European Journal of Operational Research 245 (2015) 35–45

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Scatter search with path relinking for the flexible job shop scheduling problem

Miguel A. González, Camino R. Vela, Ramiro Varela*

Department of Computing, University of Oviedo. Campus of Gijón, 33204, Gijón, Spain

ARTICLE INFO

Article history:
Received 10 May 2014
Accepted 26 February 2015
Available online 3 March 2015

Keywords:
Scheduling
Flexible job shop
Scatter search
Path relinking
Neighborhood structures

ABSTRACT

The flexible job shop scheduling is a challenging problem due to its high complexity and the huge number of applications it has in real production environments. In this paper, we propose effective neighborhood structures for this problem, including feasibility and non improving conditions, as well as procedures for fast estimation of the neighbors quality. These neighborhoods are embedded into a scatter search algorithm which uses tabu search and path relinking in its core. To develop these metaheuristics we define a novel dissimilarity measure, which deals with flexibility. We conducted an experimental study to analyze the proposed algorithm and to compare it with the state of the art on standard benchmarks. In this study, our algorithm compared favorably to other methods and established new upper bounds for a number of instances.

1. Introduction

The job shop scheduling problem (JSP) is a simple model of many real production processes. It is one of the most classical and difficult scheduling problems and it has been studied for decades (Meeran & Morshed, 2014). However, in many environments the production model has to consider additional characteristics or complex constraints. In this work we consider the possibility of selecting alternative routes among the machines, which is useful in production environments where multiple machines are able to perform the same operation (possibly with different processing times), as it allows the system to absorb changes in the demand of work or in the performance of the machines. This problem is known as the flexible job shop scheduling problem (FJSP). It was first addressed by Brucker and Schlie (1990) and it has been object of intensive research since then.

Initially, researchers proposed hierarchical approaches, in which the machine assignment and the scheduling of operations were studied separately (Brandimarte, 1993). However, most of the works in the literature consider both subproblems at the same time. Mastrolilli and Gambardella (2000) developed two neighborhood structures to improve the TS algorithm proposed by Dauzère-Pérès and Paulli (1997). More recently, Ho et al. proposed a learnable genetic architecture (LEGA) (Ho, Tay, & Lai, 2007). It is also remarkable the hybrid genetic algorithm combined with a variable neighborhood descent search (hGA) developed by Gao, Sun, and Gen (2008). Other approaches as the climbing depth-bounded discrepancy search (CDDS) algorithm proposed by Hmida, Haouari, Huguet, and Lopez (2010), the hybrid harmony search and large neighborhood search (HHS/LNS) by Yuan and Xu (2013) or the hybrid genetic algorithm combined with tabu search (GA + TS) proposed by González, Vela, and Varela (2013) also obtain good results on standard instances. Bozejko, Uchronski, and Wodecki (2010) present a parallel approach with two double-level parallel metaheuristic algorithms based on neighborhood determination ($TSBM^2h$), with good results on one standard benchmark. Gutierrez and Garcia-Magario (2011) combine a modular genetic algorithm with repairing heuristics (MGARH), and obtain new upper bounds in one standard benchmark as well. Just to have a general picture of the state of the art in FJSP, we could said that TS, hGA, CDDS, HHS/LNS, $TSBM^2h$, MGARH and GA + TS show the best performance among the aforementioned methods. However, none of them dominates the others in the sense of obtaining the best solutions or taking the lowest time for all benchmarks. Also, the performance of some of these methods strongly varies with the benchmark and some of them have not been evaluated on all the common benchmarks. For example, MGARH is among the best for one particular benchmark while it has not been evaluated on others; and hGA is also the best method for one benchmark, while it is clearly not so good in others.

In spite of that metaheuristics have been widely applied to scheduling problems, a powerful method as scatter search with path relinking has been rarely used in flexible environments. Maybe this is due to the difficulty of defining a tight distance between schedules. As far as we known, only in Jia and Hu (2014), where the authors combine path relinking with tabu search and consider multiobjective optimization, a distance is defined for the FJSP.

