Università degli Studi di Roma Tre
**Dipartimento di Informatica e Automazione**
Via della Vasca Navale, 79 – 00146 Roma, Italy

# A tabu search algorithm for scheduling pharmaceutical packaging operations

Luca Venditti[1], Carlo Meloni[2] and Dario Pacciarelli[1]

**RT-DIA-130-2008** **Giugno 2008**

(1) Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre,
via della vasca navale, 79 - 00146 Roma, Italy.
(2) Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari,
via E. Orabona, 4 - 70125 Bari, Italy.

*This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, project FIRB "Advanced tracking system in intermodal freight transportation"*

## ABSTRACT

This paper addresses a practical scheduling problem arising in the packaging department of a pharmaceutical industrial plant. The problem is modelled as a parallel machine scheduling problem with setups, release and due dates and additional constraints related to the scarce availability of tools and human operators. The objective functions are minimization of makespan and maximum tardiness in lexicographic order. Representing a solution with a directed graph allows us to devise an effective tabu search algorithm to solve the problem. Computational experiments, carried on real and randomly generated instances, show the effectiveness of this approach.

# 1 Introduction

In this paper, we address a parallel machine scheduling problem with resource constraints related to tools and operators availability, setup times, release times, due dates and deadlines. The objectives are the joint minimization of makespan and maximum tardiness, which are addressed in lexicographic order. This problem is motivated by the practical implementation of a decision support system for scheduling the production orders at the packaging department of a pharmaceutical plant located in Italy. The production of pharmaceutical packages is driven by wholesaler orders and a two-week schedule is produced every week, following a rolling horizon criterion. In the week preceding the packaging, different kinds of tablets are produced in the manufacturing area and stored in sealed bins. Then, in the packaging department, bins are re-opened and processed to produce final products. The packaging operations are executed on blister lines, which perform all operations from the production of blisters to the final individual specific packages. Each line processes one lot at a time, thus acting as a single machine. A detailed description of the problem is given in Section 2.

In the last years, there is a clear trend in the scheduling literature on extending the models to include more practical constraints [13, 18, 21]. However, to the best of our knowledge, no published work addresses exactly the problem studied in this paper, even though many algorithms have been proposed to solve relaxations of this problem, addressing different subsets of constraints. Among the papers dealing with problems most similar to the one addressed in this paper we distinguish three main streams of research, namely the parallel machine scheduling problems with setup times, release and due dates [19], the vehicle routing problem with fixed number of vehicles and time windows [1, 2, 4] and the resource constrained scheduling problem [3, 5, 6, 12].

In this paper we propose a graph representation of the problem and a tabu search algorithm, described in sections 3 and 4, respectively. The Tabu Search (TS) metaheuristic [9, 10] is a well established iterative algorithm with steepest descent criterion, which accepts non-improving moves to escape from local minima and makes use of a tabu list to restrict the neighborhood to be explored at each step. A move in the tabu list is forbidden and remains in it for a limited number of iterations, called the length of tabu list, which can be fixed or variable. This mechanism can be overruled when a solution associated with a tabu move satisfies an aspiration criterion. More sophisticated TS schemes have been proposed in the literature [7, 16], and this method is certainly among the most successful heuristics for a large number of planning and scheduling problems [14].

In Section 5 we report on our computational experiments, carried on real industrial instances and on randomly generated instances. The comparison with the industrial practice shows the effectiveness of our approach.

# 2 Description of the problem

The packaging department studied in this paper is composed of three packaging lines working in parallel. Each line, devoted to blisters preparation and packaging, can process one lot of identical products at a time, thus acting as a single machine.

A set of production orders, hereinafter called jobs, have to be scheduled on this set of parallel machines, and each job is compatible with a subset of machines. Each machine

needs two kinds of resources to process a job: a tool, which defines the size of the blister and depends on the job being processed, and a fixed number of operators, which is the same for each production line. Human resources availability is constant within a shift, but it can vary from one shift to another, the night shift being typically less supervised; machines can be unavailable in given periods for preventive maintenance operations. The considered scheduling problem is characterized by different timing constraints. Job release times are determined by the scheduled completion times at the earlier manufacturing departments. Each job has a due date, while a hard deadline is defined for urgent orders, when there is a risk of stock-out at the final customers. Each planned machine unavailability is modeled as an unavailability job characterized by a release date, corresponding to its starting time, a processing time, corresponding to its duration, and a deadline corresponding to its finish time.

Modeling the practical scheduling problem requires to consider a number of constraints and the incorporation of several details into the model. Tools are shared among families of similar products and available in a limited number of copies, in most cases there is a single copy of each tool. In each machine, a sequence-dependent setup occurs from the completion of a job to the start of another job, in order to clean and calibrate the machine and to change the tool defining the blister size. The setup is called *minor* when requiring no tool change and *major* in the other case. The time required to execute a minor setup is always smaller than the time required for a major setup. Moreover, all setup times satisfy the triangular inequality.

Cleaning operations, mechanical configurations and job processing require a given amount of work for human operators, therefore machines cannot process job nor execute setups without human resources. In order to reduce the risk of cross-contaminations, the plant policy is to assign each operator to a specific machine during the whole shift. In addition, due to technical reasons, setups cannot be interrupted. On the other hand, the processing of a job on a machine can be interrupted and resumed later on the same machine which remains unavailable for other jobs during that period. Hence, interruption of a job is used only at the end of a shift, if the number of operators in the subsequent shift is not sufficient to process the job.

Summarizing the above description, this scheduling problem can be described with the $(\alpha|\beta|\gamma)$ classification scheme of Graham *et al.* [11] as

$$P_3|r_i, d_i, D_i, s_{ij}, M_i, RA, RC|C_{max}, T_{max}$$

in which the following notation is used:

- $R_k$, unrelated parallel machines production environment with $k$ machines;
- $r_i$, release times;
- $d_i$, due dates;
- $D_i$, deadlines;
- $s_{ij}$, sequence-dependent setup times;
- $M_i$, machine eligibility;

- $RA$, Resource availability: job processing is interrupted and resumed later when the machine or the operators become unavailable but no other job can be processed in between;
- $RC$, resource constraints (tool availability);
- $C_{max}$, minimization of makespan;
- $T_{max}$, minimization of maximum tardiness.

We notice that, in the scheduling literature, even relaxed versions of this problem, such as the vehicle routing problem with release and due dates, are considered particularly difficult NP-hard problems [8, 17]. Moreover, some constraints, such as the resumable availability, are not frequently addressed in scheduling literature, at least in combination with others. In the following section we illustrate the mathematical structure of a feasible solution for this problem.

# 3 Graph representation of a solution

In this section, we introduce the notation used throughout the paper and then we formally describe the constraints satisfied by feasible solutions.

Problem data consist of the following. A set of $n$ jobs $\mathcal{J} = \{J_1, ..., J_n\}$, each consisting of a single operation, must be scheduled on a set of $m$ machines $\mathcal{M} = \{M_1, ..., M_m\}$. Each job must be processed entirely on the same machine, which can process one job at a time. We denote with $\mathcal{M}(J_j) \subseteq \mathcal{M}$ the set of machines able to process job $J_j$. In order to process $J_j$, a machine must be equipped with a fixed number $b$ of operators and with a tool of type $\tau_j$, chosen in the set $\mathcal{T} = \{T_1, ..., T_t\}$. We denote with $\mathcal{T}(J_j) \subseteq \mathcal{T}$ the set of tools of type $\tau_j$. Also, let $p_j$, $r_j$, $d_j$ and $D_j$ be the processing time, release date, due date and deadline of $J_j$, respectively.

Between two jobs $J_i$ and $J_j$, processed consecutively on the same machine $M_h$, a setup time $f_{J_iJ_j}$ is required to clean and calibrate $M_h$ and, if $\tau_i \neq \tau_j$, to change tool. Similarly, if $J_i$ is processed on $M_h$, $J_j$ is processed on $M_k$ and both use consecutively the same tool, another setup time $g_{J_iJ_j}$ is required to transfer the tool from $M_h$ to $M_k$. All setup times satisfy the triangular inequality, i.e. $f_{J_iJ_j} + f_{J_jJ_k} \geq f_{J_iJ_k}$ and $g_{J_iJ_j} + g_{J_jJ_k} \geq g_{J_iJ_k}$. Moreover, minor setups are always smaller than major setups, i.e. if $\tau_i = \tau_j \neq \tau_k$ impies $f_{J_iJ_j} < f_{J_iJ_k}$. Clearly, all setups between different machines are always major setups, i.e. $f_{J_iJ_j} < g_{J_iJ_j}$.

