure date t).

We discuss the method for found an approximation and evaluate its quality in the part (4).

3.3 Point 4 : Global model including approximations

The data included in the new model are a_k and b_k previously defined. λ_k represent the number of segments in F_k. $c_{k,s}$ and $t_{k,s}$ represented respectively the cost and the time already consumed when the s^{th} segment of function F_k is used. $\alpha_{k,s}$ represent the cost associated to each unit of time on the s^{th} segment of function F_k. Let see Fig.(1) for $t_{k,s}$, $c_{k,s}$ and $\alpha_{k,s}$. We consider t_{k,λ_k+1} is equal to b_k. Two new sets of variables are included, $x_{k,s}$ is equals to 1 if the departure date $C_{j_k,m}$ is includes in the s^{th} segment of function F_k (between $t_{k,s}$ and $t_{k,s+1}$) else 0, and $d_{k,s}$ represent the utilization time spend on this segment before the departure date of the batch B_k, $s \in \{1, \dots, \lambda_k\}$, $\forall k \in \{1, \dots, V\}$.

The objective function (10) is always composed of the inventory cost IC and the routing and tardiness cost are summarised, for each batch B_k, by the image of the departure date by the function F_k. Constraints

$$
\begin{aligned}
& min\ IC + \sum_{k=1}^{V}\sum_{s=1}^{\lambda_k}(x_{k,s}c_{k,s} + d_{k,s}\alpha_{k,s}) && && (10)\\
& s.t.\ (1), (2), (3) && && \\
& C_{j_k,m} \leq \sum_{s=1}^{\lambda_k}(x_{k,s}t_{k,s} + d_{k,s}) && \forall k \in \{1,\dots,V\} && (11)\\
& \sum_{s=1}^{\lambda_k} x_{k,s} = 1 && \forall k \in \{1,\dots,V\} && (12)\\
& d_{k,s} \leq x_{k,s}(t_{k,s+1} - t_{k,s} - 1) && \forall k \in \{1,\dots,V\},\ \forall s \in \{1,\dots,\lambda_k\} && (13)\\
& 0 \leq d_{k,\lambda_k} && \forall k \in \{1,\dots,V\} && (14)
\end{aligned}
$$

Table 2: New model including tardiness penalty cost approximation

(11) make the link between the completion time of the last job of a batch and departure time of this batch. Constraints (12) ensure that only one segment is partially used in each function F_k for each batch B_k, $\forall k \in \{1,\dots,V\}$. Constraints (13) guaranty each segment is used only if it is the one containing the departure date of the batch and the utilization of this segment cannot be overflowed.

We could add the followings cut in order to speed up the resolution of the model:

$$
\begin{aligned}
& a_k \leq C_{j_k,m} \leq b_k && \forall k \in \{1,\dots,V\} && (15)\\
& d_{k,\lambda_k} \leq t_{k,s+1} - t_{k,s} && s \in \{1,\dots,\lambda_k\}, \forall k \in \{1,\dots,V\} && (16)\\
& x_{k,s}t_{k,s} \leq C_{j_k,m} && s \in \{1,\dots,\lambda_k\}, \forall k \in \{1,\dots,V\} && (17)
\end{aligned}
$$

Cuts (15) reuse the results found during the previous model resolution and cuts (16) are upper bound. Cuts (17) add a lower bound for the departure date.

The resolution of this model lead to a solution of the integrated model. For each batch, its departure date is associated to a delivery path link the to the last segment used of the approximation function.

4 Built the F_k function

We develop three methods for build a routing cost function F_k. Two of them are exacted, the first one solve a mathematical model with a solver and the second one is an ad-hoc Branch and Bound. The last method develop an heuristic in order to propose a correct approximation for large number of job by batch.

4.1 F_k based on mathematical model

For one departure date t and one batch B_k, the mathematical model correspond in the minimisation of $PC_k + RC_k$ under the constraint (4), (5), (7) and (8). The constraint (6) is change in constraint (20) using the new departure date t.

$$\min = PC_k + RC_k \tag{18}$$

$$(4), (5), (7), (8), (9) \tag{19}$$

$$D_j \geq t + t_{k,0,j} \qquad \forall j \in \{1, \ldots, n_k\}, \forall k \in \{1, \ldots, V\} \tag{20}$$

In order to draw the function F_k on the complete windows $[a_k, b_k]$, the model is solved first with $t = a_k$ and then re optimise for each date until b_k.

