Engineering Applications of Artificial Intelligence 62 (2017) 80–95

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

An effective multi-start multi-level evolutionary local search for the flexible job-shop problem☆

S. Kemmoé-Tchomté[a], D. Lamy[b], N. Tchernev[b],*

[a] CRCGM EA 3849: Centre de Recherche Clermontois en Gestion et Management Clermont Auvergne University, Clermont-Ferrand, France
[b] LIMOS UMR 6158: Laboratoire d'Informatique de Modélisation et d'Optimisation des Système Clermont Auvergne University, Clermont-Ferrand, France

ARTICLE INFO

Keywords:
Flexible Job-shop
Metaheuristics
GRASP
Multi-level ELS
Neighbourhood structures

ABSTRACT

In this paper, an improved greedy randomized adaptive search procedure (GRASP) with a multi-level evolutionary local search (mELS) paradigm is proposed to solve the Flexible Job-shop Problem (FJSP). The FJSP is a generalisation of the well-known Job-Shop Problem with the specificity of allowing an operation to be processed by any machine from a given set. The GRASP metaheuristic is used for diversification and the mELS is used for intensification. Four different neighbourhood structures are formalised. A procedure for fast estimation of the neighbourhood quality is also proposed to accelerate local search phases. The metaheuristic has been tested on several datasets from the literature. The experimental results demonstrate that the proposed GRASP-mELS has achieved significant improvements for solving FJSP from the viewpoint of both quality of solutions and computation time. A comparison among the proposed GRASP-mELS and other state-of-the-art algorithms is also provided in order to show the effectiveness and efficiency of the proposed metaheuristic.

1. Introduction

Flexible job shop scheduling problem (FJSP) is an extension of the classical Job-shop scheduling problem (JSP) (see for instance some recent papers (Wang and Duan, 2014; Peng et al., 2015; Ku and Beck, 2016)). The FJSP, which takes into account routing flexibility, can be found in many applications in the modern manufacturing system. The specificity of FJSP is that operations related to a given job should be processed by a machine selected in a set of available ones. Considering the assignment problem resulting from this selection and the common scheduling problem relevant to the JSP, the FJSP is at least as complex as the JSP, which is known to be strongly NP-hard (Garey et al., 1976). Hence, solving such a problem advocates the use of metaheuristics in order to find valuable results, even though they are not optimal in a rather short computation time.

In this paper, an improved Greedy Randomized Adaptive Search Procedure (GRASP) is presented. Introduced by Feo et al. (1994), GRASP is a multi-start local search metaheuristic where each initial solution is constructed using a greedy randomised heuristic. The specificity of the proposed metaheuristic is to hybridise the GRASP with a multi-level Evolutionary Local Search (mELS) metaheuristic which extends the ELS procedure of Wolf and Merz (2007). Therefore, in this paper, a metaheuristic algorithm called GRASP-mELS is proposed. This metaheuristic has very effective searching ability and can balance the intensification and diversification very well by using four different neighbourhood structures during the search process. Experiments show that the proposed algorithm is able to return good solutions on a large number of instances found in the literature. The metaheuristic approach proposed in this paper differs from the FJSP literature according to the following points:

- An efficient construction heuristic, which generates good starting solutions.
- A local search based on critical path analysis, which encompasses both machine changes and operation permutations, is designed. This local search makes use of a procedure for fast estimation of the neighbours' quality.
- Definition of new neighbourhoods integrated in a highly randomised neighbourhood structure, where several permutations or machine changes are made to generate a neighbour of a solution.
- An improved intensification scheme, the multi-level ELS, embedded in a GRASP metaheuristic.

The paper is organised as follows: in the second section, the

☆ This article is an extended and improved version of a preliminary GRASP based metaheuristic provided in Kemmoé et al. (2016) which did not include procedure for fast estimation of the neighbourhood quality and neighbourhood structure formalisation. Experiments are updated and completed with Hurink benchmark.

* Corresponding author.

E-mail addresses: Sylverin.KEMMOE_TCHOMTE@u-clermont1.fr (S. Kemmoé-Tchomté), lamy@isima.fr (D. Lamy), tchernev@isima.fr (N. Tchernev).

http://dx.doi.org/10.1016/j.engappai.2017.04.002
Received 9 July 2016; Received in revised form 28 January 2017; Accepted 2 April 2017
Available online 18 April 2017

Nomenclature

n	Number of jobs to schedule
m	Number of machines
J	Set of jobs to schedule, $J=\{J_1, \ldots, J_r\}$
M	The set of all machines, $M=\{M_1, \ldots, M_m\}$
i,j,k,l	Indices for jobs
a,b,c,d	Indices for job operations
u,v,v',w,w'	Indices for machines
$M_{i,a}$	The set of available machines for operation $O_{i,a}$
n_i	Number of operations of job J_i
$O_{i,a}$	a^{th} operation of job J_i
$p_{O_{i,a},v}$	Processing time of operation $O_{i,a}$ on machine v
$s_{O_{i,a}}$	Starting date of operation $O_{i,a}$
$c_{O_{i,a}}$	Ending date of operation $O_{i,a}$
C_{max}	Total completion time of all operations
$C_{max}{}^{e}$	Estimated makespan
α	Assignment of all operations
π	Schedule of all operations
ρ	Critical path of $G(\alpha,\pi)$
$S(\alpha,\pi)$	Solution associated to α and π
$G(\alpha,\pi)$	Graph associated to a solution
$E(\alpha,\pi)$	Set of disjunctive arcs representing the processing orders on machines
Nn^{π}, $N_1^{\pi-\rho}$	Neighbourhood concerning schedules. n represents the number of times the neighbourhood is applied. ρ means the neighbourhood is applied on the critical path.
$Nn^{\alpha}N_1^{\alpha-\rho}$	Neighbourhood concerning assignments
B, B', B''	Critical blocks (consecutive operations in a critical path, assigned to the same machine)
$SM_{O_{i,a}}$	Operation succeeding $O_{i,a}$ in the assigned machine
$PM_{O_{i,a}}$	Operation preceding $O_{i,a}$ in the assigned machine

formulation of the Flexible Job-shop problem is introduced. Some related works, which motivated this study, are given in the third section. The fourth section refers to the metaheuristic and to the integrated procedures such as neighbourhood structures and the local search. Experiment results on four well known sets of problems taken from the literature are then presented and compared to the state of the art approaches. Finally, a conclusion in the sixth section of this paper summarises key points and future research.

2. Problem definition

2.1. Problem settings

The FJSP problem has been first introduced by Brucker and Schlie (1990). It is formally formulated as follows: a set J of n jobs, $J=\{J_1, \ldots, J_n\}$ must be scheduled on a set M of m machines, $M=\{M_1, \ldots, M_m\}$. Each job J_i consists in a number of n_i operations $O_{i,l}, \ldots, O_{i,\ n_i}$, which have to be processed in this order. An operation $O_{i,a}$ is allowed to be executed in any machine of a given set $M_{i,a} \subseteq M$. The processing time of an operation $O_{i,a}$ on a machine $M_v \in M_{i,a}$ is noted $p_{O_{i,a},v}$. Furthermore, each operation can be assigned to only one machine. Note that several operations of a same job could be assigned to the same machine. Considering that there is an assignment and a schedule for all the operations, the starting date of an operation is noted $s_{O_{i,a}}$ and its ending date is noted $c_{O_{i,a}}$. The objective is to minimise the total completion time, also called makespan and noted C_{max}, of all the operations. This criterion is given as follows: $C_{max} = \max_{1\le i\le r,1\le a\le n_i}(c_{O_{i,a}})$. Several assumptions are made for the classical problem, involving availability of all machines and release dates of all jobs at time 0.

2.2. Graph modelling

The classical representation of job scheduling problems is usually based on the disjunctive graph model presented by Roy and Sussmann (1964). It is possible to extend this representation to FJSP as shown by Dauzère-Pérès and Paulli (1997). Using the disjunctive graph model, any FJSP instance can be represented by a disjunctive graph $G=(V,A,E)$, where V represents the set of nodes, A the set of conjunctive (oriented) arcs and E the set of pairs of disjunctive (non-oriented) arcs. The nodes are associated with operations $O_{i,a}$. There are also two particular nodes to represent the start and the end of the schedule: (i) a source node, denoted 0 connected to the first operation of each job and (ii) a sink node, denoted *, linked with the last operation of each job. The conjunctive arcs are used to represent the precedence constraints of the different operations of the jobs and connect each pair of consecutive operations of the same job. Each pair of disjunctive arcs connects two operations, belonging to different jobs, which may be processed on the same machine. A solution corresponds to an acyclic subgraph that encompasses all conjunctive arcs and that contains at most one disjunctive arc for each pair of disjunctive operations. An optimal solution corresponds to the feasible subgraph with the minimal makespan C_{max}.

A solution can be noted $S(\alpha,\pi)$. In $S(\alpha,\pi)$, α represents a feasible assignment of each operation $O_{i,a}$ to a machine $M_v \in M(O_{i,a})$. ($v = \alpha(O_{i,a})$). A feasible assignment is a vector where each operation is assigned to one and only one machine. In $S(\alpha,\pi)$, π is a schedule of the operations, respecting job sequences, on all the machines in M. A solution Graph could be noted $G(\alpha,\pi)=(V,A,E(\alpha,\pi))$ with $E(\alpha,\pi)$ being the set of disjunctive arcs representing the processing orders on machines. The makespan of $S(\alpha,\pi)$ is the cost of any critical path in $G(\alpha,\pi)$. In the following, the notation B indicates a machine-block which is a set of consecutive operations needing the same machine in a critical path of $G(\alpha,\pi)$, following the definition of (Grabowski et al., 1986). Fig. 1 represents a solution for a problem with 4 jobs and 5 machines. Operations $O_{2,2}$, $O_{3,2}$ and $O_{4,2}$ form a machine-block since they are proceeded by the same machine M_5. Note that $O_{4,3}$ is also processed on machine M_5 but does not belong to the machine-block because of precedence constraints and therefore it is impossible to permute order of operations $O_{4,2}$ and $O_{4,3}$.

3. Related works

Two main approaches have been used in the research done so far concerning FJSP: *(i)* hierarchical solving, which consists in finding the best schedules while considering that assignments of machines are fixed (Brandimarte, 1993); or *(ii)* considering simultaneously the scheduling and assignment problems by using effective neighbourhoods (Mastrolilli and Gambardella, 2000). Actually, most of works are

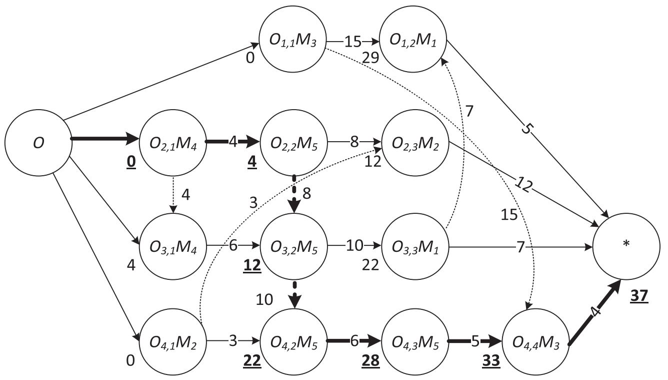

Fig. 1. A schedule of a problem with 4 jobs and 5 machines represented by a solution graph. Bold arcs show a critical path whose length, i.e.: the makespan, is 37.

focusing on this last point since the hierarchical approach appeared quite time-consuming in practice.

Teekeng and Thammano (2012) used a Genetic Algorithm (GA) to tackle the problem; the particularity of their metaheuristic is that it uses clustering of the population during the crossover phase. Zhang et al. (2011) also used a GA with good heuristic schemes to build the initial population, named Global Selection, Local Selection and Random Selection. Ishikawa et al. (2015) proposed a hierarchical multi-space competitive distributed GA to solve the problem. This algorithm consists in applying a GA on various solution spaces with exchange of information between populations.

González et al. (2013) proposed a Memetic Algorithm where the local search is based on Tabu Search (TS) to solve the FJSP with Setup Times. Li and Gao (2016) proposed a hybridisation of genetic algorithm and TS for the FJSP with the objective of minimising the makespan. Palacios et al. (2015) applied a GA hybridised with TS to the Fuzzy FJSP. The size of the neighbourhood is controlled using a filtering mechanism.

Thammano and Phu-ang (2013) designed an Artificial Bee Colony (ABC) where the initialisation is handled with a Harmony Search; this ABC extracts from local minimum using a Simulated Annealing (SA) procedure.

