International Journal of Production Research
Vol. 49, No. 23, 1 December 2011, 6963–6980

RESEARCH ARTICLE

A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem

Geoffrey Vilcot[ab]* and Jean-Charles Billaut[b]

[a]*Volume Software, 20 rue Dublineau BP2630, 37026 Tours Cedex 1, France;*
[b]*Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France*

(*Final version received August 2010*)

The problem that we consider in this article is a flexible job shop scheduling problem issued from the printing and boarding industry. Two criteria have to be minimised, the makespan and the maximum lateness. Two tabu search algorithms are proposed for finding a set of non-dominated solutions: the first is based on the minimisation of one criterion subject to a bound on the second criterion (ϵ-constraint approach) and the second is based on the minimisation of a linear combination of criteria. These algorithms are tested on benchmark instances from the literature and the results are discussed. The total tardiness is considered as a third criterion for the second tabu search and results are presented and discussed.

Keywords: scheduling; jobshop; tabu search; multi-objective

1. Introduction

In this article we consider a problem which comes from the printing and boarding industry. In this kind of industry, shops are composed of several parallel machines, mainly printing machines, carding machines and folding machines. Generally, more than one printing machine is able to perform a printing operation, more than one folding machine is able to perform a folding, etc. So, in addition to the problem of scheduling operations, i.e. of determining a starting time for each operation, there is an assignment problem, i.e. a problem of determining the performing machine. Since all the jobs to perform do not visit the machines of the shop in the same order, the routings of jobs are not identical and we say that we are faced with a *job shop* environment. This problem is known in the literature and called the Flexible Job Shop Scheduling Problem (FJSSP). Furthermore, decision makers are generally not only interested in solutions that perform well for a single objective function, but also by solutions of best compromise between several criteria like makespan or flow time and criteria involving tardiness measures. In the following, we first consider two criteria and finally three criteria in the objective function. The bicriteria problem involves the makespan and the maximum lateness minimisation and the additional criteria considered is the total tardiness. The aim is to offer to the decision maker a set of solutions that correspond to non-dominated criteria vectors, also called *Pareto optimal solutions* or *non-dominated solutions* for more convenience.

*Corresponding author. Email: g.vilcot@volume-software.com

ISSN 0020–7543 print/ISSN 1366–588X online

http://dx.doi.org/10.1080/00207543.2010.526016
http://www.tandfonline.com

The literature contains a great number of articles dealing with job shop scheduling problems, with multicriteria approaches in scheduling (T'kindt and Billaut 2006) and with scheduling-and-assignment problems. Less articles deal with FJSSPs with multiple criteria.

In Jurisch (1992), a branch-and-bound algorithm and some heuristic algorithms are proposed for the multi-purpose machine job shop scheduling problem. For the FJSSP, a hierarchical approach can be found in Paulli (1995), where the first step is to solve the assignment problem and the second step is to solve the job shop scheduling problem. Dauzère-Pérès and Paulli (1997) propose a tabu search for the FJSSP with the makespan as an objective function. Kacem *et al.* (2002) propose a hybridisation of an evolutionary algorithm and fuzzy logic. In Mastrolilli and Gambardella (2000) two neighbourhood structures for the FJSSP are proposed. One interest in the proposed neighbourhood is that any feasible solution having an empty neighbourhood is optimal. Alvarez-Valdes *et al.* (2005) propose a heuristic algorithm for a FJSSP dedicated to a glass factory industry with special constraints like *no-wait* or *overlapping*. The objective is to find a schedule with a criterion based on earliness and tardiness penalties. The authors use a two-step algorithm. In the first step priority rules are used to solve the problem and in the second step a local search improves this solution. Ho *et al.* (2007) propose an architecture where a learning module is introduced into an evolutionary algorithm for solving FJSSP. The makespan is minimised and the authors show that their method outperforms existing ones (Brandimarte 1993, Kacem *et al.* 2002) for some instances of the literature. Choobineh *et al.* (2006) propose a multi-objective tabu search algorithm for a single-machine problem with sequence-dependent setup time. We refer to T'kindt and Billaut (2006) for a more precise survey on multicriteria scheduling problems.

We propose in this article two versions of a tabu search algorithm for solving the problem. The first version consists of solving the ϵ-constraint approach of the problem. It means that one criterion is minimised assuming that the second criterion is bounded by ϵ. By modifying the bound iteratively, it is possible to determine a set of non-dominated solutions (supported by the trade-off curve or not). The second version of the tabu search algorithm consists of finding a solution that minimises a linear combination of the criteria. By modifying the weights associated to the criteria, it is also possible to determine a set of non-dominated solutions, restricted to the 'supported' ones.

In Section 2 a formal definition of the problem is presented with notations. In Section 3 the two tabu search algorithms are explained. Computational experiments are presented and discussed in Section 4. The third criterion is introduced together with the experiments since it does not influence the algorithms description. Section 5 concludes this article and gives some future research directions.

2. Problem definition and notations

The problem that we consider is the FJSSP, which is a generalisation of the classical job shop scheduling problem. We consider that a set $\mathcal{J}$ of n jobs have to be scheduled on a set $\mathcal{M}$ of m disjunctive resources. Each job J_j needs n_j consecutive operations for being completed and operation i of job J_j is denoted by $O_{i,j}$. Each operation $O_{i,j}$ can be processed by any resource of the set $\mathcal{R}_{i,j}$ which is the subset of $\mathcal{M}$ containing the resources that can perform $O_{i,j}$. We assume that the processing time of operation $O_{i,j}$, denoted by $p_{i,j}$, does not depend on the performing resource. This hypothesis could be relaxed without changing the validity of the methods. To each job J_j a due date is also associated denoted by d_j.

Table 1. List of notations.

