Timetabling of sorting slots in a logistic warehouse

Antoine Jouglet, Dritan Nace, Christophe Outteryck

Noname manuscript No.
(will be inserted by the editor)

Timetabling of sorting slots in a logistic warehouse

Antoine Jouglet · Dritan Nace · Christophe Outteryck

Abstract We study a problem that occurs at the end of a logistic stream in a warehouse and which concerns the timetabling of the sorting slots that are used to accommodate the prepared orders before they are dispatched. We consider a set of orders to be prepared in a certain number of preparation shops over a given time horizon. Each order is associated with the truck that will transport it to the customer. A sorting slot is an accumulation area where processed orders wait to be loaded onto a truck. For a given truck a known number of sorting slots is needed from the time the first order for this truck begins to be prepared, right up until the truck's scheduled departure time. Since several orders destined for different trucks are processed simultaneously, and since the number of sorting slots is limited, the timetabling of these resources is necessary to ensure that all orders can be processed over the considered time horizon. In this paper we describe the general industrial context of the problem and we formalize it. We state that some particular cases of the problem are polynomially solvable while the general problem is NP-complete. We then propose optimization methods for solving the problem.

Keywords warehouse management · sorting slots · scheduling · timetabling

1 Introduction

Logistic warehouses are becoming more and more complex installations: flows are continually increasing, systems are becoming more automated and logistic processes globalized. In an increasingly competitive environment, warehouse management software (comprising Warehouse Management Systems (WMS) and Ware-

Antoine Jouglet, Dritan Nace
Université de Technologie de Compiègne, UMR CNRS 7253 Heudiasyc
60200 Compiègne, France
E-mail: {antoine.jouglet,dritan.nace}@utc.fr

Christophe Outteryck
a-sis, 8, rue de la Richelandière, 42000 Saint-Etienne, France
E-mail: christophe.outteryck@a-sis.com

house Control Systems (WCS)) is becoming a vital part of operational decision making.

Since the introduction of automation systems in production and distribution environments a large number of research studies into operational decision systems have been published (see for example [3–5, 8, 9] for surveys in this area). In this paper we study a sub-problem which occurs at the end of a logistic stream in a warehouse and concerns the timetabling of sorting slots. To our knowledge, this problem has so far not been studied in the literature.

We consider a warehouse composed of several order preparation shops and a set of orders to be prepared over a given time horizon. Each order is composed of several order lines. An order line corresponds to a quantity of work to be processed in one of the preparation shops. Each order is associated with the particular transportation truck that is to transport it to its final customer. This makes it necessary for the subset of all orders for a given truck to be totally processed before the truck's scheduled departure.

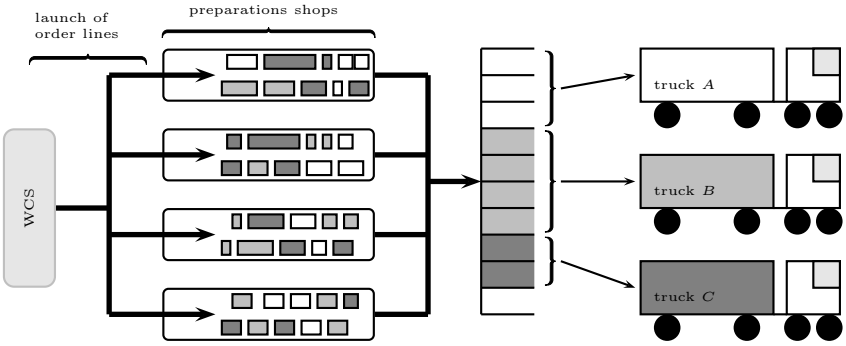

Fig. 1 Preparation of order lines in a warehouse

Once an order has been prepared it is dispatched to a sorting slot, that is to say an accumulation area where the order waits to be loaded onto the truck. The difficulty of the sorting slot timetabling problem is due to a specific resource constraint: for a given truck, from the time the first order destined for the truck begins to be prepared, and up until the truck's scheduled departure, a precise number of sorting slots must be allocated. Since several orders corresponding to different trucks are prepared simultaneously, and since the number of sorting slots is limited, timetabling the use of sorting slots is essential. The WCS dynamically makes decisions concerning the order lines to be processed in each preparation shop, and it needs to know which of these order lines can be started at a given time without adversely affecting the processing of other order lines by using up available sorting slots (see Figure 1). In other words, sorting slots need to be timetabled to ensure that all orders can be processed over the considered time horizon.

For instance, consider a simple example with only one preparation shop, 3 trucks and a capacity of 5 sorting slots. Truck 1 needs 3 sorting slots to process its orders and must leave the warehouse at time 100. Truck 2 needs 2 sorting slots to process its orders and must leave the warehouse at time 80. Truck 3 needs 2 sorting

slots to process its orders and must leave the warehouse at time 50. Suppose that

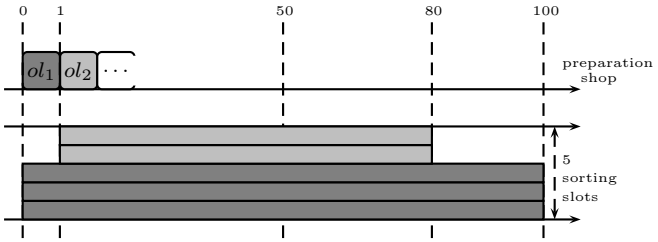

Fig. 2 A non-optimal use of sorting slots, illustrating why a timetable for sorting slots needs to be established.

the WCS has no timetable for the use of sorting slots and that it begins (at time 0) to start an order line ol_1 destined for truck 1 and, immediately afterwards (at time 1), an order line ol_2 associated with truck 2. This means that 5 sorting slots are entirely allocated to trucks 1 and 2 at least from time 1 to time 80 (the departure time of truck 2) (see Figure 2). Unfortunately, this makes it impossible to process order lines for truck 3, scheduled to depart earlier.

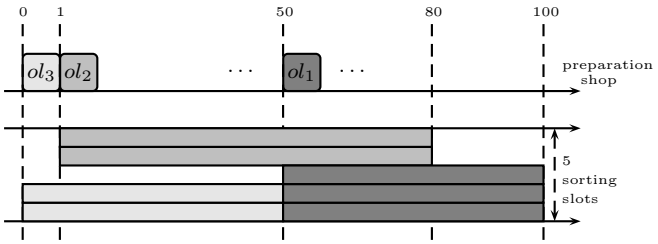

Fig. 3 Correct dynamic decisions by the WCS when the use of sorting slots is timetabled

Suppose now that a timetable for using sorting slots has been established, stipulating that 2 sorting slots are to be used by truck 2 from time 0 to time 80, 2 are to be used by truck 3 from time 0 to time 50, and 3 are to be used by truck 1 from time 50 to time 100. This timetable must make it possible for all order lines to be processed. The WCS knows which order lines it is authorized to dynamically start at a given time t, and which order lines belong to the trucks using sorting slots over an interval of time including t. In this example, order line ol_1 cannot be started before time 50 (see Figure 3).

Note that it is not our concern to schedule the order lines in the preparation shops. That is the role of the WCS, which must make this kind of decision according to the current state (not precisely predictable) of the whole system. Most of the time, a preparation shop is a very complex system involving several subsystems. One of the functions of the WCS is to start the appropriate order lines at the right time in order to balance the work between all its subsystems. This is why we shall not be assuming below that we have complete information about the set of orders. Instead, we have access to macro-information corresponding to the amount of work

related to each truck in each preparation shop. Thus, as we will see in the next section, the orders corresponding to a particular truck can be described by a set of operations (one for each preparation shop) whose processing times correspond to these amounts of work. However, it should be remembered that the schedule of these operations is not really of any importance. We simply need to be certain that for an established timetable there exists a possible scheduling of these operations. In other words, we have to find a timetable for the use of sorting slots, given an operation scheduling feasibility constraint.

The remainder of the paper is organized as follows. We first define the problem in Section 2. Two operational cases are considered: the *non-preemptive case* in which order lines for the same preparation shop and for the same transportation truck must be processed in a single operation, and *the preemptive case* in which this constraint is released. We then provide a study of the complexity of the problem in Section 3: we state that some particular cases of the problem are polynomially solvable, while the general problem is NP-complete [6]. In the remaining sections, we propose optimization methods to solve it. Thus, in Section 4, we describe a dominance rule allowing integer linear programming formulations and constraint-based scheduling methods [1] to be built to solve the problem in practice. These methods are described in Section 5 for the non-preemptive case and in Section 6 for the preemptive case. Some experimental results are provided in Section 7 and we conclude in Section 8.

2 Problem definition

The problem in hand can be formalized as follows. Let n be the number of trucks and let m be the number of preparation shops in the warehouse. From a scheduling-based modeling point of view, a truck is assimilated to a job and a preparation shop is assimilated to a machine. Thus, let $N = \{1, \ldots, n\}$ be the set of jobs (trucks) and let $M = \{1, \ldots, m\}$ be the set of machines (preparation shops).

