

STORAGE PROBLEM IN A SHAMPOO MAKING SYSTEM

R.BELAID, V.T'KINDT, C.ESSWEIN

Université Francois Rabelais Tours
Laboratoire d'Informatique
64 avenue Jean Portalis, 372300 Tours, FRANCE
rabah.belaid@univ-tours.fr,
vincent.tkindt@univ-tours.fr,
carl.esswein@univ-tours.fr.

ABSTRACT: *In this paper we address the problem of planning a temporary storage area in a shampoo making system. Everyday, the packing service of the company, which gathers several packing lines, transmits plans of orders to produce (also called batches) to the making service. Each plan corresponds to the work that will be performed by one packing line for the next production horizon. A batch, which consists of a volume of 12 tons of the same shampoo juice is produced by the making service and transferred to a tank before being sent to the corresponding packing line. There are several families of shampoo and a tank must be cleaned when the next batch to store have a different shampoo family than the last stored batch. In this temporary storage problem the goal is to minimize the number of tank cleanings and the maximum lateness of batches. Two different cases of this problem are considered: in the first the lateness is allowed, and in the second no lateness is allowed. Complexity of these two cases is studied and three solution algorithms are proposed to solve the second case.*

KEYWORDS: *Fixed Interval Scheduling, Assignment problem, Ant Colony Optimization.*

1 Introduction

In this paper we address the problem of planning the temporary storage in a shampoo making system. This problem is tackled in the context of a collaboration with an international company producing healthcare products. Everyday, the packing service of the company, which gathers several packing lines, transmits plans of orders to produce (also called *batches*) to the making service which is composed of several *making units*. A batch is produced by the making service and transferred to a storage facility before being packed in accordance with the packing line plan.

A batch is a fixed and indivisible quantity (12 tons) of the same shampoo product (*Family*). Indivisible means that each batch has to be entirely stored in one tank, and entirely packed by one packing line. A storage facility contains many tanks with different capacities (12, 20 or 24 tons) and is directly connected to a subset of packing lines. This implies that a batch can only be stored by a given set of tanks depending on the packing line on which it will be processed. There are several families of shampoo and each family has its own chemical and physical characteristics. Thus, even if the tank capacity is sufficient, two batches with different shampoo families cannot be stored in it at once. So, a tank must be cleaned when the next batch to store have a different shampoo family than the last stored batch. In this temporary storage

problem the goal is to minimize the number of tank cleanings and the maximum lateness of batches. The lateness is defined on the basis of due dates which correspond to the date at which packing lines finish to use the batches.

In this work, we first provide a model of a tank as a two machine flowshop with a limited buffer capacity, which is later on used to establish properties of the problem. Next, we consider the special case where no lateness is allowed and provide solution algorithms. Under this assumption, the problem can be seen a special case of the fixed interval scheduling problem which complexity is open.

The problem under consideration is at the crossroad of three kind of scheduling problems: scheduling with limited buffer capacities, scheduling with setup times or costs (see Allahverdi *et. al.* (2008) for a survey) and fixed interval scheduling (see Kovalyov *et. al.* (2007) and Kolen *et. al.* (2007) for a presentation of the basic problems and results). Regarding scheduling with limited buffer capacities, we were more interested on complexity issues for flowshop problems. Papadimitriou and Kanellakis (1980) show that the decision problem of the 2-machine problem with makespan is NP-complete. Gupta (1986) shows that flowshop problems with sequence dependent setup times or costs are NP-Hard whatever the number of machines, the criteria and the buffer ca-

capacity. But no complexity results is known about the particular problem addressed in this paper.

In the remainder, section 2 presents the problem statement. Solution algorithms to solve this problem are outlined in section 3, whilst section 4 is devoted to the computational evaluation of these algorithms. Finally, some conclusions are given in section 5.

2 Problem statement

In this temporary storage problem we have to assign n batches on L tanks. Let us first define the problem, noted P_ϵ , on a single tank which is modeled as a two-machine flowshop with a limited buffer capacity. This particular problem is the one to solve for each tank when the batches have been assigned to the tanks. The first machine processes the transfer operations, denoted by $o_{i,1}$, $i = 1..n$, from the making unit to the tank. The buffer capacity, denoted by b , is equal to the tank capacity and we consider that the batch starts to be in the buffer at time $t_{i,1}$, which refers to the starting time of the operation $o_{i,1}$. Processing times on the first machine are all equal to p_1 (loading time). The second machine processes the emptying operations, $o_{i,2}$, which starts at time $t_{i,2}$ and require a processing time $p_{i,2}$. According to the packing plan, this second operation have to starts after a release date r_i , ($t_{i,2} \geq r_i$), and should finish at the due date defined by $d_i = r_i + p_{i,2}$. According to the industrial problem, a bunch of additional constraints have to be answered and are listed below. The preemption is not allowed, and a minimal time lag between the starting times $t_{i,1}$ and $t_{i,2}$ of the two operations must be greater than or equal to q_i , ($q_i > 0$). Notice, that the two operations can overlap if $q_i < p_1$. This time lag may correspond to the time required to load into the tank a minimal quantity a product before enabling the packing line to start processing. Sometimes, it corresponds to the time required to make product's foam disappearing before the corresponding batch can be usable by the packing line. Let C_i be the completion time of operation $o_{i,2}$. A second minimal time lag, noted δ , has to be answered between two batches i and j if they are planned on two different packing lines x, y : thus we have $C_i + \delta \leq t_{j,1}$. This constraint is imposed by the making service to ensure that if the packing line x is late then it will not make the packing line y late. A cleaning operation between two batches is modeled by a family dependent setup noted S_f which is a constant. These setups block the two machines and the buffer when they are performed, in order to model the unavailability of the tank.