4.2 F_k based on a B&B

We will see in the result section the limitation of the CPLEX method (related to the explosion of the computation time when number of job per batch increase). In order to solve exactly more important instances, a Branch and Bound has been developed.

B&B algorithm description???

In this algorithm, the branching is applied on the next customer to served in the tour.

Because our B&B works in parallel on all the departure date in the windows $[a_k, b_k]$, we could consider we are in a multi-objective context. *In this context we don't know special exploration strategies.* So we choose a deep-first exploration strategy.

The main contribution of this work is the implementation of a the lower bound for the two parts of the objective function, the routing and the penalty cost. The minimum spanning three is use as a lower bound for the routing cost. In order build a lower bound for the penalty cost, we need to determine two things. A list of estimated delivery date for the customers not serve yet and next a assignment of this customers to this delivery date for estimated the penalty cost in the best case. So the cost of the lower bound are composed of the cost of the partial solution, which is note modified by the customers served later, plus a estimation of routing and penalty cost generate by the customers resting to served.

4.2.1 Estimation of the remaining delivery dates

Suppose the $n-1$ remaining customers and the last customers served compose a complete graph G. We looking for $n-1$ ordered delivery dates such the k^{th} one is lower or equals to the minimum time distance path of k edges in G. The idea is find a date such we can say: "the k^{th} customer served can not be served before this date". For doing this, we consider T_G a minimum spanning tree of G, and the k^{th} delivery date correspond to the sum of the distance of the k shortest edges in T_G (plus delivery date of the last job fixed). Now, let's prove this bound is correct:

Formally, let a_i $\forall i \in \{1, \ldots, n\}$ an edge of T_G. Without loss of generality, lets a_i be the distance associated to this edge and $a_i \leq a_{i+1}, \forall i \in \{1, \ldots, n-1\}$. Let's be *sumK_edge* associated to a graph T and a number k the sum of the k smallest edges of T:

$$\begin{aligned} sumK\_edge : & T \times [1, \ldots, |T|] & \rightarrow & \mathbb{R}^+ \\ & T, k & \rightarrow & \sum_{i=1}^{k} a_i \end{aligned}$$

Let's h any path in G, we want to prove: $sumK\_edge(T_G, k) \leq sumK\_edge(h, k), \forall k \in \{1, \ldots, n\}$.

Property 1: For each cycle in a graph, if the cost of one of its edge is bigger than the cost of any other edges in this cycle, then this edges doesn't belong to the minimum spanning of the graph.

Let's consider any path h of size k in G, and H the subgraph including all the node use by h. Let's be E the set of all the edges included in H but not in T_G. For each edge e in E, because T_G is a minimum spanning tree of G, adding e in T_G make a cycle in G. According to property 1, because e is the only edge of this cycle that is not in T_G, it's the edge of the cycle with the bigger cost. So there exist at least one edge with a lower or equal cost include in T_G but not in H (else H would contain a cycle while he is built from a path). This edge replaces e in the calculus of $sumK\_edge(T_G, k)$ so $sumK\_edge(T_G, k) \leq sumK\_edge(h, k)$. So the property is proved for any path, including the optimal path of distance k.

4.3 Customer assignment to delivery date

Now we have a set of delivery date. The cost of a customer assignment to a delivery date correspond to the penalty cost due to him if he is served at this date. All the couple customer-delivery date form a assignment matrix. We use Kuhn-Munkres algorithm **ref** on the matrix for solve the minimal assignment cost problem for a specific departure date.

4.4 Lower bound on a departure time windows

We choose to lead the exploration in the Branch and Bound on all the possible departure date. It means, in each new node, we check the lower bound of all the remaining date and try to cut certain before continue the exploration. A node is "cut" if there is no remaining date after the lower bound application. We reuse computational effort deploy to found a lower bound for a certain departure date during the research of the bound for the next one. The lower bound for the routing cost are not modified by the increasing of the departure date. All the estimated delivery dates increase of the difference between the old and the next departure date (even if they keep the same gap between each other). Therefore, the assignment matrix of customers-delivery date must be update.

An assignment cost rest the same during the matrix update if the target job is not late for the new delivery date. Else, the cost increases with the extra penalty we must to pay to customers (equals to the gap between the new and the old departure date multiply by the unitary penalty cost of this customers).