$\mathcal{M}$	: Set of resources
m	: Number of resources, $m = \lvert\mathcal{M}\rvert$
M_k	: Resource number k
$\mathcal{J}$	: Set of jobs
n	: Number of jobs, $n = \lvert\mathcal{J}\rvert$
J_j	: Job number j
n_j	: Number of operations in J_j
$O_{i,j}$	: Operation number i of J_j
$\mathcal{R}_{i,j}$	: Set of resources able to perform $O_{i,j}$
$p_{i,j}$	: Processing time of $O_{i,j}$
d_j	: Due date of J_j
C_j	: Completion time of J_j
$C_{\max}$	: Makespan
$L_{\max}$	: Maximum lateness
$\sum T_j$	: Total tardiness

The problem is to set a performing resource and a starting time to each operation. We denote by C_j the completion time of job J_j. To measure the quality of a schedule, three classical criteria are considered: the makespan $C_{\max}$ defined by $C_{\max} = \max_{J_j \in \mathcal{J}} C_j$, the maximum lateness $L_{\max}$ defined by $L_{\max} = \max_{J_j \in \mathcal{J}} (C_j - d_j)$ and the total tardiness $\sum T_j$ defined by $\sum_{j \in \mathcal{J}} T_j = \sum_{j \in \mathcal{J}} \max(0, C_j - d_j)$.

This problem is strongly NP-hard. The notations are summarised in Table 1.

3. The tabu search algorithms

A tabu search algorithm works as follows. It requires an *initial solution* and a *neighbourhood structure* and proceeds by transiting from one solution to another using *moves*. All the neighbours of a current solution are examined and the best *admissible move* is selected. Note that this move may decrease the quality of the solution. A *tabu list* stores all the previously exploited moves which are now forbidden. A general description of tabu search algorithms can be found in Glover (1989) and a first application to FJSSP in Brandimarte (1993).

Two tabu search algorithms are proposed:

- the first corresponds to the ϵ-constraint approach and is denoted by $TS\epsilon$.
- the second minimises a linear combination of criteria and is denoted by $TS\ell c$.

In the following, we first present the common elements of both algorithms, then we continue with the specific parts for each of them.

3.1 *Common parts of both algorithms*

The common elements are the coding of a solution, the initial solution, the neighbourhood structure and finally the tabu list definition.

3.1.1 *Coding of a solution*

A solution is represented by the classical conjunctive graph $G = (V, E)$. The set of vertices is equal to $V = \{O_{i,j}, 1 \leq j \leq n, 1 \leq i \leq n_j\} \cup \{s, t\}$ with s the source and t the sink vertex.

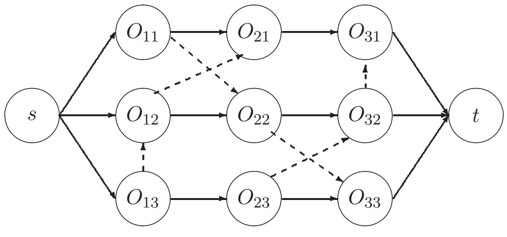

Figure 1. Coding of a solution.

There is an edge between two vertices if there is a routing constraint or a resource constraint between the corresponding operations. There is an edge between $O_{n_j,j}$ and t of length $p_{n_j,j}$; an edge between s and $O_{1,j}$ of length 0. For the other edges, the length is equal to the processing time of the operation at the origin of the edge. This graph does not contain any cycle and all the arcs have positive length. Evaluation of the makespan or the maximum lateness of the corresponding schedule can be done exactly in polynomial time (Figure 1).

3.1.2 *The initial solution*

The initial solution is given by a two-step greedy algorithm. The first step consists of solving the assignment problem for each operation, the second step consists of solving the job shop problem, without modifying the assignment of operations.

First step – firstly, the operations are sorted in $|\mathcal{R}_{i,j}|$ non-decreasing order and by non-increasing $p_{i,j}$ order to break ties. The list of operations is called $\mathcal{O}$. The workload of one resource is defined by the sum of the processing times of the assigned operations. Resources are sorted in their workload non-decreasing order, the list is called $\mathcal{R}$. The operations are taken according to list $\mathcal{O}$ and for each operation $O_{i,j}$, the first resource of $\mathcal{R}$ that belongs to $\mathcal{R}_{i,j}$ is assigned to the operation, the workload of this resource is updated as well as the sorting of the resources. The process iterates until all the operations of $\mathcal{O}$ are assigned to one resource.

Second step – the job shop problem is solved by using a greedy algorithm based on the 'slack' rule. We denote by $\mathcal{S}$ the set of candidate operations. At the beginning, $\mathcal{S}=\{O_{1,j}, 1\leq j\leq n\}$. The candidate operations are sorted in their release time non-decreasing order (and slack non-decreasing order to break ties). The release time of an operation $O_{i,j}$ is the maximum between the time the performing resource is free for processing operation $O_{i,j}$ and the completion time of operation $O_{i-1,j}$ (if it exists). The slack of operation $O_{i,j}$ is the difference between the due date of job J_j and the completion time of this job as if the remaining operations were processed after $O_{i,j}$ without idle time. The first candidate operation is scheduled, the release dates are updated, $\mathcal{S}$ is updated and the process iterates while $\mathcal{S}\neq\emptyset$.

3.1.3 *The definition of a neighbour*

A neighbour is obtained by moving an operation. This unique technique allows both the modification of the assignment and the modification of a sequence. In the

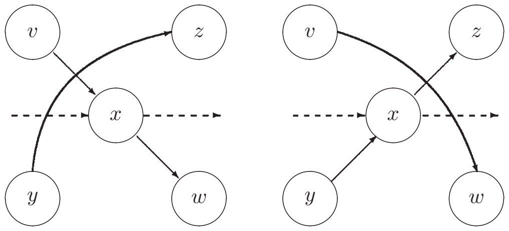

Figure 2. Moving operation x between y and z.

conjunctive graph, moving an operation x between operations y and z (this move is denoted by $\{x, y, z\}$) means:

- deleting edge (v, x) and (x, w), where v (respectively w) is the predecessor (respectively successor) of x on the performing resource,
- adding an edge (v, w),
- deleting the edge (y, z),
- adding edges (y, x) and (x, z).

Figure 2 illustrates a move in the graph.

3.1.4 *The tabu list*

The tabu list $T\ell$ is used to prevent the search from cycling between solutions. Our tabu list has a fixed size and when the list is full, the oldest element is replaced by the new element (First In First Out management of the list).

