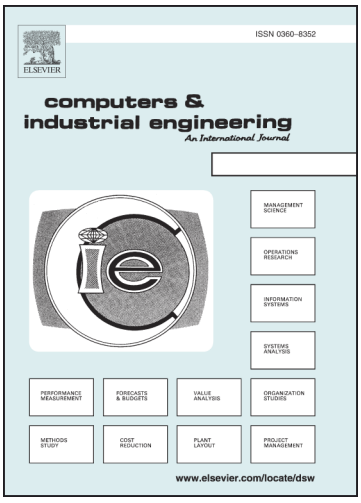
Accepted Manuscript



Metaheuristics for the Job-Shop Scheduling Problem with Machine Availability Constraints

Karim Tamssaouet, Stéphane Dauzère-Pérès, Claude Yugma

PII: S0360-8352(18)30380-2
DOI: https://doi.org/10.1016/j.cie.2018.08.008
Reference: CAIE 5357

To appear in: *Computers & Industrial Engineering*

Received Date: 15 November 2017
Accepted Date: 6 August 2018

Please cite this article as: Tamssaouet, K., Dauzère-Pérès, S., Yugma, C., Metaheuristics for the Job-Shop Scheduling Problem with Machine Availability Constraints, *Computers & Industrial Engineering* (2018), doi: https://doi.org/10.1016/j.cie.2018.08.008

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Metaheuristics for the Job-Shop Scheduling Problem with Machine Availability Constraints

Karim Tamssaouet[1,2] Stéphane Dauzère-Pérès[1,3] Claude Yugma[1]

[1]Mines Saint-Etienne, Univ. Clermont Auvergne
CNRS, UMR 6158 LIMOS
CMP, Department of Manufacturing Sciences and Logistics
F-13541 Gardanne, France
E-mail: karim.tamssaouet@emse.fr, dauzere-peres@emse.fr, yugma@semse.fr

[2]STMicroelectronics Rousset,
F-13106 Rousset, France

[3]Department of Accounting, Auditing and Business Analytics,
BI Norwegian Business School,
0484 Oslo, Norway

Abstract

This paper addresses the job-shop scheduling problem in which the machines are not available during the whole planning horizon and with the objective of minimizing the makespan. The disjunctive graph model is used to represent job sequences and to adapt and extend known structural properties of the classical job-shop scheduling problem to the problem at hand. These results have been included in two metaheuristics (Simulated Annealing and Tabu Search) with specific neighborhood functions and diversification structures. Computational experiments on problem instances of the literature show that our Tabu Search approach outperforms Simulated Annealing and existing approaches.

Keywords: Scheduling, Job Shop, Metaheuristic, Availability Constraints, Disjunctive Graph

1. Introduction

In scheduling theory, the vast majority of studies assumes that resources are continuously available for processing throughout the scheduling horizon. However, in many realistic situations, machines may be unavailable during certain periods for different reasons, such as failures or unexpected quality control problems. To avoid these failures without stopping the production too often, preventive maintenance tasks are planned on machines by trading off between planned unproductive downtimes and the risk of unscheduled downtimes due to machine failures. These preventive maintenance activities make machines unavailable for processing operations. As this work deals with deterministic scheduling, only unavailability periods that are known in advance are considered such as preventive maintenance or machine unavailabilities that are due to previous scheduling decisions within a rolling horizon framework.

The continuous availability of machines during the whole scheduling horizon is an assumption that might be justified in some cases but cannot apply to all industrial settings. Semiconductor manufacturing is an example where it is important to consider machine availability constraints. In this industry, machines are complex, thus requiring frequent preventive maintenance, and very expensive, thus must be used as much as possible (Bureau et al. (2006)). Also, due to the complexity of scheduling problems in

ACCEPTED MANUSCRIPT

this industry, a rolling horizon procedure is necessary to decompose the problem over time (Ovacik and Uzsoy (2012)). When a scheduling problem is solved, some of the decisions from a previous schedule have to be considered, making some machines unavailable at the beginning of the horizon for the newly available jobs. It is then important to consider these machine unavailabilities in order to produce robust and feasible schedules.

Surveys of scheduling problems with machine availability constraints can be found in Schmidt (2000) and Ma et al. (2010). Here, we only provide an overview of previous studies on scheduling problems with unavailability periods. Then, some relevant details are given for two papers that are used to evaluate our approach. In the research studying the integration of availability constraints in scheduling problems, different ways of modeling availability constraints have been proposed and investigated in different machine environments. Regarding the possibility for an operation to be interrupted by an unavailability period, Lee (1999) introduces the *semi-resumable* case that includes two cases: (i) *Resumable*, where the operation, after being interrupted, can be continued and without any penalty after the end date of the unavailability period and (ii) *Non-resumable*, where the operation needs to be fully restarted. A pseudo-polynomial dynamic programming algorithm and heuristic algorithms with an error bound analysis were developed to solve the two-machine flow-shop problem with availability constraints. Aggoune (2002) introduces a fourth case for an operation to be interrupted by an unavailability, called *non-preemptive*, to model the situation where an operation can be interrupted neither by another operation nor by an unavailability period. Based on a polynomial algorithm that solves the two-job job-shop scheduling problem, Aggoune (2002) proposes a heuristic and a branch-and-bound method with a lower bound to solve the general job-shop scheduling problem. In Benttaleb et al. (2016), a two-machine job-shop scheduling problem with availability constraint on one machine is considered. In the non-preemptive case and the deterministic case, some propositions on the optimality of Jackson's algorithm in the presence of availability constraints are stated. Then, a branch-and-bound algorithm is developed and proved to be more efficient than two Mixed Integer Programs. Aggoune et al. (2009) propose a polynomial algorithm for solving two-job job-shop scheduling problems with an arbitrary number of unavailability periods on each machine. Mauguière et al. (2005) introduce another terminology for unavailability periods allowing operations to be interrupted: *Crossable* and *Non crossable*. They propose a branch-and-bound algorithm for the single-machine and job-shop scheduling problems in which only some unavailability periods are crossable and some jobs are resumable, other periods are non-crossable and other jobs are non-resumable.