Each job i is composed of m operations $\{o_{i1}, o_{i2}, \ldots, o_{im}\}$. The processing time p_{ij} of operation o_{ij} corresponds to the duration of the whole set of order lines which have to be processed by the machine (preparation shop) j that corresponds to truck i, *i.e.* to the total amount of work which has to be done by shop j for truck i. Each of the m machines therefore has to process n operations (one per job). Let d_i be the known departure date of truck i. In a feasible solution of the problem, each job i is then considered to have finished by time d_i, while the start-time s_i of i is to be determined such that all operations of i can be performed. The set of sorting slots is assimilated as a cumulative resource of capacity E, E being the number of sorting slots available in the warehouse. Thus, job i needs a given number n_i of sorting slots from among E, from time s_i up until time d_i. Note that the period over which the resource constraint applies depends only on the start-time of the job, in contrast to classical scheduling problems where it depends on periods during which the job is actually being processed.

Let s_{ij} and c_{ij} be the variables representing the start time and the completion time respectively of operation o_{ij} in a particular solution. In a feasible solution, each operation o_{ij} can start after the start of job i and has to be completed before the departure time of the associated truck: $s_{ij} \geq s_i$ and $c_{ij} \leq d_i$. Each machine is disjunctive and can process only one operation at a time. Note that unlike in a

scheduling shop problem (see *e.g.* [2]), operations for the same job can be processed simultaneously (and, in fact, very often will be).
Two cases are considered:

- The non-preemptive case where, once it has begun to be processed by a machine, an operation o_{ij} must be continued until its completion ($s_{ij}+p_{ij}=c_{ij}$). This implies that all order lines in a preparation shop associated with a truck will be dealt with sequentially.
- The preemptive case, where interruptions to the processing of operations are possible. Such interruptions are useful, since they can allow part of another more urgent operation associated with another truck to be executed. This may be desirable when more flexibility is preferred in preparation shops. In this case s_{ij} is taken as the first time an element of operation o_{ij} is processed by machine j, while c_{ij} is the date at which the operation is completed. Thus we have $s_{ij}+p_{ij} \leq c_{ij}$.

The problem is to determine the values of variables $\{s_1, \ldots, s_n\}$ such that no more than E sorting slots can be used at a time, while the scheduling of operations is feasible. If variables $\{s_1, s_2, \ldots, s_n\}$ can be fixed such that all the previous constraints hold, it follows that n_i sorting slots from among E will be used from s_i to d_i for truck i. Thus, the values of variables $\{s_1, s_2, \ldots, s_n\}$ and data $\{d_1, \ldots, d_n\}$ provide the timetable for the use of the sorting slots. From now on we shall refer to this problem as "**the search problem**".

Of course, if several solutions exist, we should like to find the best solution from an operational point of view. In this context, the longer the span of time over which a truck can use sorting slots, the more flexibility the WCS has for processing the orders for the truck in question. Greater overall versatility is thus ensured by using the sorting slots for as long as possible, *i.e.* by leaving as few idle times as possible in the use of sorting slots. Let $H = \max_{i\in\{1,\ldots,n\}} d_i$ be the time horizon of an instance of the problem. One interesting objective is therefore to minimize $H \times E - \sum_{i\in N} n_i(d_i - s_i)$, which is equivalent to minimizing $\sum_{i\in N} n_i s_i$.

From now on we shall refer to the search for a solution that minimizes $\sum_{i\in N} n_i s_i$ as "**the optimization problem**".

We shall be using the following additional notation. Let $T = \{d_i | i \in N\} \cup \{0\} = \{t_0 = 0, \ldots, t_w = H\}$ be the set of departure dates, their values t_i being indexed in non-decreasing order, *i.e.* $t_0 = 0 < t_1 < \ldots < t_w = H$. Moreover, for all $i \in N$, let $[i]$ be the position of d_i in the ordered set T, *i.e.* $[i] = k$ such that $k \in \{1, \ldots, w\}$ and $d_i = t_k$.

3 Some complexity results

In this section we propose several complexity results for the general problem and for some of its special cases.

The special preemptive case where we have $\sum_{i=1}^{n} n_i \leq E$ is polynomially solvable. The resource constraint according to the number of sorting slots disappears in this case, and it only remains to determine whether the scheduling of the operations is feasible, which will enable us to solve m single-machine scheduling problems with deadlines $(1|\bar{d}_i|\_)$ which are polynomially solvable [6].

The other special case in which any $p_{ij} = 1$ is also polynomially solvable. Here the operations can be scheduled identically on all the machines. The problem is thus reduced to a problem with $m = 1$. We then have to determine whether the operations can be scheduled without violating the resource constraint on sorting slots. To this end we iteratively schedule operations from H to 0 in decreasing order of departure times. When several operations have the same departure time, they are scheduled in decreasing order of the number of sorting slots. The process stops when all operations are scheduled or when the resource constraint on the sorting slots is no longer satisfied. If the latter is the case, then the instance is not feasible.

Proposition 1 *The **preemptive** case with $m = 1$ is $NP-complete$.*

Proof To prove this result, we state that the KNAPSACK problem [6] can be reduced to the preemptive case with $m = 1$. The KNAPSACK problem can be stated as follows:
***INSTANCE**: A positive integer B. Two sets $\{a_i | i \in I = \{1, \ldots, u\}\}$ and $\{b_i | i \in I = \{1, \ldots, u\}\}$ of u strictly positive integers. Two positive integers A and B.*
***QUESTION**: Is there a subset $I' \subseteq I$ such that $\sum_{i \in I'} a_i \geq A$ and $\sum_{i \in I'} b_i \leq B$?*
The decision version of the preemptive sorting slot problem with $m = 1$ can be stated as follows:
***INSTANCE**: A set $N = \{1, \ldots, n\}$ of n jobs and a positive number E of sorting slots. Each job $i \in N$ has a completion time d_i, requires n_i sorting slots in order to be processed and has one operation whose processing time is p_i.*
***QUESTION**: Is there a preemptive single-machine schedule of the operations such that no more than E sorting slots are used at any one time, and such that each operation is completed before the completion time of its job?*

Consider an instance of the KNAPSACK problem. We build an instance of the preemptive sorting slot problem with $n = u + 1$ jobs, $E = \sum_{i \in I} a_i$ and we set $H = 1 + \sum_{i \in I} b_i$. For $i \in \{1, \ldots, u\}$ we set $d_i = H$, $p_i = b_i$ and $n_i = a_i$. We set $d_n = H - B$, $p_n = 1$ and $n_n = A$. Note that this reduction is polynomial.

Suppose that there exists a subset $I' \subseteq I$ such that $\sum_{i \in I'} a_i \geq A$ and $\sum_{i \in I'} b_i \leq B$. We process the operations as follows. Operations of jobs $i \in I'$ are processed from time $H - \sum_{i \in I'} p_i$ to time H in any order. Operations of jobs $i \in I \cup \{n\} \setminus I'$ are processed from time 0 to time $H - \sum_{i \in I'} p_i$. These jobs need $\sum_{i \in I \setminus I'} a_i + A \leq \sum_{i \in I} a_i$ sorting slots to be processed. Since jobs in I' are totally processed after time $H - \sum_{i \in I'} p_i \geq H - B$ (since $\sum_{i \in I'} p_i = \sum_{i \in I'} b_i \leq B$), job n has released A sorting slots. Hence, there remain at most $\sum_{i \in I \setminus I'} a_i$ occupied sorting slots and there are still $\sum_{i \in I'} a_i$ sorting slots available to be used from jobs in I'.

Now suppose that there exits a valid preemptive schedule for the built instance of the preemptive sorting sort problem. Since job n requires at least A sorting slots from time $H - B - 1$ to time $H - B$, then a subset $I' \subseteq I$ is totally processed after time $H - B$ letting at least A available sorting slots. This subset I' of jobs is then such that $\sum_{i \in I'} a_i \geq A$ and $\sum_{i \in I'} b_i \leq B$.

Proposition 2 *The **non-preemptive** case with $m = 1$ is strongly $NP-complete$.*

Proof To prove this result, we first establish that the 3-PARTITION problem [6] can be reduced to the general non-preemptive case with $m = 1$. The 3-PARTITION

problem can be stated as follows:
***INSTANCE**: A positive integer B. A set $A = \{a_i | i \in I = \{1, \ldots, 3q\}\}$ of $3q$ integers with $\sum_{i \in I} a_i = qB$ and $\forall i \in I, B/4 < a_i < B/2$.*
***QUESTION**: Is there a partition of I into q subsets $\{I_1, \ldots, I_q\}$ of cardinal 3 such that $\forall k \in \{1, \ldots, q\}, \sum_{j \in I_k} a_j = B$?*
The decision version of the non-preemptive sorting slot problem with $m = 1$ can be stated as follows:
***INSTANCE**: A set $N = \{1, \ldots, n\}$ of n jobs and a positive number E of sorting slots. Each job $i \in N$ has a completion time d_i, needs n_i sorting slots to be processed and has an operation whose processing time is p_i.*
***QUESTION**: Is there a non-preemptive single-machine schedule of the operations such that no more than E sorting slots are used at a time and that each job is completed before the completion time of its job?*

Consider an instance of the 3-PARTITION problem. We build an instance of the sorting slot problem with $n = 5q$ jobs, $E = 3q$ and we set $H = qB + 2q$. For $i \in \{1, \ldots, 3q\}$ we set $d_i = H$, $p_i = a_i$ and $n_i = 1$. For $k \in \{0, \ldots, q-1\}$, we set $d_{3q+2k+1} = (B+2)k+1$, $d_{3q+2k+2} = (B+2)k+2$, $p_{3q+2k+1} = p_{3q+2k+2} = 1$ and $n_{3q+2k+1} = n_{3q+2k+2} = 3(q-k)$, and so on. This reduction is polynomial.