The sequencing rules of this problem change depending on the (i) values of the buffer capacity, (ii) the families of the batches and their (iii) intended packing lines. Let j be the last batch scheduled on a tank

ℓ , i be the batch preceding j in this schedule, and k the next batch to schedule; $i, j, k = 1..n$.

Table 1 shows the rules corresponding to the different cases. The value $\Delta_{j,2}$ in the third sequencing rule correspond to the time needed to have a free capacity of $12t$ on the tank ℓ wich capacity is $20t$.

Considered case	Sequencing rule
family of $j \neq$ family of k $b_\ell \in \{12t, 20t, 24t\}$	$C_j + S_f + \delta \leq t_{k,1}$
family of $j =$ family of k $b_\ell = 12t$	$C_j + \delta \leq t_{k,1}$
family of $j =$ family of k $b_\ell = 20t$	$t_{j,2} + \Delta_{j,2} + \delta \leq t_{k,1}$
i, j, k have the same families $b_\ell = 24t$ $\delta = 0$	$(C_i \leq t_{j,2}) \wedge (C_j \leq t_{k,2})$ and, $\max(C_i, t_{j,1} + p_1) \leq t_{k,1}$
family of $j =$ family of k $b_\ell = 24t$ $\delta > 0$	$C_j + \delta \leq t_{k,1}$

Table 1: Sequencing rules

The figure 1 shows a Gantt diagram corresponding to a sequence, in a tank of 20 tons, of three batches intended for the same packing line (*i.e.*: $\delta = 0$). In this example, batches 1 and 2 have the same family and therefore can share the buffer while its capacity is sufficient. Loading operation of batch 2 starts at $\Delta_{1,2}$ units of time after the start of the emptying operation of the batch 1 as specified in the third rule of Table 1. Batch 2 and 3 have different families, so a setup time S_f have to be done before the start of batch 3 loading operation as specified by the first rule of Table 1. As we can see on the figure, the setup S_f blocks both machines and the buffer. The batch 3 completes its execution after its due date, with a lateness $L_3 = C_3 - d_3$. We have to minimize the number of setups and the maximum lateness and more precisely we are imposed to minimize the number of setups under the constraint that the maximum lateness is bounded by a given threshold. According to the notation of multicriteria scheduling problems introduced by T'kindt and Billaut (2006), the problem P_ϵ can be referred to as $F2|S_f = S, p_1 = p, r_i, q_i, \delta, b|\epsilon(\sum S/L_{max})$.

Lemma 2.1.

The $F2|S_f = S, p_1 = p, r_i, q_i, \delta, b|\epsilon(\sum S/L_{max})$ problem is NP-Hard.

Proof. Let us first start with the $1|S_f = S, r_i|C_{max}$ problem which is strongly NP-Hard (Yuan *et. al.* 2006). It directly follows that the associated decision problem $1|S_f = S, r_i, \tilde{d}_i|-$ is NP-Complete since there is a pseudo-polynomial number of possible C_{max} values. This result, implies that the $1|S_f = S, r_i, \tilde{d}_i|\sum S$ problem is also NP-Hard.

The last problem is equivalent to the $1|S_f = S, r_i|\epsilon(\sum S/L_{max})$ problem, which is a subproblem

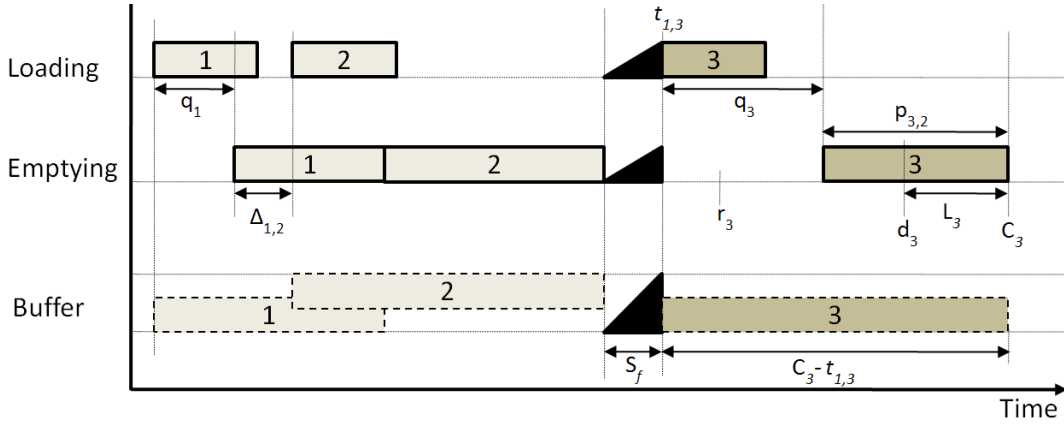


Figure 1: Example of a schedule on a tank

of P_ϵ ($p_1 = 0, \delta = 0, q_i = 0, b = 12t$). Consequently, problem P_ϵ is NP-Hard. \square

Lemma 2.1 also implies that the general problem with L tanks, referred to as P_ϵ^L , is also NP-Hard.