The number of operators $o_i$ available during shift $i = 1, \ldots, s$ is known in advance, and operators are assigned to machine for the entire duration of a shift. Therefore, at most $\left\lfloor \frac{o_i}{b} \right\rfloor$ machines can be active during shift $i$. We model this situation by introducing $m - \left\lfloor \frac{o_i}{b} \right\rfloor$ dummy *unavailability* jobs with release date, deadline and processing time equal to the shift start, end and duration, respectively. These jobs will be scheduled on the machines with all the other jobs. In some cases, the subset of inactive machines for a shift is partially specified in advance, for example when preventive maintenance operations are planned for some machines during that shift. Hence, we denote with $\mathcal{M}(u_h) \subseteq \mathcal{M}$ the set of machines compatible with the unavailability job $u_h$ and with $\mathcal{U} = \{u_{n+1}, \ldots, u_{n+q}\}$ be

the set of unavailability jobs, where

$$q = \sum_{i=1,\ldots,s} \left(m - \left\lfloor \frac{o_i}{b} \right\rfloor\right).$$

Finding a *solution* consists of:

(*i*) assigning each job $J_j$ to a machine $M_h \in \mathcal{M}(J_j)$ and to a tool $T_k \in \mathcal{T}(J_j)$, $j = 1, \ldots, n$,

(*ii*) assigning each unavailability $u_i$ to a machine $M_h \in \mathcal{M}(u_i)$, $i = n+1, \ldots, n+q$,

(*iii*) scheduling the jobs and the unavailabilities assigned to each machine $M_h$, $h = 1, \ldots, m$ and the jobs assigned to each tool $T_k$, $k = 1, \ldots, t$, i.e., defining a *starting time* $s_j$ and a *completion time* $c_j$ for each job $J_j \in \mathcal{J}$ such that jobs or unavailabilities assigned to the same machine or tool do not overlap.

Let $\mathcal{J}(M_h)$ be the set of jobs and unavailabilities assigned to $M_h$ and $\mathcal{J}(T_k)$ be the set of jobs assigned to $T_k$. Therefore we have

$$\mathcal{J} \cup \mathcal{U} = \bigcup_{i=1}^{m} \mathcal{J}(M_i), \quad \mathcal{J}(M_i) \cap \mathcal{J}(M_j) = \emptyset, i \neq j, \tag{1}$$

$$\mathcal{J} = \bigcup_{i=1}^{t} \mathcal{J}(T_i), \quad \mathcal{J}(T_i) \cap \mathcal{J}(T_j) = \emptyset, i \neq j. \tag{2}$$

A job $J_j$ assigned to machine $M_h$ and tool $T_k$ occupies $M_h$ and $T_k$ from $s_j$ to $c_j$. However, while preempting setups is not allowed, the execution of $J_j$ can be interrupted every time the machine becomes unavailable. Since the time required to resume the processing of $J_j$ is negligible, the completion time of $J_j$ equals $s_j + p_j$ plus the total time $M_h$ is unavailable for processing before the completion of $J_j$. The completion time of job $J_j \in \mathcal{J}(M_h)$ can be computed as follows. Let us denote with $u_{i_1}, \ldots u_{i_v}$ the sequence of unavailabilities in $\mathcal{J}(M_h)$ starting after $s_j$, ordered for increasing values of their release times $r_{i_1} \geq s_j, r_{i_2} \geq r_{i_1}, \ldots r_{i_v} \geq r_{i_{v-1}}$, and let $p_{i_1}, \ldots p_{i_v}$ be their respective processing time. Then, the following holds:

$$c_j = s_j + p_j + \sum_{l=1}^{k} p_{i_l} \tag{3}$$

with $k = \min\{h : s_j + p_j + \sum_{l=1}^{h} p_{i_l} \leq r_{i_{h+1}}\}$. The completion time of $J_j$ is feasible if

$$c_j \leq D_j \qquad \forall J_j \in \mathcal{J}. \tag{4}$$

In a solution, all jobs and unavailabilities in $\mathcal{J}(M_h)$ must be totally ordered, and we call $\sigma_h$ the resulting sequence, for $h = 1, \ldots, m$. Similarly, the jobs in $\mathcal{J}(T_k)$ must be ordered, thus obtaining another sequence $\pi_k$, for $k = 1, \ldots, t$. For each job $J_j \in \mathcal{J}$ we denote with $PM_j$ the job or unavailability preceding $J_j$ in $\sigma_h$, with $SM_j$ the job or unavailability succeeding $J_j$ in $\sigma_h$, and with $\overline{PM}_j$ the last job sequenced before $J_j$, possibly with some unavailability sequenced between the completion of $\overline{PM}_j$ and the starting of

$J_j$. Similarly, for each job $J_j \in \mathcal{J}$ we denote with $PT_j$ the job preceding $J_j$ in $\pi_h$ and with $ST_j$ the job succeeding $J_j$ in $\pi_h$. Finally, we define the setup time $F_{PM_jJ_j}$ between $PM_j$ and $J_j$ as follows:

$$F_{PM_jJ_j} = \begin{cases} 0 & \text{if } \left(PM_j \neq \overline{PM}_j\right) AND \\ & \left(r_{PM_j} \geq c_{\overline{PM}_j} + f_{\overline{PM}_jJ_j}\right) \\ f_{\overline{PM}_jJ_j} & \text{otherwise} \end{cases} \tag{5}$$

Equation (5) takes into account the fact that preemption is forbidden during setup execution and allowed during job processing. Hence, if $PM_j \neq \overline{PM}_j$, i.e. if $PM_j$ is an unavailability, the setup between $\overline{PM}_j$ and $J_j$ can be performed before $PM_j$ only if there is enough time to complete it. We notice that, if the condition $r_{PM_j} > c_{\overline{PM}_j} + f_{\overline{PM}_jJ_j}$ occurs in Equation (5), then it may be convenient anticipating job $J_j$ to start before the unavailability $PM_j$ and processing it until $r_{PM_j}$. In this case, the predecessor of $J_j$ is no longer the unavailability but job $\overline{PM}_j$. In other words, in the first case of Equation (5) we can limit ourselves to consider always $r_{PM_j} = c_{\overline{PM}_j} + f_{\overline{PM}_jJ_j}$.

With this notation, the starting time of $J_j$ can be computed as

$$s_j \geq \max\{r_j, \quad c_{PM_j} + F_{PM_jJ_j}, \quad c_{PT_j} + g_{PM_jJ_j}\}. \tag{6}$$

Summarizing the above discussion, a feasible solution consists of defining sets $\mathcal{J}(M_h)$ and $\mathcal{J}(T_k)$ according to (1) and (2), defining sequences $\sigma_h$ and $\pi_k$ and times $s_j$ and $c_j$ for $h = 1, \ldots, m$, $k = 1, \ldots, t$, $j = 1, \ldots, n$, such that constraints from (3) to (6) are satisfied. The performance indicators of a feasible solution are the makespan $C_{MAX} = \max_j\{c_j : j = 1, \ldots, n\}$ and the maximum tardiness $T_{MAX} = \max_j\{\max\{0, c_j - d_j\} : j = 1, \ldots, n\}$. The former indicator is more important, being a surrogate of the department productivity. Hence, we consider these two objective functions in lexicographic order: among the solutions minimizing the first objective functions, the best is the one minimizing the second.

In the remaining part of this section, we introduce the representation of a solution $S$ by using a graph $\mathcal{G}(S) = (V, E(\sigma) \cup F(\pi) \cup A)$, where: $V$ is the set of nodes, each representing an unavailability or the processing of a job fraction; $E(\sigma)$ is a set of arcs, each between two consecutive nodes processed on the same machine in $S$; $F(\pi)$ is a set of arcs, each between two consecutive jobs using the same tool in $S$; finally, $A$ is a set of additional arcs that we use to compute the objective function. Set $V$ also contains five auxiliary nodes *start*, *end*, *viol_unav*, *viol_dead*, *viol_due*. The first two nodes represent the beginning and the completion of the schedule, while the other three nodes are used to compute the maximum violation of unavailability deadlines, job deadlines and due dates. An arc from *start* to every non-auxiliary nodes in $V$ is added to $A$, as well as an arc from each unavailability node to *viol_unav*. Moreover, from each node $i$ associated to the processing of the last fraction of a job, there is an arc $(i, end) \in A$, an arc $(i, viol\_dead) \in A$ if a deadline is defined for $i$, and an arc $(i, viol\_due) \in A$ if a due date is defined for $i$.

Nodes are weighted with the processing time of the associated job fraction or with the duration of the associated unavailability. Each arc $(h, j) \in E(\sigma)$ is weighted with the quantity $F_{hj}$, computed according to (5), while each arc $(l, j) \in F(\pi)$ is weighted with the setup $g_{lj}$ between the associated jobs. This situation is shown in Figure 1, where arcs in $E(\sigma)$ are depicted with solid lines and arcs in $F(\pi)$ are depicted with dashed lines.