When unavailability constraints are due to preventive maintenance, two additional cases can be distinguished. The first case, called *deterministic*, is when maintenance periods are fixed in advance. The second case, called *flexible*, corresponds to the situation where maintenance periods also have decision variables, i.e. each maintenance period has to be scheduled in a given time window. Azem et al. (2012) propose heuristics for the job-shop problem with resource availability constraints where preemption between operations and unavailability periods are allowed and unavailability periods are flexible. Gao et al. (2006) tackle the flexible job-shop scheduling problem with non-fixed availability periods using genetic algorithms. In Zribi et al. (2008), a hierarchical approach is proposed that first solves the assignment problem and then the sequencing problem in the multipurpose machine job-shop scheduling problem. In Cui et al. (2016), two Mixed Integer Programs are given for the deterministic and the flexible cases for the non-permutation flow-shop scheduling problem where jobs are non-resumable. In the flexible case, machines are constrained to have a continuous working time that does not exceed a maximum allowed working time. Then, a hybrid incremental genetic algorithm is proposed to solve large-sized problems.

In this paper, we investigate the integration of availability constraints in the job-shop scheduling problem as the latter can be generalized to real-life applications. The problem is NP-hard as it generalizes

ACCEPTED MANUSCRIPT

the classical job-shop scheduling problem. In order to evaluate our metaheuristics, we use the problem instances proposed by Azem et al. (2007) and Mati (2010). Azem et al. (2007) propose two Mixed Integer Linear Programming (MILP) models for the non-preemptive job-shop scheduling problem with deterministic and flexible unavailability periods. The first model is based on the disjunctive graph and the second model is time-indexed. New instances are generated and used to compare the two formulations and it was found that the disjunctive model outperforms the time-indexed model. Mati (2010) proposes a tabu thresholding heuristic to solve the non-preemptive job-shop scheduling problem with machine unavailability for makespan minimization. The metaheuristic uses a new block-based neighborhood function, in which the block concept is generalized to include the unavailability periods of machines. Some sufficient conditions are also proposed to eliminate non-improving moves that involve operations at the borders of unavailability periods. To evaluate the tabu thresholding heuristic, the instances of Azem et al. (2007) and new instances are used. To evaluate our approach, the instances in these two papers are used and the results are compared with those obtained by the disjunctive MILP of Azem et al. (2007) and those obtained by the tabu thresholding heuristic of Mati (2010).

We propose an efficient metaheuristic for the job-shop scheduling problem with availability constraints. As the introduction of availability constraints results in the loss of some classical properties of critical operations that are highlighted, a novel definition and a way of labeling operations as critical are given. A second contribution is the evaluation of a move without actually making it. This allows the size of the visited neighborhood to be significantly reduced by discarding non-improving moves. The experimental results in Section 5.1 show that over 76% of non-improving moves can be ignored when the proposed evaluation is used. These propositions are embedded in two metaheuristics: Simulated Annealing (SA) and Tabu Search (TS). These metaheuristics are chosen because they are known to be successful for the classical job-shop scheduling problem, e.g., Van Laarhoven et al. (1992) for simulated annealing, Taillard (1994) and Nowicki and Smutnicki (2005) for tabu search. The experimental results show that our Tabu Search obtains good results and outperforms Simulated Annealing, the tabu thresholding of Mati (2010) and the MILP of Azem et al. (2007) that was reimplemented.

The remainder of the paper is organized as follows. Section 2 formally describes the problem and recalls the disjunctive graph model and how it can be used to model job sequences and to deduce some properties. Section 3 presents the necessary building blocks to design an efficient metaheuristic. The operation properties that are lost when considering availability constraints are highlighted, and a novel definition of critical operations is given. Then, the proposed move evaluation that allows non-improving moves to be ignored is described. In Section 4, after a brief description of the construction heuristic and the Simulated Annealing (SA) metaheuristic that was initially implemented, a Tabu Search (TS) metaheuristic is described in more details. In Section 5, after analyzing the proposed move evaluation, the two proposed metaheuristics are compared to existing approaches on various test instances. Conclusions and some research directions are given in Section 6.

2. Problem Description and Modeling

In this paper, we investigate the integration of fixed and non-preemptive unavailability periods in a job-shop scheduling environment. We consider n jobs $J = \{J_1, J_2, ..., J_n\}$ to be processed on a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$ in order to minimize the makespan. Each job J_i is composed of a linear sequence (*routing*) of n_i operations $O_i = \{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$. Each machine can process only one operation at a time and each operation O_{ij} needs only one machine $m_{ij} \in M$ during an uninterrupted processing time of $p_{ij} > 0$ units. For each machine k, there are r_k unavailability periods $H_k = \{h_{k1}, h_{k2}, \ldots, h_{kr_k}\}$. The start date S_{kl} and the duration p_{kl} of each unavailability period h_{kl} are fixed and known in advance.

ACCEPTED MANUSCRIPT

As an extension of the classical job-shop scheduling problem, our problem can be modeled using a disjunctive graph $G = (V, A, E)$, where V is the set of nodes, A the set of *conjunctive* arcs and E the set of *disjunctive* arcs. Each node is associated to an operation, i.e. V is the set of operations $O = \bigcup_{1 \leq i \leq n} O_i$ plus the dummy start and end operations 0 and $*$. The conjunctive arcs in A model the conjunctive constraints between two consecutive operations on a routing, between 0 and every first operation on a routing, and between every last operation on a routing and $*$. The disjunctive arcs in E_k model the disjunctive constraints between pairs of operations that must be processed on the same machine k, and $E = \bigcup_{1 \leq k \leq m} E_k$. Disjunctive arcs ensure that linked operations cannot be processed on the same machine simultaneously. The length of an arc from the source node to the first operation of each job is equal to 0, and the length of each remaining arc is equal to the processing time of the operation from which the arc starts.