Let us now consider the case where lateness is not allowed: more precisely the making service imposes that $L_{max} = 0$. Here an emptying operation $o_{i,2}$ has to start exactly at time r_i and we assume that there are enough tanks. We denote this problem by P_0^L and distinguish between two cases according to the tank capacities b_ℓ .

If b_ℓ is lower than twice the batch size, we can right timeshift the first operation of each batch. This leads to define a fixed processing interval for each batch whatever the tank on which it is scheduled. We obtain a partially ordered set of time intervals $I_k = [r_k - q_k; r_k + p_{k,2}]$ such that:

$$I_i \cap I_j = \emptyset \implies (i \rightarrow j) \vee (j \rightarrow i), \forall i, j = 1..n,$$

with $i \rightarrow j$ meaning that job i precedes job j . Henceforth, by finding a min-cost perfect matching in a bipartite graph, we can solve the P_0^L problem. We now details how this graph is constructed.

Let $O_\ell, \ell = 1..L$, be the fictitious initial intervals and $F_\ell, \ell = 1..L$, be fictitious final intervals. These fictitious intervals are used to model the initial and terminal states of the tanks. The fictitious initial intervals are also characterized by a shampoo family which the one of the last stored batch in the previous planning horizon. We have:

$$\begin{aligned} O_\ell < I_i; \forall i = 1..n, \forall \ell = 1..L, \\ \text{and,} \\ I_i < F_\ell; \forall i = 1..n, \forall \ell = 1..L. \end{aligned}$$

Besides, all the n intervals related to the batches and characterized by their family.

Let $G = (V_{pred}, V_{suc}, A)$ be a balanced bipartite graph with V_{pred} and V_{suc} the two sets of vertices and A the set of arcs. V_{pred} contains all the intervals which can be predecessors of other intervals, so it contains the L fictitious initial intervals and the n batch intervals. V_{suc} contains all the intervals which can be successors of other intervals and thus the L fictitious final intervals and the n batch intervals. So, $|V_{suc}| = |V_{pred}| = L + n$.

Let $i \in V_{pred}$, and $j \in V_{suc}$ be two vertices. According to the partial order, we connect an arc $a_{i,j}$ from a vertex i to a vertex j if $i \rightarrow j$ (this includes fictitious vertices). Arc $a_{i,j}$ has a cost S if i and j have different families and 0 otherwise. We will show that from a min-cost perfect matching on the graph G we can build an optimal solution for problem P_0^L . From an algorithmic side, under the assumption of the equivalence between such a matching and an optimal solution for P_0^L , we can proceed as follows to compute the latter. Starting from a fictitious initial interval O_ℓ we select the arcs of the matching which enable to create a path to a fictitious terminal interval $F_{\ell'}$. Thus, we obtain a set of L paths which correspond to L batch sequences on the tanks, such that the total cost of these paths is the total setup costs.

Lemma 2.2.

The P_0^L problem can be solved in polynomial time whenever $b_\ell \in \{12t, 20t\}, \forall \ell = 1..L$.

Proof. Let us first show that a min-cost perfect matching on the bipartite graph G corresponds to a solution for the P_0^L problem. Let M be a min-cost perfect matching for G , and Sol be the corresponding solution of P_0^L problem built according to the above procedure.

Since M is a perfect matching, all vertices are saturated and all arcs are independents. This implies that all the batch intervals are included in the L paths and that each batch interval is assigned to only one path

as a tail of an arc (a successor) and as the head of the next arc (a predecessor).

Suppose now that G has no perfect matching M . Thus, there are at least two non saturated vertices (since $|V_{suc}|=|V_{pred}|$). Let us show that these two remaining vertices do not correspond to fictitious intervals. According to the definition of the fictitious intervals we have:

$$\forall x \in V_{suc} : O_\ell < I_x, \text{ and } a_{\ell,x} \in A; \ell = 1..L.$$

$$\forall y \in V_{pred} : I_y < F_\ell, \text{ and } a_{y,\ell} \in A; \ell = 1..L.$$

This means that a fictitious interval cannot be among these two remaining vertices since there is an independent arc between every fictitious interval with all the other vertices of G . Thus the remaining vertices are batch intervals and this implies that there is no feasible solution for the P_0^L problem.