Suppose that there exists a partition of I into q subsets $\{I_0, \ldots, I_{q-1}\}$ of cardinal 3 such that $\forall k \in \{0, \ldots, q-1\}$ we have $\sum_{j \in I_k} a_j = B$. We build a feasible schedule as follows.

For $k \in \{0, \ldots, q-1\}$, operation of job $3q + 2k + 1$ is scheduled from time $(B + 2)k$ to time $(B + 2)k + 1$ and operation of job $3q + 2k + 2$ is scheduled from time $(B + 2)k + 1$ to time $(B + 2)k + 2$. Note that the departure time of each of these jobs is then respected. Moreover, the constraint on sorting slots holds, since these jobs are totally processed on disjoint intervals and never use more than $3q = E$ sorting slots. Next, let k iteratively take values in $\{0, \ldots, q-1\}$. Operations of jobs of indices in I_k are scheduled in any order within the interval $[(B+2)k+2, (B+2)(k+1)[$. Note that the machine is never idle during this interval, since $\sum_{j \in I_k} p_j = \sum_{j \in I_k} a_j = B$ and $(B+2)(k+1) - (B+2)k - 2 = B$. Moreover, at least $3(q-k+1)$ sorting slots are available from time $(B+2)k+2$, since job $3q+2k+2$ has just finished. 3 of these sorting slots are used by jobs in I_k up until time H, leaving the $3(q-k)$ other slots for the processing of jobs $3q+2(k+1)+1$ and $3q+2(k+1)+2$ within the interval $[(B+2)(k+1), (B+2)(k+1)+2[$.

*Now suppose that there exists a feasible non-preemptive schedule for the jobs. This implies that the machine is never idle on interval $[0, H[$ since $\sum_{i=1}^{i=5q} p_j = \sum_{i=1}^{i=3q} a_j + 2q = qB + 2q = H$. Recall that during interval $[H-1, H[$ all sorting slots are necessarily used by jobs $\{1, \ldots, 3q\}$. Job $5q$ requires at least $n_{5q} = n_{3q+2(q-1)+2} = 3(q-(q-1)) = 3$ sorting slots in order to be processed during the interval $[H-B-1, H-B[$. Thus, at least 3 operations from jobs among $\{1, \ldots, 3q\}$ must start during the interval $[H-B, H[$ (see Figure 4). Moreover, no more than 3 operations can be processed over this interval of size B, because $\forall i \in \{1, \ldots, 3q\}, p_i = a_i > B/4$ and **preempting is not possible**. As the machine is never idle, there is no other choice for job $5q-1$ but to use the same 3 sorting slots as job $5q$. Since $d_{5q-1} = H-B-1$ and $d_{5q} = H-B$, the operation of job $5q$ is necessarily scheduled from time $H-B-1$ to time $H-B$. Therefore exactly 3 operations from jobs x, y and z such that $p_x + p_y + p_z = B$ are necessarily processed by the machine in interval $[H-B, H[$. We then set $I_{q-1} = \{x, y, z\}$.*

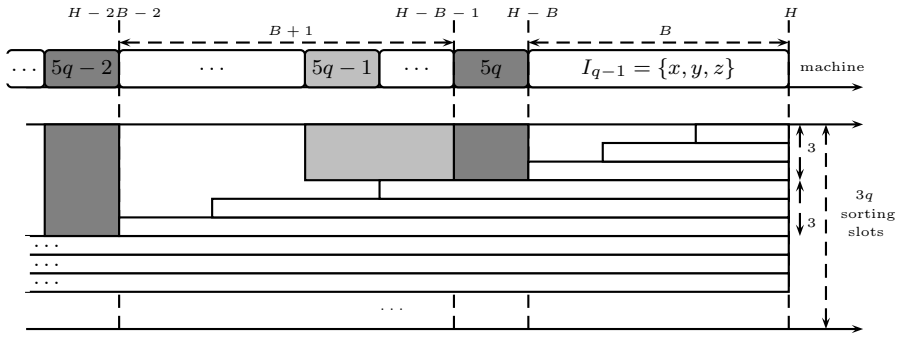

Fig. 4 Reducing the 3-PARTITION problem to the sorting slot problem (the part above the time-axis represents the schedule of jobs on the machine, while the part below represents the use of the sorting slots over the time horizon). **Case of** I_{q-1}

Note that unlike job $5q$*, job* $5q-1$ *can use these 3 sorting slots starting at a date earlier than* $H-B-2$ *if no other job requires them.*

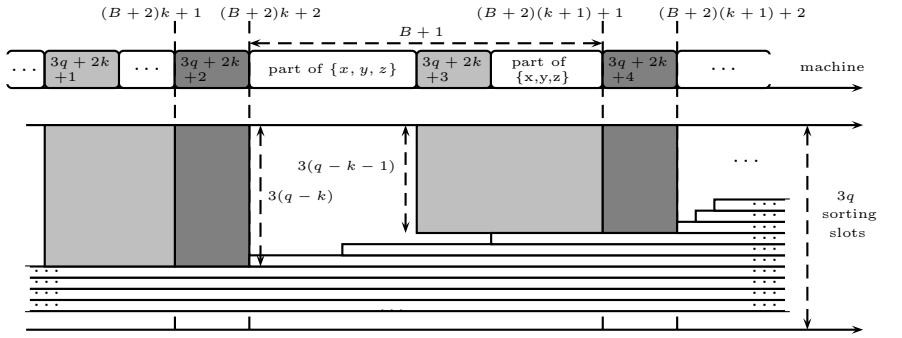

Fig. 5 Reducing the 3-PARTITION problem to the sorting slot problem (the part above the time-axis represents the schedule of jobs on the machine while the below part represents the use of the sorting slots over the time horizon). **General case.**

Consider $k \in \{0, \ldots, q-2\}$ *and let us assume that, for all* $z \in \{k+1, \ldots, q-1\}$*, job* $\{3q+2z+2\}$ *starts at* $(B+2)z+1$ *and is completed at* $(B+2)z+2$*, that* $3(q-z)$ *operations of jobs in* $\{1, \ldots, 3q\}$ *are totally processed during the interval* $[(B+2)z+2, H[$ *and that job* $3q+2z+1$ *uses* $3(q-z)$ *sorting slots from its start to its departure time* $(B+2)z+1$*. Note that this assumption is true for* $k = q-1$*, as previously shown.*

Job $3q+2k+2$ *needs at least* $3(q-k)$ *sorting slots over the interval* $[(B+2)k+1, (B+2)k+2[$ *for its execution to be possible. Exactly* $3(q-k-1)$ *jobs in* $\{1, \ldots, 3q\}$ *are totally processed after time* $(B+2)(k+1)+2$*, leaving* $3(q-k-1)$ *sorting slots which are all used by job* $3q+2(k+1)+2 = 3q+2k+4$ *over the interval* $[(B+2)(k+1)+1, (B+2)(k+1)+2[$ *and by job* $3q+2(k+1)+1 = 3q+2k+3$ *from its start time to its departure time* $(B+2)(k+1)+1$ *(see Figure 5). At most* 4 *operations belonging to jobs from among* $\{1, \ldots, 3q\}$ *may be executed over the*

interval $[(B+2)k+2,(B+2)(k+1)+1[$ *of size* $B+1$ *(the total processing time for 5 operations belonging to these jobs is necessarily strictly greater than* $B+1$*). This means that at most 4 sorting slots may be freed for job* $3q+2k+2$*, which is not sufficient, since* $3(q-k) >= 6$*. The only way that* $3(q-k)$ *sorting slots might become available for job* $3q+2k+2$ *before time* $(B+2)k+2$ *would be for 3 jobs to start in* $\{1,\ldots,3q\}$ *and for job* $3q+2(k+1)+1$ *to start during* $[(B+2)k+2,(B+2)(k+1)+1[$*, which would leave exactly the* $3+3(q-k-1)=3(q-k)$ *sorting slots needed by job* $3q+2k+2$*. However, there is no alternative to processing the operations belonging to job* $3q+2k+2$ *from time* $(B+2)k+1$ *to time* $(B+2)k+2$*, because job* $3q+2k+1$ *needs the same available sorting slots before its departure time* $(B+2)k+1$*. Since job* $3q+2k+3$ *needs one unit of time to process its operation on the machine during* $[(B+2)k+2,(B+2)(k+1)+1[$*, we must have 3 operations of jobs* x*,* y *and* z *in* $\{1,\ldots,3q\}$ *such that* $p_x+p_y+p_z=B$ *being processed on the* B *remaining unit times of this interval, thus giving the subset* $I_k=\{x,y,z\}$*.*