* Corresponding author. Tel.: +34 985182508.
E-mail addresses: mig@uniovi.es (M. A. González), crvela@uniovi.es (C. R. Vela), ramiro@uniovi.es (R. Varela).

http://dx.doi.org/10.1016/j.ejor.2015.02.052
0377-2217/

In this paper we propose new neighborhood structures for the FJSP with makespan minimization. We define feasibility and non improving conditions as well as algorithms for fast estimation of neighbors' quality. We also define a dissimilarity measure between two solutions which takes into account the flexible nature of the problem. These new structures and the dissimilarity measure are incorporated into a hybrid metaheuristic which uses scatter search with path relinking and tabu search as improvement method. We conducted an experimental study to analyze our proposal and to compare it with the state of the art.

The remainder of the paper is organized as follows. In Section 2 we formulate the problem and describe the solution graph model. In Section 3 we define the proposed neighborhood structures. Section 4 details the new dissimilarity measure and the metaheuristics used. In Section 5 we report the results of the experimental study, and finally Section 6 summarizes the main conclusions of this paper.

2. Problem formulation

In the job shop scheduling problem (JSP), there are a set of jobs $J = \{J_1, \ldots, J_n\}$ that must be processed on a set $M = \{M_1, \ldots, M_m\}$ of physical resources or machines, subject to a set of constraints. There are *precedence constraints*, so each job J_i, $i = 1, \ldots, n$, consists of n_i operations $O_i = \{o_{i1}, \ldots, o_{in_i}\}$ to be sequentially scheduled. Also, there are *capacity constraints*, whereby each operation o_{ij} requires the uninterrupted and exclusive use of one of the machines for its whole processing time.

In the flexible JSP (FJSP), an operation o_{ij} is allowed to be executed in any machine of a given set $M(o_{ij}) \subseteq M$. The processing time of operation o_{ij} on machine $M_k \in M(o_{ij})$ is $p_{o_{ij}k} \in \mathbb{N}$. Notice that the processing time of an operation may be different in each machine and that a machine may process several operations of the same job. The goal is to build up a feasible schedule which consists in assigning both a machine and a starting time to each operation in the set $O = \cup_{1 \le i \le n} O_i$, in such a way that all constraints hold. The objective function is the makespan, which should be minimized. The FJSP is NP-hard as it is a generalization of the JSP which has proven to be NP-hard (Garey, Johnson, & Sethi, 1976).

A solution can be alternatively viewed as a pair (α, π) where α represents a feasible assignment of each operation $o_{ij} \in O$ to a machine $M_k \in M(o_{ij})$, denoted $\alpha(o_{ij}) = k$, and π is a processing order of the operations on all the machines in M compatible with the job sequences.

Let $PJ_{o_{ij}}$ and $SJ_{o_{ij}}$ denote the operations just before and after o_{ij} in the job sequence and $PM_{o_{ij}}$ and $SM_{o_{ij}}$ the operations right before and after o_{ij} in the machine sequence in a solution (α,π), if they exist. The starting and completion times of o_{ij}, denoted $St_{o_{ij}}$ and $C_{o_{ij}}$ respectively, can be calculated as $St_{o_{ij}} = \max(C_{PJ_{o_{ij}}}, C_{PM_{o_{ij}}})$ (if an operation is the first in its job or in its machine sequence, the corresponding $C_{PJ_{o_{ij}}}$ or $C_{PM_{o_{ij}}}$ is taken to be 0) and $C_{o_{ij}} = St_{o_{ij}} + p_{o_{ij}k}$ being $k = \alpha(o_{ij})$. The objective is to find a solution (α,π) that minimizes the makespan, denoted as $C_{max}(\alpha, \pi) = \max_{o_{ij} \in O} C_{o_{ij}}$.