However, if the customer is ever late for all the delivery date, its associated line in the assignment matrix rest the same. Indeed, assignment decision is based on the gap between the different options of assignment for a customer, if all options cost increase in the same way, the decision don't change. Accordingly, as soon as all the jobs are late for all the delivery date, the assignment rest fix and the lower could compute for all older departure date using this assignement. In other case, the Kuhn-Munkres algorithm is used for re optimised the solution.

Let see a small example, three customers rest to assign with the following characteristics: $d_1 = 5$, $\pi_1 = 1$, $d_2 = 9$, $\pi_2 = 2$, $d_3 = 11$ and $\pi_3 = 3$. The minimum spanning three are compose of three edge of length 1, 2 and 2 (Start represent the last costumer served). For this node, the departure date $t_1 = 8$ and $t_2 = 10$ are still available.

The Fig.(4.4) show a configuration of the minimum spanning tree for the remaining customers to served. Fig(4.4) represent the different penalty costs due to client in function of their assigned delivery date for the departure date $t_1 = 8$ and Fig.(4.4) show the optimal assignment found by the Kuhn-Munkres algorithm, its associate cost is 8.

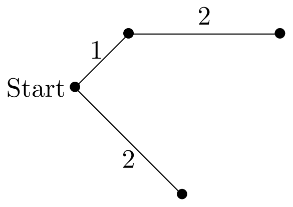

Figure 2: Minimum spanning tree

t_1	$D_1 = 9$	$D_2 = 11$	$D_3 = 13$
j_1	4	6	8
j_2	0	4	8
j_3	0	0	6

Figure 3: Assignment matrix for $t1$

t_1	$D_1 = 9$	$D_2 = 11$	$D_3 = 13$
j_1	0	2	$\boxed{0}$
j_2	$\boxed{0}$	4	4
j_3	0	$\boxed{0}$	2

Figure 4: Result of the Kuhn-Munkres algorithm for t_1

The Fig.(4.4) represented the assignment matrix for the departure date $t_2 = 10$. In Fig.(6), the underline numbers are ones modified from the matrix in Fig(4.4). Indeed, at $t_1 = 8$, the first customers will served late, no matter the delivery date, and so, its line rest unchanged. Idem for the second customers who is served, at most, just at time with the first delivery date. The third customer is in advance for the first date, so this penalty cost don't change. The last Fig.(4.4) presented the optimal assignment found, this cost is 18.

t_2	$D_1 = 11$	$D_2 = 13$	$D_3 = 15$
j_1	6	8	10
j_2	4	8	16
j_3	0	6	12

Figure 5: Assignment matrix for $t2$

t_2	$D_1 = 11$	$D_2 = 13$	$D_3 = 15$
j_1	0	2	0
j_2	0	4	4
j_3	0	$\underline{6}$	$\underline{8}$

Figure 6: Modification from Fig.(4.4)

t_2	$D_1 = 11$	$D_2 = 13$	$D_3 = 15$
j_1	2	0	$\boxed{0}$
j_2	0	$\boxed{0}$	2
j_3	$\boxed{0}$	2	6

Figure 7: Result of the Kuhn-Munkres algorithm for t_2

transition B and B heuristic

4.5 Heuristic

In order to explain how build a function F_k, let see Alg.(1) which build the function $f_{p,k}$. This function associates the departure date of batch B_k with the cost of the routing and tardiness for the path p which is feasible to delivered B_k, $\forall k \in \{1, \dots, V\}$. This algorithm introduce the notation $D(p, j)$, which represented the travelling time between the deposit and the customers associated to job J_j using path p.

The first segment (such $t = a_k$), c is defined by sum the routing cost of the path p, (which is constant) plus the penalty cost of customers already served late for a departure in a_k. α is the sum of unitary penalty cost of such customer for α. Then, the departure date is delayed (and so the delivery date of all customers) and a new segment is defined each time new customers become late. The cost c and the coefficient α are incremented. So we can note that, because α only increased, the derivation of the function only increase too.

It's obvious the optimal path change with the departure date in the window $[a_k, b_k] \forall k \in \{1, \dots, V\}$ especially if the windows is large. So, the function F_k is composed of several function $f_{k,p}$ segment issue of different path optimal for a subset of departure date.