Dauzère-Pérès and Paulli (1997) propose three types of tabu list and we use the best type of list they propose. This tabu list works as follows: after moving x between y and z, (x, y) and (z, x) are added to the tabu list. A move $\{x', y', z'\}$ is forbidden if either $(x', y') \in T\ell$ or $(z', x') \in T\ell$.

3.1.5 *The neighbourhood structure*

The neighbourhood N is defined as follows: for each operation $O_{i,j}$, this operation is swapped with its immediate predecessor on its performing resource, if is not in first position. In order to change the assignment, for each resource in $\mathcal{R}_{i,j}$ (except the current assignment), $O_{i,j}$ is moved at the first possible position of insertion which starts at or just after its actual start time. Figure 3 illustrates the assignment modification (dotted lines represent routing constraints that are not modified).

Algorithm 1 describes the neigbourhood exploration. Seq_k is the set of operations sequenced on M_k and $Seq_{k,l}$ denotes the operation in position l on resource M_k. *SearchInsertion*(r, o) finds the insertion position of operation o on resource M_r.

Algorithm 1:

```
1: for all $M_k \in \mathcal{M}$ do
2:   for $l = 1$ to $|Seq_k|$ do
3:     if $l \geq 2$ and $(Seq_{k,l}, Seq_{k,l-2}) \notin T\ell$ and $(Seq_{k,l-1}, Seq_{k,l}) \notin T\ell$ then
4:       Move $Seq_{k,l}$ between $Seq_{k,l-2}$ and $Seq_{k,l-1}$; Evaluate_the_neighbour; Undo
         the move
```

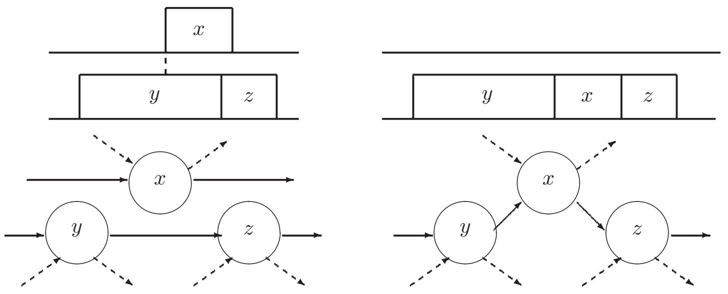

Figure 3. Assignment modification for *x*.

```
5:       end if
6:       for all $M_{k'} \in \mathcal{R}_{Seq_{kl}}$ do
7:          $l' \leftarrow SearchInsertion(k', Seq_{k,l})$
8:          if $M_{k'} \neq M_k$ and $(Seq_{k,l}, Seq_{k',l'-1}) \notin T\ell$ and $(Seq_{k,l'}, Seq_{k',l}) \notin T\ell$ then
9:             Move $Seq_{k,l}$ between $Seq_{k',l'-1}$ and $Seq_{k',l'}$; Evaluate_the_neighbour; Undo
               the move
10:         end if
11:      end for
12:   end for
13: end for
```

Procedure Evaluate_the_neighbour stores the neighbour and its evaluation if it is the best until this point. For the job in first position, only the assignment modification is explored.

Proposition 3.1 : *The neighbourhood N is connected.*

Proof: Consider an initial solution S^i and a target solution S^t. We have to show that it is possible to obtain S^t from S^i after a finite number of moves. We consider the conjunctive graphs corresponding to these schedules, denoted by G^i and G^t. From S^i it is always possible to change the assignment of the operations so that it is the same as in S^t. It is clear that such a set of moves cannot generate a cycle in the conjunctive graph. Notation S^i is used to denote the current schedule in the modification phase. We can suppose now that in S^i, the assignment of operations is the same as in S^t.

Suppose that G^t is topologically sorted. We start by considering the operations of first order in G^t. Since these operations have no predecessor, it is always possible – by using the swap move – to put them at the first order in G^i.

Now consider operation x of rank r *in* G^t and suppose that the operations of rank $r' < r$ in G^t are at the same rank in G^i. Figure 4 illustrates the 'only case' where the swap of x with its predecessor on the resource generates a cycle (dotted lines represent routing constraints). Note that the path from w to y can be composed of more than one edge. The swap move of x in G^i cannot generate the cycle of Figure 4(b) since operations z, w and y, that precede x in Figure 4(a), are of rank $r' < r$ and thus are already scheduled in G^i. Thus, it is always possible to swap x with its predecessor until its rank becomes the same in G^i as in G^t. By moving the operations of G^i in the topological order of G^t, we obtain S^t from S^i after a finite number of moves. □

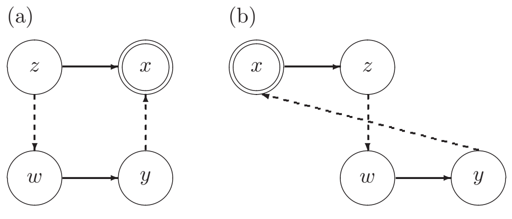

Figure 4. Cycle generation after a *swap move*.

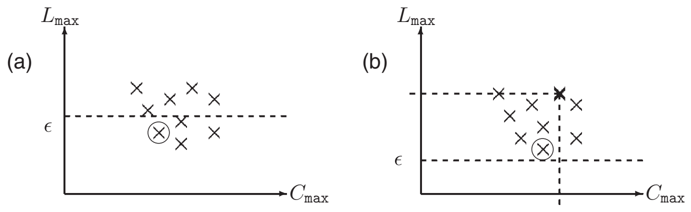

Figure 5. Best neighbour and search direction.

3.2 *TSϵ algorithm*

With the ϵ-constraint approach, the tabu search algorithm requires a particular evaluation of the neighbours. Here we assume that the objective to minimise is the makespan, subject to a bound on the maximum lateness.