A selection is obtained by fixing a direction for each disjunctive arc in E, i.e. by replacing every disjunctive arc with a conjunctive arc. The resulting selection corresponds to a feasible sequence of operations on the machines if and only if it induces an acyclic directed (conjunctive) graph. As a feasible sequence generates feasible schedules, this graph representation is used to compute the left justified schedule using the lengths of the conjunctive arcs. Before explaining how schedules are computed with availability constraints, some additional notations are necessary and notions used in other scheduling problems are recalled.

Let v be a node (operation) in the directed graph associated to an operation $O_{ij} \in O$. Let pr_v and fr_v denote, respectively, the operations preceding and following v in the routing of its associated job. Similarly, let ps_v and fs_v denote, respectively, the operations preceding and following v in the sequencing of its machine. In a conjunctive graph that represents a solution for a classical job-shop scheduling problem, a finite sequence of conjunctive arcs between two operations is called a *path*. The *length* of a path is equal to the sum of the lengths of the arcs on the path. When deterministic unavailability periods are introduced, this definition is no longer valid as this path may contain, in addition to the arc weights, some unavailability periods and a forced idle time that is due to non-preemption. Section 3.1 shows how unavailability periods are considered when computing path lengths. A *longest* path in the graph between two operations is a path with maximum length. Let $L(u, v)$ be the length of the longest path between operation u and v. $S_v = L(0, v)$ denotes the *earliest start date (head)* of v and $q_v = L(v, *) - p_v$ denotes its *delivery time (tail)*. Because of non-preemption, the *earliest completion date* of an operation v is $C_v = S_v + p_v$. A longest path between the start and finish operations 0 and $*$ is called a critical path. Its length $L(0, *)$ is equal to the minimum amount of time required to complete all jobs, i.e. the *makespan*. Operations and arcs on critical paths are called *critical operations* and *critical arcs*, respectively. The necessary and sufficient condition $C_{max} = S_v + p_v + q_v$ is a way to establish the *criticality* of a node (operation) v. Finally, a *block* consists of a maximum sequence of adjacent critical operations that are processed on the same machine.

3. Building Blocks of the Metaheuristics

3.1. Computing a Longest Path

As mentioned above, a conjunctive graph representation is used to compute heads and tails of the operations (Roy and Sussmann (1964)). However, because the unavailability periods are not explicitly modeled in the graph, the classical longest path calculation algorithm is not sufficient to obtain accurate dates. In Mauguière et al. (2005), four recursive functions are defined to determine the exact values of heads and tails in the resumable and non-resumable cases. In the case of the non-preemptive case, two recursive functions are required.

ACCEPTED MANUSCRIPT

The first one $esd(t_v, p_v, k, l)$, detailed in Algorithm 1, returns the earliest start date, i.e. the head, of operation v. The parameter t_v denotes the earliest start date of operation v that can be different from the final one. The initial value of this parameter is determined by the earliest completion times of its predecessors, i.e., $t_v = max\{C_{pr_v}, C_{ps_v}\}$. If the execution of v overlaps with an unavailability period, this function postpones its start time until non-preemptive constraints are satisfied. Also, p_v denotes the processing time of v, k the machine on which v is processed and l the index of an unavailability period of machine k that the recursive function checks if it overlaps with the processing of operation v. This recursive function is then used in the forward pass to adjust and determine accurate earliest start dates. Algorithm 3 sketches the graph traversal in the topological order and the use of the recursive function during the forward pass.

The second recursive function $lcd(t_v, p_v, k, l)$ is detailed in Algorithm 2. It returns the exact latest completion date of an operation v, which allows computing the tail q_v by subtracting the returned value from the makespan. The parameter t_v denotes the latest start date of operation v that can be different from the final one. The initial value of this parameter is determined by the latest start times of its successors, i.e., $t_v = C_{max} - max\{q_{fr_v} + p_{pr_v}, q_{fs_v} + p_{ps_v}\}$. Also, p_v denotes the processing time of v, k the machine on which v is processed and l the index of an unavailability period of machine k that the recursive function checks if it overlaps with the processing of operation v. By symmetry to the first recursive function, the latest completion date of the operation v is advanced as soon as there is an overlap between its processing interval and an unavailability period h_{kl}. This recursive function is used during the backward pass to adjust and determine accurate latest completion dates that are used to compute tails.

Algorithm 1 Recursive function to adjust earliest start date

$esd(t_v, p_v, k, l)$
if $t_v \geq S_{kl} + p_{kl}$
    return $esd(t_v, p_v, k, l+1)$
else if $S_{kl} \leq t_v < S_{kl} + p_{kl}$
    return $esd(S_{kl} + p_{kl}, p_v, k, l+1)$
else if $t_v < S_{kl}$ **and** $t_v + p_v > S_{kl}$
    return $esd(S_{kl} + p_{kl}, p_v, k, l+1)$
else
    return t_v

Algorithm 2 Recursive function to adjust latest completion date

$lcd(t_v, p_v, k, l)$
if $t_v \leq S_{kl}$
    return $lcd(t_v, p_v, k, l-1)$
else if $S_{kl} < t_v \leq S_{kl} + p_{kl}$
    return $lcd(S_{kl}, p_v, k, l-1)$
else if $t_v > S_{kl} + p_{kl}$ **and** $t_v - p_v < S_{kl} + p_{kl}$
    return $lcd(S_{kl}, p_v, k, l-1)$
else
    return t_v

ACCEPTED MANUSCRIPT

Algorithm 3 Forward Pass: Earliest start and completion dates

ComputeStartDate(G)
$S_O \leftarrow 0$
for $v \in$ *ComputeTopologicallyOrdering*$(G \setminus \{0\})$
  $t_v \leftarrow \max\{C_{pr_v}, C_{ps_v}\}$
  $S_v \leftarrow esd(t_v, p_v, m_v, 1)$
  $C_v \leftarrow S_v + p_v$

3.2. Determining Critical Operations

In addition to the computation of heads and tails that should be adapted when introducing availability constraints in the scheduling problems, it is actually also necessary to define another way to label operations as critical. As mentioned above, $C_{max} = S_v + p_v + q_v$ is a necessary and sufficient condition for an operation v to be critical in classical scheduling problems. In this case, critical operations have identical earliest and latest start dates, in other words, they do not have any *slack*, i.e. there is no margin on their start dates. Therefore, in addition to being a maximal succession of critical operations on the same machine, the block concept ensures the absence of idle times on the related machine.