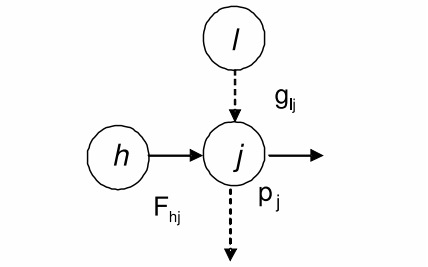


Figure 1: Graph representation of a node with sequencing arcs and weights

For each unavailability $u = n+1, \ldots, n+q$, arcs $(start, u)$ and $(u, viol\_unav)$ are weighted with $r_u$ and $-D_u$, respectively. For each node $i$ associated to the processing of the first fraction of a job, arc $(start, i)$ is weighted with the job release time $r_i$, and for each node $j$ associated to the processing of the last fraction of a job $J_j$, arc $(j, end)$ has weight 0, arcs $(j, viol\_dead)$ and $(j, viol\_due)$, if any, have weights $-D_j$ and $-d_j$, respectively. Figure 2 shows the representation of job $J_j$, whose processing is interrupted by two unavailabilities $u$ and $v$.

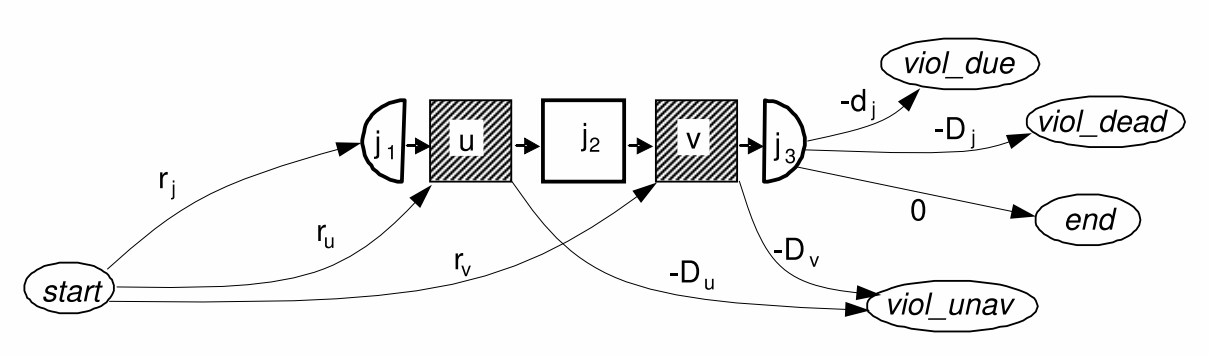


Figure 2: Processing of job $J_j$ interrupted by two unavailabilities $u$ and $v$

For a given graph $\mathcal{G}$, we can define the *head* $h(j)$ of node $j$ as the longest path in $\mathcal{G}$ from *start* to $j$ and the *tail* $q_a(j)$ as the longest path from $j$ to the auxiliary node $a \in \{end, viol\_dead, viol\_due, viol\_unav\}$. With this notation, the starting time $s_j$ of job $J_j$ in a solution can be easily computed as the head of the node associated to the processing of the first fraction of $J_j$. Then, $h(end)$ and $\max\{0, h(viol\_due)\}$ are equal to the makespan $C_{Max}$ and the maximum tardiness of the solution, respectively. The solution is feasible if $h(viol\_dead) \leq 0$ and $h(viol\_unav) \leq 0$.

Figure 3 shows the graph representation of a solution for a problem with twelve jobs and one unavailability, scheduled on three machines and four tools. Arcs in $E(\sigma)$ are depicted with horizontal solid arrows, while arcs in $F(\pi)$ are depicted with dashed arrows.

# 4 Tabu search algorithm

In this section, we describe our tabu search algorithm. The *Tabu Search* is a local search based metaheuristic, which makes extensive use of memory for guiding the search. Its basic components are the concepts of move and tabu list, which restrict the set of solutions

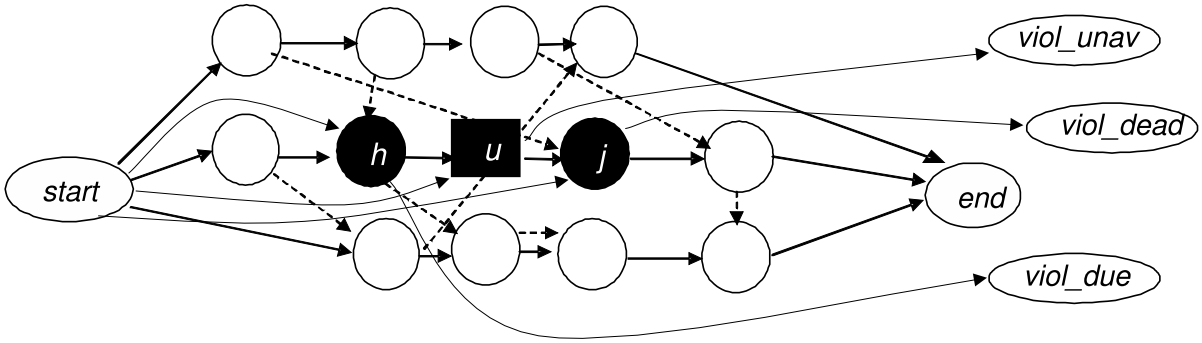


Figure 3: Graph representation of a solution

to be explored. From the incumbent solution, non-tabu moves define a set of solutions, called neighborhood, the best of which is selected as the new incumbent. The tabu list stores the inverse of the last moves performed by the algorithm, and it is used to escape from local optima and to avoid re-visiting twice solutions that have been recently visited.

Section 4.1 describes the algorithm used to find an initial solution, not necessarily feasible. Section 4.2 describes the basic moves used by the tabu search algorithm. In Section 4.3, these moves are used with a feasibility test to devise two neighborhood structures, in which the search for neighbors is limited to feasible solutions. Two procedures to evaluate the quality of a neighbor are described in Section 4.4: the first requires constant time and provides an estimate of the objective functions, while the second provides the exact schedule in linear time.

## 4.1 Greedy algorithm

A fast and simple greedy algorithm is used to obtain an initial feasible solution. The algorithm is summarized in the following steps:

1. compute an estimate workload for each machine as follows:
   (a) compute the minimum workload $W_h$ for machine $M_h \in \mathcal{M}$ summing up processing time + minimum setup time of all jobs that can be executed on $M_h$ only, i.e. $W_h = \sum_{j:\mathcal{M}(J_j)=\{M_h\}} p_j + \min_i\{f_{J_i J_j}\}$.
   (b) compute the quantity $Q$ equal to the processing time + minimum setup time of all jobs that can be executed on $M_h$ and other k machines in $\mathcal{M}$, and add to $W_h$ the quantity $\frac{Q}{k+1}$.
2. assign human operators to machines, proportionally to the values $W_h$;
3. partition the time horizon into time intervals $l = 1, \ldots, L$, each of duration $H$, and assign and schedule the jobs on the machines for each time interval $[(l-1)H, lH]$ according to the following algorithm:
   (a) let $\mathcal{J}_l$ be the set of jobs with deadline or due-date smaller than $lH$;
   (b) group into blocks the jobs in $\mathcal{J}_l$ requiring the same tool, and sequence the jobs in each block according the ERD rule (Earliest Release Date first rule);

(c) sequence the blocks one at a time by assigning a block to the next available machine according the SST rule (Shortest Setup Time first rule) until all blocks are scheduled.

## 4.2 Neighborhood structure

Our tabu search algorithm makes use of two basic moves. The first one is based on the interchange of a pair of adjacent jobs sequenced on the same machine/tool, which is commonly adopted in the tabu search literature on open and job shop scheduling problems [15]. The second move is based on removing a job from the sequence of jobs processed on a machine (tool) and inserting it in the sequence of another machine (tool). This move is commonly used when dealing with parallel machine scheduling or vehicle routing problems [4, 7].

The *interchange* move $\varphi^M(x, y)$ is defined as follows: given a solution $S$ and a pair of consecutive jobs $x$ and $y$ in $\sigma_h$, a new solution $S'$ is obtained from $S$ reversing the precedence arc $(x, y)$. This move is made more complex when job processing is split into different fractions. In this case jobs are recomposed, their position is exchanged and then are rescheduled taking into account machine unavailability. A similar move $\varphi^T(x, y)$ deals with rescheduling of consecutive jobs $x$ and $y$ in the sequence $\pi_k$ of tool $T_k$. When two jobs are processed consecutively on the same machine $M_h$ and tool $T_k$, then a move $\varphi^{MT}(x, y)$ must be applied, which exchanges their relative positions both in $\sigma_h$ and $\pi_k$, in order to avoid a cycle of precedence constraints in the processing of $x$ and $y$.