Let us now show that the solution Sol built from the min-cost perfect matching M is optimal. Suppose that Sol is not optimal and let $permu(i, j)$ be the permutation of two batches i, j which reduces the total number of setups. Let $pr(x)$, $sc(x)$ and $fm(x)$ be, respectively, the predecessor, successor and the family of a batch x , $x = 1..n$, in Sol . We also denote by $c(a_{x,y})$ the cost on the arc $a_{x,y}$, $\forall x \in V_{pred}, y \in V_{suc}$. If $fm(i) = fm(j)$ no improvement is possible. So we consider the case where $fm(i) \neq fm(j)$. In this case, an improvement can appear at least if :

$$(fm(pr(j)) = fm(i)) \text{ or } (fm(i) = fm(sc(j))) \quad (1)$$

or

$$fm(pr(j)) = fm(sc(j)) = fm(i) \quad (2)$$

Let us consider the second case in which $permu(i, j)$ improves the solution Sol by $2 \times S$. Here, if we consider the sequences on the tanks as paths, arcs $a_{i,sc(j)}$, $a_{pr(j),i}$, $a_{pr(i),j}$, $a_{i,sc(j)}$ are added whilst arcs $a_{i,sc(i)}$, $a_{pr(i),i}$, $a_{pr(j),j}$, $a_{j,sc(j)}$ are removed.

Let M' be a set of arcs such that :

$$M' = M - \{a_{i,sc(i)}, a_{pr(i),i}, a_{pr(j),j}, a_{j,sc(j)}\} + \{a_{i,sc(j)}, a_{pr(j),i}, a_{pr(i),j}, a_{i,sc(j)}\}$$

It is obvious that M' is a perfect matching since the added arcs are independents and saturate only vertices which were not saturated after the removal of arcs $a_{i,sc(i)}$, $a_{pr(i),i}$, $a_{pr(j),j}$, $a_{j,sc(j)}$ from M . So, according to the equation (2) we have:

$$\begin{aligned} & c(a_{i,sc(j)}) + c(a_{pr(j),i}) < c(a_{i,sc(i)}) + c(a_{pr(i),i}). \\ \Rightarrow & c(a_{i,sc(j)}) + c(a_{pr(j),i}) + c(a_{pr(i),j}) + c(a_{i,sc(j)}) < \\ & c(a_{i,sc(i)}) + c(a_{pr(i),i}) + c(a_{pr(j),j}) + c(a_{pr(i),j}). \\ \Rightarrow & c(M) + c(a_{i,sc(j)}) + c(a_{pr(j),i}) + c(a_{pr(i),j}) + c(a_{i,sc(j)}) < \\ & c(M) + c(a_{i,sc(i)}) + c(a_{pr(i),i}) + c(a_{pr(j),j}) + c(a_{pr(i),j}). \\ \Rightarrow & c(M) + c(a_{i,sc(j)}) + c(a_{pr(j),i}) + c(a_{pr(i),j}) + c(a_{i,sc(j)}) - \\ & c(a_{i,sc(i)}) + c(a_{pr(i),i}) + c(a_{pr(j),j}) + c(a_{pr(i),j}) < c(M). \\ \Rightarrow & c(M') < c(M). \end{aligned}$$

$\Rightarrow M$ is not a min-cost perfect matching of G which is a contradiction.

Henceforth, the P_0^L problem can be solved by solving a min-cost perfect matching in $O(n^3)$ time (or equivalently by solving an assignment problem). \square

The P_0^L problem for which there exists at least one tank ℓ with $b_\ell = 24t$, can be modeled as a multi-index assignment problem which is NP-Hard (Burkard *et. al.* 2009). However, we can prove the following straightforward result which can be applied when the assignment of batches to tanks is known.

Lemma 2.3.

The $F2|S_f = S, p_1 = p, t_{i,2} = r_i, q_i, \delta, b| \epsilon(\sum S/L_{max})$ problem can be solved in polynomial time by using the Earliest Release Time (ERT) rule. If the ERT rule fails to find a feasible solution, then the problem is infeasible.

3 Solution algorithms

We present in this section the solution algorithms developed to tackle the particular case of P_0^L when the tank capacities are equal to 24 tons.

The first algorithm is a greedy algorithm coupled with a local search based on the 2-opt neighborhood operator. The main lines of this algorithm, which exploits Lemma 2.3, are as follows: A list L^i of batches, sorted by increasing order of release dates r_i , is built and each batch in this list is assigned to the first available tank that does not require cleaning, if it exists. Otherwise, the batch is assigned to the first available tank that will inevitably be cleaned given the families of the L next batches in the list: a tank will be cleaned if the lastly assigned batch is of a family that does not appear in the next L batches in the list. If the two above situations do not happen, then the current batch is assigned to the first available tank. The calculated solution is repaired, if it is infeasible, and finally the local search is applied on it.

The availability of a tank can be defined as follows: let i be the last batch scheduled on a tank ℓ , and k be the batch preceding i in this schedule; $i, k = 1..n$. The availability of the tank ℓ for the next batch to schedule j is determined according to the followings rules:

If i and j have different families Then:

• ℓ is available $\Leftrightarrow C_i + S + \delta \leq r_j - q_j$.

Else (i and j have the same family):

If $b_\ell = 12$ t :

• ℓ is available $\Leftrightarrow C_i + \delta \leq r_j - q_j$.

If $b_\ell = 20$ t :

• ℓ is available $\Leftrightarrow t_{i,2} + \Delta_{i,2} + \delta \leq r_j - q_j$.

where $\Delta_{i,2}$ is the time needed to have a free capacity of $12t$ on the tank ℓ .