2.1. Solution graph and criticality

We define the following solution graph model for the FJSP. In accordance with this model, a machine assignment α and a feasible operation processing order π can be represented by an acyclic directed graph $G(\alpha, \pi) = (V, A \cup R(\alpha, \pi))$, where each node v in V represents either an operation of the problem, labeled with the assigned machine M_k, or one of the dummy nodes *start* and *end*, which are fictitious operations with processing time 0.

The set A contains *conjunctive arcs* representing job processing orders and the set $R(\alpha, \pi)$ contains *disjunctive arcs* representing machine processing orders. The arc (v, w) is weighted with the processing time, p_{vk}, of the operation in v on the assigned machine M_k. If w is the first operation in the processing order of its job, $J(w)$, there is an arc $(start, w)$ in $G(\alpha, \pi)$ with weight 0 and if w is the last operation in the job processing order, there is an arc (w, end) with weight p_{wk}, $k = \alpha(w)$. Fig. 1 shows a solution graph for a problem with 3 jobs and 3 machines.

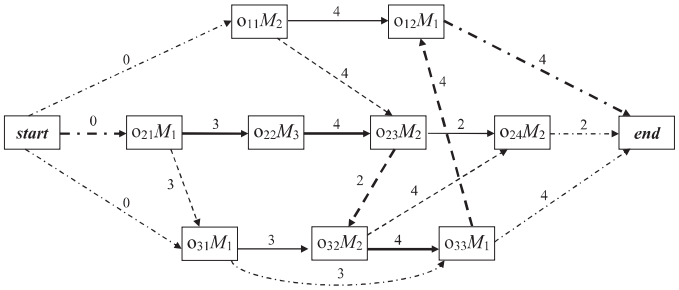

Fig. 1. A feasible schedule to a problem with 3 jobs and 3 machines represented by a solution graph. Bold-face arcs show a critical path whose length, i.e., the makespan, is 21.

The set $R(\alpha, \pi)$ is partitioned into subsets $R_k(\alpha, \pi)$, where $R_k(\alpha, \pi)$ is a minimal set of arcs defining a processing order for all operations requiring the machine M_k.

The makespan of the solution (α, π) is the cost of a critical path in $G(\alpha, \pi)$, i.e., a directed path from node *start* to node *end* having maximum cost. Bold-face arcs in Fig. 1 represent a critical path. Nodes and arcs in a critical path are also termed critical. We define a critical block as a maximal subsequence of consecutive operations in a critical path requiring the same machine. Notice that with this definition, a critical block may contain more than one operation of the same job. This makes a difference w.r.t. other definitions in the literature, as for example those given in Mastrolilli and Gambardella (2000) or in González et al. (2013), and has some influence in the critical block length and neighborhood size, as we will detail later.

The concept of critical block is important as most neighborhood structures proposed for job shop problems rely on exchanging the processing order of operations in critical blocks (Amico & Trubian, 1993; Mati, Dauzere-Peres, & Lahlou, 2011; Van Laarhoven, Aarts, & Lenstra, 1992). The structures proposed in Section 3.1 include moves of this type as well, but as we shall see, in the FJSP we have to introduce an additional type of move to deal with the machine assignment subproblem.

To formalize the description of the neighborhood structures, we introduce the concepts of head and tail of an operation v, denoted r_v and q_v respectively, which are calculated as follows:

$$r_{start} = q_{end} = 0$$
$$r_v = \max(r_{PJ_v} + p_{PJ_v k_1}, r_{PM_v} + p_{PM_v k})$$
$$k = \alpha(v),\ k_1 = \alpha(PJ_v)$$
$$r_{end} = \max_{v \in PJ_{end}, k = \alpha(v)} \{r_v + p_{vk}\}$$
$$q_v = \max(q_{SJ_v} + p_{SJ_v k_2}, q_{SM_v} + p_{SM_v k})$$
$$k = \alpha(v),\ k_2 = \alpha(SJ_v)$$
$$q_{start} = \max_{v \in SJ_{start}, k = \alpha(v)} \{q_v + p_{vk}\}$$

Abusing notation, SJ_{start} (PJ_{end}) denotes the set consisting of the first (last) operation processed in each of the n jobs. A node v is critical if and only if $C_{max} = r_v + p_{v\alpha(v)} + q_v$.