For illustrate the Alg.(1) and the two functions minimum operators, let see the following example. A batch B is composed of three customers a, b and c. Each customer has its due date ($d_a = 16$, $d_b = 16$ and $d_c = 22$) and its unitary tardiness penalty cost ($\pi_a = 1, \pi_b = 2, \pi_c = 3$). The matrix of distance associated to this customers and the deposit is presented Fig.(4.5), we considers that the distance matrix and the cost matrix are equals. Let consider two paths, p_A which serve customers in the order b, c, a and p_B which used

Algorithm 1 (p, B_k, a_k, b_k) Get function $f_{p,k}$ from a path p for batch B_k

```
1:  Initialisation
2:  $List \leftarrow B_k$
3:  $f_{p,k} \leftarrow \{\}$
4:  # Initial cost
5:  $c \leftarrow RC(p)$
6:  for $j \in B_k$ do
7:      if $a_k + D(p,j) \geq d_j$ then $c \leftarrow c + \pi_j(a_k + D(p,j) - d_j)$
8:  end for
9:  $t \leftarrow a_k;\ t' \leftarrow t;\ \alpha \leftarrow 0$
10: Start
11: while $List \neq \emptyset$ AND $t < b_k$ do
12:     # Addition unitary tardiness cost of the late jobs
13:     $\alpha \leftarrow \alpha + \sum_{j \in List | t+D(p,j) \geq d_j} \pi_j$
14:     # Increments of the total cost
15:     $c \leftarrow c + \alpha(t - t')$
16:     # Remove such jobs
17:     $List \leftarrow List\ /\ \{j | t + D(p,j) \geq d_j\}$
18:     # Increments of the function
19:     $f_{p,k} \leftarrow f_{p,k} + \{t,\ c, \alpha\}$
20:     # Increments of $t$ with the smallest advance
21:     $t' \leftarrow t$
22:     $t \leftarrow t + min(d_j - D(p,j) - T_j | j \in List)$
23: end while
```

the ordehttps://v2.overleaf.com/project/5b962c4379823d6602532e43r a, c, b.

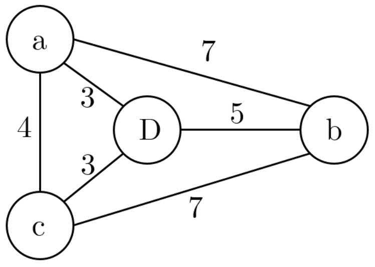

Figure 8: Distance and cost matrix

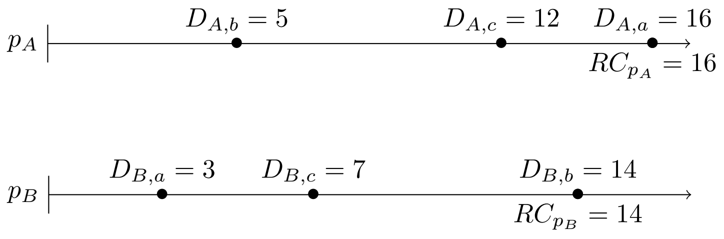

Figure 9: Travel time and routing cost

The departure windows for batch B is fixed to $[1, 13]$. For path p_A, it's initial departure time $t_{A,1} = 1$, customers a is late for 1 units of time and the travel cost is 16. Because $\pi_a = 1$, $c_{A,1} = 16 + 1 \times 1 = 17$ and $\alpha_{A,1} = \pi_a = 1$. If we delay the departure time, customers c starts to be served late for $t_{A,2} = 10$, because the penalty cost due to customers a increase during this 9 units of time $c_{A,2} = 17 + 9 \times 1 = 26$ and $\alpha_{A,2}$ increase of π_c ($\alpha_{A,2} = 4$). Next, customers b are late since $t_{A,3} = 11$, $c_{A,3} = 26 + 1 = 30$ and $\alpha_{A,2} = 4 + 2 = 6$).

For path p_B, there no initial tardiness, so $c_{B,1} = RC(p_B) = 14$ and $\alpha_{B,1} = 0$. We have customers b tardy for $t_2 = 2$, because $\alpha_{B,1} = 0$, $c_{B,2}$ rest unchanged and $\alpha_{B,2}$ gets π_b.

In conclusion, The path p_A must be chosen if the departure date of the batch B is in the intervals $[1, 6]$ and [12, 13] and the path p_B must be chosen if the departure date is in the interval $[6, 11]$.