Sobeyko and Mönch (2016) tackle the FJSP with Total Weighted Tardiness as the optimisation criterion. Their metaheuristic is a hybridisation of Shifting Bottleneck procedure, Variable Neighbourhood Search, and an Iterative Local Search using Simulated Annealing to escape from local minima.

Hmida et al. (2010) proposed a search method based on tree exploration, named Climbing Depth-bounded Discrepancy Search (CDDS). This algorithm is tested with various neighbourhood structures. Yuan and Xu (2013a, 2013b) proposed two different approaches in two papers, one consisting in a Hybrid Differential Evolution algorithm for small and medium size instances, and the other consisting in a Hybrid Harmony Search combined with Large Neighbourhood Search for large instances. Kamble et al. (2015) worked on a multi-objective FJSP implying minimisation of makespan, maximal load on machines, total load, idle time on machines and total tardiness. A hybrid algorithm based on Particle Swarm Optimisation (PSO) and SA is proposed. Singh and Mahapatra (2016) worked on a quantum behaved PSO which uses mutation to prevent early stagnation of the particles in local minima. (González et al., 2015) proposed a Scatter Search with Path Relinking and TS (SSPR); their method seems to be one of the most efficient published for FJSP.

Furthermore, a state of the art proposed by Chaundry and Khan (2015) shows that most researches have been conducted between 2010 and 2013 whereas the problem has been known since 1990. This is mainly due to the need for more flexible production systems. What springs from their work, is that most of metaheuristics designed for the FJSP are population based metaheuristics (González et al., 2015; Palacios et al., 2015; Li and Gao, 2016). According to Chaundry and Khan (2015), only two papers from Rajkumar et al. (2010, 2011) are tackling extensions of the FJSP by using a GRASP based metaheuristic. However, no results are given concerning the performance of multi-start metaheuristics.

In an earlier paper Kemmoe-Tchomte et al. (2016) proposed a GRASP embedding a bi-level ELS for solving FJSP. This research proved that an optimisation approach using a single solution based metaheuristic (for more details see (Boussaïd et al., 2013)) is a promising research direction. Therefore, in this paper an improved metaheuristic is introduced. Here, the proposed GRASP-mELS uses a new procedure for makespan estimation that accelerates the neighbourhood exploration process, as the observed processing times for the local search procedure are two times faster. Also in this paper a deeper neighbourhood mathematical formalisation is presented. This formalisation allows to obtain an efficient local search and better computational results. Thus experimental results are better than the previous work of Kemmoe-Tchomte et al. (2016).

Researchers often find new upper bounds for available set of instances which shows that the problem is still wide open. However, as stressed by Chaundry and Khan (2015) all existing solutions referring to literature instances are not yet proven to be optima. As mentioned by González et al. (2015), none of the previously mentioned metaheuristics is providing the best solutions in all the literature set of instances. Furthermore, various extensions of the problem are yet to be studied. However before dealing with new problems involving Flexible Job-shops structures, it is important to tackle the basic problem. Thus a metaheuristic has been designed to provide near optimal solutions to this challenging problem. The key points of this metaheuristic are presented in the following section and are intrinsically linked to effective metaheuristic procedures.

4. GRASP-mELS metaheuristic

In scheduling problems, it is common to use concepts taken from the literature and adapted to the problem under study. Among the approaches used, disjunctive graph and its properties are addressed in several works. Also, numerous metaheuristics are used, such as Tabu Search, or Variable Neighbourhood Search for instance, and neighbourhood structures (Grabowski et al., 1986; Van Laarhoven et al., 1992; Dell'Amico and Trubian, 1993; Nowicki and Smutnicki, 1996; Mastrolilli and Gambardella, 2000; Gao et al., 2008; Karimi et al., 2012; González et al., 2015). The quality of the results depends of the way this knowledge is used (Yuan and Xu, 2013a).

For solving the FJSP, much attention has been given to the metaheuristics such as TS (Dauzère-Pérès and Paulli, 1997; Mastrolilli and Gambardella, 2000), GRASP (Rajkumar et al., 2011), GA (Pezzella et al., 2008) and so on. Most of them rely on existing metaheuristics which are improved to respond to encountered issues, thus leading to hybrid-metaheuristics (Li and Gao, 2016; González et al., 2015; Palacios et al., 2015). Recently, the well-known GRASP×ELS proposed by Prins (2009) has been used to provide solutions for Job-shop problem (Chassaing et al., 2014). The first idea was to use this metaheuristic for the FJSP. However, although a wide range of parameters have been tested in a Design Of Experiment (DOE), results were not convincing. Nevertheless, it has been chosen to explore deeply the track of the multi-start based metaheuristic structures by including several level of decision during the search process. Hence, the metaheuristic presented in this paper is an improved version of the GRASP×ELS, where GRASP procedure is jointly used with a multi-level ELS approach and thus this metaheuristic is noted GRASP-mELS. One of the feature of this metaheuristic algorithm is that it works on coding space and solution space alternatively (for more details see (Cheng et al., 1996)). Key points of the proposed GRASP-mELS for solving FJSP problems are:

- Graph representation such as disjunctive/conjunctive graph (Section 2.2)
- A Quasi-Direct Representation of Solution that is not a whole solution of the problem but a compact representation, such as a sequence of nodes or operations (called Bierwirth vector (Bierwirth, 1995));
- Efficient neighbourhood structures that allow to explore the solution space by switching from a solution to another;
- An effective metaheuristic, which allows a good balance between intensification and diversification.

In the following section a deep analysis is made to propose efficient GRASP-mELS, starting with neighbourhood structures used in this work.

4.1. Neighbourhood structures

Usually, metaheuristics embed various algorithms to explore the solution space. Neighbourhood structures are part of these algorithms. If in JSP, neighbourhood structures are operations oriented, in FJSP the assignment of operations on machines should also be taken into account. Thus, various neighbourhood structures can be used for FJSP (Mastrolilli and Gambardela, 2000; Karimi et al., 2012). In this study four neighbourhood structures are applied. Some properties about feasibility are also reminded.

4.1.1. Neighbourhood for the sequencing sub-problem

Two neighbourhood structures noted N_n^{π} and $N_1^{\pi-\rho}$ are defined here, where n corresponds to the number of times the neighbourhood is applied (1 is for a unique application of the neighbourhood), and ρ denotes the critical path. N_n^{π} and $N_1^{\pi-\rho}$ extend the neighbourhoods of Van Laarhoven et al. (1992) and Nowicki and Smutnicki (1996). These neighbourhoods were first defined for the classical JSP and are based on permutation of operations of a machine-block. Recall that a machine-block is a set of consecutive operations assigned to the same machine in a critical path of $G(\alpha,\pi)$.

In order to preserve feasibility of solutions when moving an operation $O_{j,b}$ before another operation $O_{i,a}$ in a machine-block, one must ensure that there is no path anymore from $O_{i,a}$ to $O_{j,b}$, otherwise there would be a cycle in the new graph. A sufficient condition for feasibility is given in Property 1.

Property 1. Let $S(\alpha,\pi)$ be a solution (i.e.: a schedule for which the directed graph $G(\alpha,\pi)$ is acyclic (Dauzère-Peres et al., 1997)) and let $(B \cup O_{i,a} \cup B' \cup O_{j,b} \cup B'')$ be a machine-block, where B, B' and B'' are sub-blocks of critical operations that may be empty. Let $G(\alpha,\pi)$ be the graph associated to $S(\alpha,\pi)$ and let $S'(\alpha,\pi')$ be a schedule created from S by moving $O_{j,b}$ just before $O_{i,a}$. Let s be the starting date of an operation and p the processing time of the operation once assigned to a machine v. $SM_{O_{j,b}}$ stands for "operation succeeding $O_{j,b}$ in the assigned machine". Then, $G'(\alpha,\pi')$ has no cycles if the following condition holds:

$$\left(s_{O_{j,b-1}} < s_{O_{k,c+1}} + p_{O_{k,c+1},v} + H\right) \wedge J_{O_{j,b}} \neq J_{O_{k,c}},\ \forall\ O_{k,c} \in O_{i,a} \cup B'$$

$$\text{where } H = \begin{cases} 0 & \text{if } SM_{O_{k,c+1}} = O_{j,b-1} \\ \min\left\{p_{O_{k,c+2},u},\ p_{SM_{O_{k,c+1}},v}\right\} & \text{otherwise} \end{cases}$$
$$\text{with } u = \alpha(O_{k,c+2}) \text{ and } v = \alpha(O_{k,c+1})$$

Definition 1. N_n^{π} structure: Let O^1 and O^2 be two sets of operations, with $O_i^1 \neq O_i^2$ and where O_i^1 and O_i^2 are assigned to the same machine. In a neighbouring solution the processing order of O_i^1 and O_i^2 is reversed for each i, provided that the condition of feasibility given in Property 1 holds for each couple $\{O_i^1; O_i^2\}$.

Property 2. Let $S(\alpha,\pi)$ and $S'(\alpha,\pi')$ be two solutions such that S' is obtained from S without modifying the processing order of any operation in a critical path of $G(\alpha,\pi)$. Then, $C_{max}(S') \geq C_{max}(S)$.

Considering Property 2, many of the neighbours generated by N_n^{π} will not improve the optimisation criterion. Hence, this structure will not be useful in a Local search procedure. However, it is particularly appropriate for diversification during the Neighbour generation in an ELS procedure, to allow the exploration of various solutions in the solution space.

Property 3. Let $S(\alpha,\pi)$ be a solution and ($B\ O_{i,a}\ B'\ O_{j,b}\ B''$) a machine-block in $G\ (\alpha,\pi)$, where B, B' and B'' are sequences of operations of the form $B = (O_{k,c}, \ldots, O_{l,d})$, $B' = (O_{k',c'}, \ldots, O_{l',d'})$ and $B'' = (O_{k'',c''}, \ldots, O_{l'',d''})$, with $|B|$, $|B''| \geq 1$ (B' could be an empty set). Even if the schedule $S'(\alpha, \pi')$ obtained from S by inserting $O_{j,b}$ just before $O_{i,a}$ is feasible, the makespan of S' cannot be lower than the makespan of S.

The same goes for a neighbour created by moving an operation $O_{i,a}$ after an operation $O_{j,b}$. Therefore, a move in N_1^{π}(only one permutation done) may only produce an improvement in the makespan if it does not involve inner modifications of the critical blocks. Considering the neighbourhood proposed by Nowicki and Smutnicki (1996), which consists in applying permutations of operations at the end of the Grabowsky's machine-block (Grabowski et al., 1986), the following reduced neighbourhood structure can be defined:

Definition 2. $N_1^{\pi-\rho}$ structure: Let $S(\alpha,\pi)$ be a solution. Let O be the set formed of all consecutive operations $O_{i,a}$ and $O_{j,b}$ at the beginning or at the end of each machine-block B of $G(\alpha,\pi)$. A neighbouring solution of $S(\alpha,\pi)$, denoted $S'(\alpha,\pi')$, is obtained by applying the permutation of two consecutive operations in O if it leads to the best improvement in the makespan of S, given that Property 1 is verified.

4.1.2. Neighbourhood for the assignment sub-problem

In the work of Mastrollili and Gambardella (2000) the notion of k-insertion is proposed, where the objective is to change the machine assignment of an operation to another machine m and then find the best insertion for this operation among the other operations scheduled on m. Here two simpler neighbourhood structures are used. They consist in changing the assignment of one or several operations.

Definition 3. N_n^{α} structure: Let $S(\alpha,\pi)$ be a schedule, let O be a set of n operations $O_{i,a}$. Let $\alpha(O,v)$ be the assignment obtained from α after reassigning each $O_{i,a}$ to a new machine $v \in M_{i,a}$. Then, $S'(\alpha(O, v), \pi)$ is a neighbouring solution.

As can be easily stated, a change in the assignment of an operation out of the critical path will not lead to a direct improvement in the optimisation criterion. A restriction of N_n^{α} is given in (González et al., 2015) and redefined below.

Definition 4. $N_1^{\alpha-\rho}$ structure: Let $S(\alpha,\pi)$ be a schedule, let $O_{i,a}$ be one critical operation in any critical path ρ of $G(\alpha,\pi)$. A neighbouring solution is obtained by applying N_1^{α} to this specific $O_{i,a}$.

4.2. Solution representation

Let us consider an example of FJSP composed of 3 jobs, which have respectively, 3, 2 and 3 operations as defined in Table 1. For each job, this table gives the set of operations and for each operation the possible machines (ranging from m$_1$, to m$_5$) and the processing time on these machines.