3. Neighborhood structures

In this section we propose several neighborhood structures for the FJSP, some of them focus on the sequencing subproblem and so they rely on changing the processing order of operations on a machine,

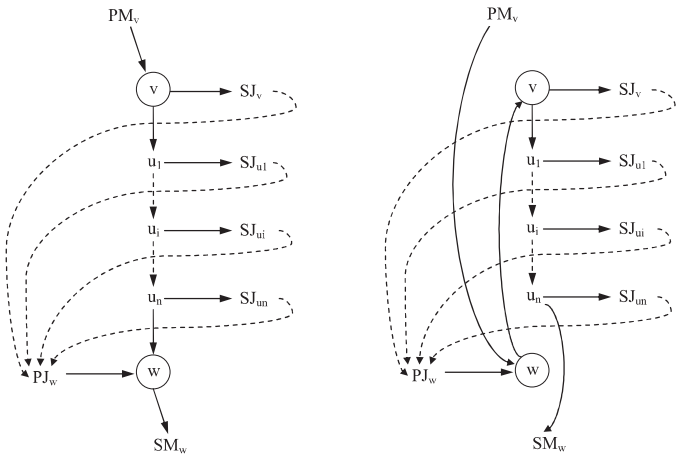

Fig. 2. Potential alternative paths between two operations v and w belonging to the same machine that would create a cycle in the neighbor built by inserting w before v. The initial schedule is shown in the left side while the neighbor is in the right side.

while others deal with the assignment subproblem, hence their moves consist in changing the machine assignment of an operation. For these structures we prove conditions of feasibility and non-improvement, and propose methods for fast estimation of the neighbors quality. We also analyze different strategies for combining these structures.

3.1. Structures for the sequencing subproblem

We propose two structures, termed N^π and N^π_r, which extend some of the structures defined by Amico and Trubian (1993) to the FJSP. In both cases a neighbor is created by moving a critical operation to another position in its critical block, keeping the machine assignment. They are based on the following results.

Proposition 1. *Let $S = (\alpha, \pi)$ and $S' = (\alpha, \pi')$ be two feasible schedules such that S' is obtained from S so that it keeps the processing order of all operations in a critical path in S. Then, $C_{max}(S') \geq C_{max}(S)$.*

Therefore, we will only consider changes in the processing orders of operations in a critical path to get potentially improving schedules. Let us now consider the portion of the graph in the left side of Fig. 2 where all the operations of the block ($v\ u_1\ \ldots\ u_n\ w$) have the same machine assigned and w belongs to different job than the other operations. Discontinuous arrows represent potential alternative paths. Moving w just before v (right side of Fig. 2) requires checking that none of these paths exist, otherwise there would be a cycle in the resulting graph. The following result gives a sufficient condition for no cycles.

Proposition 2. *Let $S = (\alpha, \pi)$ be a feasible schedule and let $(B'\ v\ B\ w\ B'')$ be a sequence of consecutive operations in the solution graph $G(\alpha, \pi)$ all of them assigned to the same machine, where B, B' and B'' are themselves sequences of operations such that any of them may be null. Let (α, π') be a schedule created from S by moving w just before v. Then, $G(\alpha, \pi')$ has no cycles if the following condition holds:*

$$r_{PJ_w} < r_{SJ_u} + p_{SJ_u k_1} + C \wedge J(u) \neq J(w), \quad \forall u \in (vB), \tag{1}$$

where $C = 0$ if $SM_z = PJ_w$, and $C = min\{p_{SJ_z k_2}, p_{SM_z k_1}\}$ otherwise, being $z = SJ_u$, $k_1 = \alpha(SJ_u)$, $k_2 = \alpha(SJ_z)$.