If ($b_\ell = 24$ t) and ($\delta = 0$) :

- ℓ is available $\Leftrightarrow \max(C_k, t_{i,1} + p_1) \leq r_j - q_j$.
- If ($b_\ell = 24$ t) and ($\delta > 0$) :
- ℓ is available $\Leftrightarrow C_i + \delta \leq r_j - q_j$.

The framework of this greedy algorithm is given in Algorithm 1.

Algorithm 1: Greedy algorithm

L^i : list of batches;
 T_ℓ : first available tank ;
 T_ℓ^f : first available tank which does not require a cleaning;
 T_ℓ^{list} : first available tank which does not require a cleaning taking account of the L next batches in the list;
 n : Number of batches;
 L : Number of tanks;
 B : Current batch;
 sol : Solution;
 $repar()$: Repairing method;
 $local_s()$: Local search method;
begin
 while $L^i \neq \emptyset$ **do**
 $B = L^i[0]$;
 $L^i = L^i - B$;
 if T_ℓ^f exists **then**
 assign B on T_ℓ^f ;
 else
 if (T_ℓ^{list} exists) **then**
 assign B on T_ℓ^{list} ;
 else
 assign B on T_ℓ ;
 end
 end
 end
 if (sol is infeasible) **then**
 $repar(sol)$;
 end
 $sol = local_s(sol)$;
 return(sol);
end

The second proposed solution algorithm is based on an Ant Colony Optimization (ACO) approach which uses features of the simulated annealing search as initially proposed by T'kindt *et. al.* (2002) and which gave very good results on a 2-machine flow-shop scheduling problem. As in classical ACO heuristics (see Dorigo and Stutzle (2004) for a comprehensive survey of this metaheuristic), ants try to build solutions with a constructive procedure and communicate by means of a shared memory called the pheromone matrix and noted τ . Here an element $\tau_{i,j}$, $\tau_{i,j} \in [\tau_{min}, \tau_{max}]$, is the probability of having batch j as a successor of a batch i on the same tank in a good solution. The assignment of a batch to a tank

at each step of the constructive procedure is done according to either an intensification mode or a diversification mode. In the intensification mode, an ant chooses among all batches j which can be assigned to a tank ℓ , the batch with highest value of $\tau_{i,j}$ with i the batch lastly scheduled on tank ℓ . In the diversification mode, the choice of the assignable batch is made by using a classic wheel process. Notice that we apply the principles of simulated annealing to select between these two modes: this leads to enforce diversification at the first iterations and intensification at the last iterations. For each global iteration of the ACO heuristic, if no feasible solution is built yet, we apply a best fit rule repairing method on $\rho\%$ of the best incomplete solutions. After that, a local search is applied on the best solution built by ants or later on repaired. Finally, the pheromone matrix is updated with the classic evaporation and enforcement processes of ACO heuristics.

The stopping criterion of this heuristic is the maximum number of iterations. The framework of this ACO algorithm is given in Algorithm 2.

The third proposed algorithm exploits the result stated in Lemma 2.2 and is based on the observation that in the industrial instances the general problem can be decomposed into subproblems, some of them involving a subset of batches to be assigned on set of $12t$ or $20t$ tanks. First, for all batches we compute the time intervals I_i and solve the assignment problem described in Lemma 2.2. Consequently, for some $24t$ tanks this may results into infeasible assignments: some batches i , called *infeasible batches*, are assigned to a tank but are not available at there release date r_i . These infeasible assignments are next repaired. The repairing method is based on a dynamic priority rule. We remove all infeasible batches from the solution found by the assignment algorithm, and we assign a priority to each of them. after that we try to assign each infeasible batch to tank in order to meet the problem constraints. To do so, we define two insertion operators. The first one, denoted by *1-mov*, inserts directly an infeasible batch i in the sequence of a tank ℓ if possible. The second one, denoted by *2-mov*, inserts an infeasible batch i into the sequence of a tank ℓ after removal of an already scheduled batch j . The latter must be necessarily inserted into the sequence of another tank before the insertion of batch i is realized. Clearly, before removal of batch j on tank ℓ the insertion of batch i was not possible due to the overlap of time intervals I_i and I_j .

The priority of an infeasible batch i reflects the facility to insert it into a sequence of tank. It is defined as the number of possible insertions using the two operators defined above. So, the batch which has the less chance of being inserted (the lowest priority value) will be inserted first by the repairing method. If multiple insertions are possible, the algorithm chooses the

Algorithm 2: S.A.C.O

```

τ: Pheromone matrix;
F : Number of ants;
L: Number of tanks;
n: Number of batches;
ITERMAX: Maximum iteration number;
bestit: Best solution found at the current iteration;
bestsol: Best solution;
funproba(): Function returning the construction
mode probability;
reparρ(): Repairing method;
locals(): Local search;
evap(τ): Evaporation function with quantity τ;
enfor(bestit): Enforcement function on bestit;
begin
  for (iter from 1 to ITERMAX) do
    σ = funproba(iter);
    for (f from 1 to F) do
      for (i from 1 to n) do
        Generate random number 0 ≤ P ≤ 1;
        if (σ > P) then
          Schedule an unscheduled batch
          using the diversification mode;
        else
          Schedule an unscheduled batch
          using the intensification mode;
        end
      end
    end
    if (No feasible solution is built) then
      | reparρ();
    end
    bestit = locals(bestit);
    evap(τ);
    enfor(bestit);
    if (bestit < bestsol) then
      | bestsol = bestit
    end
  end
  return(bestsol);
end

```

one that less deteriorates the objective function. The priorities of the remaining batches is updated after an insertion is realized and the algorithm stops when all the infeasible batches are inserted or if a batch cannot be inserted. In this last case, we consider that the algorithm cannot solve this instance. The framework of this third solution algorithm is given in Algorithm 3.