The non-oriented disjunctive graph illustrating this example is given in Fig. 2. In this graph, an arc (in full line) between two successive operations ($O_{i,a}$, $O_{i,a+1}$) represents the precedence (routing) constraint of the Job i. It is not weighted by any distance between these operations as no machines are assigned yet. Each pair of disjunctive arcs is represented with a dotted edge and represents the constraint between two operations sharing the same possible machine resource.

A solution is an oriented conjunctive-disjunctive graph which encompasses arcs only. As the FJSP is an assignment and a scheduling problem, first, a machine assignment is done as stated in Fig. 3. Such an assignment is represented by a vector MA (Machine Assignment). Each element of MA is the number of the machine assigned to a given operation, as proposed by Chen et al. (1999). Hence, the assignment shown in Fig. 3 would be represented in this way: 1 3 2 3 4 1 4 2. This vector means that the first operation of Job 1 is processed on Machine M_1; the second operation of Job 1 is processed by machine M_3; the third operation of Job 1 is proceeded by machine $M_{2,}$ and so on for Job

Table 1
Example of a Flexible Job-shop problem. Each couple (M_u; p) refers to the possible machine assignment for an operation and the corresponding processing time.

Op. Jobs	$O_{i,1}$	$O_{i,2}$	$O_{i,3}$
i=1	(M_1; 10), (M_2; 5)	(M_3; 4)	(M_4; 8) (M_2; 5)
i=2	(M_3; 3) (M_5; 4)	(M_4; 7) (M_3; 9)	
i=3	(M_1; 11) (M_5; 5) (M_3; 4)	(M_4; 12)	(M_2; 21) (M_4; 10)

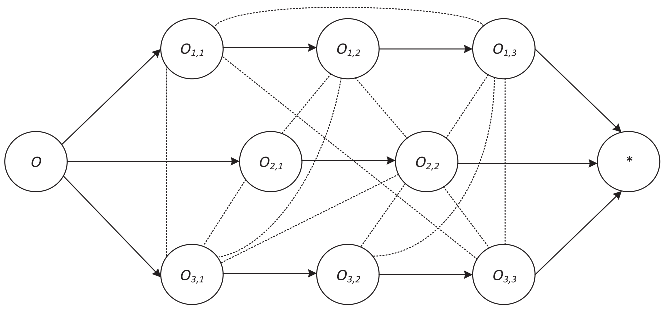

Fig. 2. A problem with 3 jobs and 5 machines represented with a disjunctive graph.

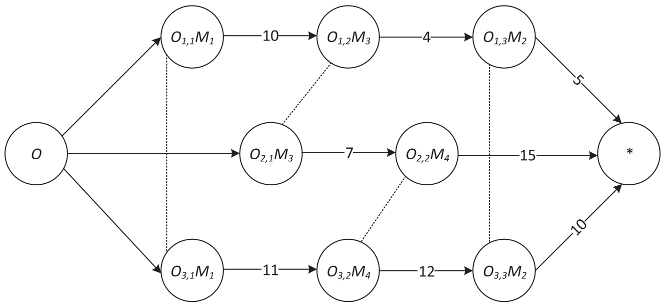

Fig. 3. A problem with 3 jobs and 5 machines represented with a disjunctive graph where each operation is assigned to one and only one machine.

2 and Job 3.

Fig. 3 shows the new non-oriented disjunctive graph after machine assignments. It can be seen that only edges relevant to this assignment are still present as all disjunctive edges which are not relevant have been removed because they are not involved in the schedule. Thus, $O_{1,1}$ and $O_{3,1}$ should be scheduled on the same machine M_1, and so there is still an edge linking these operations. Finally, the acyclic conjunctive-disjunctive graph of a solution is given in Fig. 4. In this graph, all pairs of disjunctive edges are reduced to one arc and represent the sequence of operations processed on the same machine. Arcs in bold constitute the critical path and underlined numbers represent earliest starting times of operations.

The arcs between operations of jobs which use the same machines define the operations sequence on machines. The graph of Fig. 4 represents a solution in which:

1. on machine M_1 the sequence is: $O_{1,1}$, $O_{3,1}$;
2. on machine M_2 the sequence is: $O_{1,3}$, $O_{3,3}$;
3. on machine M_3 the sequence is: $O_{2,1}$, $O_{1,2}$;
4. on machine M_4 the sequence is: $O_{2,2}$, $O_{3,2}$;
5. no operations scheduled on machine M_5.

Since each operation belongs to one job only, a common representation of a solution consists in giving the job sequence on machines. With such notation, the previous solution is noted: Machine 1: job 1, job 3; Machine 2: job 1, job 3, etc. However, Bierwirth (1995) introduces an alternative representation as a sequence of job number. Based on his proposal, the solution of Fig. 4 is encoded to: 1 3 2 2 1 1 3 3. This kind of representation is called: sequence with repetition. Such sequence is represented by a vector *OS* (Operation Selection), read from left to right. In this vector the first 1 corresponds to the first operation of Job 1; the second value is a 3 which refers to the first operation of Job 3; the third value is the first operation of Job 2, followed by the second operation of Job 2, and so on. The main advantage of such a vector is that it allows to obtain an acyclic oriented disjunctive graph and therefore this sequence can be efficiently managed by a greedy algorithm or a metaheuristic. Please note that there exist several such sequences that lead to the same oriented disjunctive graph. In the Fig. 5, these data structures are represented.

As can be seen in this figure, the first and second operations of the first job are scheduled on machines 1 and 3 respectively, the first and second operations of job 2 are scheduled on machines 3 and 4, hence, any permutation in the sequence of operations will not affect the Machine Assignment vector. Moreover, changing a given machine assigned to a given job operation does not have an impact on the Operation Selection vector either.

After presenting the coding of solutions, an important part is the metaheuristic that will allow to explore the solution space.

4.3. Components for GRASP-mELS

In this section, an original metaheuristic approach based on efficient neighbourhoods for the FJSP with makespan minimisation is presented. Algorithms for improving solutions' quality as well as fast estimation of neighbours' makespan are proposed. These components are incorporated into a metaheuristic. This metaheuristic uses GRASP with multi-level ELS as improvement method.

Fig. 6 and Algorithm 1 illustrate the GRASP-mELS principles and implementation in pseudo-code, where *nb_g* stands for the number of GRASP restarts (diversification step), *nb_ol* and *nb_els* stand for the number of iterations of mELS and *nb_n* corresponds to the number of neighbours that must be generated (intensification step). In this algorithm, the function *Q* refers to the objective function that is optimised in the process (i.e.: makespan). The GRASP-mELS iterations are carried out in line 1–26. In line 1, the variable that stores the best solution found is initialised.

Algorithm 1. GRASP-mELS

```
Input
Data:       structure storing all problem information;
nb_g :      number of restart (GRASP);
nb_ol :     number of iterations of first mELS level;
nb_els :    number of iterations of second level of mELS;
nb_n :      number of neighbours generated in mELS;
Ouptut:
S* :        Best found solution;
Variables:
S, S' :     a Solution and its neighbour used;
nS, nS*     allowed makespan to select jobs to insert in the
:           sequence;
elS* :      best found solution during ELS loop;
BEGIN
1.          S*:= Ø;
2.          For g := 1 TO nb_g Do
3.          S:= Construction_Phase;
4.          S:= Local_Search_Phase;
5.          IF (S* == Ø OR Q(S) < Q(S*)) THEN S*:= S END IF
6.          FOR ol := 1 TO nb_ol DO
7.          S':= Apply_Perturbation_On_S;
8.          elS*:= ø;
```

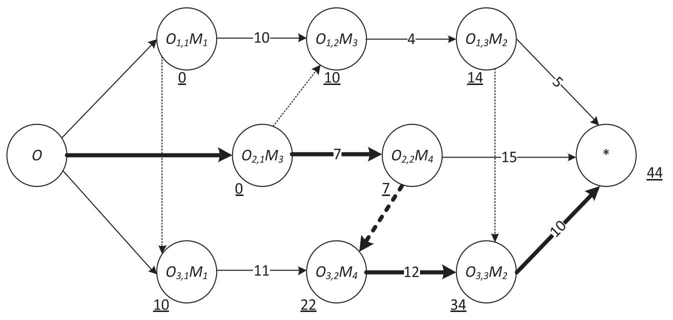

Fig. 4. A schedule of a problem with 3 jobs and 5 machines represented by a solution graph. Bold-face arcs show a critical path whose length, i.e.: the makespan, is 44.

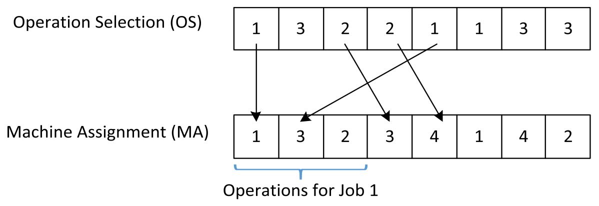

Fig. 5. Coding of a solution. The arcs model the correspondence between operations in the Operation Selection vector, and the machine assigned to each operation in the Machine Assignment vector.

```
9.    FOR $els$ := 1 TO $nb\_els$ DO
10.   $nS^* := \emptyset$;
11.   FOR $n$ := 1 TO $nb\_n$ DO
12.   $nS$ := Apply_Perturbation_On_$S'$;
13.   $nS$ := Local_Search_Phase;
14.   IF($Q(nS) < Q(nS^*)$) THEN $nS^* := nS$ END IF
15.   ENDFOR
16.   $S' := nS^*$;
17.   IF ($Q(nS^*) < Q(elS^*)$) THEN $elS^* := nS^*$ END IF
18.   END FOR
19.   $S := elS^*$;
20.   IF ($Q(S) < Q(S^*)$) THEN
21.   $S^* := S$;
22.   $ol := 1$;
23.   END IF
24.   END FOR
25.   END FOR
26.   return $S^*$;
END
```

The block of instructions between lines 2 and 25 is executed iteratively, where each iteration consists of three phases as follows:

(i) Construction phase (line 3): initial solutions are built, one element at a time, with a greedy randomised heuristic. At each construction iteration the next element to be added is determined by ordering all elements in a candidate list. The probabilistic component of a GRASP in this phase is characterised by randomly choosing one of the candidates in the list, not always the best.

(ii) Local search phase (line 4): since the solution returned after construction phase is not guaranteed to be locally optimal, a local search is performed. In line 5, the quality of the obtained solution is compared to the current best found and, if necessary, the solution is updated.

(iii) Multi-level Evolutionary local search (mELS) phase (line 6–31): to better investigate the neighbourhood of the local optimum, a neighbour of the current solution is generated. This neighbour is then used as starting point of the classical ELS phase (line 8–18). In line 20, the quality of the obtained solution is compared to the current best found and, if necessary, the best solution is updated and the first level mELS is restarted in order to further explore the neighbourhood of the newly best found solution.

In Fig. 6, a first ELS (mELS1 box) is applied after constructing an initial solution. During this ELS, the best neighbour obtained is saved (filled circle). Once the ELS reaches a stopping criterion, a neighbour of the best neighbour (previously saved) is generated, and a new ELS (mELS2 box) is applied on this new solution. This could be repeated several times. Once the number of mELS restarts is reached, a new solution is generated, and the exploration process may start again.

The following sub-sections aim at presenting each of the 3 phases proposed before.