Since we try to solve an ϵ-constraint problem, the definition of the best neighbour depends on the structure of the set of neighbours. The best neighbour of a solution is the one with the minimum $C_{\max}$ among the neighbours with $L_{\max} < \epsilon$ (see Figure 5 (a) where the best neighbour is represented in a circle). If all the neighbours are such that $L_{\max} \geq \epsilon$, then the best neighbour is the one with the minimum lateness, with a makespan value not greater than the makespan value of the current solution, if possible (see Figure 5 b). In all other cases, we choose the neighbour that minimises $L_{\max}$. By using this definition of the best neighbour, we first try to respect the ϵ bound and then to improve the makespan under the ϵ constraint.

3.3 *TSℓc algorithm*

A solution is evaluated by a linear combination of criteria. The considered criteria are: the makespan ($C_{\max}$) and the maximum lateness ($L_{\max}$).

One of the difficulties with such an approach lies in the criteria range. For instance, the makespan can be equal to 1000 and the maximum lateness can be equal to 10. In this case, the makespan eclipses the maximum lateness. Furthermore, it can be impossible to estimate *a priori* the range of each criterion. To bypass this difficulty, the coefficients of the linear combination have been normalised. The evaluation of a solution is given by

$$Z(S) = \frac{\alpha \times C_{\max}(S)}{\max(C_{\max}) - \min(C_{\max})} + \frac{\beta \times L_{\max}(S)}{\max(L_{\max}) - \min(L_{\max})}$$

with

- $\{\alpha, \beta\}$ the coefficients,
- $C_{\max}(S)$ and $L_{\max}(S)$ the makespan and the maximum lateness of the solution,
- and $\max(C_{\max})$, $\min(C_{\max})$, $\max(L_{\max})$ and $\min(L_{\max})$ that are computed as follows.

We denote by S^i_{best} the selected neighbour at iteration i, let k be the current iteration number of *TSℓc* algorithm. The maximum and minimum value of criteria are computed as follows:

$$Y = \{S^i_{\text{best}},\ \forall i, i < k\}\ ;\ \max(C_{\max}) = \max_{y \in Y} C_{\max}(y)\ ;\ \min(C_{\max}) = \min_{y \in Y} C_{\max}(y)$$

$$\max(L_{\max}) = \max_{y \in Y} L_{\max}(y)\ ;\ \min(L_{\max}) = \min_{y \in Y} L_{\max}(y)$$

This normalisation method is inspired by Deb *et al.* (2002) for the computation of the crowding distance.

Sometimes, the tabu search falls into a deep local optimum and is unable to leave this solution. In this case, we use a diversification operator, inspired by the *strategic oscillation* (which consists of allowing the exploration of non-feasible solutions for a short time, (Downsland 1998)). The diversification we have implemented works as follows. In *TSℓc* algorithm, after some iterations without improvement, the search restarts with the best known solution S^+ and new coefficients α, β randomly chosen for the linear combination. By using this technique, some solutions which would have never have been explored with the initial coefficients can be explored. The search with these new coefficients continues until a better solution (with the correct coefficients) than S^+ is found or until a limit is reached. In the first case, *TSℓc* continues starting from this solution and with the initial coefficients, otherwise it is the end of the search.

4. Computational experiments

Four data sets are used to evaluate the tabu search algorithms. These data sets are based on Hurink's instances (Hurink *et al.* 1994), derived from Lawrence's instances. In order to introduce due dates in these data sets, the method that has been used is inspired by Demirkol *et al.* (1998). The due date of J_j is defined by

$$d_j \hookrightarrow \mathcal{U}\left[\mu_j \times \left(1 - \frac{R}{2}\right); \mu_j \times \left(1 + \frac{R}{2}\right)\right]$$

with

$$\mu_j = \left(1 + \frac{T \times n}{m}\right) \times \sum_{i=1}^{n_j} p_{i,j}$$

where T and R are two parameters that have been fixed to $T = 0.3$ and $R = 0.5$.

The four data sets are the following:

- `sdata`: instances of Lawrence (single resource per operation), no assignment problem,
- `edata`: instances with few flexibility, near to the original instances of Lawrence,

- `rdata`: instances with two possible resources per operation in average,
- `vdata`: instances with $m/2$ possible resources per operation in average.

The size of the instances la01 to la05 is 5×10, i.e. $m = 5$, $n = 10$, instances la06 to la10 is 5×15, instances la11 to la15 is 5×20, instances la16 to la20 is 10×10, instances la21 to la25 is 10×15, instances la26 to la30 is 10×20, instances la31 to la35 is 10×30, and instances la36 to la40 is 15×15. Experiments have been made with an *Intel Core Duo T*2400, 1.84 GHz, 2Go RAM (with only one core used).

4.1 *Indicators definition*

4.1.1 *Weak outperformance.* A set of non-dominated solutions $S_\times$ weakly outperforms a set $S_\circ$ if no solution in $S_\times$ is dominated by a solution in $S_\circ$ and at least one solution in $S_\times$ dominates a solution in $S_\circ$. Figure 6 illustrates the notion of weak outperformance: set $S_\times$ (solutions are represented by a cross) weakly outperforms set $S_\circ$ (solutions are represented by a circle). $Ow(S_\times, S_\circ)$ is equal to 1 if $S_\times$ weakly outperforms $S_\circ$, 0 otherwise. Notice that $Ow(S_\times, S_\circ) + Ow(S_\circ, S_\times)$ may be equal to 0 or 1 (but cannot be equal to 2).

Let $\exp_i$ and $\exp_j$ be two test series and $S_{\exp_i}$ (respectively $S_{\exp_j}$) the set of non-dominated solutions for $\exp_i$ (respectively $\exp_j$). For all $j \neq i$, $SOw(\exp_i, \exp_j)$ is the sum for all considered instances of $Ow(S_{\exp_i}, S_{\exp_j})$. Indicator $SOw(\exp_i)$ is the sum of $SOw(\exp_i, \exp_j)$ for all $j \neq i$. If N is the number of test series, then we have

$$SOw(\exp_i) = \sum_{j=1, j\neq i}^{N} SOw(\exp_i, \exp_j)$$

This indicator represents the number of times the set of non-dominated solutions obtained for the considered parameter set $\exp_i$ weakly outperforms the others. The greater this indicator, the best $S_{\exp_i}$.