4 Computational evaluations

In this section, we present the computational experiments carried out to evaluate the efficiency of the proposed heuristics. Let us denote by H_{GRE} the greedy algorithm coupled with the local search (Al-

Algorithm 3: ASSIGN

```

n : Number of batches;
L : Number of tanks;
Li: List of the batch to insert, sorted in increasing
order of priorities;
sol: Solution;
B : Selected batch;
Assign(): Classical Assignment Algorithm;
Check_inf_batch(sol): create Li with removing the
infeasible batches from the solution;
Update(Li): Computes the priorities;
Insert(B): inserts the batch B;
begin
  sol = Assign();
  Li = Check_inf_batch(sol);
  Update(Li);
  while (Li ≠ ∅) do
    if (Li[0] = 0) then
      | return(Infeasible);
    else
      B = Li[0];
      Li = Li - B;
      Insert(B);
      Update(Li);
    end
  end
  return(sol);
end

```

gorithm 1), H_{SACO} the ACO heuristic (Algorithm 2) and H_{ASSI} the heuristic based on the assignment problem coupled with the repairing method (Algorithm 3).

Preliminary experiments were conducted to optimize the behavior of the H_{SACO} algorithm for which several versions have been compared. These versions differ depending on (i) the probability function used in the construction process done by an ant to build a solution, (ii) the type of evaporation and enforcement processes applied, (iii) the impact of the repairing method, (iv) the value of other parameters as the maximum number of iterations and the number of ants.

The probability function, denoted by $fun_{proba}()$ in Algorithm 2, influences the global search strategy applied by the ants. In a classic ACO heuristic, this function returns a constant value which is a parameter of the algorithm. In H_{SACO} we apply the same principle than in Simulated Annealing algorithm, *i.e.* we make varying this probability among the search. The idea is to enforce diversification at the beginning of the search and intensification at the end. We tested the three followings functions:

$$f_1(iter) = \frac{\log(iter)}{\log(ITER_{MAX})};$$

$$f_2(iter) = \frac{iter}{ITER_{MAX}};$$

$$f_3(iter) = \frac{\log(ITER_{MAX} - iter + 1)}{\log(ITER_{MAX})};$$

Where $iter$ is the current iteration number, and $ITER_{MAX}$ is the maximum iteration number.

The second parameter studied is the type of evaporation and enforcement processes. We compared two modes with several values. The first mode uses a percentage of pheromone while the second mode uses a fixed quantity of pheromone to add in the enforcement process and to remove in the evaporation process.

We also studied the impact of the repairing method on the solution quality and the computational time. We tested these followings values: $\rho = 0, \rho = 1, \rho = 30\%, \rho = 60\%$. The last studied parameters are the number of ants and the number of iterations. For the first one, we tested three values (10, 15 and 20 ants) whilst for the second one we tested four values (50, 100, 150 and 200 iterations).

The experimental results show that the choice of a probability selection function is related to the type of evaporation and enforcement processes. The best results are obtained with the function f_2 , a fixed quantity evaporation τ defined by $\tau = \frac{\tau_{max} - \tau_{min}}{ITER_{MAX}}$ and an enforcement with $(2 \times \tau)$. The repairing method with $\rho = 60\%$ provides better results, without deteriorating the required CPU time since infeasible solutions are mainly built during the first iterations of H_{SACO} . Besides, the suitable number of ants is 20 and the maximum iteration number which provides the best results is equal to 150.

After having calibrated the H_{SACO} heuristic, we evaluate the quality of the three heuristics. We compare these algorithms on industrial-like instances, *i.e.* instances as close as possible to the industrial process (for confidential reasons, real industrial instances were not available for extensive testing purposes). In such instances, the number of batches is about 80: we generated instances with 75, 80, 85 and 90 batches. For each batch size, 500 instances have been randomly generated trying to follow as close as possible the features of the industrial instances. For this purpose, the $R_1 = (\text{number of batches}/\text{number of tanks})$ ratio and $R_2 = (\text{number of batches}/\text{number of families})$ ratio remain equal to that of the industrial instances. The details of the generation process are the following: ratio R_1 is drawn at random using a uniform law in the set $\{\frac{8}{3}, \frac{9}{3}, \frac{10}{3}, \frac{11}{3}\}$ and ratio R_2 is set to 3. This implies that the number of tanks depends on the number of batches and takes integer values within the interval [20.45; 33.75]. The number of families takes values in the set {25; 26; 28; 30}. The processing times p_1 (tank loading time) and $p_{i,2}$ (batch emptying time) are generated as fractions of the considered planning horizon which is of 4320 minutes (72 hours). This is done in order to generate data close to the real-life instances: $p_1 = \frac{756}{3 \times R_1}$ and $p_{i,2}$ is drawn at random using a uniform law in the interval $[\frac{2116}{3 \times R_1}, \frac{3931}{3 \times R_1}]$. The time lags