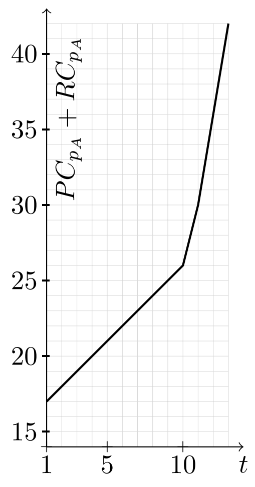

Figure 10: Path p_A

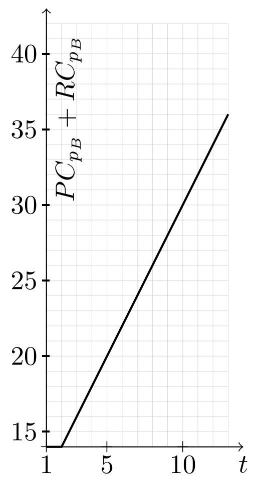

Figure 11: Path p_B

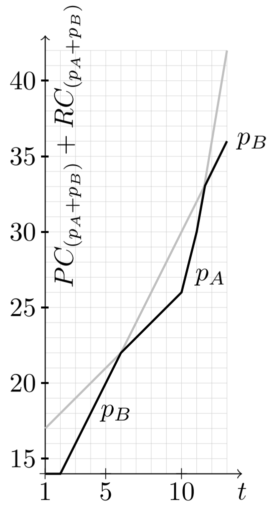

Figure 12: Merge(p_A+p_B)

4.6 Global algorithm to get an approximation from one initial solution

We need to found interesting paths p for all departure dates in the interval $[a_k, b_k]$ and merge their associated function $f_{p,k}$ together in order to obtain the global approximation F_k, $\forall k \in \{1, \dots, V\}$. A good path criteria for a specific departure date is this path is at least locally optimal for this date and for a certain neighbourhood. For guaranty this criteria, we design an algorithm based on local search. The function F_k is initialised with a specific $f_{p,k}$ where path p is determine by the user (it could be equal to the EDD sequence, near insertion sequence or random sequence). Then we apply a local search on path p in order to minimise the sum of the routing (RC_p) and tardiness (PC_p) cost for the specific departure date, (which is a_k in the initial situation). The local search is based on a best improvement. We merge each function associated to each path evaluated during the local search with the main function F_k. The local search stop as soon as a local optima is found (depending of the specific neighbourhood used). This local optima is associated to a function $f_{p^*,k}$ (with p^* the locally optimal path found). All segments of this function present in the main function F_k are locally optimal for all the departure date where the best result is given by this function.

The proof is, let d^* be the departure date associated to the local search those p^* is the result and considers now another departure date d' where the associated segment in F_k is a part of the function $f_{p^*,k}$. Suppose you want apply a local search in d' starting from the path p_*, you will explore the complete neighbourhood around p_* and merge all the associated function with F_k. But all this function have already be merged with F_k during the local search in d^* which produce p^*, and no one have improved $f_{p^*,k}$ in d'. So a second local is useless and so $f_{p^*,k}$ is locally optimal in d'.

The algorithm continue by improving each segment which are not proved as locally optimal until f_k are locally optimal for all departure date. This algorithm is described in Alg.(2)

Algorithm 2 Approximation algorithm for batch B_k (p_{init})

```
1: Initialisation
2: $F_k \leftarrow f_{p_{init},k}$
3: for $S'$ in $F_k$ do $S'.localy\_optimal \leftarrow False$
4: $S \leftarrow first\_segment\_not\_locally\_optimal(F_k)$
5: while $S \neq \emptyset$ do
6:     $p^* \leftarrow local\_search(S.departure\_date, S.path)$
7:     for $S'$ in $F_k$ do
8:         if $S'.path$ equals $p^*$ then $S'.localy\_optimal \leftarrow True$
9:     end for
10:    $S \leftarrow first\_segment\_not\_locally\_optimal(F_k)$
11: end while
```

For illustration, let see the following example in Fig. (13, 14, 15, 16, 17). The dark line are the product of the current local search (several function at each local search). The dashed line represent the segments of function F_k which are locally optimal and the gray one represent the function F_k at the previous step. At any moment, F_k is the lowest part of all the different functions represented. The abscissa of black dots represent the departure date on which the local search will be done during the next step and all the segments of the function found during this local search are automatically consider as locally optimal. In the Fig. (17), the local search does not achieve to improve F_k but it proves the current segment are locally optimal.