A similar result can be proven with regards to the insertion of operation v just after w. So, we can define the following neighborhood structure.

Definition 1. N^π **structure.** Let operation v be a member of a critical block B. In a neighboring solution v is moved to another position in B, provided that the sufficient condition of feasibility given in Proposition 2 holds.

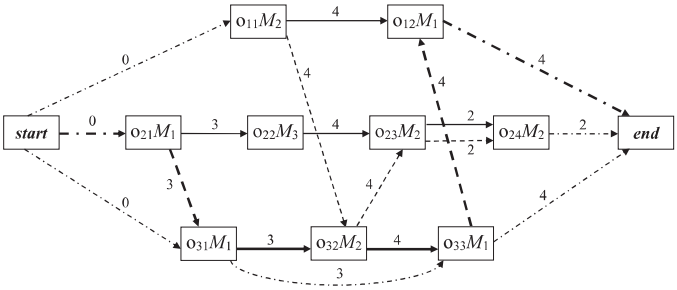

Fig. 3. A neighbor of the solution of Fig. 1 created with N^π_r. The makespan is reduced from 21 to 18.

In order to reduce the effective size of the neighborhood, we propose the following condition for non-improvement, which can be efficiently evaluated.

Proposition 3. *Let $S = (\alpha, \pi)$ be a solution and $(B'\ v\ B\ w\ B'')$ a critical block in $G(\alpha, \pi)$, where B, B' and B'' are sequences of operations of the form $B = (u_1, \ldots, u_n)$, $B' = (u'_1, \ldots, u'_{n'})$ and $B'' = (u''_1, \ldots, u''_{n''})$, with $n', n'' \geq 1$. Even if the schedule $S' = (\alpha, \pi')$ obtained from S by inserting w just before v is feasible, the makespan of S' cannot be lower than the makespan of S.*

An analogous result can be established for a neighbor created by inserting an operation v after an operation w. Therefore, a move in N^π may only produce an improvement in the makespan if an internal operation is moved either before the first or after the last operation of the critical block, or if an operation at the extreme of a critical block is moved to another position in the block. Hence, we can define the following reduced neighborhood structure.

Definition 2. N^π_r **structure.** Let v be an operation in a critical block B. If v is internal to B, then a neighboring solution is obtained by moving v before the first or after the last operation in B. If v is the first or the last operation in B, a neighboring solution is obtained by moving v to another position in B. In both cases, the sufficient condition of feasibility given in Proposition 2 must hold.

Fig. 3 shows an example where a neighbor of the schedule of Fig. 1 is created with N^π_r. In this case, o_{23} is inserted after o_{32} and the makespan improves.

Let us now introduce the new neighborhood, termed N^π_2, which extends that proposed in Van Laarhoven et al. (1992).

Definition 3. N^π_2 **structure.** Let v and w be consecutive operations of the same machine. In a neighboring solution the processing order of v and w is reversed, provided that the sufficient condition of feasibility given in Proposition 2 holds.

In accordance with Proposition 1, many of the neighbors generated by N^π_2 will be non improving ones and so this structure will not be useful for tabu search. However, it may be good for path relinking, as we will see in Section 4.4.

Notice that, for any of the defined structures, the order π' of the selected neighbor can be built by reconstructing only a portion of π. For example, if the neighbor is created with N^π by inserting v after w, or by inserting w before v, the sequences of operations before v and after w in π remain in π' (similar reconstruction is detailed in Mattfeld (1995) for the classical JSP).