After analyzing the impact of introducing availability constraints, it appears that labeling an operation with zero slack can be misleading. Considering availability constraints makes the property of having a zero slack neither a necessary nor a sufficient condition for an operation to be critical. The example in Figure 1 is used to illustrate the loss of these properties. In this example, three jobs, each with two operations, must be scheduled on three machines. Represented by red rectangles, Machine M_1 is unavailable in the periods $h_{1,1} = [4,6]$ and $h_{1,2} = [11,12]$. In the Gantt Chart, operation $O_{i,j}$ refers to operation i of job j. In this solution, there is one critical path that contains operations $O_{1,2}$, $O_{2,2}$ and $O_{2,3}$. If Operation $O_{2,3}$ starts it processing just after the completion time of Operation $O_{2,2}$, it cannot be completed before the start of the unavailability period $h_{1,2}$. Its start time is then delayed to time 12. Because of the availability constraint, the critical operation $O_{2,2}$ has a slack of one time unit. Note that swapping these two operations leads to a better solution with a makespan of 14. This example shows that a zero slack is not a necessary condition for an operation to be critical.

Operation $O_{1,1}$ is not critical as it does not belong to the critical path. In the classical job-shop scheduling problem, because it is a non-critical operation, $O_{1,1}$ can be delayed by its strictly positive slack without increasing the makespan. When availability constraints are considered, delaying this non-critical operation by only one time unit increases the makespan by two time units. Due to the unavailability period $h_{1,1}$, delaying the start time of $O_{1,1}$ by one unit makes it impossible for this operation to be completed before the start of the unavailability period. $O_{1,1}$ will then start its processing after the end of the unavailability period. This shows that an operation may have a zero slack without being a critical one, which means that this property is also not a sufficient condition.

As a zero slack is neither a necessary nor a sufficient condition, it cannot be used to identify and label critical operations. Instead, a critical operation is "only" an operation that belongs to the critical path, which is tracked during the backward pass. This is done by defining an operation v as critical if one of its successors w is critical and the completion time C_v is the largest among those of the predecessors of w.

ACCEPTED MANUSCRIPT

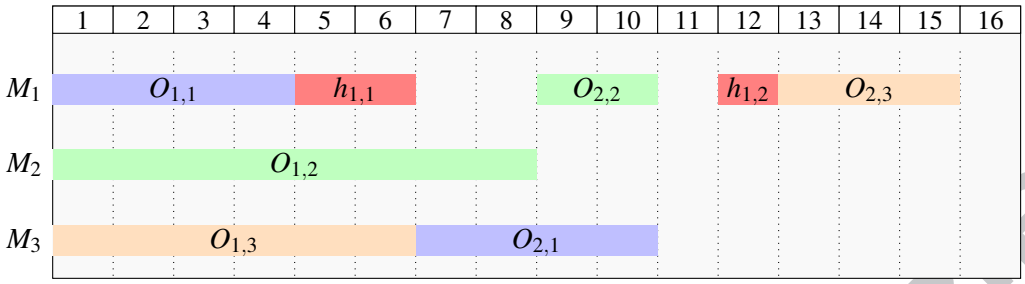


Figure 1: Example of a non critical operation $O_{1,1}$ without *slack*.

3.3. Neighborhood Function

A neighborhood function (or operator) typically provides a set of solutions that are enabled through moves from a currently feasible solution to new and hopefully better feasible solutions. A neighbor solution in our approach is a solution obtained by permuting two successive and critical operations on the same machine. This *swap* move is among the most basic neighborhood definitions that have been proposed for machine scheduling problems. Different reasons motivate this choice. Van Laarhoven et al. (1992) prove two important properties of this neighborhood function. First, there is no need to check for the feasibility of solutions obtained with the related moves since they are always feasible. Second, this neighborhood function is *connected*, which means that an optimal solution can be reached starting from any initial solution via a sequence of moves. However, the permutation of non-critical operations cannot improve the objective function and may create a directed cycle in the graph, i.e. an unfeasible solution. As these properties hold when considering availability constraints, this neighborhood function is the one used in this work.

3.4. Evaluating Moves

The set of moves will typically be quite large and calculating the makespan for all neighbors of a solution can be very time-consuming. Therefore, one can conduct preliminary neighbor evaluations to identify promising ones, as in Taillard (1994) for the classical job-shop scheduling problem and Dauzère-Pérès and Paulli (1997) for the flexible job-shop scheduling problem. Hence, to assess the effectiveness of a given swap, the value of the new longest path containing one of the two operations associated to the move is used. The length of this path is a valid lower bound of the makespan of the new solution. Compared to most neighborhood operators, the swap move allows the exact value of the longest path to be computed quickly. Whereas it is calculated in $O(N)$ time for other neighborhood operators, the length of the longest path is computed in constant time in the case of the classical job-shop scheduling problem. If E is the length of such a path, its length is computed as in Equation (1) (Taillard (1994)). The symbol $\tilde{}$ is used for the data that are related to the solution resulting from the arc swap.

$$E = \max\left(\tilde{S}_u + p_u + \tilde{q}_u, \tilde{S}_v + p_v + \tilde{q}_v\right) \tag{1}$$

The same reasoning can be applied to solve our problem. An estimation of the length of the new longest path containing one of the re-sequenced operations can be used to formulate the condition for a move to be accepted. This condition is given in (2) and interpreted as follows: Making a swap (u, v) in the case of the job-shop scheduling problem with availability constraints is considered to be a promising move only if the estimation of the length of the longest path E is smaller than the current makespan C_{max}.