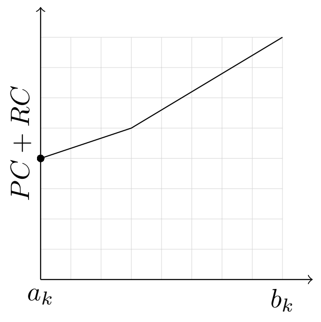

Figure 13: Start in a_k with the initial path

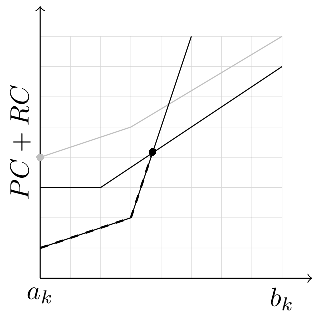

Figure 14: Result of local search in a_k, two functions are added

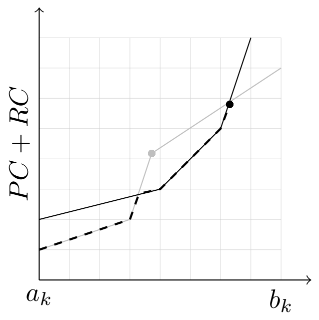

Figure 15: Result of the second local search

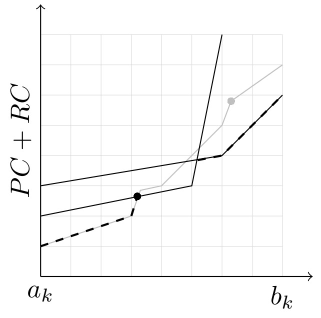

Figure 16: Third local search: some segments are reoptimised

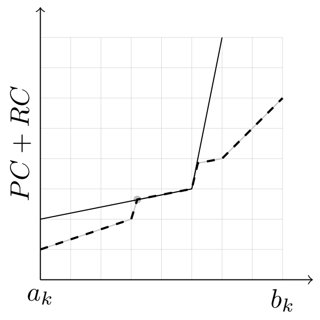

Figure 17: Last local search: we are completely locally optimal

In the rest of the experimentation, a reinsertion operator is used without restriction.

Reinsertion of 2^{nd} job at the 5^{th} job level: $1, \underline{2}, 3, 4, 5, 6, 7 \rightarrow 1, 3, 4, 5, \underline{2}, 6, 7$

Reinsertion of 5^{th} job at the 2^{nd} job level: $1, 2, 3, 4, \underline{5}, 6, 7 \rightarrow 1, \underline{5}, 2, 3, 4, 6, 7$

4.7 Multi start mechanism

In order to improve the quality of the heuristic, we could easily merge different functions F_k produce by Alg. (2) with different initial paths p_{init}. Rather than used random initial path, we propose to use the path produce with Earliest Due Date order (EDD), and with nearest insertion order (near) . We suppose they are good quality initial solution and use it speed up the heuristic. But on the other hand, the heuristic has more chance to found good solution if it explore a larger part of the solution space. Indeed, during a local search for a specific departure date, all the function tested are added to the main function F_k with a chance to improve this function on all the other departures dates.

In order to balance the research between efficiency and large exploration, we propose different variation of the same initial path (for example: 1, 2, 3, 4, 5, 6).

1. the initial path itself: 1, 2, 3, 4, 5, 6
2. the reverse path: 6, 5, 4, 3, 2, 1,
3. the begin-end exchange: 4, 5, 6, 1, 2, 3
4. the reverse begin-end exchange: 3, 2, 1, 6, 5, 4

We call $p_{init}X4$, the method merging the four approximation get from the different variants of a same path.

5 Computational study

This experimentation have been made on computer with Intel Core i7-7820HQ, CPU 2.90GHz and 16,0 Go RAM.

5.1 Instance description

a completer

uniforme ou rparti

- nombre de machines
- nombre de jobs
- processing time
- cout d'inventaire
- Due date

- Taille du carr

5.2 Experimentation on mathematical model

5.3 Experimentation on the branch and bound

Methode	CPU	%
PLS + CPLEX		
PLS + B and B		
PLS + Heuristic		

Comparaison of areas on $+\infty, -\infty$, for the different heuristic quality

5.4 Approximation quality of the routing and delivery function

As a reminder, an approximation function F_k is dedicated to a specific batch on a specific time windows. So We test here only one batch instance. The evaluation of the heuristic quality is based on the area under the curve of the function F_k.