4.4. Construction phase

The GRASP construction phase aims at building a feasible solution, by assigning and scheduling one operation at each step until all operations are planned. It is based on the proposal of Binato et al. (2001) for the JSP where the objective is to build at each step a Restricted Candidate List (RCL). From the set containing the jobs with all prior operations already scheduled a subset of jobs is selected respecting the criterion “which is the smallest increase in the makespan of the already scheduled jobs”. Recall that $O_{i,a}$ is the a^{th} operation of job J_i, $M_{i,a}$ is the set of available machines for $O_{i,a}$. Let us consider $O_{i,a,v}$

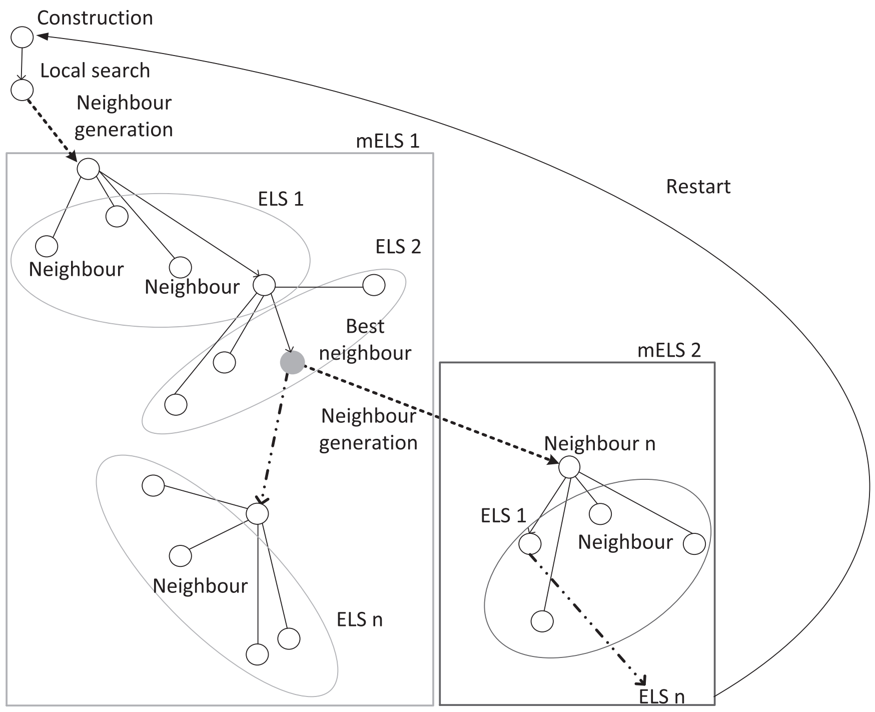

Fig. 6. Graphic visualisation of GRASP-mELS run. First an Evolutionary Local Search is applied. At the end of this algorithm, the best neighbour obtained is mutated to generate a new solution. A new Evolutionary Local Search is then applied starting from this solution. The process continues until a new solution is constructed, or until a stopping criterion is met.

the operation $O_{i,a}$ processed on machine v, $v \in M_{i,a}$. At a given iteration of the construction phase, at most $|J|$ operations can be planned, since each operation of a job depends of prior operations of this job. This set of assignable/schedulable operations is noted O^{as}, while operations already planned are in set O^{p}. At each iteration of the construction phase, a list L is built. This list is the subset of operations that could be processed at this iteration. Then, a candidate list CL is constructed, where each operation is assigned to its most promising machine:

$$CL = \left\{ O_{i,a} \in L | C_{\max}(O_{i,a}) = \min_{v \in M_{i,a}}(C_{\max}(O_{i,a,v})) \right\}$$

The smallest makespan among all the operations in CL is noted $\underline{C_{\max}}$; the highest is noted $\overline{C_{\max}}$. The restricted candidate list (RCL) is then defined as in the following equation. $RCL = \{O_{i,a} \in CL | \underline{C_{\max}} \le C_{\max}(O_{i,a}) \le \underline{C_{\max}} + \lambda(\overline{C_{\max}} - \underline{C_{\max}})\}$, where $\lambda \in [0,1]$. The next operation entering the subset O^{p} is then chosen randomly among the operations in RCL. As stressed in the work of Binato et al. (2001), the selection of an element in the RCL could follow any probability distribution. In this work, a uniform distribution is chosen in order to give each element the same probability of being selected. Algorithm 2 provides the construction phase procedure in a detailed approach.

Algorithm 2. Generate_Solution

```
Input:
Data :            structure storing all problem information;
λ :               threshold value to compute h during the
                  algorithm; h is a variable corresponding to the
                  computed allowed makespan used to select jobs
                  to insert in the sequence;
Ouptut:
S :               empty solution at the beginning, containing
                  sequence at the end;
Variables:
minMkpn,          minimal and maximal makespan at each step;
maxMkpn:
h :               allowed makespan to select jobs to insert in the
                  sequence;
RCL :             Restricted Candidate List built with selected jobs;
infinite :        large constant;
mkpn :            temporary makespan when searching for an
                  appropriate machine for the job to be processed;
BEGIN
1.                S := new Solution;
2.                FOR i :=1 TO ProblemSize DO
3.                minMkpn := infinite; maxMkpn := 0;
4.                FOR EACH job that could be scheduled DO
5.                mkpn := infinite;
6.                FOR EACH machine that could process the job
                  DO
7.                Compute each important value relevant to the job
8.                insertion with the current machine (makespan,
9.                starting date,...);
10.               IF the makespan is better than mkpn THEN
11.               Update mkpn; Store the computed useful values;
12.               END IF
13.               END FOR
14.               Update minMkpn if the makespan is lower;
15.               Update maxMkpn if the makespan is larger;
16.               END FOR
17.               Compute h := minMkpn+ λ *(maxMkpn-
                  minMkpn);
18.               Build RCL with jobs whose makespan are lower
                  than h;
19.               Choose randomly a job in the RCL;
20.               Insert the chosen job into the operation sequence
                  of S;
21.               Insert the selected machine into the machine
                  sequence of S;
22.               Save other definitive values (start date, end date,
                  ...);
23.               END FOR
24.               return S;
END
```

4.5. Local search phase

To improve the quality of the solutions, it is necessary to exchange the order of two consecutive operations sharing the same machine and/or change the machine assignment of a given operation on the critical path. In other words, to improve the solution it is essential to solve two problems simultaneously: the machine assignment and the job sequencing. For the incumbent problem, the local search is exploring the critical path. At each node the algorithm tries to reallocate the operation to another machine (assignment problem) using neighbourhood structure $N_1^{\alpha-\rho}$ and then applies a local search based on the neighbourhood structure $N_1^{\pi-\rho}$. Highlights of this local search are given in Algorithm 3. In this algorithm, the neighbourhood of a solution is explored in two ways: by doing machine changes, and by applying permutations on disjunctive operations. The line 1 consists in applying a specific local search based on permutation of machine-block operations (LocalSearch_Disj). The objective in using this first local search based on disjunctions in the graph is to start the local search from a solution which is a local minimum considering the current assignment. Then the local search starts exploring the critical path (lines 3–18). During this exploration, the local search successively changes the machine assignment of a critical operation and performs a disjunctive local search in order to schedule efficiently all the operations. Since the local search has been applied at the beginning of the algorithm (line 1), only assignments that are different from the current one are tested for a given operation. As the Disjunctive local search is a key procedure of the local search scheme, it is presented in Algorithm 4.

Algorithm 3. LocalSearch

```
Input/Output:
S :          solution to improve;
data :       structure storing all problem information;
Variables:
op :         operation on critical path;
ok :         boolean representing improvement of solution;
sav,         temporary solutions;
newS :
BEGIN
1.           LocalSearch_Disj(S,data);
2.           Initialise op at the end of critical path of S;
3.           WHILE op != 0 DO
4.           IF ( number of possible machines for op > 1) THEN
5.           ok := false;
6.           sav := S;
7.           FOREACH machine different than the current one
             DO
8.           Change machine of operation op in S;
9.           ok := LocalSearch_Disj(S,data); // if S is improved
             during LocalSearch_Disj, then ok is set to true;
10.          IF ok THEN
11.          op := last op on critical path of S;
12.          ELSE
13.          op := predecessor of op in critical path of S;
14.          END IF
15.          END FOR
```

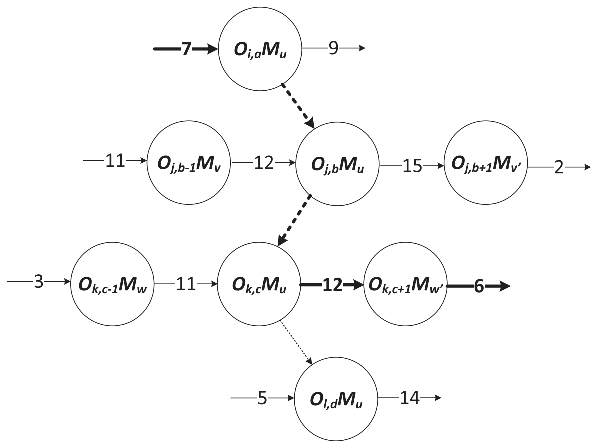

Fig. 7. Example of disjunctive arcs in a graph (arcs in bold are on the critical path).

```
16.     ELSE
17.     op ≔ predecessor of op in critical path of S;
18.     END IF
19.     END WHILE
END
```

In Algorithm 4, the first WHILE loop (lines 3–24) explores the critical path from the sink node * to the beginning node 0. The second WHILE loop (lines 5–15) is searching for permutations of operations which could improve the current solution. Finally, if a permutation that could improve the quality of the solution is found during the WHILE loop (line 5–15), this permutation is applied and an evaluation is done to preserve the integrity of the solution (line 16–23). The evaluation algorithm for the FJSP is relying on a longest path computation. This algorithm is a Bellman-Ford-like algorithm which uses the structure of the previously mentioned representations to make an evaluation in *O(n)*. For each operation, the longest path computation sets the earliest starting time of all operations and the makespan is evaluated.

Algorithm 4. LocalSearch_Disj

```
Input/Output:
S :             solution to improve;
Data :          structure storing all problem information;
Output:
ok :            boolean representing improvement of solution;
Variables:
op, prec :      operation and its predecessor on critical path;
mkpn, :         temporary makespan after estimation;
savOp,
savPrec:        save operation and its disjunctive predecessor;
newS, sav :     temporary solutions;
BEGIN
1.      mkpn ≔ Q(S); //initialise mkpn to the value of S makespan
2.      data.lPath // longest path from any operation to sink node
3.      WHILE !ok DO
4.      op := *; //initialise op at the sink of critical path;
5.      WHILE op != 0 DO
6.      IF (op has a disjunctive predecessor) THEN
7.      prec ≔ disjunctive predecessor of op;
8.      $C_{max}^e$ ≔ Estimate_Cmax(prec, op, data.lPath);
9.      IF $C_{max}^e$ is better than mkpn THEN
10.     mkpn ≔ $C_{max}^e$;
11.     savOp ≔ op; savPrec ≔ prec; ok ≔ true;
12.     END IF
13.     END IF
14.     op ≔ predecessor of op in critical path;
15.     END WHILE
16.     IF ok THEN
17.     newS ≔ S;
18.     Apply permutation of savOp and savprec in newS; // $N_1^{\pi-\rho}$
19.     Evaluate newS;
20.     IF Q(newS) is better than Q(S) THEN
21.     S ≔ newS; ok ≔ false; mkpn ≔ Q(S);
22.     END IF
23.     END IF
24.     END WHILE
25.     return ok;
END
```

As can be noted, a lot of disjunctive arcs could be permuted during the local search (line 5–15), hence evaluating each of these permutations could be time consuming. Thus, during the local search phase, each permutation of two consecutive operations on the critical path that could improve the quality of the makespan is estimated. As can be seen in line 8, a call to a procedure named Estimate_Cmax is done. Estimation procedures are really efficient in saving computation time, and are faster than an evaluation procedure. An estimation function was implemented and is presented in Algorithm 5. The best estimated permutation is kept (line 9–12) and applied (line 16–23). This estimation is based on the length of paths going from each operation to the sink node. In Fig. 7, a sub-graph of a global problem is given where the critical path is going through operations $O_{i,a}$, $O_{j,b}$, $O_{k,c}$ and

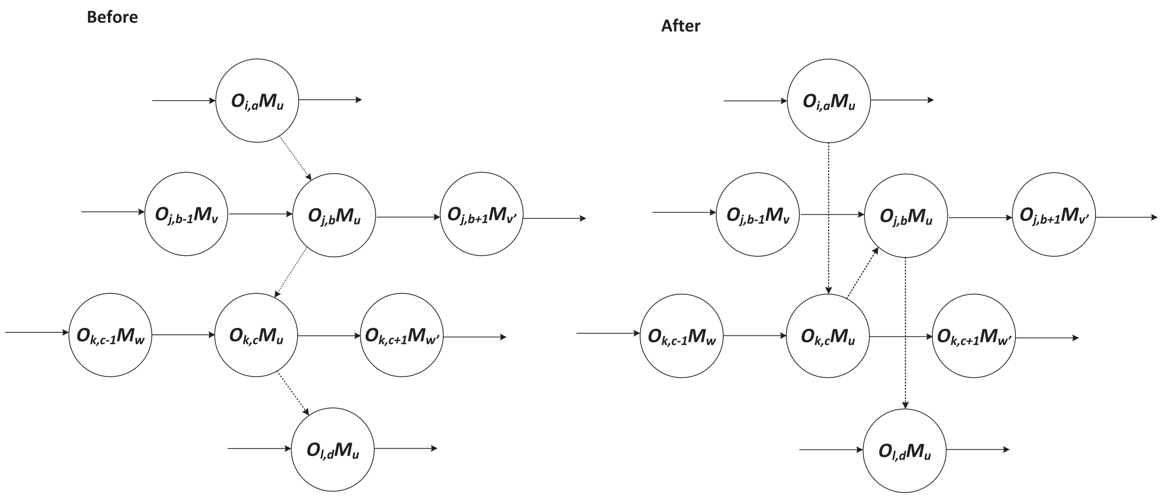

Fig. 8. Example of a swap between operations on a critical path. The arc between operations $O_{j,b}$ and $O_{k,c}$ in Fig. 7 (Before) is reversed (After).