ACCEPTED MANUSCRIPT

$$E < C_{max} \tag{2}$$

The heads and tails have to be adjusted to consider the unavailability periods and the non-preemption of the operations. Using the two recursive functions in Algorithms 1 and 2, the new heads and tails of the swapped operations are calculated as follows:

$$\tilde{S_v} = \text{esd}(max(C_{pr_v}, C_{ps_u}), p_v, m_v, 1) \tag{3}$$

$$\tilde{S_u} = \text{esd}(max(C_{pr_v}, \tilde{C_v}), p_u, m_u, 1) \tag{4}$$

$$\tilde{q_u} = C_{max} - \text{lcd}(C_{max} - max(q_{sm_v} + p_{sm_v}, q_{sr_u} + p_{sr_u}), p_u, m_u, |h_{m_u}|) \tag{5}$$

$$\tilde{q_v} = C_{max} - \text{lcd}(C_{max} - max(\tilde{q_u} + p_u, q_{sr_v} + p_{sr_v}), p_v, m_v, |h_{m_v}|) \tag{6}$$

Contrary to the classical job-shop scheduling problem, E is only an estimation of the path length, not its exact value, when availability constraints are considered. While the new earliest start times are exactly computed using (3) and (4), the tails computed with (5) and (6) are only estimations that can be larger or smaller than the exact values. This is due to the fact that the length of the longest path between operations v and $*$ is not only equal to the sum of the processing times of operations belonging to this path, but can also potentially contain unavailability periods and idle times before unavailability periods because of the non-preemption constraint. As operation processing times are fixed, advancing or delaying the start time of an operation does not change the length of the longest path between v and $*$ in the classical job-shop scheduling problem. In the presence of availability constraints, advancing the earliest start time of an operation can increase or decrease the length of the longest path between v and $*$ by increasing or decreasing idle times before unavailability periods, adding or removing unavailability periods to or from the path. This also applies when delaying the start time of an operation.

Instead of being in constant time, the computation of E, and consequently checking the improvement potential of a swap move, can be done in $O(r_k)$, where r_k is the number of unavailability periods on machine k on which operations u and v are processed. As the computation of longest paths is the most time-consuming part of a local search heuristic, this allows significant computational times to be saved by discarding uninteresting moves. As the length of the path is only an estimation, the risk of discarding improving moves should be taken into consideration. The computational results in Section 5.1 show that this risk is negligible, while the move evaluation can discard a significant number of non-improving moves.

4. Neighborhood-Based Metaheuristics

4.1. Initial Solution

To generate an initial solution, a simple construction heuristic is implemented. It starts with the conjunctive graph that only includes the conjunctive arcs in A, i.e. associated to the routing precedence constraints. At each iteration, the candidate operations are the ones without a route predecessor or whose predecessors are already scheduled. The operation with the minimum completion time is determined and scheduled on its machine. The selected operation is removed from the list of candidate operations and its route successor, if it exists, becomes a candidate operation. The complexity of this algorithm is $O(|O|^2)$. With such a simple priority rule-based dispatching heuristic, it is not expected to obtain good initial solutions. This is confirmed in the computational experiments. However, we believe an efficient metaheuristic should be able to converge towards very good solutions even it starts from a poor initial solution. Starting from a good initial solution should "only" help to accelerate the search.

ACCEPTED MANUSCRIPT

4.2. Simulated Annealing

Simulated annealing (SA) belongs to the class of randomized local search algorithms and has been developed to handle hard combinatorial problems. It was initially chosen to test the efficiency of the proposed neighborhood function because of its relative ease of implementation. However, the numerical results were not competitive enough. The analysis showed that its natural diversification mechanism is not strong enough if the neighborhood is restricted to the moves that satisfy Condition (2). To overcome this limitation, a diversification phase was added. This phase is triggered when the number of iterations since the last improvement is larger than a given maximum number *MaxIter*. Preliminary experiments show $MaxIter = (n+m)^2$ leads to good results. In this case, a given number of swaps of arcs on critical resources are randomly performed without considering the preliminary move evaluation. In the computational experiments, the number of iterations t during this phase is randomly picked in the interval $[0.8*(m*n), 1.2*(m*n)]$. This modification helped us to get significant improvements. However, it seemed more natural to apply the same idea in a tabu search metaheuristic which, as shown in Section 5, led to better results.

4.3. Tabu Search

Tabu Search (TS) was a natural choice because of its success for the classical job-shop scheduling problem (see e.g. Nowicki and Smutnicki (2005) and Zhang et al. (2007)). In Tabu Search, the best-allowed neighbor is always chosen. Whereas Simulated Annealing avoids being trapped in a local optimum by accepting non-improving moves with a certain probability, Tabu Search avoids cycling and escapes local optima by forbidding moves already performed. This is done within a *short-term memory* framework by keeping the forbidden (tabu) moves in a structure called *tabu list* (Glover and Taillard (1993)).

Tabu Search can be improved through varying levels of sophistication by incorporating more complex aspects, e.g. aspiration level criterion or long-term memory. Our implementation is rather simple as we only use the most basic idea of tabu search, i.e. the tabu list. The size of this list is crucial to the success of a tabu search metaheuristic. For a too small tabu list, the search will tend to cycle and visit the same solutions many times whereas, if it is too large, the lack of available moves may lead to the search being stuck. To avoid these problems, it is useful to modify the size of the tabu list during the search (Talbi (2009)). In our computational experiments, the size of the tabu list is randomly changed between m and n. The tabu list is used during the search which is divided into two phases: Intensification and diversification. In the intensification phase, only the neighbors which are generated by allowed and promising moves, i.e. those satisfying (2), are visited. When the set of these neighbors is empty, it is increased by considering all the allowed swaps of critical arcs without considering (2). This is the first level of diversification. If the set of neighbors is empty or if the number of iterations since the last improvement exceeds a given maximum number *MaxIter*, the second level of diversification starts. As for simulated annealing, good results are obtained with $MaxIter = (n+m)^2$. This second phase of diversification first randomly selects a number of iterations $t \in [0.8*(m*n), 1.2*(m*n)]$ and repeats the selection of moves for t iterations or until a new best solution is found. It is during this phase that the size of the tabu list is randomly changed.