The *rerouting* move $\theta^M(x, y)$ is applied to a pair of jobs or unavailabilities $x$ and $y$ processed on different machines $M_h$ and $M_k$, respectively, with $M_k \in \mathcal{M}(x)$. It consists of removing $x$ from $\sigma_h$ and inserting it before $y$ in $\sigma_k$. Similarly, move $\theta^T(x, y)$ is applied to jobs $x$ and $y$ processed on different tools $T_h$ and $T_k$, respectively, with $T_k \in \mathcal{T}(x)$.

### 4.2.1 Acyclicity property

A solution $S$ is feasible only if the corresponding graph $\mathcal{G}(S)$ is acyclic. Our tabu search algorithm does not perform moves leading to cyclic graphs. To this aim, a ciclicity test is preliminarily checked before each move. We next describe the test for the two moves separately. In the following we denote $S$ the incumbent solution, $S'$ the solution obtained after a move, $h(i)$ the head of node $i$ in $S$ and $h'(i)$ the head of node $i$ in $S'$.

- Interchange move: the following lemma holds:

  **Theorem 4.1** *Given an arc $(x, y)$ on the critical path, the interchange move $\varphi^M(x, y)$ does not create cycles*

  **Proof.** Applying a move $\varphi(x, y)$ a cycle is created in $\mathcal{G}(S')$ if and only if there is a path in $\mathcal{G}(S)$ from $x$ to $y$, other than arc $(x, y)$. Three cases are possible:

  - $\varphi^{MT}(x, y)$ move: there are no other arcs outgoing $x$, other than those ingoing $y$ or the auxiliary nodes, and therefore this move cannot cause cycles in $\mathcal{G}(S')$
  - $\varphi^M(x, y)$ move: let us suppose that a path $p$ from $x$ to $y$ exists $\mathcal{G}(S)$. This path must include at least arc $(x, ST_x)$, because this is the only arc outgoing from $x$ other than $(x, SM_x)$, and arc $(PT_y, y)$. There are only two possibilities: either

$\tau_x \neq \tau_y$ or $\tau_x = \tau_y$. In the former case, path $p$ includes at least a tool change and the processing time $p_{ST_x} > 0$. From the triangular inequality, the weight of all setups occuring in $p$ is larger or equal to $f_{xy}$. Therefore, the weight of $p$ is strictly larger than the weight of the critical arc $(x, y)$, a contradiction. In the latter case, the weight of arc $(x, y)$ only include the minor setup $f_{xy}$, while path $p$ includes at least the processing time $p_{ST_x}$ and a major setup $g_{xST_x}$ for moving $\tau_x$ from $M_x$to $M_{ST_x}$, again the weight of $p$ is larger than the weight of the critical arc $(x, y)$, a contradiction.

- $\varphi^T(x, y)$ move: let us suppose that a path $p$ from $x$ to $y$ exists in $\mathcal{G}(S)$. This path must include at least arc $(x, SM_x)$ because it is the only arc outgoing from $x$ other than $(x, ST_x)$, and an arc corresponding to a transfer time for a tool because $M_x \neq M_y$. Hence the length of path $p$ is greater than the weight of the critical arc $(x, ST_x)$, a contradiction. So path $p$ does not exists if $(x, y)$ is critical.

- Rerouting moves: there is a cycle in $\mathcal{G}(S')$ after move $\theta^M(x, y)$ if and only if there is a path in $\mathcal{G}(S)$ from $y$ to $PT_x$ or from $ST_x$ to $PM_y$. We notice that, if $y = ST_x$ the acyclicity of $\mathcal{G}(S)$ implies that of $\mathcal{G}(S')$. Otherwise, a sufficient condition for the acyclicity of $\mathcal{G}(S')$ is given by the following theorem, proved in [5]:

  **Theorem 4.2** *After a move $\theta^M(x, y)$, $\mathcal{G}(S')$ is acyclic if $(h(y) + p_y > h(PT_x))$ and $(h(ST_x) + p_{ST_x}) > h(PM_y)$ holds.*

  Similar conditions hold for $\theta^T(x, y)$. When sufficient conditions do not hold the algorithm explicitly checks the existence of a path from $x$ to $y$ on $\mathcal{G}(S)$, different from arc $(x, y)$. If such a path exists, the move is not evaluated.

### 4.2.2 Connectivity

In this section, we show that the neighborhood defined by interchanging consecutive jobs on a critical path is connected, i.e. we prove that it is possible to reach a global minimum starting from any feasible solution. Our proof is similar to that reported in [5].

In this section, we denote with $a$ a resource assignment, i.e. the specification of a set $\mathcal{J}(M_h)$ of jobs and unavailabilities for each $M_h \in \mathcal{M}$, and a set $\mathcal{J}(T_k)$ of jobs for each $T_k \in \mathcal{T}$.

We call $S_a$ and $S_a^*$ a generic and an optimal solution associated to $a$, respectively, i.e. an optimal sequencing of the elements in $\mathcal{J}(M_h)$ and $\mathcal{J}(T_k)$ for each $M_h \in \mathcal{M}$ and $T_k \in \mathcal{T}$. We also call $\mathcal{G}(S)$ the graph representation of $S$ and $\mathcal{G}(\overline{S})$ its transitive closure, i.e. the graph obtained from $\mathcal{G}(S)$ including all the redundant arcs. A first result is the following:

**Lemma 4.1** *Given a resource assignment $a$, an optimal solution $S_a^*$ and a solution $S_a$, if $S_a$ is not optimal there is an arc $(i, j)$ such that $(i, j)$ is critical in $\mathcal{G}(S_a)$ and such that $(j, i) \in \mathcal{G}(\overline{S}_a^*)$.*

**Proof.** Let suppose that all critical arcs of $\mathcal{G}(S_a)$ also belong to $\mathcal{G}(\overline{S}_a^*)$ as well. Therefore all the critical paths of $\mathcal{G}(S_a)$ also belong to $\mathcal{G}(\overline{S}_a^*)$, thus implying that $S_a^*$ is not optimal,

a contradiction. Hence, there must be at least a critical arc $(j,i) \in \mathcal{G}(S_a)$ that does not belong to $\mathcal{G}(\overline{S}_a^*)$, i.e. such that $(j,i) \in \mathcal{G}(\overline{S}_a^*)$. ∎

From this lemma the following theorem can be proved.

**Theorem 4.3** *Given a resource assignment $a$ and a solution $S_a$, if $S_a$ is not optimal there is a sequence of interchange moves leading from $S_a$ to an optimal solution $\overline{S}_a^*$.*

**Proof.** The proof directly follows from Lemma 4.1 and Theorem 4.1. Starting from $S_a$ and given an optimal solution $\overline{S}_a^*$, Lemma 4.1 guarantees that there exists an pair of consecutive jobs that can be reversed and Theorem 4.1 guarantees that the resulting solution, say $\tilde{S}_a$, is feasible. If $\tilde{S}_a$ is optimal the thesis follows, otherwise the same procedure can be repeated starting from $\tilde{S}_a$, finally leading to an optimal solution. ∎

We have shown how an optimal sequencing for a given assignment $a$ can be reached starting from any solution. We next show that an optimal assignment can be reached starting form any assignment. To this aim, we need the following preliminary result.

**Theorem 4.4** *Given a solution $S$, a machine $M_k$, a tool $T_h$ and a job $x$ compatible with $M_k$ or $T_h$, if $x \notin \mathcal{J}(M_k)$ [if $x \notin \mathcal{J}(T_h)$] there always exists a rerouting move that moves $x$ to $\mathcal{J}(M_k)$ [respectively, to $\mathcal{J}(T_h)$].*

**Proof.** Theorem 4.2 provides sufficient conditions for acyclicity after a rerouting move. Therefore, to demonstrate the thesis, it is sufficient to show that, if $x$ is compatible with machine $M_h$ [with tool $T_k$], and given $\sigma_h$ [given $\pi_k$], there are always in $\sigma_k$ [in $\pi_k$] two consecutive jobs $z$ and $y$ such that $x$ can be inserted between $z$ and $y$.