3.2. Structures for the assignment subproblem

Changing the machine assignment of one operation may give rise to an improved schedule. This possibility was already exploited in Mastrolilli and Gambardella (2000) where the authors consider a set of moves, called k-insertion. A k-insertion assigns an operation v to

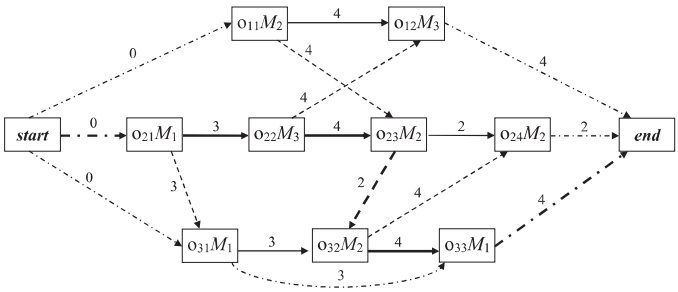

Fig. 4. A neighbor of the solution of Fig. 1 created with N^{α}. The makespan is reduced from 21 to 17.

a new machine $M_k \in M(v)$ and then looks for the optimal position for v in the machine sequence of k.

We propose here a simpler and less time consuming method: given a solution $S = (\alpha, \pi)$, a neighbor is obtained by assigning a new machine $M_{k'} \in M(v)$ to a critical operation v. The new position for v in the machine sequence of $M_{k'}$ is chosen in such a way that the topological order of the graph after the move remains identical, that is, the order π is the same after the new assignment; so ensuring feasibility. Notice that changing the machine of non-critical operations cannot produce an immediate improvement in the makespan. This structure is defined as follows.

Definition 4. N^{α} **structure.** Let $S = (\alpha, \pi)$ be a schedule, let v be a critical operation and let $\alpha_{<v,k>}$ be the assignment obtained from α after reassigning v to a new machine $M_k \in M(v)$, $k \neq \alpha(v)$. Then, $(\alpha_{<v,k>}, \pi)$ is a neighboring solution.

As an example, let us consider that the operation o_{12} can be processed on the machine M_3 in the schedule of Fig. 1. If o_{12} switches from M_1 to M_3, as o_{12} is after o_{22} in the topological order, then o_{12} is inserted after o_{22} in the new machine sequence of M_3. The result is the schedule of Fig. 4, whose makespan is better than that of the original schedule.

As we have done for the sequencing subproblem, we define the neighborhood N_2^{α} to be used in path relinking. This structure is the same as N^{α} except that the operation v may not be critical.

3.3. Combining sequencing and assignment structures

We also consider combinations of the previous structures to be used at different points of the proposed algorithm. For example, in the tabu search algorithm (see Section 4.5) it is convenient a neighborhood structure with a high chance of generating improving schedules. So, we define the following structure for this purpose.

Definition 5. $N_r^{\alpha\pi} = N^{\alpha} \cup N_r^{\pi}$

However, in the path relinking procedure the priority is not only to reduce the makespan but also to reduce the distance between two solutions. For this reason, some of the neighbors discarded by N^{α} or N_r^{π} may be very attractive in terms of distance or dissimilarity. This consideration leads us to define the following structures.

Definition 6. $N^{\alpha\pi} = N^{\alpha} \cup N^{\pi}$

Definition 7. $N_2^{\alpha\pi} = N_2^{\alpha} \cup N_2^{\pi}$

In Section 4.4 we will describe how to integrate these structures in the path relinking procedure.