5. Computational Experiments

This section presents an empirical analysis of the performance of our approaches on the problem instances in Azem et al. (2007) and Mati (2010). First, in Section 5.1, the experimental validation of the benefit of using the proposed move evaluation are reported. The results show that this evaluation allows a

ACCEPTED MANUSCRIPT

large proportion of non-improving moves to be discarded while the probability of discarding improving moves is very low. Section 5.2 reports the experimental results on the instances of Azem et al. (2007) and the comparison of the Simulated Annealing proposed in Section 4.2 and the Tabu Search proposed in Section 4.3 with the results of Mati (2010) and those obtained with the MILP of Azem et al. (2007). As this section shows that Tabu Search outperforms Simulated Annealing, Section 5.3 report the results obtained by Tabu Search on the instances of Mati (2010).

The metaheuristics were implemented in C++ and the numerical experiments were carried out on a personal computer with a 1.90 GHz processor and 8 Gb RAM. As in Mati (2010), the metaheuristics were run five times for each problem instance during 60 seconds. Only the best makespan is reported for the instances of Mati (2010), while the mean is also reported for the instances of Azem et al. (2007). Instead of comparing with the results reported in Azem et al. (2007), the MILP proposed in Azem et al. (2007) was solved using a more recent version (12.6) of the standard solver IBM ILOG CPLEX. The MILP is used to solve the instances of the two benchmarks with a computational time limited to $1,800$ seconds. We use the following notation for the different approaches: YM stands for the tabu thresholding metaheuristic of Mati (2010), MILP stands for the results obtained by the Mixed Integer Linear Program, SA stands for Simulated Annealing and TS stands for Tabu Search.

5.1. Potential benefit of the move evaluation

The potential benefit and risk of using the proposed move evaluation are investigated by performing some computational experiments. Simulated Annealing is used to solve the problem instances of Azem et al. (2007) and Mati (2010). At each iteration, a random critical arc on a machine is swapped without any preliminary evaluation. A move is defined as improving if the obtained neighbor is better than the solution before the swap. During the whole execution of the metaheuristic, the number of improving moves and the number of non-improving moves are tracked. Using these results, two indicators are defined: (1) The number of non-improving moves found by the evaluation over the total number of non-improving moves, and (2) The number of improving moves not discarded by the evaluation over the total number of improving moves. A value of 100% means that the move evaluation does not consider an improving move as a non-improving one, and a value lower than 100% means that the proposed move evaluation can discard good moves.

Table 1: Analysis of the move evaluation

Class	m	n	% of found non-improving moves	% of improving moves not discarded
1	5	5	82.86%	99.90%
2	5	10	76.12%	99.91%
3	10	10	73.47%	99.92%
4	10	15	78.32%	99.85%
5	10	20	77.02%	99.86%
6	10	30	68.09%	99.88%
7	15	15	80.12%	99.88%
Average			76.34%	99.88%

The results of the computational experiments are shown in Table 1. The 80 problem instances are grouped into different classes according to the number of machines and number of jobs. The values

ACCEPTED MANUSCRIPT

for each class are averages. The results show that the move evaluation allows more than 76% of non-improving moves to be avoided. Compared to the sufficient conditions that were given in Mati (2010), our proposed move evaluation is more powerful. In fact, the experimental results that were conducted in Mati (2010) shows that the sufficient conditions apply on average for 28% of the moves and could be used in 72%. In others words, the sufficient conditions may only discard less than 21% of non-improving moves. These results also show that, while the percentage of found non-improving moves is high, there is only a risk of 0.12% of missing improving moves. Moreover, the results are rather stable for the different classes of instances. In conclusion, it is promising to use the move evaluation to quickly obtain good solutions, and this is confirmed by the computational results of the following sections.

5.2. Results on Azem et al. (2007) Instances

The problem instances of Azem et al. (2007) are composed of four classes of benchmarks. The number of jobs (resp. machines) of each class is 5 (resp. 5), 10 (resp. 5), 10 (resp. 10) and 15 (resp. 10). Table 2 reports the results, where the column *LB* (resp. C_{max}) corresponds to the lower (resp. upper) bounds determined by the MILP. For the metaheuristics, the columns C_{init}, C_{mean} and C_{max} respectively give the initial solution for the 5 runs, the mean makespan and the best makespan. As the same construction heuristic is used within Simulated Annealing and Tabu Search, only one column C_{init} is necessary.

For the first class of instances with 5 jobs and 5 machines, the four approaches find the optimal solutions in negligible computational times. For the second class, both metaheuristics find all the optimal solutions whereas YM obtains these optimal only for four instances. There is only one problem instance of the third class (10m_10n_4) for which our approaches cannot determine solutions at least as good as the ones of YM. Except for this instance, all approaches find optimal solutions during one of the five runs. Tabu Search starts to dominate the two other metaheuristics when considering C_{mean} for this third class of instances. Out of five instances, TS outperforms SA in three cases and YM in two cases. The difference in the performances is larger for the last class. TS outperforms the two other metaheuristics both in terms of C_{max} and C_{mean}. Compared to the MILP, TS is outperformed for two instances and find better solutions for two other instances. In addition to the improvement of the solutions, it is worth mentioning that 18 out of the 20 initial solutions are worse than the ones of the YM procedure.