4.1.2 *Net front contribution.* Considering two sets of non-dominated solutions $S_\times$ and $S_\circ$, S_* represents the set of non-dominated solutions in $S_\times \cup S_\circ$. $NFC(S_\times, S_\circ)$ is the proportion of solutions from $S_\times$ in S_*. $\overline{NFC}(\exp_i, \exp_j)$ denotes the average value of $NFC(S_{\exp_i}, S_{\exp_j})$ for all considered instances. $\overline{NFC}(\exp_i)$ is defined by

$$\overline{NFC}(\exp_i) = \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} \overline{NFC}(\exp_i, \exp_j)$$

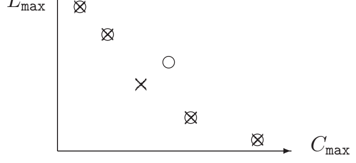

Figure 6. Notion of weak outperformance.

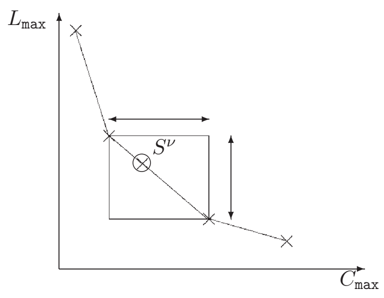

Figure 7. Computation of the crowding distance.

Indicator $\overline{NFC}$ corresponds to the average Net front contribution. The greater this indicator, the best $S_{\exp_i}$.

4.1.3 *Crowding distance.* The crowding distance (Deb *et al.* 2002) can be used to evaluate the diversity of a Pareto front. Let us consider a solution S^{ν} in the criteria space. The crowding distance is the sum of the width and height of the rectangle between the solution 'on the right' and the solution 'on the left' (Figure 7). The extreme solutions have a crowding distance equal to infinity.

Indicator $\overline{CD}$ is the average crowding distance between all solutions in the curve, except for the extremum solutions. The greater this indicator, the best the set of non-dominated solutions. However, note that a method generating few solutions may have an important average crowding distance. This indicator is also considered 'after' SOw and $\overline{NFC}$.

4.2 *Preliminary experiments*

Preliminary experiments have been conducted in order to fix the parameters of the tabu search algorithms. The parameters are #Max the number of iterations without improvement before the search is stopped (#Max $\in \{100, 200, 500, 800, 1000\}$) and $|T\ell|$ the size of the tabu list ($|T\ell| \in \{10, 40, 70, 100, 150\}$). These preliminary experiments have been realised for $TS\epsilon$ and $TS\ell c$ algorithms. The instances that have been used for the tests are la01, la06, la11, la16, la21, la26, la31 and la36 for `sdata`, `edata`, `rdata` and `vdata` (each data set corresponds to a problem size).

4.2.1 *Results of $TS\epsilon$ algorithm*

Table 2 shows the results for $TS\epsilon$ algorithm depending on the variation of #Max. For these experiments, the size of $T\ell$ has been fixed to 100.

Since the computation time of the algorithm is strongly related to parameter #Max, we have to fix this parameter for having a good compromise between the quality of the solutions and the computation time. Figure 8 illustrates the variation of SOw and $\overline{NFC}$ with the computation time.

We can see that the improvement after 500 s of computation is not really significant for both performance measures : the improvement between #Max = 500 and #Max = 800 for SOw is 41% and for $\overline{NFC}$ is 15%, but the computation time increases by 42%. So for $TS\epsilon$ algorithm, #Max has been fixed to 500.

Table 2. Variation of #Max for *TSϵ* algorithm.

#Max	*SOw*	$\overline{NFC}$ (%)	Average computation time (s)
100	3	35	53
200	13	51	137
500	24	66	339
800	41	78	593
1000	48	81	818

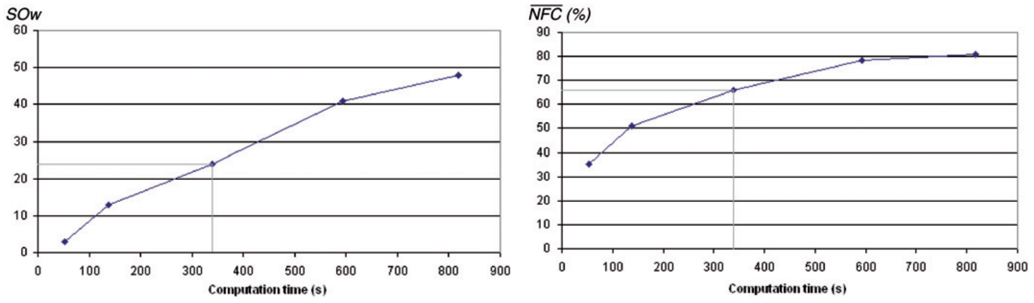

Figure 8. Quality versus computation time for the variation of the #Max for *TSϵ*.

Table 3. Variation of $|T\ell|$ for *TSϵ* algorithm.

$\lvert T\ell \rvert$	*SOw*	$\overline{NFC}$ (%)
10	0	15
40	37	55
70	46	67
100	53	70
150	53	71

Table 3 shows the results of *TSϵ* depending on the size of the tabu list. For these experiments, the maximum number of iterations without improvement has been fixed to 500.

Since the size of the Tabu list has an influence on the solutions quality, Figure 9 shows the variation of *SOw* and $\overline{NFC}$ with $|T\ell|$. Parameter $|T\ell|$ has been fixed to 100.

4.2.2 *Results of TSℓc algorithm*

Table 4 shows the results of *TSℓc* algorithm depending on the value of #Max. For these experiments, the tabu list size was fixed to 100. Figure 10 shows the variation of *SOw*

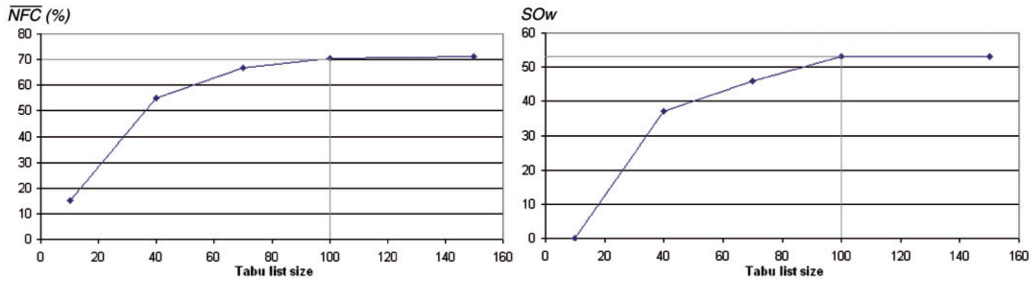

Figure 9. Quality versus $|T\ell|$ for the variation of the $|T\ell|$ for $TS\epsilon$.