3.4. Makespan estimate

To estimate the makespan of the neighbors created with N_r^{π}, N^{π} and N_2^{π} we propose to use a direct extension of the procedure *lpath* given in Amico and Trubian (1993) for the classical JSP. Let (α, π') be a schedule obtained from (α, π). In order to estimate the makespan of (α, π'), the procedure takes as input a sequence of operations requiring the same machine $(x\ Q_1 \dots Q_q\ y)$ in π', where $Q_1 \dots Q_q$ is a permutation of the operations $O_1 \dots O_q$ appearing as $(x\ O_1 \dots O_q\ y)$ in π. It is important to notice that the heads of operations before x and the tails of operations after y do not change in the neighbor. So, for each $i = 1 \dots q$, the procedure estimates the cost of the longest path from node *start* to *end* through Q_i, and the maximum of these values is taken as the makespan estimate for the neighboring schedule. This estimate is not necessarily a lower bound. Additionally, for N_2^{π} it may happen that the processing order of critical operations is not changed, and in that case the estimate can be refined a little bit more, as it cannot be lower than the makespan of the original schedule. For additional details on the *lpath* procedure we refer the interested reader to Amico and Trubian (1993).

Let us now discuss the estimate of the neighbors created with N^{α} and N_2^{α}. Given a schedule $S = (\alpha, \pi)$ and an operation x with $k = \alpha(x)$, assigning x to another machine $M_{k'} \in M(x)$ produces a feasible schedule $S' = (\beta, \pi)$ where $\beta = \alpha_{<x,k'>}$, i.e., $\beta(y) = \alpha(y)$ for all $y \neq x$ and $\beta(x) = k'$. As before, the heads of the operations before x and the tails of the operations after y do not change, and the head and the tail of x after the move are given by the following expressions:

$$\begin{aligned} r'_x &= \max\{r_{PJ_x} + p_{PJ_xk_1}, r_{PM_x} + p_{PM_xk'}\} \\ q'_x &= \max\{q_{SJ_x} + p_{SJ_xk_2}, q_{SM_x} + p_{SM_xk'}\} \end{aligned} \tag{2}$$

where $k_1 = \alpha(PJ_x)$ and $k_2 = \alpha(SJ_x)$. Therefore, the makespan of $S' = (\beta, \pi)$ can be estimated as $C^e_{max}(S') = r'_x + p_{xk'} + q'_x$. In this case the estimate is a lower bound on $C_{max}(S')$.

Additionally, in the case that x is not critical (as it may occur in the N_2^{α} structure), the makespan cannot be lower than that of the original schedule, therefore in that case the estimate can be refined as $C^e_{max}(S') = \max\{r'_x + p_{xk'} + q'_x, C_{max}(S)\}$.

In Section 5.2 we report some experimental results showing the accuracy of the proposed estimates.

4. Scatter search for the FJSP

Scatter Search (SS) is a population-based evolutionary metaheuristic recognized as an excellent method at achieving a proper balance between intensification and diversification in the search process. SS was first proposed by Glover (1998) and it has been successfully applied to a number of problems, in particular to scheduling (González, Vela, Varela, & González-Rodríguez, 2015; Nowicki & Smutnicki, 2006; Sels, Craeymeersch, & Vanhoucke, 2011; Yamada & Nakano, 1996).

The SS five-method template proposed in Glover (1998) has been the main reference for most SS implementations to date, including ours. This template consists of (1) a diversification-generation method, (2) an improvement method, (3) a reference-set update method, (4) a subset-generation method and (5) a solution-combination method.

Algorithm 1 starts by creating an initial set of solutions P which are improved using TS (Tabu Search). Then, a reference set *RefSet* is obtained selecting the best solutions from P. The algorithm stops after a given number of iterations without improvement. At each iteration, a pair of solutions from *RefSet* (selected according to the subset generation method) is combined using PR (Path Relinking) to generate a new solution, which is also improved by TS. Then, the reference set update method is applied. Additionally, if all possible pairs of solutions in *RefSet* have already been combined without introducing any new solution to the set, a diversification phase is applied. Further details on the algorithm are given in the following sections.

Require: A FJSP instance
  Build the set P with $PSize$ random solutions;
  Apply Tabu Search to every solution of P;
  Build the reference set $RefSet$;
  while not Stop Condition **do**
    if All possible pairs of solutions in $RefSet$ were already combined **then**
      Apply diversification phase;
    end if
    Choose two solutions S_{ini} and S_{end} from $RefSet$;
    Apply PR to S_{ini} and S_{end} to obtain a new solution S;
    Apply Tabu Search to S to obtain S^*;
    Update $RefSet$, if necessary, with S^*;
  end while
  return The best solution in $RefSet$;

Algorithm 1: Scatter search.