$O_{k,c+1}$.

The objective is to estimate the potential improvement in makespan once the processing order of $O_{j,b}$ and $O_{k,c}$ has been changed on Machine M_u. During this estimation, the hypothesis is that the disjunctive arc between these two operations is reversed, as shown in Fig. 8. Figs. 8–10 show the principles of the estimation procedure, considering that operations $O_{j,b}$ and $O_{k,c}$ are in disjunctions in a critical path. Values in the arc are no longer present as they are not useful to understand the procedure. The concepts of head ($s_{o_{k,c}}$) and tail ($q_{o_{k,c}}$) of an operation $O_{k,c}$, needed to formalise the estimation procedure are reminded below.

$$s_{start} = 0, \;\; q_{end} = 0$$

$$s_{O_{k,c}} = \max\left(s_{O_{k,c-1}} + p_{O_{k,c-1},\alpha(O_{k,c-1})} \,,\; s_{PM_{O_{k,c}}} + p_{PM_{O_{k,c}},\alpha\left(PM_{O_{k,c}}\right)} \right)$$

$$s_{end} = \max_{O_{k,c}\in PJ_{end}, v=\alpha(O_{k,c-1})}\left\{ s_{O_{k,c}} + p_{O_{k,c},v} \right\}$$

$$q_{O_{k,c}} = \max\left(q_{O_{k,c+1}} + p_{O_{k,c+1},\alpha(O_{k,c+1})},\; q_{SM_{O_{k,c}}} + p_{SM_{O_{k,c}},\alpha\left(SM_{O_{k,c}}\right)} \right)$$

$$q_{start} = \max_{O_{k,c}\in SJ_{start}, v=\alpha(O_{k,c+1})}\left\{ q_{O_{k,c}} + p_{O_{k,c},v} \right\}$$

s_{start} is the release date of the starting node; q_{end} is the length of the path going from the sink node to itself. SJ_{start} and PJ_{end} respectively denote the set composed by the first and the last operations of each job. *SM* stands for "Successor operation in Machine sequence" (i.e.: $SM_{O_{k,c}} = O_{l,d}$ in Fig. 8), and *PM* is for "Predecessor operation in Machine Sequence".

Fig. 8 shows the partial disjunctive graphs with the hypothetical swap between $O_{j,b}$ and $O_{k,c}$.. Please note that the graph representation is *as if* the permutation was really done in the *OS* vector. Considering this, the estimated makespan based on the path from $O_{k,c}$ to sink node *(i.e. $C_{max}{}^{e}(O_{k,c})$) is computed as follows.

$$s_{O_{k,c}} = \max\left\{ s_{O_{k,c-1}} + p_{O_{k,c-1},M_w},\; s_{O_{i,a}} + p_{O_{i,a},M_u} \right\}$$

$$C_{\max}{}^{e}(O_{k,c}) = s_{O_{k,c}} + p_{O_{k,c},M_u} + q_{O_{k,c+1}} + p_{O_{k,c+1},M_{w'}}$$

In Fig. 10, two values are computed: the estimated makespan using the length of the path from $O_{j,b+1}$ to *(i.e. $C_{max}{}^{e}$ $(O_{j,b+1})$) and the estimated makespan (i.e. $C_{max}{}^{e}(O_{l,d})$) from the new disjunctive successor of $O_{j,b}$ (which was the successor of $O_{k,c}$ in the beginning) to *. This is done as follows:

$$s_{O_{j,\ b}} = \max\left\{ s_{O_{j,\ b-1}} + p_{O_{j,\ b-1}},\, M_v,\; s_{O_{k,c}} + p_{O_{k,c}},\, M_u \right\}$$

$$C_{\max}{}^{e}(O_{j,b}) = s_{O_{j,b}} + p_{O_{j,b},M_u} + q_{O_{j,b+1}} + p_{O_{j,b+1},M_{v'}}$$

$$C_{\max}{}^{e}(O_{l,d}) = s_{O_{j,b}} + p_{O_{j,b},M_u} + q_{O_{l,d}} + p_{O_{l,d},M_u}$$

The highest value between $C_{max}{}^{e}(O_{k,c})$, $C_{max}{}^{e}(O_{j,b})$ and $C_{max}{}^{e}(O_{l,d})$ is then returned as the result of the estimation of permuting the arc between $O_{j,b}$ and $O_{k,c}$.

$$C_{\max}{}^{e} = \max\{\, C_{\max}{}^{e}(O_{k,c}),\; C_{\max}{}^{e}(O_{j,b}),\; C_{\max}{}^{e}(O_{l,d})\}$$

Principles and implementation in pseudo-code of the algorithm are given in Algorithm 5. This algorithm begins with variables initialisation (lines 1–3). There are three **IF** statements (lines 4–8; 12–16; 17–21) which estimate the makespan after hypothetical permutation of two operations sharing the same machine at the extremities of a given machine block. The algorithm ends with estimated makespan line 23.

Algorithm 5. Estimate_Cmax

```
Input/Output:
opFather,       consecutive operations on the critical path to be
opSon :         permuted;
data.lPath[]    longest path from any operation to sink node;
Output:
cmax_esti :     currently scheduled after opFather;
Variables:
sDate, eDate    estimated starting or ending date of an operation;
cmaxOp :        estimated makespan from opSon;
cmaxOpFC :      estimated makespan from opFather (conjunctive
                arc);
cmaxOpFD :      estimated makespan from opFather (disjunctive
                arc);
op :            temporary operation;
BEGIN
1.              sDate ≔ starting date of opSon supposing that opF
                is no
2.              longer its predecessor;
3.              eDate ≔ ending date of opSon;
4.              cmaxOp ≔ sDate + data.lPath[opSon];
5.              If opSon has a conjunctive successor Then //
                refine estimation
6.              op ≔ conjunctive successor of opSon;
7.              cmaxOp ≔ eDate + data.lPath[op];
8.              End If
9.              sDate ≔ starting date of opFather considering
                opSon as
10.             its new predecessor;
11.             eDate ≔ ending date of opFather;
12.             cmaxOpFC ≔ sDate; cmaxOpFD ≔ sDate;
13.             If opFather has a conjunctive successor Then
14.             // refine estimation
15.             op ≔ conjunctive successor of opFather;
16.             cmaxOpFC ≔ eDate + data.lPath[op];
17.             End if
18.             If opSon has a disjunctive successor THEN //
                refine estimation
19.             op ≔ disjunctive successor of opSon;
20.             cmaxOpFD ≔ eDate + data.lPath[op];
21.             End If
22.             cmax_esti ≔ max(cmaxOpFD, cmaxOpFC,
                cmaxOp);
23.             Return cmax_esti;
END
```

A computational experiment has been conducted to assess the quality of the estimation procedure. The estimated makespan $C_{max}{}^{e}$ is compared with the evaluated one C_{max}. During this experiment 300

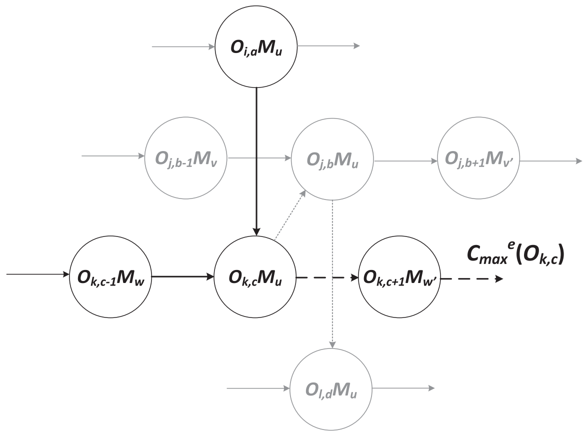

Fig. 9. Example of computing the estimated makespan from $O_{k,c}$ with the estimation procedure. $C_{max}{}^{e}(O_{k,c})$ is the sum of the estimated starting date of operation $O_{k,c+1}$ plus the length of the path going from $O_{k,c+1}$ to the sink node *.

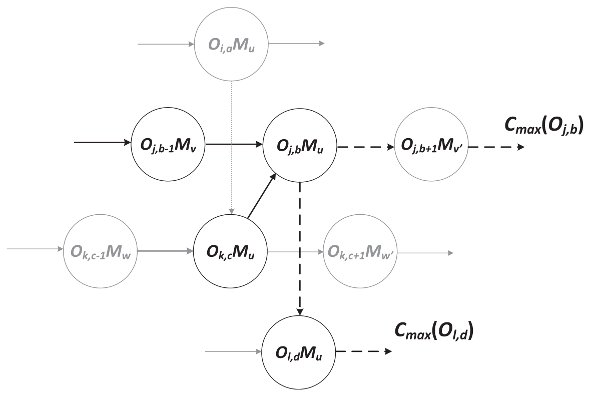

Fig. 10. Example of computing the estimated makespan from $O_{j,b}$. $C_{max}{}^e(O_{j,b})$ is computed by summing the estimated starting date of $O_{j,b+1}$ and the length of the path going from $O_{j,b+1}$ to the sink node *. The same applies for $O_{l,d}$ to obtain $C_{max}{}^e(O_{l,d})$.

million permutations on critical path operations were done. Results show that: 84% $C_{max}{}^e=C_{max}$; 16% $C_{max}{}^e < C_{max}$ and never $C_{max}{}^e > C_{max}$. Therefore, the estimation algorithm is really effective. In addition, computational time used for local search is reduced by 50% thanks to this estimation procedure. Using this estimation procedure the GRASP-mELS is allowed to search the solution space longer considering the time limit, and the quality of solutions has been improved by 23% in average.

As stressed before, after having estimated the quality of the permutation of all disjunctive operations in the critical path, the best move is applied (line 18 of Algorithm 4). One could think that such a move consists only in swapping the two jobs corresponding to the operations in disjunction. However, working on permutations inside the *OS* vector implies doing more than a simple swap. Actually, because of the implicit renumbering of operations inside the *OS* vector some movements could lead to unwanted results as stressed in Fig. 11. The first occurrence of Job 3 appears before the third occurrence of Job 2 in the vector, and the corresponding operations are in disjunction, thus the operation related to Job 3 is processed before the one of Job 2. However, after performing the swap, it appears that the first operation of Job 3 is still scheduled before the second of Job 2 as shown in the second part of Fig. 11.

To avoid this case, a simple procedure reorders the vector between the index of first occurrence of Job 3 and third of Job 2 according to the starting and ending dates of operations between the two that must be swapped. Actually, all operations that end before the ending date of the first operation to be swapped (Job 3 in the first part of Fig. 12) are placed before it, and all the operations that start after the last one (Job 2 in the first part of Fig. 12) are placed at the end of the sub-sequence. This leads to having both operations to be exchanged on the machine being side by side in the vector as shown on the second part of the Fig. 12. This operator and the estimation procedure have been used into the local search (Algorithm 3).

4.6. Neighbour generation

In the GRASP-mELS, neighbours of solutions are generated at two moments: at line 9 of Algorithm 1, and during the ELS phase. In this study, two different neighbourhood structures are used for this purpose. The first one consists in applying the N_n^π structure where n is randomly selected between 1 and the average number of machines per operation. The second neighbourhood consists in applying N_n^α in order to change the machine assignments. This is done randomly the same way as in the first neighbourhood.

These two neighbourhood structures are chosen randomly based on the flexibility (*flex*) of the instance of a problem that must be solved. The flexibility corresponds to the average machine number per operation in a problem. Then, given a random real number r selected in [0,20] – 20 is a value empirically chosen using a computational experiment, and a number a is calculated by multiplying a random number selected in [0,1] by *flex*; if r is lower than a then N_n^α is applied otherwise N_n^π is selected. Selection of the neighbourhood is done as stressed in Algorithm 6.

Algorithm 6. NeighbourGeneration

```
Input:
a,r :          random numbers
Input/Output:
NS :           copy of the current solution;
BEGIN
1.             IF $r < a$ THEN
2.             apply $N_n^\alpha$ on NS; //n is chosen randomly in [1, flex]
3.             ELSE
4.             apply $N_n^\pi$ on NS;
5.             END IF
END
```

4.7. Differences from the literature

As stressed by Chaundry and Khan (2015), metaheuristics have been successfully used to provide good solutions to the FJSP. Hence, this work relies on some approaches taken from previously published methods, and contains several improvements. For example, the first idea was to use the well-known GRASP×ELS proposed by Prins (2009), which has been used to provide solutions to the JSP (Chassaing et al., 2014). Although a wide range of parameters have been tested in a DOE, results were not convincing. Nevertheless, it has been chosen to explore deeply the track of the GRASP×ELS structure by including several levels of decision during the search process. Hence, the metaheuristic presented in this paper is an improved version of the GRASP×ELS, where the GRASP procedure is jointly used with a multi-level ELS approach. Thus, this metaheuristic is noted GRASP-mELS.