Table 4. Variation of #Max for $TS\ell c$ algorithm.

#Max	SOw	$\overline{NFC}$ (%)	Average computation time (s)
100	1	19	77
200	20	43	155
500	47	68	369
800	48	74	547
1000	63	81	650

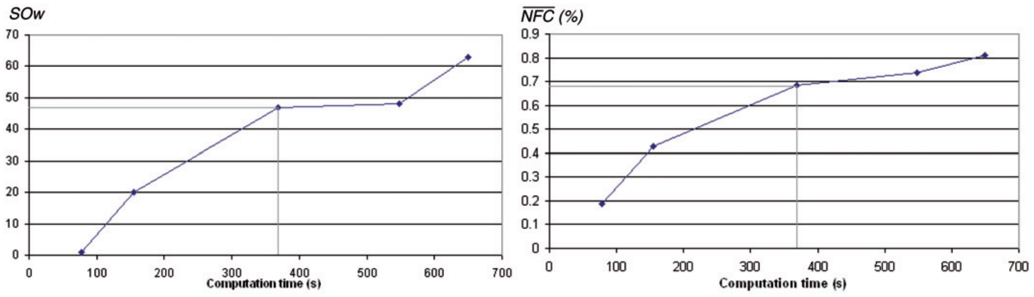

Figure 10. Quality versus computation time for the variation of the #Max for $TS\ell c$.

and $\overline{NFC}$ with the computation time. As for algorithm $TS\epsilon$, parameter #Max has been fixed to 500.

Table 5 shows the results for the tabu search with a linear combination approach on the variation of the tabu list size. For these experiments, the maximum number of iterations without improvement has been fixed to 500.

Figure 11 shows the results of $TS\ell c$ depending on the size of the tabu list. For these experiments, the maximum number of iterations without improvement was fixed to 500. We can see that a size of 100 is the best value. For both tabu search algorithms, the parameter values are fixed to $\#\text{Max} = 500$ and $|T\ell| = 100$.

Table 5. Variation of $|T\ell|$ for $TS\ell c$ algorithm.

$\|T\ell\|$	SOw	$\overline{NFC}$ (%)
10	4	20
40	23	50
70	37	67
100	47	69
150	44	64

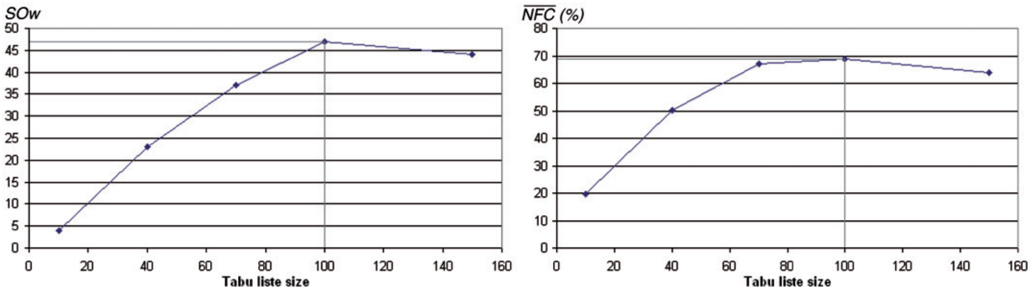

Figure 11. Quality versus computation time for the variation of the $|T\ell|$ for $TS\ell c$.

4.3 *Results*

In this section we give the computational results for three types of tests: considering a single criterion (makespan), considering two criteria (makespan and maximum lateness) and considering three criteria (plus total tardiness).

4.3.1 *Makespan minimisation*

In this section we compare $TS\epsilon$ and $TS\ell\ c$ to the tabu search algorithm proposed in Dauzère-Pérès and Paulli (1997). This algorithm only optimises the makespan. For the linear combination approach, the parameters are $\alpha = 1$ and $\beta = 0$. For the ϵ-constraint approach, $\epsilon = \infty$. Table 6 shows the average makespan deviation of the two versions of our tabu search with the tabu search in Dauzère-Pérès and Paulli (1997), denoted by $TSdpp$:

$$\Delta(TS) = (C_{\max}(TS) - C_{\max}(TSdpp))/C_{\max}(TS)$$

with $TS \in \{TS\epsilon, TS\ell c\}$. If this value is positive, algorithm $TSdpp$ performs better than TS. Column $C_{\max}(TS\epsilon) \leq C_{\max}(TS\ell c)$ indicates the number of times (over 40 instances) algorithm $TS\epsilon$ performs better than or equal to $TS\ell c$. Column Δ indicates the average deviation between these two algorithms:

$$\Delta = \frac{\max(C_{\max}(TS\epsilon), C_{\max}(TS\ell c)) - \min(C_{\max}(TS\epsilon), C_{\max}(TS\ell c))}{\max(C_{\max}(TS\epsilon), C_{\max}(TS\ell c))}$$

Table 6. Average makespan deviation with *TSdpp*.

Data set	$\Delta(TS\epsilon)(\%)$	$\Delta(TS\ell c)(\%)$	$C_{\max}(TS\epsilon) \leq C_{\max}(TS\ell c)$	$\Delta(\%)$
sdata	10.5	9.8	11	1.7
edata	11.1	11.0	15	2.64
rdata	8.5	9.1	24	2.22
vdata	5.3	6.7	28	2.34

We can see that *TSdpp* outperforms $TS\epsilon$ and $TS\ell c$. This is not surprising since our tabu search algorithms are multicriteria oriented and implement a basic neighbourhood structure, whereas *TSdpp* only considers the makespan minimisation and implements specific techniques for evaluating neighbours. Furthermore, we can see that $TS\epsilon$ and $TS\ell c$ perform similarly for the makespan minimisation considered as a single criterion.