q_i are drawn at random in the interval $[p_{i,2}; 2p_{i,2}]$ and the time lag δ is set to 240. The family depend setup time S is set to 120. To generate the release dates r_i we randomly generate sequences of batches on the packing lines in order to have on the average $3 \times R_1$ batches per line. If two batches of the same family are sequenced consecutively on the same line, no idle time is inserted between these two batches. Otherwise an idle time equal to 60 minutes is inserted (this model the duration required to clean the packing line as we change the family). From these schedules we deduce the values of the r_i 's.

Besides, on the result tables in which the heuristic H_{ASSI} is concerned, for a given batch size we separate the instances depending on the number of infeasible batches p_{inf} obtained after the assignment problem has been solved. By the way, we obtained two intervals of values, given as a percentage of the total number of batches: $]0\%, 10\%]$ and $\in]10\%, 20\%]$.

To evaluate the efficiency of the heuristics, we use the relative deviation $RELA$ (in percentage) and the absolute deviation ABS . We have: $RELA = (OBJ_H - OBJ_{best})/OBJ_{best}$ and $ABS = OBJ_H - OBJ_{best}$ with OBJ_{best} the best objective function value among the compared heuristics, and OBJ_H the objective function value calculated by a given heuristic H . We provide the average and maximum deviations. Clearly, these deviations are only calculated on instances on which the compared algorithms succeeded in finding a feasible solution. We also compute the percentage, denoted by $BEST$, of instances for which a heuristic is the only one to provide the best solution. All the tests have been done on a Pentium Core 2 Duo 2.4Ghz with 2Go of RAM.

Table 2 provides the results of the comparison between H_{GRE} and H_{SACO} . The first column contains the number of batches. The two next columns provide the percentage of best solutions found by the heuristics. As the value of column $BEST$ for the H_{GRE} heuristic is always equal to 0 (which means that there is no instance for which it performs better than H_{SACO}), we only focus in the remaining columns on the relative and absolute deviation of H_{GRE} from H_{SACO} . It appears that, on the 2000 tested instances, H_{GRE} never outperforms H_{SACO} .

Tables 3 and 4 present the computational results of the comparison of H_{SACO} and H_{ASSI} . Table 3 gives the results for the relative deviation while Table 4 presents the results for the absolute deviation. The first column of these two tables contains the number of batches. For each algorithm we next provide the average and maximum (columns AVG and MAX) deviations. For algorithm H_{ASSI} we also present $INFEASI$, the average percentage of infeasible batches p_{inf} . These results are calculated on instances for which both algorithms have found a fea-

#BATCHES	H_{SACO}	H_{GRE}	RELA DEV of H_{GRE} (%)		ABS DEV of H_{GRE}	
	BEST (%)	BEST (%)	AVG	MAX	AVG	MAX
75	89.00	0.00	9.74	29.03	3.00	9
80	81.80	0.00	7.66	26.67	2.51	8
85	84.60	0.00	8.40	30.00	2.97	10
90	81.20	0.00	6.70	30.56	2.56	11

Table 2: Comparison of H_{GRE} and H_{SACO}

sible solution. Remind that for each batch size, two lines appear as we separate the instances between the value intervals of p_{inf} as explained above.

These two tables highlight the fact that H_{ASSI} outperforms the H_{SACO} heuristic both in terms of absolute and relative deviations. For only approximately, on the average, 10% of the instances, H_{SACO} provides better solutions. However, the efficiency of H_{ASSI} is seriously affected by the percentage of infeasible batches p_{inf} : when it increases the global performances of the heuristic decreases, even it remains better on the tested instances than H_{SACO} .

The percentages of instances for which a heuristic was not able to find a feasible solution are presented in Table 5. The H_{SACO} heuristic provides the best results since it always computes a feasible solution whilst H_{GRE} and H_{ASSI} failed to solve some instances. Comparing H_{GRE} and H_{ASSI} , we observe that H_{ASSI} gives better results.

#BATCHES	H_{SACO}	H_{GRE}	H_{ASSI}
75	0.00	8.00	7.80
80	0.00	6.40	3.60
85	0.00	8.20	5.40
90	0.00	7.00	6.60

Table 5: Percentage of infeasible solutions

Table 6 presents the average computational time, in seconds, of the three proposed heuristics. As we can see, the average computational time of H_{GRE} and H_{ASSI} never exceed 1 second while H_{SACO} exceed 100 second for the largest instances. However, in practice even this computational time is acceptable for the shampoo production company.