We limit ourselves to prove this for $\theta^M(x,y)$ moves only, the proof for $\theta^M(x,y)$ moves being very similar. First observe that every arc in $\sigma_k$ always satisfies at least one of the two conditions in theorem 4.2. In fact, if the first condition is not true, then $(h(y)+p_y \leq h(PT_x))$, it follows that $h(z) < h(y)+p_y \leq h(PT_x) < (h(ST_x)+p_{ST_x})$. Similarly, if the second condition is not true, then $(h(ST_x)+p_{ST_x}) \leq h(z)$, which implies $(h(y)+p_y \geq h(z)+p_z+p_y > h(z) \geq h(ST_x)+p_{ST_x} > h(PT_x))$. In particular the second condition is always satisfied for node *start* and the first for at least an auxiliary node *end*, *viol_unav*, *viol_dead* or *viol_due*. Note also that the auxiliary nodes cannot satisfy the second condition. Then, in $\sigma_k$ there must be a last node $z$ such that $(h(ST_x)+p_{ST_x}) > h(z)$ holds and $(h(ST_x)+p_{ST_x}) > h(SM_z)$. Therefore, $y = SM_z$ must satisfy the condition $(h(y)+p_y > h(PT_x))$, which concludes the proof. ∎

**Theorem 4.5** *The neighborhood defined by moves $\varphi$ and $\theta$ is connected.*

**Proof.** Consider an optimal solution $S^*$. Theorem thm:resequence shows that starting from any solution $S_a$ it an be reached an optimal sequencing $S_a^*$ for the given assignment $a$. If $S_a^*$ is not globally optimal, there must be at least a job $x$ which is assigned to a different resource in $S_a^*$ and $S^*$. Theorem 4.4 guarantees that there exists a rerouting move that moves $x$ to the same machine as in $S^*$, thus leading to a new feasible solution $S'$. If $S'$ is optimal the thesis follows, otherwise the same procedure can be repeated starting from $S'$, finally leading to an optimal solution. ∎

## 4.3 Neighborhood exploration

At each step of the algorithm we restrict the interchange move to consecutive jobs ($x$ and $\overline{PM}_x$ or $PT_x$) on a critical path, i.e. a longest path from *start* to one of the other auxiliary nodes. This is a common strategy, e.g., when solving job shop scheduling problems [15]. In fact, as observed, e.g., in [20], resequencing consecutive jobs that are not on the longest path from *start* to *end* cannot improve the makespan. Specifically, we analyze a critical path from *start* to *viol_dead* when $h(viol\_dead) > 0$. Otherwise, we explore the paths from *start* to *end* and *viol_due*. The head of node *viol_unav* is always maintained equal to zero using the algorithm described in Section 4.4.2.

When dealing with the *rerouting* move $\theta^M(x, y)$, we restrict the choice of $x$ to the jobs or unavailabilities positioned on a critical path and $y$ to those nodes such that one of the two following conditions is satisfied:

- there is a strictly positive slack time $\Delta_y > 0$ before $y$, on the machine processing it, during which the machine is available and not busy;
- $y = ST_x$, i.e., $y$ is a job using the same tool of $x$, immediately after it.

Clearly, moving an unavailability $x \in \mathcal{U}$ from machine $M_h$ to machine $M_k$ corresponds to moving the operators from $M_k$ to $M_h$ during the corresponding shift.

### 4.3.1 Critical paths and extended critical paths

When there are unavailabilities in the graph representation $\mathcal{G}(S)$ of a solution $S$, we define two different critical paths. One is the classical longest path on $\mathcal{G}(S)$ from *start* to a specific auxiliary node, i.e. the path for which the sum of the arc and node weights is maximum. However, this path might be too short, as shown in the example of Figure 4.

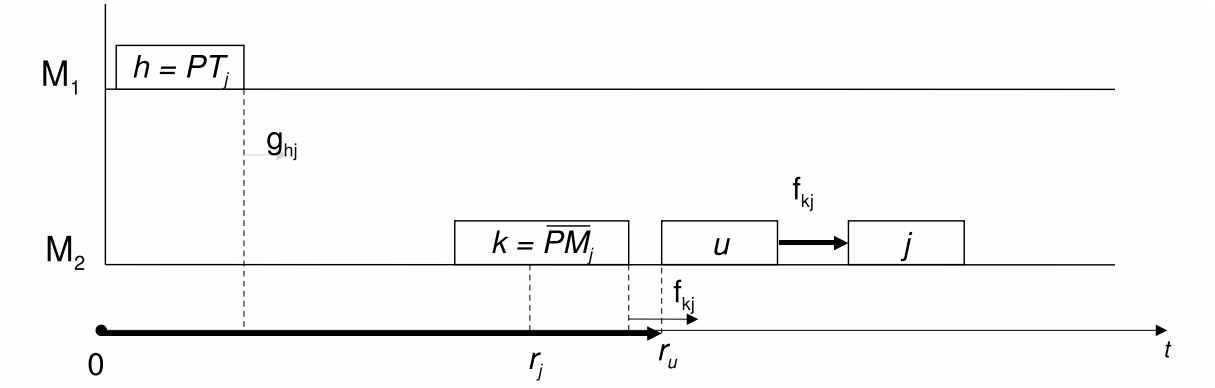


Figure 4: Unavailability $u$ belonging to a critical path

In this case, the unavailability $u$ is followed by the job node $j$, associated to job $J_j$. Arc $(u, j)$ is weighted with the setup between $J_k = \overline{PM}_j$ and $j$, which does not fit between $c_k$ and $r_u$. Hence, $h(j) = D_u + f_{J_k J_j}$, and $h(j) > c_k + f_{J_k J_j} + p_u$. Moreover, let us assume that $c_j > D_j$ and that the deadline violation $c_j - D_j$ is the largest among all the other jobs. It follows that $\mathcal{G}(S)$ contains a single critical path from *start* to *viol_dead*, shown in Figure 5, including only $u$ and $j$ as intermediate nodes.

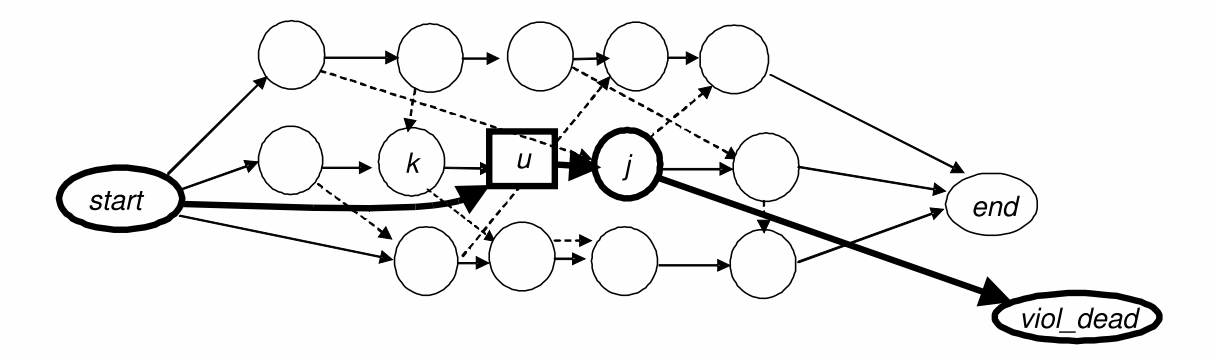


Figure 5: Critical path on a graph with the classical definition

According to the classical definition of critical path, the only way to recover the deadline violation in the example of Figure 5 is to reroute either $u$ or $J_j$. However, the starting time of job $J_j$ might be anticipated as well by exchanging $J_k$ and $J_j$ or by rerouting $J_k$. In other words, incorporating in the critical path the path from *start* to node $k$, would allow a larger possibility to improve upon the incumbent solution.

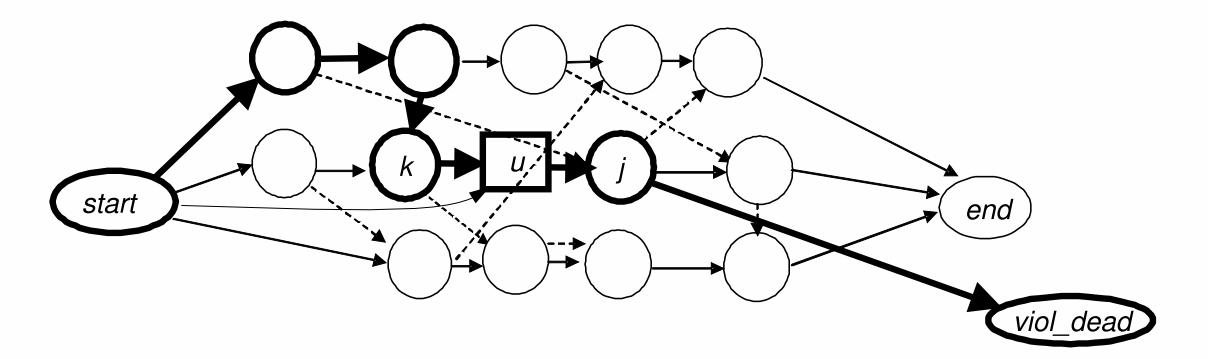


Figure 6: An extended critical path

We call the new path shown in Figure 6, an *extended critical path*. More formally, the extended critical path includes all the nodes on the critical path and, when the first node is an unavailability $u$, the critical path from *start* to $PM_u$.