4.3.2 *Bicriteria problem*

In order to find an approximation of the Pareto front, two approaches have been used depending on the tabu search algorithm.

For $TS\epsilon$, the value of ϵ is fixed to ∞ at the beginning. At a given step k:

- ϵ is fixed to $L_{\max}^{k-1}$ with $L_{\max}^{k-1}$ the $L_{\max}$ value obtained at step $k-1$
- the initial solution at step k is the best solution found at step $k-1$.

If no feasible solution is found, the tabu is re-launched with the initial solution obtained by the greedy algorithm (Section 3.1) and $\epsilon = L_{\max}^{k-1}$. This process iterates until a feasible solution is found.

For $TS\ell c$, different α and β values have been tested (Table 7).

Tables 8–10 show the comparison between $TS\ell c$ and $TS\epsilon$ for the three considered performance indicators: SOw, $\overline{NFC}$ and $\overline{CD}$, respectively. In Table 8, note that each value cannot exceed 5 (number of instances of size ($m \times n$)). In Table 9, each value cannot exceed 100%. The best values are indicated in bold in Tables 8–10. The results of Table 8 show that $TS\ell c$ generally weakly outperforms $TS\epsilon$.

The results of Table 9 show that $TS\epsilon$ contributes to the Pareto front more than $TS\ell c$. It means that in most of the cases the set of solutions returned by $TS\epsilon$ is better than the one returned by $TS\ell c$ in terms of quality criteria. This result confirms the results found in Table 8 in the sense that when $TS\ell c$ weakly outperforms $TS\epsilon$, the $\overline{NFC}$ indicator is generally better for $TS\ell c$, and *vice-versa*.

The results of Table 10 show that the average crowding distance is higher for $TS\ell c$ algorithm than for $TS\epsilon$ algorithm. These results confirm that $TS\ell c$ is better than $TS\epsilon$.

To conclude these experiments in the bicriteria context, we can say that $TS\ell c$ is better than $TS\epsilon$ in terms of solutions quality. This result can be surprising since generally the ϵ-constraint approach is considered as a better method for obtaining a set of non-dominated solutions (all the non-dominated solutions can be reached by using an exact method), whereas the linear combination algorithm only allows obtaining supported ones. Concerning computation times, both methods perform similarly, even if $TS\ell c$ takes slightly more time than $TS\epsilon$.

Table 7. Sets of tested parameters for the linear combination.

α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
β	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0

Table 8. *SOw* performance indicator.

	TSℓc				*TSϵ*			
$m \times n$	`sdata`	`edata`	`rdata`	`vdata`	`sdata`	`edata`	`rdata`	`vdata`
5×10	2	**3**	0	0	2	1	0	0
5×15	1	1	0	0	**2**	1	**1**	**1**
5×20	0	0	0	0	**2**	**1**	**1**	**1**
10×10	0	0	1	0	**1**	1	0	0
10×15	**1**	**2**	**1**	0	0	1	0	0
10×20	1	**3**	0	0	1	1	**1**	0
10×30	**3**	**3**	**4**	**3**	2	1	0	0
15×15	**3**	2	0	**2**	0	2	0	0

Table 9. $\overline{NFC}$ performance indicator.

	TSℓc				*TSϵ*			
$m \times n$	`sdata(%)`	`edata(%)`	`rdata(%)`	`vdata(%)`	`sdata(%)`	`edata(%)`	`rdata(%)`	`vdata(%)`
5×10	68	**73**	**47**	38	**85**	45	56	**62**
5×15	62	**56**	36	30	**87**	55	**64**	**70**
5×20	38	37	34	34	**78**	**77**	**66**	**67**
10×10	**53**	42	48	52	52	**63**	**52**	48
10×15	52	**64**	43	25	**58**	36	**57**	**75**
10×20	47	**72**	28	37	**53**	28	**72**	**63**
10×30	**70**	**65**	**92**	**81**	40	35	8	19
15×15	**74**	50	49	**68**	33	50	**51**	32

Table 10. $\overline{CD}$ performance indicator.

	TSℓc				*TSϵ*			
$m \times n$	`sdata`	`edata`	`rdata`	`vdata`	`sdata`	`edata`	`rdata`	`vdata`
5×10	**1.47**	**1.25**	**0.79**	0.64	1.29	1.00	0.56	**0.69**
5×15	**0.92**	**1.23**	**0.90**	**0.59**	0.88	0.75	0.61	0.38
5×20	1.40	**1.38**	**0.67**	**0.71**	**1.69**	1.14	0.44	0.40
10×10	1.17	1.19	0.81	1.22	**1.61**	**1.38**	**1.15**	**1.47**
10×15	**1.18**	1.17	**1.32**	**0.93**	0.87	**1.36**	1.17	0.65
10×20	1.01	**0.62**	0.61	0.94	**1.05**	0.40	**1.22**	**0.98**
10×30	**0.72**	**0.80**	**0.99**	**0.84**	0.40	0.65	0.60	0.60
15×15	1.16	0.93	**1.42**	**1.19**	**1.55**	**1.41**	0.75	0.60

Table 11. Sets of tested parameters for the linear combination for the tri-criteria experiments.