5.3. Results on Mati (2010) Instances

Mati (2010) generalizes 20 medium sizes instances of Lawrence (1984) by adding unavailability periods. The original instances are composed of four classes. The number of jobs (resp. machines) of each class is 15 (10), 20 (10), 30 (10) and 15 (15). Based on these instances for the classical job-shop scheduling problem, three sets of instances that differ by the number of unavailability periods were generated: In *ldata*, each machine has on average less than two unavailability periods, exactly two unavailability periods in *mdata* instances, and between two and three unavailability periods in *hdata* instances.

Table 3 reports the results that were found by YM, the MILP, and TS. The results obtained by SA are not reported as they are outperformed by those obtained by TS. The boldfaced values in Table 3 indicate the best solution values. Table 4 summarizes the average and maximum relative errors over lower bounds of the three approaches and for each of the three different sets: ldata, mdata and hdata. The Relative Error (RE) is calculated as follows: $RE = 100 * \frac{C_{max} - LB}{LB}$.

The results reported in the two tables show a clear dominance of TS, which obtains the best solution for 40 out of 60 instances. More precisely, TS outperforms YM in 47 instances and the MILP in 42 instances. In addition to the analysis on the number of instances, the indicators on the relative errors confirm that TS dominates. When TS is outperformed, its solutions are not far from the best solutions.

ACCEPTED MANUSCRIPT

Table 2: Results on Azem et al. (2007) instances

Instances	YM			SA			TS		MILP	
	C_{init}	C_{mean}	C_{best}	C_{init}	C_{mean}	C_{best}	C_{mean}	C_{best}	LB	C_{max}
5m_5n_1	977	894	894	1177	894	894	894	894	894	894
5m_5n_2	1095	1095	1095	1225	1095	1095	1095	1095	1095	1095
5m_5n_3	1271	1069	1069	1261	1069	1069	1069	1069	1069	1069
5m_5n_4	1386	1146	1146	1388	1146	1146	1146	1146	1146	1146
5m_5n_5	1331	1201	1201	1368	1201	1201	1201	1201	1201	1201
5m_10n_1	1779	1360	1360	1872	1360	1360	1360	1360	1360	1360
5m_10n_2	1814	1454	1454	1924	1454	1454	1454	1454	1454	1454
5m_10n_3	1756	1606	1606	1901	1606	1606	1606	1606	1606	1606
5m_10n_4	1776	1536	1536	1917	1506	1506	1510	**1506**	1506	1506
5m_10n_5	1744	1414	1414	1894	1414	1414	1414	1414	1414	1414
10m_10n_1	2349	1970.2	1947	2590	1966	1947	1952	1947	1947	1947
10m_10n_2	2446	1916	1916	2591	1920	1916	1916	1916	1916	1916
10m_10n_3	2187	1874	1874	2391	1874	1874	1874	1874	1874	1874
10m_10n_4	2227	1786	1771	2620	1786	1786	1786	1786	1771	1771
10m_10n_5	2264	1958	1932	2404	1965	1960	1943	1932	1932	1932
10m_15n_1	2807	2121	2118	2666	2127	2120	2117	2106	1898	**2076**
10m_15n_2	2961	2280.6	2259	3025	2298	2284	2285	2252	2085	**2232**
10m_15n_3	2882	2210	2182	3017	2232	2212	2197	2182	1940	2182
10m_15n_4	2631	2133.2	2119	2785	2145	2131	2125	**2117**	1827	2123
10m_15n_5	2624	2210.2	2184	2974	2238	2224	2186	**2171**	2051	2204

ACCEPTED MANUSCRIPT

Table 3: Results on Mati (2010) instances

Data	n	m	ldata			mdata			hdata		
			YM	MILP	TS	YM	MILP	TS	YM	MILP	TS
la21	15	10	1296	1296	**1205**	**1182**	**1182**	1185	**1296**	**1296**	1320
la22	15	10	1206	1127	**1084**	1148	**1065**	1082	**1127**	**1127**	1162
la23	15	10	1184	1257	**1182**	**1131**	1202	1202	1257	1257	**1250**
la24	15	10	1206	1172	**1072**	1270	**1091**	1118	1319	**1172**	**1172**
la25	15	10	1402	1201	**1108**	1277	**1114**	1115	**1201**	**1201**	1214
la26	20	10	**1289**	1435	1334	1457	1368	**1348**	**1334**	1435	1427
la27	20	10	1391	1499	**1379**	1615	1449	**1391**	1462	1499	**1446**
la28	20	10	1483	1465	**1391**	1534	1429	**1389**	1733	1465	**1416**
la29	20	10	1696	1399	**1330**	1844	1394	**1324**	1577	**1399**	1430
la30	20	10	1630	1592	**1466**	1728	1520	**1519**	1851	1592	**1543**
la31	30	10	2159	2205	**1843**	2235	2135	**1946**	2141	2205	**1927**
la32	30	10	2300	2215	**2059**	2218	2260	**2030**	2704	2215	**2068**
la33	30	10	2642	2285	**1816**	2447	2260	**1897**	2041	2285	**1924**
la34	30	10	2085	2097	**1853**	2308	2158	**1917**	2291	2097	**1973**
la35	30	10	2352	2413	**2033**	2652	2109	**2018**	2322	2413	**2110**
la36	15	15	1657	1450	**1431**	1533	**1463**	1490	1615	**1450**	1465
la37	15	15	1672	1697	**1586**	**1481**	1560	1557	1960	**1697**	1718
la38	15	15	1804	1418	**1357**	1518	**1333**	1374	1716	**1418**	1475
la39	15	15	1492	1452	**1415**	**1360**	1361	1446	1479	**1452**	1491
la40	15	15	1620	**1234**	1246	**1229**	**1229**	1233	**1229**	1234	1233

Table 4: Comparing Average and Maximum Relative Errors (RE %) on lower bounds

	ldata			mdata			hdata		
	YM	MILP	TS	YM	MILP	TS	YM	MILP	TS
Average	14.16%	8.86%	0.32%	12.50%	4.14%	1.37%	9.66%	3.99%	1.23%
Maximum	45.48%	25.83%	3.49%	39.27%	19.14%	6.32%	30.75%	18.76%	6.97%

ACCEPTED MANUSCRIPT

As it can be observed in Table 4, the average Relative Error for TS (resp. YM) is 0.32% (resp. 14.16%) for the set ldata, 1.37% (resp. 12.50%) for the set mdata and 1.32% (resp. 9.662%) for the set hdata. Also, when considering the worst case, TS finds a solution which is less than 7% from the lower bound, whereas YM finds solutions that are more than 30% from the best-found solution. Similarly to the instances in Azem et al. (2007), the initial solutions of TS are of bad quality.