This shows that there exists a partition of I *into* q *subsets* $\{I_0,\ldots,I_{q-1}\}$ *of cardinal 3 such that for all* $k\in\{0,\ldots,q-1\}$ *we have* $\sum_{j\in I_k} a_j = B$*.*

4 Dominance rules

Proposition 3 *Let* S *be a feasible solution of an instance of the problem in which a job* i *is executed from* s_i *to* d_i*. It is possible to transform* S *into a feasible solution* S' *in which job* i *uses* n_i *sorting slots from* $s'_i=\max\{t_k|t_k\in T\wedge t_k\leq s_i\}$ *and where the processing of operations is not modified.*

Proof In a feasible solution S*, note that the times at which some sorting slots become free after use can occur only at points* $\{t_1,\ldots,t_k\}$ *of the considered time horizon. If the cumulative constraint on sorting slots for a job* i *is satisfied from*

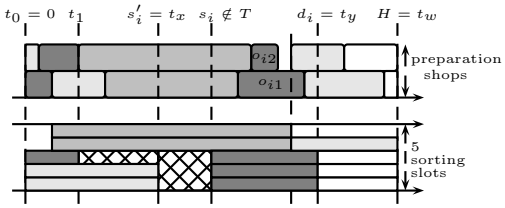

Fig. 6 Starting times for sorting slots correspond to the different values in set T

s_i *to* d_i*, then the corresponding sorting slot* n_i *can start to be used at time* $s'_i=\max\{t_k|t_k\in T\wedge t_k\leq s_i\}$ *without violating the cumulative constraint (see Figure 6).*

We can deduce the following dominance rules.

Proposition 4 *There exists at least one solution of the search problem in which the starting times for sorting slots for the different jobs correspond to the values in set* T*.*

Proof This proposition is a direct consequence of Proposition 3.

Proposition 5 *All optimal solutions of the optimization problem are such that the starting times for sorting slots for the different jobs correspond to the values in set T.*

Proof Consider an optimal solution of the optimization problem in which the start time s_i of job i is not in T. As shown in Proposition 3, if the cumulative constraint on sorting slots is respected from s_i to d_i for a job i, then the corresponding sorting slot n_i can start to be used at time $s'_i = \max\{t_k | t_k \in T \wedge t_k \leq s_i\}$ without violating the cumulative constraint. Since $s'_i < s_i$, we obtain a new solution with a smaller value of the objective function ($\sum_{i \in N} n_i s_i$), contradicting the hypothesis that the initial solution was optimal.

The previous two propositions can be used in several ways. From a practical point of view, they offer a way of simplifying the timetabling of the sorting slots, since timetables can be established in terms of periods whose time points correspond to the departure times of the different trucks. Of course, the additional use (in principle, futile) of sorting slots by a job can be seen as additional versatility in the preparation process. From the point of view of the problem resolution, these dominance rules may be used to establish efficient models, as shown in the following sections.

5 Models and approaches for the non-preemptive case

5.1 A mixed integer linear programming approach

In this section, we propose a first mixed integer linear programming model which allows the problem to be solved.

We could have proposed a classical time-indexed mixed-integer formulation for modeling the use of sorting slots by the jobs, where, for example, some binary variables $x_{i,t}$ are used to determine at any time t whether or not job i is using sorting slots. However, thanks to the dominance rule presented in the previous section, we know that if a job i uses sorting slots at time t, then it can use the same sorting slots at time $t-1$, if $t-1$ is not the departure time of a job. It is consequently possible to avoid taking the constraint on sorting slots into account, other than at each interval $[t_k, t_{k+1}[$, with $k \in \{0, \ldots w-1\}$.

This can be achieved through the use of binary variables $x_{i,k}$ for each job i and each interval $[t_k, t_{k+1}[$, which are equal to 1 if job i uses n_i sorting slots on $[t_{k-1}..t_k[$, and 0 otherwise. The constraints linked to these variables can be formulated as follows.

$$x_{i,k} = 0, \forall i \in N, \forall k \in \{[i]+1, \ldots, w-1\} \quad (1)$$

$$x_{i,[i]} = 1, \qquad \forall i \in N \quad (2)$$

$$x_{i,k} \leq x_{i,k+1}, \quad \forall i \in N, \forall k \in \{0, \ldots, [i]-1\} \quad (3)$$

$$\sum_{i \in N} x_{i,k} n_i \leq E, \qquad \forall k \in \{0, \ldots, w-1\} \quad (4)$$

Equation (1) ensures that job i cannot use sorting slot after the departure d_i. Equation (2) means that job i should use sorting slots in the period $[t_{[i]-1}, t_{[i]}[$ (in

the interval just before departure of truck i). Equation (3) ensures that there is no interruption in the use of sorting slots from the start of job i until the departure time d_i. Equation (4) ensures that in each interval $[t_k, t_{k+1}[$, the total number of used sorting slots is at most E. Note that these equations are also valid in the preemptive case and will be used in the next section.

To compute possible values for the starting times s_i of the jobs we can introduce variables $\{s_i | i \in N\}$ into the MIP by adding the following constraints, implying that the starting time of a job i that does not use sorting slots in interval $[t_k, t_{k+1}[$ with $t_{k+1} < d_i$ is greater than or equal to t_{k+1}:

$$s_i \geq (1 - x_{i,k})t_{k+1}, \forall i \in N, \forall k \in \{0, \ldots, [i]\} \tag{5}$$

We note that an operation o_{ij} can be processed only in the periods in which job i is active, *i.e.* in intervals $[t_k..t_{k+1}[$ such that $x_{i,k} = 1$. We then need to show that the scheduling of operations is possible according to the values of the job start times. For this we use a classical scheduling model, with binary variables $b^j_{il} \in \{0, 1\}$, $j \in M, i, l \in N, i < l$ such that $b^j_{il} = 1$ if operation o_{ij} starts before operation o_{lj} on machine j, otherwise $b^j_{il} = 0$. The search version of the problem can be mathematically formulated as follows:

$$s_{ij} + p_{ij} \leq d_i, \qquad \forall i \in N, \forall j \in M \tag{6}$$

$$s_i \leq s_{ij}, \qquad \forall i \in N, \forall j \in M \tag{7}$$

$$s_{ij} + p_{ij} \leq s_{lj} + \Omega \times (1 - b^j_{il}), \forall j \in M, \forall i, l \in N, i < l \tag{8}$$

$$s_{lj} + p_{lj} \leq s_{ij} + \Omega \times b^j_{il}, \forall j \in M, \forall i, l \in N, i < l \tag{9}$$

$$b^j_{il} \in \{0, 1\}, \forall j \in M, \forall i, l \in N, i < l \tag{10}$$

Ω is an arbitrary big number which can be H. Equations (6) and (7) ensure that an operation o_{ij} starts after s_i and ends before d_i. Equations (8) and (9) deal with the disjunctive constraints of operations on each machine and ensure that operations can be non-preemptively scheduled. If $b^j_{il} = 1$, then operation o_{ij} starts before operation o_{lj} on machine j, $s_{ij} + p_{ij} \leq s_{lj}$ and $s_{lj} + p_{lj} \leq s_{ij} + \Omega$. If, on the other hand, $b^j_{il} = 0$, then operation o_{lj} starts before operation o_{ij} on machine j, $s_{ij} + p_{ij} \leq s_{lj} + \Omega$ and $s_{lj} + p_{lj} \leq s_{ij}$.

The optimization version of the problem is obtained by setting the minimization of $\sum_{i \in N} n_i s_i$ as objective function.

5.2 A constraint-based scheduling approach

Constraint programming has been widely applied in the area of scheduling [1]. In our constraint-based scheduling model, an activity of duration p_{ij} is associated with each operation o_{ij} and two variables s_{ij} and c_{ij} represent respectively its start time and its completion time. The constraint $s_{ij} + p_{ij} = c_{ij}$ is maintained (since we are in the non-preemptive case). Let $Dom(V)$ be the set of values which can be taken by a variable V. The minimum and maximum values of the domain of s_{ij} are respectively denoted as:

- $est_{ij} = \min_{v \in Dom(s_{ij})} v$: the earliest starting time of operation o_{ij},
- $lst_{ij} = \max_{v \in Dom(s_{ij})} v$: the latest starting time of operation o_{ij}.

Similarly, the minimum and maximum values of the domain of c_{ij} are respectively denoted as:

- $eet_{ij} = \min_{v \in Dom(c_{ij})} v = est_{ij} + p_{ij}$: the earliest completion time of operation o_{ij},
- $let_{ij} = \max_{v \in Dom(c_{ij})} v = lst_{ij} + p_{ij}$: the latest completion time of operation o_{ij}.