For a small number of job, we develop a Branch and Bound dedicated for found the optimal function F_k^*. All our result are compared with this optimal function.

The number of machines, the processing time and inventory cost of the job are not taken in account. Concerning the other parameters, the number of jobs is in the set [10, 20], about the square where the deposit and the different customers are randomly placed, its size is in [10, 100]. We propose two kinds of penalty cost, uniform and equal to 1 or following a normal distribution with the distribution mean equals to 5 and with a standard deviation equals to 2 (with a penalty cost at least equal to 1). We describe how the due dates are fixed later. Each line of the Tab.(5.4) present the average result for sets of at least 10 instances. The left column describe the characteristic of each instance set : $(Number\ of\ job)$j_s$(Size\ of\ square)$_c$(Type\ of\ penalty\ cost)$

The approximation quality are tested here under a certain condition, the goal is used a departure time windows large enough to contain two critical date. The last date of the first segment where all are in advance and where the routing cost is minimum, and the first date of the last segment where all the job are late.

In order to define such time window, we consider the complete delivery time ($CTime$) equals to the number of jobs multiply by the size of the square. We choose to fix due dates of the jobs randomly between $CTime$ and $2CTime$. So, if we used the time windows $[a_k = 0,\, b_k = 2CTime]$, start at time a_k lead to serve the last customer before the first due date, so all customer are in advance, and start at time b_k means start after the last due date, so all the job are late. Let see Fig.18, the two black dotes represent the two critical dates presented below.

In other part, the approximation is tested depending the different initial path p_{init} their take as input.

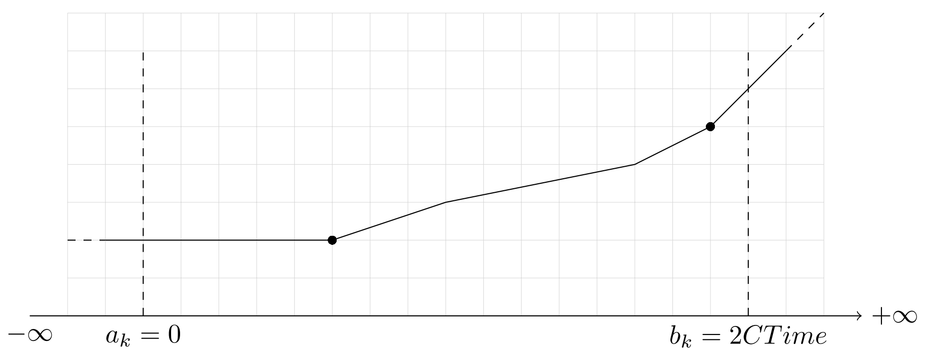

Figure 18: Departure time window for approximation

We use EDD path and nearest insertion ($near$) path and the method associated $EDDX4$ and $nearX4$. We test also the merge of the function from EDD and $near$, call $EDD + near$. Idem for their variants $X4$, ($EDDX4 + nearX4$).

For each method, we give two results, the deviation between the area under the curve of the result function and the optimal function's one, and the number of function $f_{k,p}$ generated during the method execution. (each generation of function is associated with a merge with the main function F_k).

instance type	EDD		EDDX4		near		nearX4		EDD+near		EDDX4+nearX4	
10j_s10_c5-2	1.9M	0.77%	7.9M	0.36%	1.5M	0.54%	6.8M	0.25%	3.3M	0.26%	14.8M	0.09%
10j_s10_c1	1.2M	1.43%	5.3M	0.44%	0.8M	0.93%	4.2M	0.32%	2.0M	0.43%	9.6M	0.07%
10j_s100_c5-2	2.4M	0.5%	10.2M	0.21%	2.0M	0.29%	9.1M	0.19%	4.5M	0.16%	19.3M	0.04%
10j_s100_c1	1.4M	1.1%	6.0M	0.45%	0.9M	0.77%	4.7M	0.31%	2.3M	0.34%	10.7M	0.07%
20j_s10_c5-2	27M	1.54%	103M	0.56%	18M	1.97%	87M	0.52%	45M	1.05%	191M	0.28%

5.5 Global computational time

5.6 Result

6 Conclusion