If we now analyze the results for both benchmarks, it is possible to note that TS is less efficient with "square" problems ($n \cong m$). However, when the problem grows and becomes "rectangular", the efficiency of TS is increasing. Taillard (1994) made the same observation concerning his approach for the classical job-shop scheduling problem.

6. Conclusion

In this paper, we investigated the job-shop scheduling problem with availability constraints. We highlight some changes related to the introduction of unavailability periods. These changes led to the redefinition of some known concepts. After proposing a move evaluation that allows ignoring a large proportion of non-improving moves, two metaheuristics (Simulated Annealing and Tabu Search) were proposed to solve the problem. Experiments carried out on instances of the literature showed the efficiency of the proposed Tabu Search.

In future work, we would like to extend the proposed approach to take into account, in the same model, all possible cases for an operation to be interrupted by an unavailability period, and also the cases in which an availability period is crossable or not, and is fixed or flexible. Another important future work is to consider other regular objective functions than the makespan and to extend our approach to solve the flexible job-shop scheduling problem with availability constraints.

References

Aggoune, R., 2002. Ordonnancement dateliers sous contraintes de disponibilité des machines. University of Metz, France .

Aggoune, R., Mati, Y., Dauzère-Pérès, S., 2009. A reduced-complexity algorithm for two-job shop scheduling problems with availability constraints. IFAC Proceedings Volumes 42 (4), 1190–1195.

Azem, S., Aggoune, R., Dauzère-Pérès, S., 2007. Disjunctive and time-indexed formulations for non-preemptive job shop scheduling with resource availability constraints. In: Industrial Engineering and Engineering Management, 2007 IEEE International Conference on. IEEE, pp. 787–791.

Azem, S., Aggoune, R., Dauzère-Pérès, S., 2012. Heuristics for job shop scheduling with limited machine availability. IFAC Proceedings Volumes 45 (6), 1395–1400.

Benttaleb, M., Hnaien, F., Yalaoui, F., 2016. Two-machine job shop problem for makespan minimization under availability constraint. IFAC-PapersOnLine 49 (28), 132–137.

Bureau, M., Dauzère-Pérès, S., Mati, Y., 2006. Scheduling challenges and approaches in semiconductor manufacturing. IFAC Proceedings Volumes 39 (3), 739–744.

Cui, W.-W., Lu, Z., Zhou, B., Li, C., Han, X., 2016. A hybrid genetic algorithm for non-permutation flow shop scheduling problems with unavailability constraints. International Journal of Computer Integrated Manufacturing 29 (9), 944–961.

ACCEPTED MANUSCRIPT

Dauzère-Pérès, S., Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Annals of Operations Research 70, 281–306.

Gao, J., Gen, M., Sun, L., 2006. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. Journal of Intelligent Manufacturing 17 (4), 493–507.

Glover, F., Taillard, E., 1993. A user's guide to tabu search. Annals of operations research 41 (1), 1–28.

Lawrence, S., 1984. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie Mellon University, Pittsburgh, PA .

Lee, C.-Y., 1999. Two-machine flowshop scheduling with availability constraints. European Journal of Operational Research 114 (2), 420–429.

Ma, Y., Chu, C., Zuo, C., 2010. A survey of scheduling with deterministic machine availability constraints. Computers & Industrial Engineering 58 (2), 199–211.

Mati, Y., 2010. Minimizing the makespan in the non-preemptive job-shop scheduling with limited machine availability. Computers & Industrial Engineering 59 (4), 537–543.

Mauguière, P., Billaut, J.-C., Bouquard, J.-L., 2005. New single machine and job-shop scheduling problems with availability constraints. Journal of scheduling 8 (3), 211–231.

Nowicki, E., Smutnicki, C., 2005. An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8 (2), 145–159.

Ovacik, I. M., Uzsoy, R., 2012. Decomposition methods for complex factory scheduling problems. Springer Science & Business Media.

Roy, B., Sussmann, B., 1964. Les problemes dordonnancement avec contraintes disjonctives. Note ds 9.

Schmidt, G., 2000. Scheduling with limited machine availability. European Journal of Operational Research 121 (1), 1–15.

Taillard, E. D., 1994. Parallel taboo search techniques for the job shop scheduling problem. ORSA journal on Computing 6 (2), 108–117.

Talbi, E.-G., 2009. Metaheuristics: from design to implementation. Vol. 74. John Wiley & Sons.

Van Laarhoven, P. J., Aarts, E. H., Lenstra, J. K., 1992. Job shop scheduling by simulated annealing. Operations research 40 (1), 113–125.

Zhang, C., Li, P., Guan, Z., Rao, Y., 2007. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers & Operations Research 34 (11), 3229–3242.

Zribi, N., El Kamel, A., Borne, P., 2008. Minimizing the makespan for the mpm job-shop with availability constraints. International Journal of Production Economics 112 (1), 151–160.

ACCEPTED MANUSCRIPT

- The job-shop scheduling problem with availability constraints is solved
- A novel definition and procedure to label critical operations are introduced
- An efficient move evaluation without making the move is presented
- New metaheuristics (Simulated Annealing and Tabu Search) are proposed
- Numerical experiments show that Tabu Search outperforms existing approaches

ACCEPTED MANUSCRIPT