Thus, each operation o_{ij} has to be executed within the time window $[est_{ij}, let_{ij})$ whose bounds can be initialized with $est_{ij} = 0$ and $let_{ij} = d_i$.

Each machine j is modeled with a disjunctive unary resource, each operation o_{ij} requiring the resource from its start time to its completion time. Edge-finding propagation techniques [1] which are able to adjust the time-windows of operations according to the disjunctive constraints between operations and their possible starting times are used. Each job i is also represented by an activity whose start time is denoted by the variable s_i (its completion time being already fixed at d_i). In contrast to the activities corresponding to the operations, the durations of the activities corresponding to the jobs are not fixed, and these are denoted by the variable p_i. The constraint $s_i + p_i = d_i$ is maintained. The sorting slots are represented by a discrete resource of capacity E. Each job i requires n_i units of this resource from its start time to its completion time. Note that the domains of variables s_i, p_i, s_{ij} and c_{ij} will be reduced through the use of propagation algorithms as the search proceeds.

In the optimization problem an additional variable $\overline{\Phi}$ represents the objective function $\overline{\Phi} = \sum_{i=1}^{n} n_i s_i$ to be optimized. To find an optimal solution we use a branch and bound algorithm solving successive variants of the decision problem. At each iteration we try to improve the best known solution and to this end we add an additional constraint stipulating that $\overline{\Phi}$ is lower than or equal to the best solution minus 1. At each new iteration the search resumes at the last-visited node of the previous iteration.

At each node of the enumeration procedure, the operation which can be scheduled at the latest time is scheduled at its latest start time. When the created branch fails, the operation is backward-postponed pending modification of its latest start time following the propagation of decisions. When all operations have been scheduled, the job which can start the earliest among the others, is scheduled at its earliest start time. When the created branch fails, the job is forward-postponed pending modification of its earliest start time following the propagation of decisions.

To improve the behavior of the enumeration procedure, we try to limit the enumeration of (partial) equivalent solutions. Consider a machine j, and suppose that 3 operations o_{xj}, o_{yj}, o_{zj} are consecutively scheduled on machine j (*i.e.* $c_{xj} = s_{yj}$ and $c_{yj} = s_{zj}$). Suppose also that over the interval $[s_{xj}, c_{zj}[$ jobs x, y and z use sorting slots (*i.e.* $s_x \leq s_{xj}$, $s_y \leq s_{xj}$, $s_z \leq s_{xj}$ and $d_x \geq c_{zj}$, $d_y \geq c_{zj}$, $d_z \geq c_{zj}$. Then clearly operations o_{xj}, o_{yj} and o_{zj} can be permuted (in any order) and scheduled from the initial value of s_{xj} without changing the validity (or the optimality, in the case of the optimization problem) of the solution (see Figure 7).

The following dominance rule can therefore be established:

Proposition 6 *There is at least one (optimal) solution in which* $\forall j \in M$ *and* $\forall x, y \in N$ *such that* $c_{xj} = s_{yj}$, *at least one of the* 3 *conditions holds:*

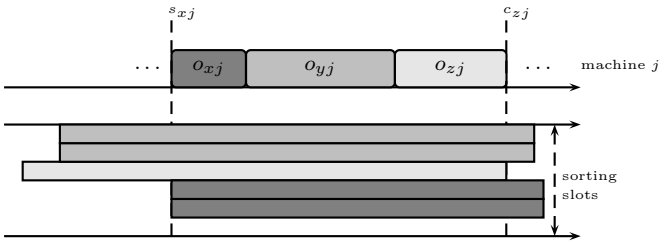

Fig. 7 Equivalent solutions when permuting operations o_{xj}, o_{yj} and o_{zj}.

1. $x < y$;
2. $s_y > s_{xj}$;
3. $d_x < c_{yj}$.

Proof Consider a solution in which there exist two operations o_{xj} and o_{yj} scheduled on machine j such that $y > x$, $s_y \leq s_{xj}$ and $d_x \geq c_{yj}$. It is possible to build an equivalent solution in which operation o_{yj} is scheduled at the initial time s_{xj} and where operation o_{xj} is scheduled immediately after o_{yj} and is completed at the initial value of c_{yj}. Since in the new schedule operations o_{xj} and o_{yj} are scheduled in the interval during which their jobs are using sorting slots, and since the schedule of the other operations does not change, the obtained schedule remains valid. Moreover, the start time of jobs x and y does not change. Therefore, if the initial solution was optimal, the obtained solution is also optimal.

This dominance rule is implemented as a propagation algorithm to ensure that only solutions belonging to the dominant subset of solutions are enumerated.

6 Models and appoaches for the preemptive case

In the previous section the start times of operations were computed simply in order to establish these operations could be scheduled. The values of the start times obtained were not really required. In the preemptive case this computation is unnecessary. Ensuring that no more than $t_{k+1} - t_k$ units of jobs are processed on each machine $j \in M$ and within each interval $[t_k, t_{k+1}[$ is sufficient to ensure that such a schedule exists. We now propose two models based on this property.

6.1 A flow-based mixed integer linear programming approach

In this model we propose a flow-based mixed integer linear programming. Just like in Section 5.1, this is implemented using binary variables for each job i and for each interval $[t_k, t_{k+1}[$. The variable $x_{i,k}$ is equal to 1 if job i uses n_i sorting slots on $[t_{k-1}..t_k[$, otherwise it is 0. The same constraints described by equations (1-4) are used on these variables to manage the constraint regarding the number of available sorting slots. In addition we use the set of variables $\{s_i | i \in N\}$ to represent the start time of jobs, along with the set of constraints described by Equation (5).

Now, suppose that the values of variables x_{ik} are known.The problem of the feasibility of operations on each machine j can then be seen as a kind of max flow

problem. From this point of view, the network flow corresponding to each machine j is a connected, directed graph $G(X, U)$, where X is the set of vertices and U the set of edges. Set X is composed of a vertex $Source$, a vertex $Sink$, n vertices o_{ij} with $i \in \{1, \ldots, n\}$ associated with the operations to be processed by machine j, and w vertices I_k with $k \in \{0, \ldots, w-1\}$ corresponding to the consecutive intervals $I_0 = [t_0, t_1[, I_1 = [t_1, t_2[, \ldots, I_{w-1} = [t_{w-1}, t_w = H[$ that partition interval $[0, H[$ (see Figure 8).

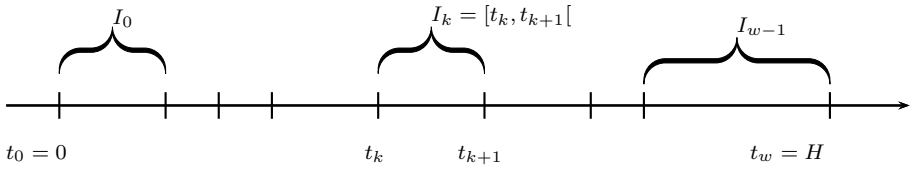

Fig. 8 Intervals I_0, I_1, ..., I_{w-1}

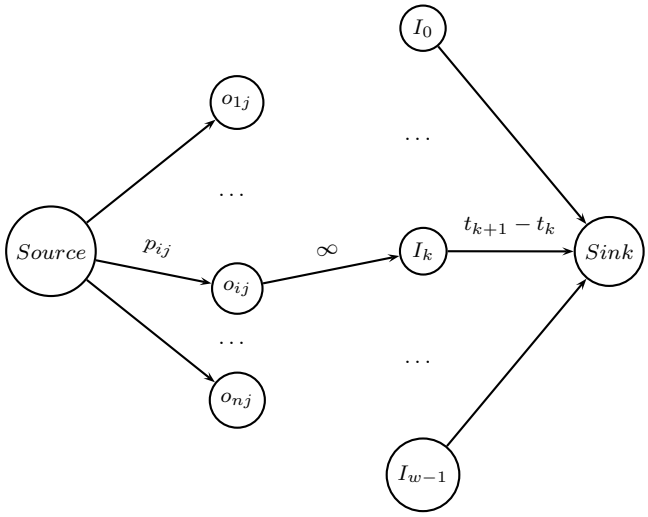

Fig. 9 A network flow representation of the preemptive problem (each edge is labeled with its capacity).

Vertex $Source$ is connected to each vertex o_{ij} by an edge of capacity p_{ij}. Each vertex I_k is connected to vertex $Sink$ by an edge of capacity $t_{k+1} - t_k$. Finally, each vertex o_{ij} is connected to vertex I_k by an edge of infinite capacity if $t_{k+1} \leq d_i$ (see Figure 9). It is easy to see that the preemptive scheduling of the operations on machine j is feasible if and only if the maximal flow which can be sent through the network is equal to $\sum_{i \in N} p_{ij}$.