## 4.4 Move evaluation

In the neighborhood of the incumbent solution $S$ we look for a solution $S'$ minimizing the following penalty function.

$$\begin{aligned} f(S') = \quad & \alpha * max\{0, h(viol\_dead)\} \quad + \\ & \beta * max\{0, h(viol\_due)\} \quad + \\ & \gamma * h(end). \end{aligned} \tag{7}$$

We developed two alternative algorithms for evaluating $f(S')$ after a move. The first one gives an estimate of $f(S')$ in constant time, while the second provides the exact value of $f(S')$ in linear time. At each step of the tabu search, the neighbor with the smallest value of $f(S')$ is selected as the new incumbent, and job precedences are updated accordingly.

Then, following the algorithm of Section 4.4.2, job starting and completion dates are updated in $\mathcal{G}(S)$. We next describe the two evaluation algorithms.

### 4.4.1 Approximate evaluation

The approximate evaluation of $f(S')$ consists of using heads and tails of $\mathcal{G}(S)$ to estimate the maximal length of the longest paths from *start* to auxiliary nodes *end*, *viol_dead*, and *viol_due*. We address each move separately, and use the notation $h(x)$ and $q_a(x)$ [respectively, $h'(x)$ and $q'_a(x)$] to define the head and the tail of job $x$ to node $a \in \{end, viol\_dead, viol\_due\}$ in $\mathcal{G}(S)$ [in $\mathcal{G}(S')$]. With respect to move $\varphi(x, y)$, the estimate $Z'_a$ of the length of a longest path in $\mathcal{G}(S')$ from *start* to auxiliary node $a$ is computed as an estimate of the longest path passing through $x$ or $y$:

$$Z'_a = max\{h'(x) + q'_a(x), h'(y) + q'_a(y)\}. \tag{8}$$

As for the solutions obtained with the $\varphi^M(x, y)$ move, we have:

$$\begin{array}{rcll}
h'(y) & = & max\{ & h(PM'_y) + p_{PM'_y} + f_{PM'_y y}, \\
 & & & h(PT_y) + p_{PT_y} + g_{PT_y y},\ r_y \quad \} \\
h'(x) & = & max\{ & h'(y) + p_y + f_{yx}, \\
 & & & h(PT_x) + p_{PT_x} + g_{PT_x x},\ r_x \quad \} \\
q'_a(x) & = & max\{ & p_x + f_{xSM'_x} + q_a(SM'_x), \\
 & & & p_x + g_{xST_x} + q_a(ST_x) \quad \} \\
q'_a(y) & = & max\{ & p_y + f_{yx} + q'_a(x), \\
 & & & p_y + g_{yST_y} + q_a(ST_y) \quad \}.
\end{array}$$

Observing that $PM'_y = PM_x$ and $SM'_x = SM_y$, these four quantities can be computed in constant time using the information available from $\mathcal{G}(S)$. Similarly, for move $\varphi^T(x, y)$ we have:

$$\begin{array}{rcll}
h'(y) & = & max\{ & h(PM_y) + p_{PM_y} + f_{PM_y y}, \\
 & & & h(PT'_y) + p_{PT'_y} + g_{PT'_y y},\ r_y \quad \} \\
h'(x) & = & max\{ & h(PM_x) + p_{PM_x} + f_{PM_x x}, \\
 & & & h'(y) + p_y + g_{yx},\ r_x \quad \} \\
q'_a(x) & = & max\{ & p_x + f_{xSM_x} + q_a(SM_x), \\
 & & & p_x + g_{xST'_x} + q_a(ST'_x) \quad \} \\
q'_a(y) & = & max\{ & p_y + f_{ySM_y} + q_a(SM_y), \\
 & & & p_y + g_{yx} + q'_a(x) \quad \}.
\end{array}$$

Observing that $PT'_y = PT_x$ and $ST'_x = ST_y$, these four quantities can be computed in constant time using the information available from $\mathcal{G}(S)$. As for move $\varphi^{MT}(x, y)$ we have:

$$\begin{array}{rcll}
h'(y) & = & max\{ & h(PM'_y) + p_{PM'_y} + f_{PM'_y y}, \\
 & & & h(PT'_y) + p_{PT'_y} + g_{PT'_y y},\ r_y \quad \} \\
h'(x) & = & max\{ & h'(y) + p_y + max\{f_{yx}, g_{yx}\},\ r_x \quad \} \\
q'_a(x) & = & max\{ & p_x + f_{xSM'_x} + q_a(SM'_x), \\
 & & & p_x + g_{xST'_x} + q_a(ST'_x) \quad \} \\
q'_a(y) & = & & p_y + max\{f_{yx}, g_{yx}\} + q'_a(x).
\end{array}$$

Observing that $PM'_y = PM_x$, $SM'_x = SM_y$, $PT'_y = PT_x$ and $ST'_x = ST_y$, these four quantities can be computed in constant time using the information available from $\mathcal{G}(S)$.

As far as the *rerouting* move $\theta^M(x,y)$ is concerned, we compute $Z'_a$ estimating the length of the longest path passing through $x$ or $y$ and the path passing through $PM_x$ and $SM_x$, i.e.:

$$\begin{array}{rl} Z'_a = max\{ & h'(x) + q'_a(x), h'(y) + q'_a(y), \\ & h(PM_x) + p_{PM_x} + f_{PM_xSM_x} + \\ & q_a(SM_x) \qquad\qquad\qquad\qquad \}. \end{array}$$

with

$$\begin{array}{rcll} h'(x) & = & max\{ & h'(PM'_x) + p_{PM'_x} + f_{PM'_xx}, \\ & & & h(PT_x) + p_{PT_x} + g_{PT_xx},\, r_x \quad \} \\ h'(y) & = & max\{ & h(x) + p_x + f_{xy}, \\ & & & h'(PT'_y) + p_{PT'_y} + g_{PT'_yy},\, r_y \quad \} \\ q'_a(y) & = & q_a(y) & \\ q'_a(x) & = & max\{ & p_x + f_{xy} + q_a(y), \\ & & & p_x + g_{xST_x} + q'_a(ST'_x) \qquad \}. \end{array}$$

We observe that $PM'_x = PM_y$, while for $PT'_y$ and $ST'_x$ there are two cases: if $PT_y = x$ then $h'(PT'_y) = h'(x)$ and $q'_a(ST'_x) = q'_a(y)$, otherwise $h'(PT'_y) = h(PT_y)$ and $q'_a(ST'_x) = q_a(ST_x)$. All these quantities are computed in constant time using the information available from $\mathcal{G}(S)$. Similarly, for $\theta^T(x,y)$ we have:

$$\begin{array}{rl} Z'_a = max\{ & h'(x) + q'_a(x), h'(y) + q'_a(y), \\ & h(PT_x) + p_{PT_x} + f_{PT_xST_x} + \\ & q_a(ST_x) \qquad\qquad\qquad\qquad \}, \end{array}$$

which can be computed in constant time by using similar arguments.

### 4.4.2 Exact evaluation

We first notice that the starting/completion time of every unavailability are fixed in any feasible solution. Therefore, moving a job from a machine to another and simply updating the heads of every nodes would lead to unfeasible solutions in most case, i.e. the head of at least one unavailability would be likely different from its release time. For this reason, after each move our algorithm keeps fixed the sequencing of the jobs and adjusts their starting/completion times taking into account unavailabilities. Our exact evaluation strategy executes this computation not only after the application of a move, but also for evaluating the neighborhood. We next show that these exact values can be computed in linear time if the nodes of $\mathcal{G}(S')$ are numbered according to a topological order.

**Theorem 4.6** *The exact evaluation of function f(S') after a move can be computed in time $O(n+q)$.*

**Proof.** Let $TO(j)$ be the position of node $j$ in a topological order of the nodes of $\mathcal{G}(S')$. We notice that, for both kinds of moves $\varphi(x,y)$ and $\theta(x,y)$, the starting times of the nodes preceding $\min\{TO(x), TO(y)\}$ remain unchanged when passing from $S$ to $S'$. For what concerns the other nodes, their exact starting times can be computed as follows. Let us first remove all unavailabilities from the topological order and then re-compute the starting time of each job one at the time, following the topological order.

For each job $J_j$ to be processed on $M_h$, let $s_j$ be a tentative starting time computed by taking into account its release time $r_j$ and its predecessors $PM_j$ and $PT_j$. If the setup times can be accommodated before the next unavailability on $M_h$, this starting time is accepted, otherwise the setup is performed after the unavailability and the starting time $s_j$ is updated consequently. Therefore, the exact starting time of $J_j$ can be computed in constant time. Given $s_j$, the completion time $c_j$ can be computed in linear time according to equation (3). Equation (3) requires computing the value $k$ by inserting the unavaliabilities $u_{i_l}$ one at a time, starting from $s_j$. Each $u_{i_l}$ is not considered again in subsequent computations. Since each job or unavailability is considered exactly once in the iterative computation of all job completion times, the overall evalutation requires a total of $O(n+q)$ steps, and the thesis follows. □

In Figure 7 the pseudocode of the overall algorithm is presented.