The construction of a solution consists in a generalisation of the approach proposed in the work of Binato et al. (2001) for the JSP; it generates good starting points for the metaheuristic. The neighbourhood structures (Section 4.1) for the assignment problem consist in changing the machine of any operation in order to explore a larger solution space; only the N^α structure used in the local search is taken from the work of González et al. (2015). Concerning the scheduling problem, the neighbourhood structures are relying on the works of Van Laarhoven et al. (1992) and Nowicki and Smutnicki (1996), but extend them into more general neighbourhood structures that allow several permutations rather than only one. An estimation procedure based on the work of Dell'Amico and Trubian (1993) is used, which accelerates the local search procedure. Furthermore, it has been shown in Section 4.5 that the repetition vector presents some drawbacks that must be handled during the algorithm. Last but not least, the approach aims at proposing an effective single-solution based metaheuristic while most of the algorithms proposed until now are population based (Chaundry and Khan, 2015).

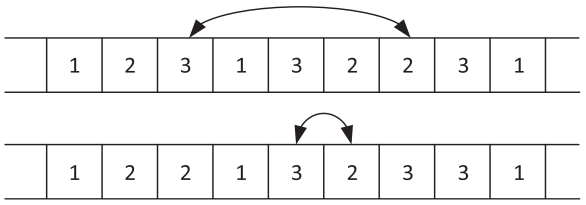

Fig. 11. Permutation of operations in *Operation Selection* vector. If the first 3 and third 2 are exchanged in the vector at the top of the figure, it has no impact in the process order because the first 3 still appears before the third 2 in the vector (bottom).

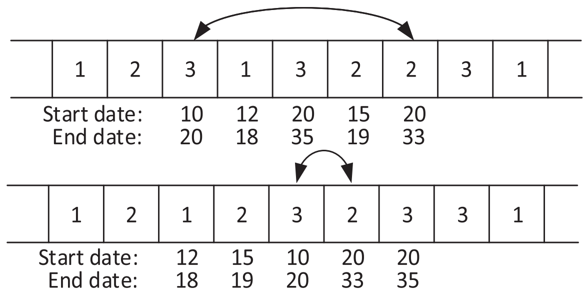

Fig. 12. Exchanging operations in the *Operation Selection* vector. The procedure ranges job occurrences in the vector in order to apply a permutation of operations by avoiding cases where the simple swap does not work.

5. Computational experiments

In this section, the following notations are used in the different tables:

DOE	Design of Experiment
INS.	Instance tested
LB	Lower Bound
BFS	Best solution (makespan) found by the algorithms
AVG	Average makespan found by the algorithms
CPU	Average computation time in seconds for each instance
CPU_{AVG}	Average computation time on an entire dataset
RPD_{BFS}	Average relative deviation between best found solution and the lower bound, in percentage
RPD_{AVG}	Average relative deviation between the average makespan and the lower bound, in percentage
Magnitude	number of optimal solutions found
CI-CPU	Normalised computation time

Instances used in this study are the well-known BCdata, DPdata, BRdata and HUdata instances, respectively proposed by Barnes and Chambers (1996), Dauzère-Pérès and Paulli (1997), Brandimarte (1993) and Hurink et al. (1994). The first data set (BRdata) is a set of 10 problems where the number of jobs varies from 10 to 20, and the number of unrelated machines, from 4 to 15. The average number of machines per operation (flexibility) ranges between 1.43 and 4.10.

The second data set (BCdata) is composed of 21 problems constructed from three of the classical job shop scheduling problems (Fisher and Thompson, 1963; Lawrence, 1984) (mt10, la24, la40) by replicating machines. The processing times are equal between operations since machines are supposed to be identical. The number of jobs varies from 10 to 15, and the number of machines varies from 11 to 18. The flexibility per operation varies between 1.07 and 1.30.

The third test sample (DPdata) is a set of 18 tough problems. The flexibility per operation varies between 1.13 and 5.02 and the number of operations varies from almost 200–400.

The fourth test sample (HUdata) is a set of 129 problems divided into three sets of test problems: edata, rdata and vdata. The problems were obtained from three problems proposed by Fisher and Thompson (1963) and 40 problems proposed by Lawrence (1984) by allowing other machines to process the operations. For each operation, processing times on the different machines are equal. The first set contains the problems with the least amount of flexibility (1.15), whereas the average flexibility is equal to 2 in Rdata and varies from 2.5 and 7.5 in Vdata Machines are identical.

Before applying the metaheuristic to all the instances proposed in these datasets, a Design of Experiment (DOE) has been conducted for parameters tuning. For this DOE, several configurations of the GRASP-mELS have been tested with *nb_g* ranging from 20 to 60 per steps of 20; *nb_ol* ranging from 25 to 50 per steps of 5; *nb_els* ranging from 100 to 200 per steps of 2; and *nb_n* ranging from 10 to 40 per steps of 1. The DOE resulted in taking *nb_g*=50, *nb_ol*=50, *nb_els*=140 and *nb_n*=14. The parameter α used for generating a solution is fixed at 0.33 in order to have good starting solutions. For each instance, 20 replications are made. The maximum computation time is fixed to 90 s for BRdata and BCdata instances, and 300 s for DPdata and HUdata instances in order to have computational conditions close to the ones proposed in the literature.

Table 2
Comparisons of GRASP×ELS, GRASP-mELS, and other metaheuristics' average performances.

		Mean performances of other methods[a]	**GRASP×ELS**	**GRASP-mELS**
BRdata	Magnitude	8	8	8
	CPU_{AVG}	17	13	15
	RPD_{BFS}	0.54	1.04	0.6
	RPD_{AVG}	0.85	1.4	0.83
BCdata	Magnitude	13	16	21
	CPU_{AVG}	14	30	18
	RPD_{BFS}	0.08	0.06	0
	RPD_{AVG}	0.19	0.24	0.07
DPdata	Magnitude	2	0	3
	CPU_{AVG}	144	190	138
	RPD_{BFS}	1.43	2.1	1.49
	RPD_{AVG}	1.66	2.53	1.73

[a] Results of (Yuan and Xu, 2013a, 2013b) are not considered as they drastically increase the average computation time.

In the following tables, **INS**. refers to the instance tested, **LB** to its Lower Bound; these values refer to the ones provided on the Quintiq webpage.[1] An asterisk [a] is used for optimal solutions. **BFS** and **AVG** are referring to the Best Found Solution and to the average makespan over 20 replications. **CPU** is the average computation time in seconds for each instance, while **CPU_{AVG}** is the average computation time on all the dataset. **RPD_{BFS}** is the average relative deviation between best found solutions and the lower bound LB in percentage; **RPD_{AVG}** is the average relative deviation between the average makespan to the lower bound in percentage. **Magnitude** is the number of optimal solutions found. **CI-CPU** is the average normalised computation time. It is obtained by applying the speed factor, calculated in Table 3, to the **AVG_{CPU}** in each dataset.

In this section, results are compared to valuable recent works, which are the following ones:

- Climbing Depth-bound Discrepancy Search (CDDS) of Hmida et al. (2010);
- Hybrid Differential Algorithm (HDE-N2) of Yuan and Xu (2013b) (HDE-N2 stands for "second neighbourhood" which is the one presenting the best results in their work);
- Memetic Algorithm (MA) of González et al. (2013)
- Scatter Search with Path Relinking (SSPR) of González et al. (2015);
- Genetic Tabu Search (HGTS) of Palacios et al. (2015);
- Hybrid Algorithm (HA) of Li and Gao (2016).

First, an overview of the performances of the proposed GRASP-mELS, compared to the GRASP×ELS and the above-mentioned metaheuristics, is presented below.

5.1. Overview of GRASP-mELS performance

To demonstrate the efficiency of the proposed GRASP-mELS, the behaviour of this metaheuristic is compared to the mean performance of GRASP×ELS and other metaheuristics.

[1] http://www.quintiq.com/optimisation/flexible-job-shop-scheduling-problem-results.html

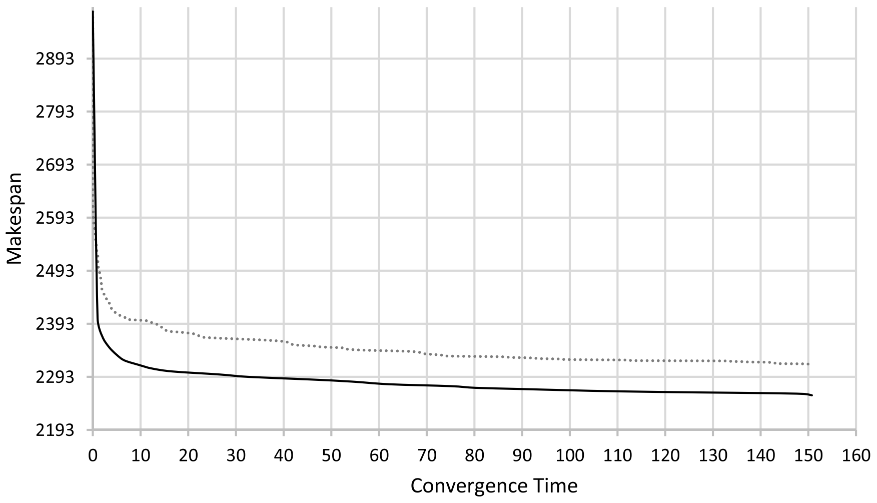

Fig. 13. Convergence of GRASP-mELS (black curve) and GRASP×ELS (grey dotted curve) on 16a instance. The GRASP-mELS reaches better solutions faster than the classical GRASP×ELS.

As shown in Table 2, the GRASP×ELS is comparable to the GRASP-mELS and other metaheuristics concerning the number of best found solutions (magnitude) for the BRdata instances. As the GRASP×ELS uses less time in intensification phase than the GRASP-mELS, it has a consequence in computation times for easy instances (as the ones of BRdata). However, the relative percentage deviation (RPD values) to the best known lower bounds to date is not as good. Concerning BCdata, the GRASP×ELS is slightly better in average than the other metaheuristics concerning the number of best found solutions. It is also better concerning the average deviation to the lower bound, but it is 40% slower than the GRASP-mELS and it finds less optimal solutions. Also, compared to the classical GRASP×ELS approach, the difference in RPD_{BFS} and RPD_{AVG} is in favour of the GRASP-mELS in all instance sets. Finally, the GRASP×ELS is ineffective in finding good solutions for the DPdata instances, with no optimal solution found whereas the GRASP-mELS found three. This validates the approach of the GRASP-mELS which has a better behaviour than the GRASP×ELS in a less computational effort. In Fig. 13, a comparison of convergence between GRASP×ELS and GRASP-mELS is given on an instance from the DP dataset (16a). A detailed comparison of the GRASP-mELS with recent state of the art metaheuristics is given in the next sub-section.

5.2. Comparison with state of the art methods

Algorithms have been implemented in C++ and have been executed on a computer with an i7-4800MQ processor, running Windows 7. In order to have a fair comparison between methods, computation times are normalised according to the processor used. The speed factor is computed using the data from Berkeley university[2] which applied the Whetstone Benchmark on numerous computers. A comparison with the study of Dongarra (2013) shows that the GFLOPS obtained with the Whetstone Benchmark is comparable (Intel Core 2 Q6600 given at 2.4GFLOPS in (Dongarra, 2013), 2.6GFLOPS with Whetstone) however, there are more information available on computers on the Berkeley websites which justifies its use. Furthermore, as each GFLOPS reported in Table 3 is obtained with the same benchmark, computation times can be easily compared, and the speed factor should be comparable with other benchmarks. This speed factor is obtained by dividing GFLOPS of the computers used for comparison by the GFLOPS of the computer used for testing the GRASP-mELS.

When the denomination of the processor used in a paper is not precise enough, the lowest GFLOPS is selected in order not to

Table 3
Performance and speed factor of computers used in other works.

	This paper.	Hmida et al. (2010)	González et al. (2013)	Yuan and Xu (2013b)	Li and Gao (2016)	Palacios et al. (2015)
Processor	i7 4800MQ 2.7 GHz	Core 2 duo 2.9 GHz	E6750 2.66 GHz	Xeon 2.83Ghz	Core 2 duo 2.0 GHz	Xeon E5520
GFLOPS	3.51	2.73	2.41	2.20	1.62	2.02
Speed Factor	1.00	0.78	0.69	0.63	0.46	0.58

disadvantage previous works. However it should be noted, as stated by Yuan and Xu (2013b) that information on computation times is purely indicative as it depends of various factors such as programming skills, coding language, or even the operating system.