This flow-max subproblem is modeled using variables f_{ijk}, $\forall i \in N, \forall j \in M, \forall k \in \{0, \ldots, w-1\}$ corresponding to the quantity of operation o_{ij} which is

processed by machine j during interval $[t_k, t_{k+1}[$. This subproblem is mathematically formulated by adding the following constraints:

$$f_{ijk} \leq x_{ik}(t_{k+1} - t_k), \forall i \in N, \forall j \in M, \forall k \in \{0, \ldots, w-1\} \quad (11)$$

$$\sum_{k \in \{0,\ldots,w-1\}} f_{ijk} = p_{ij}, \qquad \forall i \in N, \forall j \in M \quad (12)$$

$$\sum_{i \in N} f_{ijk} \leq t_{k+1} - t_k, \qquad \forall j \in M, \forall k \in \{0, \ldots w-1\} \quad (13)$$

Equation (11) ensures that an operation o_{ij} cannot be processed during an interval $[t_k, t_{k+1}[$ in which job i does not use sorting slots. Equation (12) ensures that operation o_{ij} is totally processed in $[0, H[$. Equation (13) ensures that machine j processes at most $t_{k+1} - t_k$ unit of operation on interval $[t_k, t_{k+1}[$.

The optimization version of the problem is obtained by adding the minimization of $\sum_{i \in N} n_i s_i$.

6.2 A multiperiod approach

In this model we propose a kind of multiperiod approach. In the flow-based model described above, although we do not know exactly how the component parts of operations are scheduled in each interval $[t_k, t_{k+1}[$, we nevertheless know what quantity of each operation o_{ij} is scheduled. Actually, we can go further in the non-knowing thing by verifying only that the amount of operations processed by each machine j on each interval $[t_k, t_l[$ (with $t_k < t_l$) is less than or equal to $t_l - t_k$.

To implement this idea, we use binary variables $y_{i,k}, \forall i \in N, \forall k \in \{0, \ldots w-1\}$ such that y_{ik} is equal to 1 if job i begins to use sorting slots in interval $[t_k, t_{k+1}[$ and to 0 in the other case. Note that $y_{ik} = 1$ means that job i will use n_i sorting slots in interval $[t_k, t_{[i]}[$. The search version of the problem can be mathematically formulated as follows:

$$\sum_{k \in \{0,\ldots,w-1\}} y_{ik} = 1, \qquad \forall i \in N \quad (14)$$

$$\sum_{i \in N | d_i > t_k \wedge d_i \leq t_l} p_{ij} \sum_{z=k}^{l} y_{iz} \leq t_l - t_k, \qquad \forall j \in M, \forall k \in \{0, \ldots, w-1\}, \forall l \in \{k+1, \ldots, w\} \quad (15)$$

$$\sum_{i \in N | d_i > t_k} n_i \sum_{l=0}^{k} y_{il} \leq E \quad \forall k \in \{0, \ldots, w-1\} \quad (16)$$

Equation (14) ensures that job i begins to use sorting slots in only one interval. Equation (15) ensures that within each interval $[t_k, t_l[$ such that $(t_k, t_l) \in T^2$ (with $t_k < t_l$), the amount of operation which is processed by each machine j is less than or equal to $t_l - t_k$. Note that on machine j, the operations which are totally processed within interval $[t_k, t_l[$ belong to the jobs whose start times are in $[t_k, t_l[$ (*i.e.* those where $\sum_{z=k}^{l} y_{iz} = 1$) and whose completion times are in $]t_k, t_l]$ (*i.e.* jobs i such that $d_i > t_k \wedge d_i \leq t_l$). Equation (16) ensures that the jobs which start at a

date less than or equal to t_k (*i.e.* the jobs i such that $\sum_{l=0}^{k} y_{il} = 1$) and which end after t_k (*i.e.* jobs i such that $d_i > t_k$) use at most E sorting slots on $[t_k, t_{k+1}[$. In conventional multiperiod models some term introduced in some constraint (period) will appear in all subsequent constraints (periods). In our model, the multiperiod aspect is to be found essentially in constraint (16), where each constraint indexed at k involves the previous one (indexed at $k-1$), subject to some condition on d_i.

Recall that $y_{ik} = 1$ means that job i will begin to use sorting slots at time t_k. The optimization version of the problem is therefore obtained by adding the minimization of $\sum_{i \in N} n_i (\sum_{k \in \{0,\ldots,w-1\}} t_k y_{ik})$.

7 Experimental results

7.1 Generating benchmarks

All experimental results described in this paper were computed on a Dell Precision M2400 PC with an Intel(R) Core(TM) 2 Duo T9900 3.06Ghz, running Windows 7. The constraint programming model has been implemented on the top of IBM Ilog Scheduler 6.7. The MIP algorithm of IBM Ilog CPLEX 12.1 has been used to solve the mixed integer linear programming approaches. In aim to fairly compare the CP and MIP approaches and to obtain reproducible results, the parallelization option of Cplex has been disabled (Scheduler is not parallelized and parallelizing Cplex leads to important computational time variations when running on the same instances).

The methods have been successfully tested on a small number of industrial instances. However, it is well known that practical cases are often less of a challenge than benchmarks that have been generated specifically to test the limits of solvers. In this section we propose two schemes for generating instances. As shown in the following sections, these test instances would appear to be much harder to solve than instances encountered in practice.

7.1.1 Generating instances using the proof of complexity

In the first generation scheme, we propose drawing inspiration from the proof of the NP-completeness of the problem (see Section 3) with the 3-PARTITION problem for the non-preemptive version of the problem.

In this case, we have only one machine ($m = 1$). For given values of q and B, an instance has $n = 5q$ jobs with 1 operation, and the number of sorting slots is equal to $E = 3q$. The time horizon is fixed at $H = qB + 2q$. For $i \in \{1, \ldots, 3q\}$ we set $d_i = H$ and $n_i = 1$ and the processing time of the sole operation o_{i1} of job i is randomly generated in $]B/4, B/2[$. For $k \in \{0, \ldots, q-1\}$, we set $d_{3q+2k+1} = (B+2)k + 1$, $d_{3q+2k+2} = (B+2)k + 2$, $p_{3q+2k+1} = p_{3q+2k+2} = 1$ and $n_{3q+2k+1} = n_{3q+2k+2} = 3(q-k)$.

We generated two types of instance (feasible and unfeasible) using this scheme. In a first generation scheme (instances of this type will be referred to below as "3PART_F"), we make sure that there is a solution to the problem by randomly generating buckets of 3 operations, the sum of whose processing times is equal to B. Let lb and ub respectively be the smallest and largest integers in $]B/4, B/2[$. For each bucket (a, b, c) of processing times, a is generated from a uniform distribution

on interval $[lb, min(B - 2lb, ub)]$, then b is generated from a uniform distribution on interval $[lb, min(B - a - lb, ub)]$, and finally c is set to $B - a - b$.

In a second generation scheme (instances of this type will be referred to below as "3PART_U"), the processing times are randomly generated from a uniform distribution on interval $]B/4, B/2[$ (without applying a control as for instances of type 3PART_F). Then we keep only instances which are proved unfeasible by a solver.

For each type (3PART_F and 3PART_U), for each $q \in \{2, 4, 6\}$ (*i.e.* $n \in \{10, 20, 30\}$) and for each $B \in \{28, 52, 100, 140\}$ we generate 5 instances.

7.1.2 Generating more general instances

In this section we propose a more general generation scheme used both for the non-preemptive and the preemptive cases, and which can be either feasible or unfeasible (instances of this type will be referred to below as "GEN_F&U").

For each $n \in \{10, 20, 30, 40, 50\}$, $m \in \{1, 2, 5, 10\}$, we generated 1000 instances as follows.

For each instance, we randomly generate a number E of available sorting slots from the uniform distribution on interval $[10, 20]$. A maximum processing time P is randomly generated from the uniform distribution on interval $[10, 50]$ and the processing times of operations o_{ij} with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ are then randomly generated from a uniform distribution on interval $[0, P]$. We fix the time horizon H to $(1 + \gamma) \max_{j \in M} \{\sum_{i \in N} p_{ij}\}$, where γ is randomly generated from a continuous distribution on interval $[0, 0.25]$ and used to tighten or loosen the scheduling of operations on the bottleneck machine (*i.e.* the most loaded).

The completion time of the first job is fixed to H and its required number of sorting slots is generated from a decreasing triangular distribution on $[1, \frac{1}{2}E]$, *i.e.* the required number of sorting slots can take a value in $[1, \frac{1}{2}E]$, but the probability that any integer in this interval will be generated linearly decreases according to the value of that integer.