# 5 Computational Experiments

In this section we describe the performance of different versions of our tabu search algorithm, developed by varying the following parameters:

- the size of the neighborhood, choosing either the classical critical path (option A) or the extended one (option B), as described in Section 4.3.1,
- the estimate of the penalty function $f(S)$ of equation (7), choosing either the approximate evaluation (option C) or the exact one (option D), as described in Section 4.4,
- the tabu list length (option E)
- the number of non-improving iterations before applying a restart (option F)

As far as $\alpha, \beta, \gamma$ are concerned in the evaluation of the penalty function $f(S)$, we fixed the values $\alpha = 10^{10}$, $\beta = 1$, $\gamma = 10^5$ for all experiments. This corresponds to giving maximum priority to the respect of deadlines and then considering makespan and tardiness in lexicographical order, since a tardiness of two weeks is penalized less than increasing the makespan by one minute. In Section 5.2 we report on a preliminary set of experiments, carried out to generate the best configuration of the algorithm for the four pairs of options A-C, A-D, B-C and B-D, for varying D and E. In Section 5.3 we compare these four versions of our tabu search algorithm and draw several conclusions. All versions of the algorithm were coded in C language and were run on a Pentium® 4, 3.0 GHz with 1024 MB Ram. Two different sets of problem instances, described in the following section, were generated: the first set comes from the industrial practice and consists of 2 real instances. For these instances we can compare the actual performance attained at the plant when scheduling by hand and by computer. The second set of instances is randomly generated, in order to evaluate the performance of our algorithm in a more general context.

**Algorithm TABU_SEARCH**

**Input:**

$S$ current solution,
$S^*$ current optimal solution,
$\mathcal{G}(S)$ graph representation of $S$
$f(S)$ penalty of $S$
$cp^S(x)$ critical path in $\mathcal{G}(S)$ from *start* to node $x$
$\mathcal{N}(S)$ neighborhood
*iter* current iteration
$ITER\_MAX$ maximum number of iterations

**begin**
  *iter*:=0
  **while** $iter \leq ITER\_MAX$ **do**
  **begin**
    **if** $h(viol\_dead) > 0$
    **then** $x = viol\_dead$
    **else** $x = end$
    calculate $cp^S(x)$ ;
    construct the neighborhood $\mathcal{N}(S)$ (classical or extended);
    remove from $\mathcal{N}(S)$ all moves causing cycles
    **if** $\mathcal{N}(S) \neq \emptyset$ **then**
    **begin**
      remove from $\mathcal{N}(S)$ all tabu moves
      **if** $\mathcal{N}(S) \neq \emptyset$ **then**
      **begin**
        evaluate $\mathcal{N}(S)$ (approximate or exact)
        choose the best move $m$ in $\mathcal{N}(S)$
        apply $m$ to $S$ to obtain the new solution $S'$ and set $S = S'$;
        **if** $f(S^*) > f(S')$
        **then** set $S^* = S'$
      **end**
      **else** apply aspiration criterion
    **end**
    **else** perform a perturbation
    *iter*:=*iter*+1;
  **end**
**end**

Figure 7: Algorithmic scheme of TABU_SEARCH

## 5.1 Data set description

The first data set consists of two instances, corresponding to the real production plan for several weeks of production during September and October 2006. The first instance concerns with the production from Tuesday of the first week to Friday of two weeks later, for a total of 18 days during which 26 production orders are scheduled. The second instance concerns with the production from Thursday of the first week to Friday of two weeks later, for a total of 16 days during which 19 production orders are scheduled. In both instances, no urgent orders have to be scheduled and therefore no deadlines exist in these instances. All the jobs are available from the first day and the due dates are fixed equal to the end of the last Friday. Operators availability in each week allows to activate three blister lines from Monday to Friday for two shifts of 7 hours in each day, plus a short shift of three hours in which two blisters can be activated from Tuesday to Friday only.

The second data set consists of 120 random instances, obtained by generating 10 instances for each pair $(n, m)$, with the number of jobs $n$ varying in the set $\{20, 40, 60, 100\}$, and the number $m$ of parallel machines varying in $\{2, 3, 4\}$. The number $t$ of tools in each instance is set equal to $n$, each job is compatible with all machines, i.e., $|\mathcal{M}(J_j)| = m$, and for each job there is a single tool available, i.e., $|\mathcal{T}(J_j)| = 1$.

The duration of a shift is set to seven hours for each shift. Working days range from Monday to Friday, and in each day there are operators sufficient to activate $m$ machines in the first two shifts, from 6:00 a.m. to 8:00 p.m., and $m-1$ machines in the third shift, from 8:00 p.m. to 3:00 a.m. of the following day. There is therefore one unavailability of seven hours in the third shift compatible with all machines. Then, from 3:00 a.m. to 6:00 a.m., there are three hours in which all the machines are unavailable, due to the lack of personnel. The time horizon available is fixed equal to two weeks, i.e. to 30 shifts.

A tool is randomly assigned to each job, and therefore in each instance there are tools that are not used and tools that are shared among several jobs. For each job $J_j$, the values $r_j$, $d_j$, $D_j$ and $p_j$ are randomly generated. For 50% of jobs, release dates $r_j$ are set equal to the first hour of the time horizon while random values in the first week are assigned to remaining jobs. Similarly for 50% of jobs, due dates $d_j$ are set equal to the last hour of the time horizon. For 40% of remaining jobs, random due dates in the second week are generated. The last 10% of jobs have assigned a random deadline in the second week.

Processing and setup times are randomly generated with the aim of obtaining a minimum workload approximately equal to 90% of the total processing time available on all machines. The number $W$ of hours available for processing on all machines is $W = 7 * 10 * (3 * m - 1) = 210 * m - 70$. We reserve approximately 70% of $W$ to the job processing times by generating, for each job $J_j$, a processing time $p_j$ in the range $[1, 1.4 * \frac{W}{n}]$. For each pair of jobs $J_i$,$J_j$ setup times $f_{ij}$ are randomly generated in the range $[0.5, 0.4 * \frac{W}{n}]$ for $i \neq j$, and set equal to 0.2 for $i = j$. These values correspond to reserving approximately 20% of $W$ to the setup times.

## 5.2 Parameter analysis

In this section, we analyze the performance of different configurations of our tabu search algorithm varying the tabu list length $\lambda$ in the set $\{5, 10, 20\}$, and the number $\nu$ of non-improving moves examined before applying a restart strategy in the set $\{100, 500, 1000\}$.

We focus on a set of 12 instances from the previous section, choosing the first instance for each combinations of $(n, m)$. We set the time limit of the tabu search algorithm equal to 60 seconds.

Table 1 shows the results of these experiments. For each version of the algorithm, columns $C_{MAX}$ and $T_{MAX}$ describe the makespan and maximum tardiness (in hours) for each pair $(\lambda, \nu)$ obtained as average values on the 12 instances.

Table 1: Performance of the tabu search algorithm for varying $\lambda$ and $\nu$

| | | A-C | | A-D | | B-C | | B-D | |
|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $\nu$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ |
| 5 | 100 | **253.08** | **0** | 248.28 | 5.33 | 253.62 | 0.79 | 249.12 | 0 |
| 5 | 500 | 253.41 | 7.55 | 248.73 | 1.94 | 253.31 | 5.29 | 251.58 | 0 |
| 5 | 1000 | 254.52 | 9.43 | 248.89 | 0.07 | 253.93 | 1.31 | 250.91 | 5.15 |
| 10 | 100 | 253.43 | 0.31 | 247.34 | 9.98 | 253.84 | 8.16 | 248.90 | 0 |
| 10 | 500 | 254.03 | 1.86 | 246.90 | 9.17 | 253.30 | 7.02 | 248.35 | 0.9 |
| 10 | 1000 | 253.74 | 0 | 247.31 | 7.77 | 253.62 | 4.32 | **248.87** | **0** |
| 20 | 100 | 255.10 | 9.94 | 246.96 | 0.65 | **254.38** | **0** | 248.11 | 4.37 |
| 20 | 500 | 254.39 | 4.97 | **246.85** | **0.65** | 254.95 | 9.68 | 249.22 | 1.7 |
| 20 | 1000 | 253.90 | 7.31 | 247.11 | 0.65 | 254.19 | 16.33 | 248.44 | 0.49 |

Table 1 clearly shows that the exact evaluation D yields better performance than the approximate one. For each of the four configurations, we selected the values of $\lambda$ and $\nu$ highlighted in boldface in the table to be used in the more extensive computational experiments reported in Section 5.3.