Results presented in Table 4 show that the method works well on the BRdata instances, providing results comparable to the ones of the other metaheuristics. Furthermore, computation times in some instances such as Mk01, Mk02, and Mk08 are really effective as the best solutions are almost always found at the beginning of the algorithm. Compared to all authors, only the method proposed by Li and Gao (2016) outperforms the GRASP-mELS in both CI-CPU and Magnitude values.

The method is far more efficient on the **BC** and **DP** instances in term of behaviour regarding other metaheuristics as shown in Table 5 and Table 6. As shown in Table 5, the metaheuristic is able to provide state of the art solutions for the **BC** instances. Actually, it reached all optimal solutions at least once in a rather short computational time. Hence it was able to return optimal solutions for *setb4xyz*, *seti5c12* and *seti5xx*. Regarding normalised computation time, the metaheuristic is comparable to recent works even though being not as fast as the work of Hmida et al. (2010) or the recent work of Li and Gao (2016). However, the GRASP-mELS reached the optimal solutions 21 times over 21 instances when papers mentioned above reached respectively 7 and 15 optimal solutions.

It can be seen in Table 6, that the metaheuristic is able to provide good solutions for the instances of **DP** set. The GRASP-mELS provided better computation times than the works of Palacios et al. (2015), Hmida et al. (2010), Yuan and Xu (2013b) or González et al. (2013) while reaching more or the same number of optimal solutions than these works. Even though the metaheuristic of Li and Gao (2016) has better computation times, it has one less optimal solution than the GRASP-mELS. The metaheuristic also has better average RPD for BFS

[2] https://setiathome.berkeley.edu/cpu_list.php

Table 4
Results on instances proposed by Brandimarte (1993).

INS.	LB	Hmida et al. (2010) CDDS			Yuan and Xu (2013b) HDE-N2			González et al. (2013) GA+TS			González et al. (2015) SSPR			Palacios et al. (2015) HGTS			Li and Gao (2016) HA			This paper GRASP-mELS		
		BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU
Mk01	40[a]	40[a]	40	–	40[a]	40	4	40[a]	40	5	40[a]	40	11	40[a]	40	5	40[a]	–	0	40[a]	40	0
Mk02	26[a]	26[a]	26	–	26[a]	26	6	26[a]	26	15	26[a]	26	15	26[a]	26	15	26[a]	–	1	26[a]	26	10
Mk03	204[a]	204[a]	204	–	204[a]	204	31	204[a]	204	2	204[a]	204	24	204[a]	204	2	204[a]	–	0	204[a]	204	0
Mk04	60[a]	60[a]	60	–	60[a]	60	13	60[a]	60	10	60[a]	60	19	60[a]	60	10	60[a]	–	0	60[a]	60	0
Mk05	172[a]	173	174	–	172[a]	173	38	172[a]	172	18	172[a]	172	57	172[a]	172	18	172[a]	–	5	172[a]	173	15
Mk06	57[a]	58	59	–	57[a]	59	98	58	58	63	57[a]	58	40	57[a]	58	63	57[a]	–	54	58	58	36
Mk07	139[a]	139[a]	139	–	139[a]	139	26	139[a]	139	33	139[a]	141	84	139[a]	139	33	139[a]	–	20	139[a]	140	32
Mk08	523[a]	523[a]	523	–	523[a]	523	189	523[a]	523	3	523[a]	523	83	523[a]	523	3	523[a]	–	0	523[a]	523	0
Mk09	307[a]	307[a]	307	–	307[a]	307	123	307[a]	307	24	307[a]	307	52	307[a]	307	24	307[a]	–	1	307[a]	307	0
Mk10	189	197	198	–	198	202	266	199	200	104	196	197	94	198	199	104	197	–	33	197	199	59
RPD_{BFS} – RPD_{AVG}		0.66	0.94		0.48	1.1		0.7	0.76		0.37	0.74		0.48	0.71		0.42	–		0.60	0.83	
Magnitude		7			9			8			9			9			9			8		
CPU_{AVG}		15			79			28			48			28			11			15		
CI-CPU		12			50			19			33			16			5			15		

[a] denotes optimal solutions

and AVG values than the works of Hmida et al. (2010), Yuan and Xu (2013b) or González et al. (2013). Only the application of SSPR (González et al., 2015) shows a better behaviour in average on this dataset for the FJSP than the GRASP-mELS.

To conclude this experiment, three data sets of Hurink et al. (1994) have also been tested, which are vdata, edata, and rdata. As each of these data sets are composed of 43 instances, it is common to have aggregated results in the literature. Thus it is difficult to have a fair comparison with valuable works such as the one of Hmida et al. (2010) and the one of Yuan and Xu (2013b). However, since González et al. (2015) provided detailed results on their websites, and because their work is to our knowledge the best on this dataset, the solutions found by the GRASP-mELS on these data sets are compared to theirs. As can be seen, the metaheuristic performs well on this dataset with good results in a shorter computation time than the SSPR in average, even if it provided less best solutions on the vdata set. Comparison is shown in Table 7, where it can be seen that the approach is also relevant on this set of instances.

To sum up, 178 instances of the FJSP have been considered for which the best known solution has been reached in 125 of them.

For all these data sets, detailed results and structure of optimal solutions obtained could be found online.[3] However, ways to improve computational time, quality of solutions, and consistency of the metaheuristic, specifically concerning instances with high flexibility, could be explored in future studies. Indeed, the metaheuristic generally proved to be the most competitive on instances where the average number of machine per operation is not really high. Nevertheless, results obtained in this first study are promising.

6. Conclusions

In this study, a metaheuristic is proposed to tackle the Flexible Job-shop Problem. This metaheuristic is relying on a GRASP scheme, but is improved to provide better solutions resulting in the GRASP-mELS metaheuristic. The purpose of the metaheuristic is to further explore the neighbourhood of solutions by executing multi-level ELS on mutations of a solution. Two neighbourhoods are proposed for the exploration process and are embedded in a more global neighbourhood structure which selects one or the other randomly. Two other neighbourhood structures are used during the local search algorithm. An estimation procedure is proposed to accelerate the local search phase in order to avoid the evaluation of all the solutions explored during this procedure. The proposed single solution based metaheuristic provides valuable results both in term of computation times and quality of the solutions, which shows the relevance of such an approach among all the other population-based metaheuristics.

Several studies could be conducted to further improve the GRASP-mELS such as the proposition of other neighbourhood than the suggested ones in order to provide more diversity in the diversification phases. The track of a random local search is also being explored and is supposed to be less sensitive to the structures of instances by avoiding to rapidly fall into local minima. Actually, if it is not problematic for the BC dataset, it hinders the metaheuristic from reaching better solutions on some DP instances (12a, 18a). As the purpose of the mELS is to execute several ELS on neighbours of the solutions, a distance metric could be added to define the distance between two starting points in the mELS. Another idea would be to generate more initial solutions, reducing the number of outer loop and ELS but the first studies conducted so far did not bring relevant results yet.

In future studies, the FJSP could also be extended to various recent issues observed in manufacturing especially concerning energy efficiency of production systems.

[3] damienlamy.com/works/FJSSP

Table 5
Results on instances proposed by Barnes and Chambers (1996).

INS.	LB	Hmida et al. (2010) CDDS			Yuan and Xu (2013b) HDE-N2			González et al. (2013) GA+TS			González et al. (2015) SSPR			Palacios et al. (2015) HGTS			Li and Gao (2016) HA			This paper GRASP-mELS		
		BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU
mt10c1	927[a]	928	929	–	927[a]	928	179	927[a]	927	14	927[a]	928	26	927[a]	927	13	927[a]	–	12	927[a]	927	8
mt10cc	908[a]	910	911	–	908[a]	911	180	908[a]	909	14	908[a]	908	20	908[a]	910	13	908[a]	–	10	908[a]	909	17
mt10x	918[a]	918[a]	918	–	918[a]	919	179	918[a]	922	18	918[a]	918	23	918[a]	918	15	918[a]	–	11	918[a]	918	2
mt10xx	918[a]	918[a]	918	–	918[a]	918	170	918[a]	918	16	918[a]	918	19	918[a]	918	12	918[a]	–	11	918[a]	918	2
mt10xxx	918[a]	918[a]	918	–	918[a]	918	160	918[a]	918	19	918[a]	918	20	918[a]	918	12	918[a]	–	11	918[a]	918	2
mt10xy	905[a]	906	906	–	905[a]	906	174	905[a]	905	16	905[a]	906	21	905[a]	905	13	905[a]	–	11	905[a]	905	26
mt10xyz	847[a]	849	851	–	847[a]	851	166	849	850	21	847[a]	847	20	847[a]	850	18	847[a]	–	9	847[a]	847	26
setb4c9	914[a]	919	919	–	914[a]	917	338	914[a]	914	22	914[a]	916	28	914[a]	914	16	914[a]	–	15	914[a]	914	11
setb4cc	907[a]	909	911	–	907[a]	910	336	907[a]	907	22	907[a]	907	21	907[a]	908	15	907[a]	–	15	907[a]	907	29
setb4x	925[a]	925[a]	925	–	925[a]	926	354	925[a]	925	18	925[a]	925	19	925[a]	925	15	925[a]	–	13	925[a]	925	4
setb4xx	925[a]	925[a]	925	–	925[a]	926	330	925[a]	925	19	925[a]	925	21	925[a]	925	14	925[a]	–	5	925[a]	925	2
setb4xxx	925[a]	925[a]	925	–	925[a]	926	315	925[a]	925	20	925[a]	925	22	925[a]	925	15	925[a]	–	9	925[a]	925	3
setb4xy	910[a]	916	916	–	910[a]	914	313	910[a]	910	25	910[a]	912	32	910[a]	910	19	910[a]	–	12	910[a]	910	18
setb4xyz	902[a]	905	907	–	903	905	317	905	905	19	905	905	21	905	905	15	905	–	14	902[a]	904	11
seti5c12	1169[a]	1174	1175	–	1171	1175	1113	1171	1173	41	1170	1173	25	1170	1171	41	1170	–	31	1169[a]	1172	39
seti5cc	1135[a]	1136	1137	–	1136	1138	1079	1136	1137	40	1135[a]	1136	29	1136	1137	34	1136	–	17	1135[a]	1136	24
seti5x	1198[a]	1201	1202	–	1200	1206	1087	1199	1200	43	1198[a]	1199	41	1199	1201	38	1198[a]	–	27	1198[a]	1999	36
seti5xx	1194[a]	1199	1199	–	1197	1203	1251	1197	1198	38	1197	1199	37	1197	1198	34	1197	–	29	1194[a]	1197	26
seti5xxx	1194[a]	1197	1198	–	1197	1202	1244	1197	1197	40	1194[a]	1198	38	1197	1198	31	1197	–	19	1194[a]	1197	27
seti5xy	1135[a]	1136	1138	–	1136	1138	1141	1136	1137	39	1135[a]	1136	29	1136	1137	34	1136	–	17	1135[a]	1136	28
seti5xyz	1125[a]	1125[a]	1125	–	1125[a]	1130	1223	1127	1128	41	1125[a]	1126	35	1125[a]	1126	43	1125[a]	–	33	1125[a]	1127	42
$RPD_{BFS} - RPD_{AVG}$		0.19	0.26		0.05	0.3		0.08	0.14		0.03	0.12		0.07	0.13		0.05	–		0	0.07	
Magnitude		7			13			12			18			14			15			21		
CPU_{AVG}		15			555			26			26			22			16			18		
CI-CPU		12			349			18			18			13			7			18		

[a] Denotes optimal solutions.

Table 6
Results on instances proposed by Dauzère-Pérès and Paulli (1997).