We then iteratively generate the completion time and the number of sorting slots required by the other jobs i as follows. Considering that jobs in $\{1, \ldots i - 1\}$ (jobs with already computed departure times) are re-indexed in non-decreasing order of departure time, let k be the first index such that $t_k = \max_{j \in \{1, \ldots m\}} p_{ij} + \sum_{l=1}^{k} p_{lj} \leq d_k$. Thus t_k corresponds to the earliest possible completion time that does not obviously lead to an unfeasible instance with respect to the departure times already generated (see for example the algorithm for solving $1||\sum U_i$ in [7]). A completion time d_i is then randomly generated from the increasing triangular distribution $[t_k, H]$ (higher values in the interval have a higher probability of being generated). Then, knowing that each job $l \in \{1, \ldots, i - 1\}$ requires sorting slots at least from time $d_l - \max_{j \in \{1, \ldots, m\}} p_{lj}$ to time d_l, we look at an upper bound of the maximum number

$$\bar{n_i} = E - \max_{t \in [d_i - \max_{j \in \{1, \ldots, m\}} p_{ij}, d_i]} \sum_{\{l | t \in [d_l - \max_{j \in \{1, \ldots, m\}} p_{lj}, d_l]\}} n_l$$

of sorting slots which could be still available from time $d_i - \max_{j \in \{1, \ldots, m\}} p_{ij}$ to time d_i (corresponding to the smallest interval of time over which job i will require its sorting slots). If $\bar{n_i} = 0$, another completion time d_i is randomly generated

as previously described. Otherwise, the number of sorting slots n_i is randomly generated from a decreasing triangular distribution on interval $[1, \max(1, \frac{1}{2}\bar{n_i})]$ (lower values in the interval have a higher probability of being generated).

7.2 Results for the non-preemptive case

In Table 1 results are shown for 3PART_F and 3PART_U instances tested with different methods proposed for the non-preemptive case. The results obtained with the mixed integer linear programming approach described in Section 5.1 are shown in column "MIP". To show the usefulness of the dominance rule described in Section 4, in column "ti_MIP" we also provide the results obtained by the same method but in a time-indexed version (that is without using the above dominance rule) where a variable x_{it} is introduced for each $t \in \{0, 1, \ldots, d_i\}$ meaning that job i uses sorting slots at time t if $x_{it} = 1$. The results obtained with the constraint-based scheduling approach described in Section 5.2 are provided in column "CP". To show the usefulness of dominance rule described in Proposition 6, we also give (in column "CP_no_DR") the results obtained by the same method but without using the dominance rule. For 3PART_F and 3PART_U instances, we give results for the search problem. For 3PART_F instances, we also provide results for the optimization problem.

Note that when generating instances 3PART_U with $n = 30$, either the instances were feasible (and then not retained for the set of instances 3PART_U), or our approaches were not able to prove the feasibility (or the infeasibility) of the instances even after several hours of computation. This indicates that there is a clear limit to the size of unfeasible instances that can be solved within the time limit, and it explains why we provide results only up to $n = 20$ for instances 3PART_U.

The average number of generated nodes ("*nodes*") and the average computation time in milliseconds ("*cpu*") over the 5 generated instances are shown for each type of instance and each solution context (for 3PART_F instances, for each method, and for each couple (q, B)). A time limit of 100 seconds was used. The number of instances over 5 which were not solved within the time limit is given in column "uns". For unsolved instances, the number of nodes and the computation times were not taken into account in the computation of the averages. Then "_" in a column means that no instance over the 5 ones was solved within the time limit.

As expected, dominance rules are effective in reducing computation time and enabling a greater number of instances to be solved within the time limit. We can see that the CP method appears to be more efficient than the MIP method for these instances. We also remark that unfeasible instances are much more difficult to solve than the feasible ones, due to the proof of infeasibility to be established for these instances. Thus, from $n = 20$ and $m = 1$ onwards, the MIP method was unable to solve any of the unfeasible instances. Note that from $n = 30$ onwards we also begin to find unsolved instances for method CP.

In Table 2, we provide results obtained using the GEN_UF instances for the MIP and CP methods, for both the search and the optimization problems. For each method, for each resolution context, and for each couple (n, m), the average number of generated nodes ("*nodes*") and the average computation time in milliseconds ("*cpu*") over the generated instances are shown (1000 in the case of the

Table 1 Test of methods and dominance rules in the non-preemptive case on 3PART_U and 3PART_F instances search and optimization problems.

3PART_U, search problem

			ti_MIP			MIP			CP_no_DR			CP		
q	n	B	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu
2	10	28	0	335	118	0	362	31	0	103	3	0	35	3
		52	0	293	165	0	302	28	0	101	9	0	34	3
		100	0	225	293	0	265	25	0	96	3	0	33	0
		140	0	198	389	0	247	25	0	95	3	0	33	3
4	20	28	5			5			5			0	23349	1154
		52	5	–	–	5	–	–	0	2260940	44117	0	4767	315
		100	5	–	–	5	–	–	0	1174739	22326	0	2742	184
		140	5	–	–	5			0	419399	11681	0	1409	56

3PART_F, search problem

			ti_MIP			MIP			CP_no_DR			CP		
q	n	B	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu
2	10	28	0	24	46	0	17	12	0	1	6	0	1	0
		52	0	14	81	0	26	9	0	3	3	0	3	3
		100	0	28	137	0	15	6	0	5	0	0	4	0
		140	0	33	196	0	28	9	0	2	0	0	1	0
4	20	28	0	1861	20298	0	7967	2957	0	37592	727	0	302	12
		52	3	2342	22425	0	63116	22189	0	537	12	0	45	0
		100	3	1202	15319	1	16050	8213	0	16892	321	0	189	3
		140	2	411	38022	2	6438	13525	0	38673	742	0	437	53
6	30	28	4	529	10498	4	203	234	1	190480	5815	1	933	43
		52	5			5			1	319	43	0	147328	4617
		100	5	–	–	5	–	–	1	65	19	0	29584	954
		140	5	–	–	5	–	–	3	5200	640	0	21674	1351

3PART_F, optimization problem

			ti_MIP			MIP			CP_no_DR			CP		
q	n	B	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu	uns	nodes	cpu
2	10	28	0	46	99	0	109	15	0	121	3	0	41	0
		52	0	42	143	0	10	9	0	113	0	0	39	0
		100	0	84	549	0	69	18	0	112	0	0	39	3
		140	0	112	1198	0	68	15	0	105	3	0	36	0
4	20	28	5			0	16988	11606	5			0	67163	3981
		52	5	–	–	0	836	2003	5	–	–	0	18435	798
		100	5	–	–	2	58034	42765	3	2607247	48852	0	5572	393
		140	5	–	–	1	16983	22889	1	2641537	59198	0	5768	365
6	30	28	5	–	–	4	25742	52260	5			5		
		52	5	–	–	5			5	–	–	5	–	–
		100	5	–	–	5	–	–	5	–	–	3	1575292	50489
		140	5	–	–	5	–	–	5	–	–	3	1130425	61612

search problem). Of course, whatever the method, the test on the optimization problem is only performed on the instances which have been proved to be feasible. Thus, column "nb" indicates the number of instances concerned. Once again, a time limit of 100 seconds was used. The percentage of instances which were not solved within the time limit is given in column "%uns". For unsolved instances, the number of nodes and the computational times were not taken into account in the computation of the averages. It will be remarked that the difficulty of the problem increases as the number n of jobs and the number m of machines increase. It is for this reason that we did not perform the tests for higher parameters m or n when more than 50% of the instances could not be solved within the time limit. In this case "_" in a column means that no test was performed for these parameters.

These results confirm that for the search problem the CP method outperforms the MIP method. However, this is not the case for the optimization problem, where MIP performs much better, as shown by the results for $n = 10$ and $m = 5$.

Table 2 Test of methods MIP and CP (non-preemptive case) on GEN_F&U instances, search and optimization problems.

		MIP							CP						
		search			optimization				search			optimization			
n	m	**%uns**	**nodes**	**cpu**	**nb**	**%uns**	**nodes**	**cpu**	**%uns**	**nodes**	**cpu**	**nb**	**%uns**	**nodes**	**cpu**
10	1	0	1085	120	707	0	1200	197	0	0.95	0	707	0	2940	32
	2	0.1	3430	477	698	0.72	4233	856	0	22	1	699	0	199104	2630
	5	3	10350	2611	656	9	8636	2709	2	19040	453	641	77	194517	4959
	10	11	9301	4251	646	16	7646	3335	10	9320	378	_	_	_	_
20	1	41	20221	8518	348	51	14186	23616	0	126	4	455	63	1616711	25090
	2	50	11560	10006	_	_	_	_	0.3	19545	603	_	_	_	_
	5	54	3222	7079	_	_	_	_	22	47785	2412	_	_	_	_
	10	_	_	_	_	_	_	_	47	31122	2721	_	_	_	_

Note that some instances with $n = 10$ and $m = 5$ remain unsolved for the search problem for both methods.

In Table 3 we compare the performance of the MIP and CP methods. For each method, we indicate the percentage of times where the method outperforms its rival (column "%best"), *i.e.* the percentage of times where the method found a solution either with a shorter computation time or where the other method failed to find a solution at all. We also indicate this proportion with respect to unfeasible instances ("%unf") and to feasible instances ("%feas"). We report the average difference in computation time (in milliseconds) when one method is better than the other ("cpu_dist"). Where the other method failed to find a solution a difference of 100s is taken. Column "eq" shows the percentage of instances for which both methods are equivalent. Finally "mix" shows what the results obtained would be, in terms of the percentage of unsolved instances ("%uns") and of the average computation time in milliseconds ("cpu"), if for each instance we were able to choose the best method.