## 5.3 Computational results

In this section we report on our computational experience on the real and random instances described in Section 5.1. We compare the four versions A-C, A-D, B-C, B-D using the values of $\lambda$ and $\nu$ obtained from Section 5.2.

### 5.3.1 Real Instances

We carried out a comparison with the existing system by asking the human scheduler of the packaging department to produce by hand the schedules for several weeks. We obtained the two instances described in Section 5.1 with the corresponding schedules produced by hand. We compared these schedules with those produced automatically and asked the human scheduler to comment on the quality of the solutions produced by the tabu search algorithm. In both cases the computerized schedules were considered feasible and of good quality. These results are presented in Table 2. For each instance, we report on the makespan and the maximum tardiness for the manual solution and for the four configurations of the algorithm selected in Section 5.2. We also report on the number of iterations needed by the tabu search algorithms to find the best solution within one minute of computation, and the corresponding time, expressed in seconds. We do not know exactly the computation time needed by the human scheduler to develop a solutions, which was however strictly larger than one hour.

Table 2: Comparison between manual and computerized schedules

| | Manual | | A-C | | A-D | | B-C | | B-D | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ |
| 1 | 408.75 | 0 | 377 | 0 | 377 | 0 | 377 | 0 | 377 | 0 |
| 2 | 440.25 | 0 | 424.25 | 0 | 424 | 0 | 424 | 0 | 424 | 0 |
| | *Iter.* | *Time* | *Iter.* | *Time* | *Iter.* | *Time* | *Iter.* | *Time* | *Iter.* | *Time* |
| 1 | – | $> 3600$ | 264 | 4 | 2 | $< 1$ | 52 | $< 1$ | 2 | $< 1$ |
| 2 | – | $> 3600$ | 806 | 11 | 78 | 1 | 10 | $< 1$ | 239 | 3 |

We notice that every versions of the tabu search algorithm clearly outperform the manual schedules, with reference to both makespan and computation time. Similar solutions are always found within a few seconds, while the human scheduler needed more than one hour to find the manual schedules. Three versions of the tabu search algorithm find the same solutions, while the A-C version, i.e., the one using the classical definition of longest path and the approximate neighbors evaluation, is slightly less performing than the others.

### 5.3.2 Random Instances

The second set of instances consists of the 120 random instances described in Section 5.1. Table 3 shows the results of these experiments when using the four configurations of the tabu search algorithm selected in Section 5.2. For each version of the algorithm, columns $C_{MAX}$ and $T_{MAX}$ report the average values of makespan and maximum tardiness, respectively, on the 10 instances associated to each pair $(n, m)$.

Table 3: Performance of the tabu search algorithm for varying $n$ and $m$

| | | A-C | | A-D | | B-C | | B-D | |
|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ | $C_{MAX}$ | $T_{MAX}$ |
| 20 | 2 | 246.26 | 10.65 | 242.47 | 0.19 | 246.34 | 1.58 | 240.91 | 0.52 |
| 40 | 2 | 246.33 | 8.27 | 243.29 | 3.45 | 247.68 | 11.63 | 241.36 | 0.75 |
| 60 | 2 | 242.97 | 5.17 | 239.13 | 8.48 | 245.20 | 5.78 | 237.34 | 0 |
| 100 | 2 | 271.31 | 6.55 | 268.38 | 0 | 271.97 | 26.14 | 267.68 | 0 |
| 20 | 3 | 256.86 | 14.13 | 250.82 | 6.66 | 252.53 | 10.67 | 244.06 | 4.17 |
| 40 | 3 | 254.81 | 7.92 | 250.98 | 0 | 258.12 | 7.42 | 248.34 | 0 |
| 60 | 3 | 250.07 | 1.67 | 246.51 | 0 | 249.90 | 3.45 | 243.57 | 1.22 |
| 100 | 3 | 259.19 | 5.64 | 254.69 | 2.71 | 259.97 | 9.35 | 252.77 | 6.94 |
| 20 | 4 | 250.88 | 15.64 | 245.44 | 6.37 | 256.94 | 4.51 | 243.35 | 0 |
| 40 | 4 | 250.64 | 0 | 244.16 | 2.60 | 251.47 | 2.62 | 242.36 | 1.09 |
| 60 | 4 | 240.45 | 2.72 | 234.04 | 8.70 | 240.57 | 0.40 | 232.04 | 10.24 |
| 100 | 4 | 252.25 | 0 | 248.11 | 11.23 | 253.35 | 3.43 | 247.12 | 0.43 |
| Average | | 251.71 | 6.16 | 247.21 | 4.24 | 252.67 | 7.31 | 244.95 | 2.13 |

A few comments are in order.

- The combination B-D clearly outperforms the others with respect to both makespan and maximum tardiness. Specifically, computing the exact value of the objective

function for all neighbors provide much better results than using an approximate evaluation.

- Using extended paths provides better results than using the classical paths only in combination with the exact evaluation. When using the approximate evaluation, better performance are provided by the classical path. In other words, exact evaluation performs better with a larger neighborhood, while approximate evaluation performs best with a smaller neighborhood, the improvements being reached after several restart phases.

- It is interesting analyzing the time needed by the algorithm to reach the best solution found within one minute of computation. With combinations A-C and A-D the best solutions are found after 23.57 and 20.91 seconds, respectively. With combinations B-C and B-D the best solutions are found after 27.71 and 24.12 seconds, respectively. We notice that using the extended path requires larger computation time to find the best solution, probably due to the fact that the larger number of solutions to explore in each iteration causes a larger time spent to evaluate all the solutions.

- The average number of iterations performed by the four versions of the algorithm to find the best solutions are 6630 for combination A-C, 7158 for B-C, 3632 for A-D and 4122 for B-D. Clearly, the exact evaluation is much more time consuming than the approximate evaluation, requiring an average of 5.8 milliseconds to execute a single iteration, while with the approximate evaluation only 3.7 milliseconds are needed. However, the smaller number of iterations needed to reach the best solution makes the exact evaluation more efficient, other than more effective.

# 6 Conclusions

In this paper, we studied a practical scheduling problem arising in the packaging department of a pharmaceutical production plant. The problem is formulated as a parallel machine problem with sequence-dependent setup times, machine unavailability and with additional resources. A tabu search algorithm is able to find very good solutions on all real-life instances within short computation time, compared to the solutions found by hand. Several variants of the tabu search algorithm have been compared on randomly generated instances, showing that exact evaluation of the quality of each solution in the neighborhood provides much better results than a fast approximate evaluation, and that a larger neighborhood outperforms the smaller one. On the other hand, when dealing with the approximate evaluation of the neighborhood, a more randomized search, attainable with the small neighborhood and restart strategy, is better than searching within a larger neighborhood. These results confirm that scheduling technology is mature to solve real problems. In fact, there is a clear trend in the academic literature towards richer scheduling models, but practical scheduling problems still include more details than the typical academic ones. This is a research direction that needs further development. From the algorithmic point of view, future research should address the development of more sophisticated tabu search algorithms, including different neighborhoods, aspiration criteria, granularity and dynamic tabu lists.

# References


[1] O. Braysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[2] O. Braysy and M. Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[4] W.C. Chiang and R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[5] S. Dauzère-Pérès, W. Roux, and J.B. Lasserre. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305, 1998.

[6] K.H. Ecker. Scheduling of resource tasks. *European Journal of Operational Research*, 115(2):314–327, 1999.

[7] P. M. Franca, M. Gendreau, G. Laporte, and F. M. Muller. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43:78–89, 1996.

[8] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP completeness*. Freeman, 1979.

[9] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and operations research*, 13:533–549, 1986.

[10] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic machine scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[12] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Project scheduling: Recent model, algorithms and applications*, pages 147–178. Kluwer, 1999.

[13] C. Y. Lee, L. Lei, and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997.

[14] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems*. John Wiley & Sons, 1993.

[15] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop scheduling problem. *Management Science*, 42:797–813, 1996.

[16] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.

[17] M. Pinedo. *Scheduling. Theory, algorithms, and systems.* Prentice-Hall, 1995.

[18] M. Pinedo. *Planning and Scheduling in Manufacturing and Services.* Springer, 2005.

[19] J. M. J. Schutten and R. A. M. Leussink. Parallel machine scheduling with release dates, due dates and family setup times. *International Journal of Production Economics*, 46:119–125, 1996.

[20] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[21] K. P. White and R. V. Rogers. Job-shop scheduling: limits of the binary disjunctive formulation. *International Journal of Production Research*, 28:2187–2200, 1990.