#BATCHES	H_{SACO}	H_{GRE}	H_{ASSI}
75	61.36	0.10	0.5400
80	77.60	0.13	0.7609
85	88.19	0.15	0.8583
90	102.33	0.18	0.9578

Table 6: Average computational time (in seconds)

From the above computational results, we can deduce that heuristic H_{GRE} even if simple is strongly outperformed by H_{ASSI} both in terms of computational time, deviations and number of calculated feasible solutions. Notice that, often, scheduling and planning software make extensive uses of greedy rules to pro-

vide solutions, which is shown here to be irrelevant. The comparison of H_{ASSI} and H_{SACO} does not yield such a strong conclusion: even if H_{ASSI} is faster and, on the feasible instances, more efficient than H_{SACO} , the latter requires a computational time acceptable by the industrials and provides more often a feasible solutions (which is, in a sense, more important in practice). Our conclusion is that, on industrial instances, we should consider both heuristics.

5 Conclusions

In this paper we have tackled a scheduling problem arising the industrial context of producing shampoo products. This problem consists in assigning batches to tanks before they enter the packing system of the company. This assignment is done in order to answer dedicated constraints and to minimize the total number of tank cleanings (family setup times). We have provided a bicriteria model of this problem and proved that under special condition on the tank capacities, the problem can be solved in polynomial time by solution of an assignment problem. For the general problem, as it occurs in practice, we prove its NP-Hardness and provide three different heuristics: a greedy heuristic, an ACO heuristic and a dedicated heuristic based on the solution of an assignment problem. Computational experiments show that the best heuristic, in terms of efficiency and computational time, is the third one. However, as the second one provides more often feasible solutions it is also considered for solving industrial instances.

The efficient solution of this storage problem enables us to start with the solution of the more general scheduling problem arising at the making service. This problem consists in scheduling the whole making phase, from the batch production to its storage in the tanks before sending them to the packing service. Clearly, the heuristics presented in this paper will be useful in this context.

References

- A. Allahverdi, C.T. Ng, T.C.E. Cheng, M.Y. Kovalyov., 2008, "A survey of scheduling problems with setup times or costs.", *European Journal of Operational Research*, vol. 187, pp. 985–1032.
- R. Burkard, M. Dell Amico and S. Martello., 2009, "Assignment problems.", *Society for Industrial*

#BATCHES	H_{SACO} Results (%)			H_{ASSI} Results (%)			
	BEST	RELA AVG	RELA MAX	INFEASI	BEST	RELA AVG	RELA MAX
75	15.18	3.13	11.53	8.02	71.72	0.68	10.34
	23.63	2.07	11.11	12.38	58.63	1.16	11.11
80	4.76	4.85	19.04	7.47	88.88	0.20	6.89
	18.35	2.93	14.00	12.20	71.48	1.01	12.12
85	6.07	3.75	14.28	8.10	82.24	0.23	8.57
	20.07	2.59	11.67	12.46	68.72	0.97	13.89
90	1.21	5.27	14.28	7.64	95.12	0.05	5.55
	7.92	3.82	15.62	11.99	82.17	0.31	8.57

Table 3: Comparison of H_{ASSI} and H_{ACO} : The relative deviation.

#BATCHES	H_{SACO} Results		H_{ASSI} Results		
	ABS AVG	ABS MAX	INFEASI (%)	ABS AVG	ABS MAX
75	0.86	3	8.02	0.19	3
	0.60	3	12.38	0.35	3
80	1.38	4	7.47	0.06	2
	0.90	4	12.20	0.33	4
85	1.19	4	8.10	0.07	3
	0.85	4	12.46	0.34	5
90	1.73	4	7.64	0.01	2
	1.34	5	11.99	0.11	3

Table 4: Comparison of H_{ASSI} and H_{SACO} : The absolute deviation.

and Applied Mathematics.

M.Dorigo and T.Stutzle, 2002, "Ant Colony Optimization.", *Massachusetts Institute of Technology press*.

J.N.D Gupta., 1986, "Flowshop schedules with sequence dependent setup times", *Journal of the operations research of Japan*, vol. 29, pp. 206-219.

C.H. Papadimitriou and P.C. Kanellakis, 1980, "Flowshop scheduling with limited temporary storage.", *Journal of the Association for Computing Machinery*, vol. 27, pp. 533-549.

A.W.J. Kolen, J.K. Lenstra, C.H. Papadimitriou, F.C.R. Spieksma., 2007, "Interval Scheduling: A Survey.", *Naval Research Logistics*, vol. 54, pp. 530-543.

M.Y. Kovalyov, C.T. Ng and T.C. Edwin Cheng., 2007, "Fixed interval scheduling: Models, applications, computational complexity and algorithms.", *European Journal of Operational Research*, vol. 178, pp. 331-342.

V. T'kindt and J.-C. Billaut., 2006, "Multicriteria Scheduling: Theory, Models and Algorithms.", 2nde edition, *Springer*.

V. T'kindt, N. Monmarché, F.Tercinet and D. Laugt., 2002, "An Ant Colony Optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem.", *European Journal of Operational Research*, vol. 142, pp. 250-257.

J.J. Yuan, Z.H. Liu, C.T. Ng, T.C.E. Cheng., 2006, "Single machine batch scheduling problem with family setup times and release dates to minimize makespan.", *Journal of Scheduling*, vol. 9, pp. 499-513.