INS.	LB	Hmida et al. (2010) CDDS			Yuan and Xu (2013b) HDE-N2			González et al. (2013) GA+TS			González et al. (2015) SSPR			Palacios et al. (2015) HGTS			Li and Gao (2016) HA			This paper GRASP-mELS		
		BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU	BFS	AVG	CPU
01a	2505[a]	2518	2525	–	2505[a]	2513	838	2505[a]	2511	74	2505[a]	2508	68	2505[a]	2505	122	2505[a]	–	108	2505[a]	2505	62
02a	2228[a]	2231	2235	–	2230	2231	973	2232	2234	120	2229	2230	100	2230	2234	205	2230	–	133	2229	2231	86
03a	2228[a]	2229	2232	–	2228[a]	2229	1165	2229	2230	143	2228[a]	2228	110	2228[a]	2230	181	2229	–	97	2228[a]	2230	94
04a	2503[a]	2503[a]	2510	–	2506	2506	850	2503[a]	2504	72	2503[a]	2504	57	2503[a]	2503	112	2503[a]	–	87	2503[a]	2503	31
05a	2192	2216	2218	–	2212	2215	931	2219	2221	123	2211	2215	112	2214	2218	208	2212	–	116	2212	2215	126
06a	2163	2196	2203	–	2187	2192	1167	2200	2204	157	2183	2192	181	2193	2198	260	2197	–	93	2195	2200	181
07a	2216	2283	2296	–	2288	2303	1547	2266	2286	201	2274	2285	139	2270	2280	344	2279	–	204	2276	2284	127
08a	2061[a]	2069	2069	–	2067	2074	1906	2072	2075	197	2064	2066	181	2070	2074	318	2067	–	184	2069	2072	144
09a	2061[a]	2066	2067	–	2069	2073	943	2066	2067	291	2062	2063	213	2067	2069	376	2065	–	201	2069	2071	170
10a	2212	2291	2303	–	2297	2302	1590	2267	2273	240	2269	2287	120	2247	2266	369	2287	–	238	2263	2278	110
11a	2018	2063	2072	–	2061	2067	1826	2068	2071	222	2051	2058	193	2064	2069	294	2060	–	181	2065	2068	170
12a	1969	2031	2034	–	2027	2036	914	2037	2041	266	2018	2020	280	2027	2033	486	2027	–	151	2039	2045	148
13a	2197	2257	2260	–	2263	2269	2900	2271	2276	241	2248	2257	119	2250	2264	416	2248	–	293	2252	2263	158
14a	2161[a]	2167	2179	–	2164	2168	3238	2169	2171	340	2163	2164	269	2170	2173	396	2167	–	210	2170	2174	191
15a	2161[a]	2165	2170	–	2163	2166	2112	2166	2166	470	2162	2163	376	2168	2169	523	2163	–	192	2172	2174	173
16a	2193	2256	2258	–	2259	2266	2802	2266	2271	253	2244	2253	131	2246	2257	384	2249	–	160	2243	2258	151
17a	2088	2140	2146	–	2137	2141	3096	2147	2150	333	2130	2134	299	2142	2146	483	2140	–	203	2145	2152	190
18a	2057	2127	2132	–	2124	2128	2489	2138	2141	488	2119	2123	409	2129	2133	650	2132	–	133	2146	2151	164
RPD_{BFS} – RPD_{AVG}		1.55	1.8		1.5	1.73		1.59	1.77		1.18	1.4		1.34	1.59		1.43	–		1.49	1.73	
Magnitude		1			2			2			3			3			2			3		
CPU_{AVG}		200			1738			235			187			340			166			138		
CI-CPU		156			1095			162			129			197			76			138		

[a] Denotes optimal solutions.

Table 7
Results on instances proposed by Hurink et al. (1994).

	edata				rdata				vdata			
	GRASP-mELS		SSPR		GRASP-mELS		SSPR		GRASP-mELS		SSPR	
	RPD_{BFS}	RPD_{AVG}	RPD_{BFS}	RPD_{AVG}	RPD_{BFS}	RPD_{AVG}	RPD_{BFS}	RPD_{AVG}	RPD_{BFS}	RPD_{AVG}	RPD_{BFS}	RPD_{AVG}
mt06/10/20	0	0	0	0.04	0	0	0	0	0	0	0	0
la01-la05	0	0	0	0	0	0.07	0.07	0.09	0	0	0	0
la06-la10	0	0	0	0	0	0	0	0.01	0	0	0	0
la11-la15	0	0	0	0	0	0	0	0	0	0	0	0
la16-la20	0	0	0	0	0	0	0	0.03	0	0	0	0
la21-la25	0	0.24	0.08	0.23	2.63	3.27	2.53	2.91	0.49	0.8	0.23	0.35
la26-la30	0.33	0.61	0.43	0.66	0.36	0.71	0.36	0.48	0.17	0.24	0.06	0.08
la31-la35	0.05	0.11	0.01	0.07	0.05	0.12	0.04	0.05	0.04	0.07	0.01	0.02
la36-la40	0	0.04	0	0.05	0.36	1.22	0.66	0.90	0	0	0	0
Magnitude	38		36		25		24		30		34	
CI-CPU	21		26		47		32		28		44	

Acknowledgments

This work was partially supported by the Public Investment Bank (BPI – Banque Publique d'investissement) under project ECOTHER. The authors would like to thank all anonymous referees for a number of careful remarks and insightful comments which helped improving the presentation in this paper.

References

Barnes, J., Chambers, J., 1996. Flexible Job Shop Scheduling by Tabu Search. Technical Report Series: ORP96-09. Graduate Program in Operations Research and Industrial Engineering. The University of Texas at Austin.

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. Oper.-Res.-Spektrum 17, 87–92.

Binato, S., Hery, W.J., Loewenstern, D.M., Resende, M.G.C., 2001. A greedy randomized adaptive search procedure for job shop scheduling. IEEE Trans. Power Syst. 16, 247–253.

Boussaïd, I., Lepagnot, J., Siarry, P., 2013. A survey on optimization metaheuristics. Inf. Sci. 237, 82–117.

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by tabu search. Ann. Oper. Res. 41, 157–183.

Brucker, P., Schlie, R., 1990. Job-shop scheduling with multi-purpose machines. Computing 45 (4), 369–375.

Chassaing, M., Fontanel, J., Lacomme, P., Ren, L., Tchernev, N., Villechenon, P., 2014. A GRASP ELS approach for the job-shop with a web service paradigm packaging. Expert Syst. Appl. 41, 544–562.

Chaundry, I.A., Khan, A.A., 2015. A research survey: review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. 00, 1–41.

Chen, H., Ihlow, J., Lehmann, C., 1999. A Genetic Algorithm for Flexible Job-Shop Scheduling. IEEE International Conference on Robotics & Automation, Detroit, Michigan.

Cheng, R., Gen, M., Tsujimura, Y., 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms – I representation. Comput. Ind. Eng. 30, 983–997.

Dauzère-Pérès, S., Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann. Oper. Res. 70 (3), 281–306.

Dell'Amico, M., Trubian, M., 1993. Applying tabu search to the job-shop scheduling problem. Ann. Oper. Res. 41, 231–252.

Dongarra, J., 2013. Performance of Various Computers Using Standard Linear Equations Software (Tech. rep.). Computer Science Department. University of Tennessee, Knoxville, Tennessee.

Feo, T.A., Resende, M.G.C., Smith, S.H., 1994. A, greedy randomized adaptive search procedure for maximum independent set. Oper. Res. 42, 860–878.

Fisher, H., Thompson, G.L., 1963. Probabilistic learning combinations of local job shop scheduling rules. In: Muth, J.F., Thompson, G.L. (Eds.), Industrial Scheduling. Prentice-Hall, Englewood Cliffs, NJ, 225–251.

Gao, J., Sun, L., Gen, M., 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Oper. Res. 35, 2892–2907.

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flow shop and job shop scheduling. Math. Oper. Res. 1, 117–129.

González, M.A., Vela, C.R., Varela, R., 2013. An efficient memetic algorithm for the flexible job shop with setup times. In Proceedings of ICAPS, 91–99.

González, M.A., Vela, C.R., Varela, R., 2015. Scatter search with path relinking for the flexible job shop scheduling problem. Eur. J. Oper. Res. 245, 35–45.

Grabowski, J., Nowicki, E., Zdrzalka, S., 1986. A block approach for single machine scheduling with release dates and due dates. Eur. J. Oper. Res. 26, 278–285.

Hmida, A., Haouri, M., Huguet, M.J., Lopez, P., 2010. Discrepancy search for the flexible job shop scheduling problem. Comput. Oper. Res. 37, 2192–2201.

Hurink, J., Jurisch, B., Thole, M., 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. OR Spectr. 15 (4), 205–215.

Ishikawa, S., Kubota, R., Horio, K., 2015. Effective hierarchical optimization by a hierarchical multi-space competitive genetic algorithm for the flexible job-shop scheduling problem. Expert Syst. Appl. 42, 9434–9440.

Kamble, S.V., Mane, S.U., Umbarkar, A.J., 2015. Hybrid multi-objective particle swarm optimization for flexible job shop scheduling problems. I. J. Intell. Syst. Appl. 4, 54–61.

Karimi, H., Rahmati, S.H.A., Zandieh, M., 2012. An efficient knowledge-based algorithm for the flexible job shop scheduling problem. Knowl.-Based Syst. 36, 236–244.

Kemmoe-Tchomte, S., Lamy, D., Tchernev, N., 2016. A GRASP embedding a bi-level ELS for solving Flexible Job-shop Problems. To appear in 8th IFAC Conference on Manufacturing Modelling, Management and Control (MIM).

Ku, W.Y., Beck, J.C., 2016. Mixed integer programming models for job shop scheduling: a computational analysis. Comput. Oper. Res. 73, 165–173.

Lawrence, S., 1984. Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques. GSIA, Carnegie Mellon University, Pittsburgh, PA.

Li, X., Gao, L., 2016. An effective hybrid genetic algorithm and tabu search for the flexible job shop scheduling problem. Int. J. Prod. Econ. 174, 93–110.

Mastrolilli, M., Gambardella, L., 2000. Effective Neighborhood functions for the flexible job shop problem. J. Sched. 3 (1), 3–20.

Nowicki, E., Smutnicki, C., 1996. A fast taboo search algorithm for the job shop scheduling problem. Manag. Sci. 42, 797–813.

Palacios, J.J., González, M.A., Vela, C.R., González-Rodríguez, I., Puente, J., 2015. Genetic tabu search for the fuzzy flexible job shop problem. Comput. Oper. Res. 54, 74–89.

Peng, B., Lü, Z., Cheng, T.C.E., 2015. A tabu search/path relinking algorithm to solve the job shop scheduling problem. Comput. Oper. Res. 53, 154–164.

Pezzella, F., Morganti, G., Ciaschetti, G., 2008. A genetic algorithm for the flexible job-shop scheduling problem. Comput. Oper. Res. 35, (3302-3212).

Prins, C., 2009. A GRASP x evolutionary local search hybrid for the vehicle routing problem. In: Pereira, F.B., Tavares, J. (Eds.), Bio-Inspired Algorithms for the Vehicle Routing Problem 161. Studies in computational intelligence, 35–53.

Rajkumar, M., Asokan, P., Vamsikrishna, V., 2010. A GRASP algorithm for flexible job-shop scheduling with maintenance constraints. Int. J. Prod. Res. 48, 6821–6836.

Rajkumar, M., Asokan, P., Anilkumar, N., Page, T., 2011. A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. Int. J. Prod. Res. 49, 2409–2423.

Roy, B., Sussmann, B., 1964. Les problèmes d'ordonnancement avec contraintes disjonctives, In: Note DS N°9 bis, SEMA, Paris.

Singh, M.R., Mahapatra, S.S., 2016. A quantum behaved particle swarm optimisation for the flexible job shop scheduling. Comput. Ind. Eng. 93, 36–44.

Sobeyko, O., Mönch, L., 2016. Heuristic approaches for scheduling jobs in large-scale flexible job shops. Comput. Oper. Res. 68, 97–109.

Teekeng, W., Thammano, A., 2012. Modified Genetic Algorithm for the Flexible Job-Shop Scheduling Problem. Procedia Comput. Sci. 12, 122–128.

Thammano, A., Phu-ang, A., 2013. A hybrid artificial bee colony algorithm with local search for the flexible job-shop scheduling problem. Procedia Comput. Sci. 20, 96–101.

Van Laarhoven, P., Aarts, E., Lenstra, K., 1992. Job shop scheduling by simulated annealing. Oper. Res. 40, 113–125.

Wang, X., Duan, H., 2014. A hybrid biogeography-based optimization algorithm for job shop scheduling problem. Comput. Ind. Eng. 73, 96–114.

Wolf, S., Merz, P., 2007. Evolutionary Local Search for the Super-Peer Selection Problem and the P-hub Median Problem. Lecture Notes in Computer Science 4771. Springer, Berlin, 1–15.

Yuan, Y., Xu, H., 2013a. An integrated search heuristic for large-scale flexible job shop scheduling problems. Comput. Oper. Res. 40, 2864–2877.

Yuan, Y., Xu, H., 2013b. Flexible job shop scheduling using hybrid differential evolution algorithms. Comput. Ind. Eng. 65, 246–260.

Zhang, G., Gao, L., Shi, Y., 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst. Appl. 38, 3563–3573.