Table 3 Comparing MIP and CP methods (non-preemptive case) on GEN_F&U instances, search problem.

		MIP				CP					mix	
n	m	**%best**	**%unf**	**%feas**	**cpu dist**	**%best**	**%unf**	**%feas**	**cpu dist**	**%eq**	**%uns**	**cpu**
10	1	1	0	1	15	91	89	92	131	8	0	0.18
	2	0	0	0	70	95	89	97	606	5	0	1
	5	6	2	8	38376	90	85	92	5835	5	0	19
	10	12	7	15	81202	84	81	85	17250	4	0.4	235
20	1	0	0	0	437	98	97	100	46917	2	0	3
	2	2	3	1	13739	92	88	99	58691	5	0.2	400
	5	16	22	14	47467	62	65	86	66877	8	14	2778

These results show that while CP is on average better than MIP, MIP dominates CP on a non-negligible set of instances, especially when the number of machines m is increased. For example, for $n = 10$ and $m = 10$, although neither method is able to solve more than 10% of the instances (see previous table), MIP is better than CP for 12% of the instances, taking on average 81 seconds less computational time than method CP. Then, taking the best method for each of these instances, only 0.4% of the instances still remain unsolved and the computation time is drastically reduced. This illustrates a certain complementarity between the two methods. From the results it is not possible to draw any conclusions regarding the best method and the feasibility or infeasibility of the instances.

7.3 Results for the preemptive case

For the preemptive case, Tables 4 and 5 give the same measures for the flow-based mixed interger linear programming (FLOW) and the multiperiod (MULTIP) approaches methods as given above for the non-preemptive case (see tables 2 and 3).

Table 4 Test of methods FLOW and MULTIP (preemptive case) on GEN_F&U instances, search and optimization problems.

		FLOW							MULTIP						
		search			optimization				search			optimization			
n	m	**%uns**	**nodes**	**cpu**	**nb**	**%uns**	**nodes**	**cpu**	**%uns**	**nodes**	**cpu**	**nb**	**%uns**	**nodes**	**cpu**
10	1	0	0.12	3	750	0	28	21	0	0.00	3	750	0	2	8
	2	0	0.11	4	730	0	21	23	0	0.00	5	730	0	1	9
	5	0	0.19	9	683	0	19	40	0	0.00	9	683	0	1	16
	10	0	0.21	19	716	0	19	76	0	0.04	17	716	0	1	29
20	1	0	20	84	558	0	1172	6614	0	0.40	48	558	0	225	339
	2	0	56	419	474	1	1389	12317	0	0.40	75	474	0	366	759
	5	0.3	74	1790	402	10	1256	24843	0	3	179	402	0	502	1533
	10	2	77	4725	326	38	978	37469	0.1	4	424	326	0.31	557	3596
30	1	3	235	4411	327	83	2181	44586	0	2	232	327	2	2174	6998
	2	8	229	9287					0	4	399	195	8	2488	11519
	5	26	80	12253	–	–	–	–	0	7	1147	117	29	2149	22029
	10	35	16	8146	–	–	–	–	0.2	3	2637	86	42	948	33311
40	1	21	268	13270					0	3	602	131	21	3163	24774
	2	28	82	10916	–	–	–	–	0.1	3	1046	68	54	2109	32370
	5	41	7	5227	–	–	–	–	0	2	3577				
	10	41	1	6516	–	–	–	–	0.6	1	11449	–	–	–	–
50	1	40	115	12395					0	2	1270	42	81	1630	35564
	2	45	16	6623	–	–	–	–	0	3	2853				
	5	42	0.30	5566	–	–	–	–	0.1	0.38	16007	–	–	–	–
	10	42	0.00	16380	–	–	–	–	23	0.00	40863	–	–	–	–

For the preemptive case, it would appear that the MULTIP method is better than the FLOW method. This is because this method uses fewer constraints with respect to the feasibility of the associated scheduling problem. We note that from $n = 50$ for the search problem and from $n = 30$ for the optimization problem, the problem becomes very hard to solve within the time limit.

Table 5 (with the same colums as Table 3) reveals a complementarity between the two methods. When trying to determine the best method for each instance a certain balance can be observed. Note, however, that in cases where FLOW is better, the computation time for MULTIP is only marginally greater. The difference tends to be more acute for higher values of m and n. On the other hand, when MULTIP is better, the difference in computation time between the two methods tends to be large. Consequently, using the best method for each instance yields only slightly better results than using MULTIP exclusively. This enables us to solve only a small number of additional instances within the time limit, and reduces the computation time in only a limited number of instances and by a ratio of not more than 2. Note also that, for higher values of m and n, the FLOW method is of interest only for unfeasible instances.

Table 5 Comparing the FLOW and MULTIP methods (preemptive case) on GEN_F&U instances, search problem.

		FLOW				MULTIP					mix	
n	m	**%best**	**%unf**	**%feas**	**cpu dist**	**%best**	**%unf**	**%feas**	**cpu dist**	**%eq**	**%uns**	**cpu**
10	1	20	14	22	15	20	16	21	15	60	0	0.02
	2	30	20	34	15	24	16	27	15	46	0	0.18
	5	31	38	28	15	31	27	33	16	38	0	4
	10	29	45	23	18	33	26	35	20	38	0	12
20	1	59	64	55	32	28	24	31	196	13	0	30
	2	57	58	55	46	32	28	35	1175	12	0	49
	5	57	55	58	93	40	42	37	4895	4	0	126
	10	51	53	48	219	47	46	49	13782	2	0.1	312
30	1	53	51	57	137	45	47	41	15085	1	0	159
	2	47	47	48	232	51	51	51	31158	1	0	289
	5	42	43	35	634	58	57	65	59056	0	0	881
	10	43	45	21	1625	57	55	79	67482	0	0.2	1945
40	1	42	44	29	386	58	56	71	53863	0	0	440
	2	46	47	29	673	54	53	71	64818	0	0.10	739
	5	46	47	20	2554	54	53	80	76856	0	0	2405
	10	48	49	11	7355	52	51	89	71255	0	0.6	7920
50	1	39	41	7	874	61	59	93	76165	0	0	927
	2	43	43	6	2133	57	57	94	81033	0	0	1942
	5	50	50	20	12267	50	50	80	70912	0	0.1	9867
	10	49	58	_	35687	36	42	_	41595	0	15	25580

8 Conclusion and extensions

In this paper we have formalized and studied a new combinatorial problem. While the study of this problem has a real application in warehouse management, its combinatorial structure is interesting from a theoretical point of view. We have shown the NP-completeness of the problem and we have proposed several methods for solving instances of the problem in both the preemptive and non-preemptive cases. Not only we can always determine which of the methods is better on average for the case in question, but we have also shown a certain complementarity between them. Moreover, although these methods are effective for instances encountered in practice, we have shown in our experimental study that the solvers can be challenged by theoretical instances of small dimensions.

The technical part of the paper describes some initial approaches to solving the problems. As far as future research is concerned, these methods might be improved by working on the branching schemes of the methods and by attempting to find efficient lower bounds.

There are also several ways in which this research work might be extended. First, some other objective functions could be examined. For example, seeking a solution which balances between the different trucks the extra time available for using sorting slots might improve the robustness of solutions. Secondly, important questions arise when the instance in hand is not feasible. How can decision makers be assisted in making the right choices? Several possibilities might be investigated. For example, minimizing the (weighted) number of late jobs might help to determine which trucks might be canceled (ideally, as few as possible). Similarly, minimizing the total (weighted) tardiness can indicate which orders need to be canceled (again, as few as possible) when all truck departures are maintained.

While the approaches have been described in the context of solving a particular warehouse management problem, they can also be used to solve other optimization problems in which any particular resource is required during the processing of a

set of operations on other resources, even if this processing is intermittent. For instance, a special case of multi-skill project management can fall in this category. Each project (job) is composed of tasks (operations) necessiting separated skills and resources (preparation shops). In top of this, one should affect some special resources (for instance, project manager, commercial agent, hotline number) from the start of the project until its delivery (a given deadline).

References

1. Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling, applying constraint programming to scheduling problems*, volume 39 of *International Series in Operations Research and Management Science*. Kluwer, 2001.
2. P. Brucker. *Scheduling Algorithms*. Springer Lehrbuch, 1995.
3. G. Cormier and E.A. Gunn. A review of warehouse models. *European Journal of Operational Research*, 58:3–13, 1992.
4. R. De Koster, T. Le-Duc, and K.J. Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182:481–501, 2007.
5. T. Ganesharajah, N.G. Hall, C. Sriskandarajah Design and operational issues in AGV-served manufacturing systems *Annals of Operations Research*, 76:109–154, 1998.
6. M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
7. J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.
8. B. Rouwenhorst, V. Reuter, V. Stockrahm, G.J. Van Houtum, R.J. Mantel, and W.H.M. Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122:515–533, 2000.
9. J.P. Van den Berg. A literature survey on planning and control of warehousing systems. *IIE Transactions*, 31:751–762, 1999.