α	β	γ	α	β	γ	α	β	γ	α	β	γ
1	0	0	0.5	0.5	0	0.8	0.1	0.1	0.3	0.4	0.3
0	1	0	0.4	0.6	0	0.6	0.2	0.2	0.4	0.2	0.4
0.5	0.5	0	0.3	0.7	0	0.4	0.3	0.3	0.5	0	0.5
0.9	0.1	0	0.2	0.8	0	0.2	0.4	0.4	0.1	0.1	0.8
0.8	0.2	0	0.1	0.9	0	0	0.5	0.5	0.2	0.2	0.6
0.7	0.3	0	0	0	1	0.1	0.8	0.1	0.3	0.3	0.4
0.6	0.4	0	0.3	0.3	0.3	0.2	0.6	0.2	0.4	0.4	0.2

Table 12. Results for the linear combination with three criteria.

	sdata			edata			rdata			vdata		
$m \times n$	$\overline{\#Sol}$	$\overline{CPU}$ (s)	$\overline{CD}$	$\overline{\#Sol}$	$\overline{CPU}$ (s)	$\overline{CD}$	$\overline{\#Sol}$	$\overline{CPU}$ (s)	$\overline{CD}$	$\overline{\#Sol}$	$\overline{CPU}$ (s)	$\overline{CD}$
5×10	6	8.9	0.600	6	12.9	0.451	12	27.7	0.361	12.2	43.1	0.337
5×15	5.8	29.8	0.440	7	36.1	0.402	13.4	71.2	0.316	15	96.4	0.262
5×20	3.6	63.6	0.137	7.8	75.4	0.295	11.8	142.6	0.332	11.2	191.8	0.362
10×10	8.6	59.7	0.364	5.8	66.5	0.329	4.4	141.1	0.450	4.2	327.9	0.602
10×15	3.8	174.8	0.063	7.6	228.6	0.352	7.4	490.7	0.440	6.6	1204.8	0.502
10×20	5.2	334.1	0.402	4	444.1	0.387	7	1014.5	0.460	10.4	2337.1	0.313
10×30	3.4	733.9	0.049	4.8	966.2	0.556	7.6	2578.8	0.207	11	5805.3	0.337
15×15	5.2	425.8	0.411	4.8	479.9	0.280	5.2	1182.9	0.700	6.2	4129.3	0.336

4.3.3 *Considering three criteria*

Experiments have only been made with the linear combination tabu search algorithm on three criteria: makespan, maximum lateness and sum of tardiness.

$$Z(S) = \frac{\alpha \times C_{\max}(S)}{\max(C_{\max}) - \min(C_{\max})} + \frac{\beta \times L_{\max}(S)}{\max(L_{\max}) - \min(L_{\max})} + \frac{\gamma \times \sum T_j(S)}{\max(\sum T_j) - \min(\sum T_j)}$$

with:

- $\{\alpha, \beta, \gamma\}$ the coefficients,
- $C_{\max}(S)$, $L_{\max}(S)$ and $\sum T_j(S)$ are the makespan, the maximum lateness and the total tardiness of the solution,
- $\max(C_{\max})$ and $\min(C_{\max})$ are the greater and lower makespan value found until this point (the same for $L_{\max}$ and $\sum T_j$).

Table 11 shows the values of the 28 parameters values that have been tested. So, each instance of sdata ∪ edata ∪ rdata ∪ vdata (160 instances) is run 28 times.

In Table 12, column $\overline{\#Sol}$ indicates the average number of different solutions that have been found (maximum 28), column $\overline{CPU}$ indicates the average computation time for the 28 launches (in seconds) and column $\overline{CD}$ the average diversity.

We can note that the number of different dominant solutions is clearly below the maximum (28) and this number increases with the flexibility (except for 10×10 problems). The computation time is very short for small instances: 8.9 s on average to complete the

28 launches on a `sdata` 5×10, which means 317 ms in average for each launch. On the greatest instances `vdata` 10×30, about 5805 s (in average) are needed to complete the 28 launches, which means 207 s for each launch.

The diversity ($\overline{CD}$ indicator) does not seem to be dependent from the size of the instance or its flexibility.

To see the interest of the diversification introduced for $TS\ell c$ algorithm, the number of restarts during the search has been counted. For 48% of the tested problems, at least one restart has been realised. It means that in half of the cases this method helps improve the quality of the final solution. For these 48% of instances, the maximum number of restarts is 13, the average number of restarts is 2.01, and only one restart is made in 53% of the cases.

5. Conclusion

In this article we have solved a multi-criteria FJSSP. Two versions of a tabu search algorithm have been proposed for providing a set of non dominated solutions. One algorithm is based on the ϵ-constraint approach and the other on a linear combination of criteria. Experiments have been made for a single criterion problem, a problem with two criteria and a problem with three criteria. The methods are not very competitive in the case of a single criterion ($C_{\max}$). For two criteria, the best method for the quality of the solutions is the one based on the linear combination of criteria.

In the printing and boarding industry, some additional resources have generally to be taken into account when solving the problem. The future research direction consists of considering that some operations need more than one resource for being performed. The problem becomes an FJSSP with multi-processor operations. We will try to extend the proposed algorithms for considering this additive constraint.

References

Alvarez-Valdes, R., *et al.*, 2005. A heuristic to schedule flexible job-shop in glass factory. *European Journal of Operational Research*, 165, 525–534.

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 22, 158–183.

Choobineh, F., Mohebbi, E., and Khoo, H., 2006. A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup time. *European Journal of Operational Research*, 175, 318–337.

Dauzère-Pérès, S. and 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, 3–20.

Deb, K., *et al.*, 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2), 182–197.

Demirkol, E., Mehta, S., and Uzsoy, R., 1998. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109, 137–141.

Downsland, K., 1998. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106, 393–407.

Glover, F., 1989. Tabu search–Part I. *ORSA Journal on Computing*, 1, 190–206.

Ho, N.B., Tay, J.C., and Lai, E.M.-K., 2007. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179, 316–333.

Hurink, J., Jurisch, B., and Thole, M., 1994. Tabu search for the job-shop scheduling problem with multipurpose machines. *OR Spektrum*, 15, 205–215.

Jurisch, B., 1992. Scheduling Jobs in shops with multi-purpose machines. Thesis PhD. Germany: University of Osnabrük.

Kacem, I., Hammadi, S., and Borne, P., 2002. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60, 245–276.

Mastrolilli, M. and Gambardella, L.-M., 2000. Effective neighborhood function for the flexible job shop problem. *Journal of Scheduling*, 3 (1), 3–20.

Paulli, J., 1995. A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research*, 86, 32–42.

T'kindt, V. and Billaut, J.-C., 2006. *Multicriteria scheduling. Theory, models and algorithms*. 2nd ed. Berlin: Springer.