4.1. Dissimilarity measure

The definition of "distance" between two solutions in the presence of flexibility is not straightforward. Although having properties of a metric is desirable, a dissimilarity measure can be quite effective without being a metric, that is, a distance (Goshtasby, 2012).

In this paper we propose for the FJSP that the "difference" between two solutions S_1 and S_2 (denoted by $d(S_1, S_2)$) is an ordered pair of integer numbers, one representing the dissimilarity in the sequencing subproblem (denoted by $d^\pi(S_1, S_2)$) and the other one representing the distance on the assignment subproblem (denoted by $d^\alpha(S_1, S_2)$). The first one is an extension of the disjunctive graph distance, or Hamming distance, defined by Mattfeld (1995) for the classical JSP, as the number of pairs of operations requiring the same machine which are processed in different order in S_1 and S_2. This definition in the flexible case does not fulfill the triangle inequality, and then it is not a *distance* but a *dissimilarity coefficient* (Webb, 2002). Regarding the assignment distance $d^\alpha(S_1, S_2)$, it is defined as the number of operations having a different machine assignment in S_1 and S_2.

Considering these two measures, d^α and d^π, we will adopt a lexicographic approach and define a precedence relation between two pairs $d_1 = (d_1^\alpha, d_1^\pi)$ and $d_2 = (d_2^\alpha, d_2^\pi)$ as follows:

$$d_1 \prec d_2 \Leftrightarrow d_1^\alpha < d_2^\alpha \vee \left(d_1^\alpha = d_2^\alpha \wedge d_1^\pi < d_2^\pi\right) \tag{3}$$

We will use this precedence relation to compute the minimum "difference" (i.e., the maximum similarity) as $\min_\prec(d_1, d_2) = d_1$ if $d_1 \prec d_2$ and $\min_\prec(d_1, d_2) = d_2$ otherwise. The relation $\min_\prec$ can be naturally extended for an arbitrary number of elements and also the $\max_\prec$ relation can be defined accordingly.

In Jia and Hu (2014), the authors define the distance between two solutions as the number of operations that are located at different positions on the coding strings. So, for a problem instance with 10 machines and 10 jobs each one with 10 operations, the maximum distance between two solutions would be 100. Hence, the PR (Path Relinking) procedure may only visit a maximum of 100 candidate solutions in the path from one solution to another. However, with our definition, the maximum d^π would be 4500[1] and the maximum d^α would be 100. In this way, a strategy such as $N_2^{\alpha\pi}$ will allow for a fine-grain exploration of the search space.

[1] In the worst case all 100 tasks are performed on the same machine, so considering that all pairs of tasks belonging to different jobs may be processed in different order in the two schedules, the maximum dissimilarity is calculated as $(100 \times 99)/2 - 10 \times (10 \times 9)/2 = 4500$.

4.2. Initial reference set construction

As suggested in Martí, Laguna, and Glover (2006), the reference set must contain a collection of high quality and diverse solutions. To achieve this, an initial set P is generated with $PSize$ random solutions which are all improved using TS. Then, the reference set $RefSet$ (of size $RSSize$) is built taking solutions from P one at a time such that, if possible, each solution fulfill the condition $d^\pi(S, S_{RS}) > MinD^\pi$ or $d^\alpha(S, S_{RS}) > MinD^\alpha$ for all S_{RS} already in $RefSet$, being $MinD^\pi$ and $MinD^\alpha$ parameters. Firstly, half of the solutions are taken from the best ones in terms of makespan and the remaining are taken from the most distant[2] to the solutions already included in $RefSet$. If $RefSet$ cannot be completed with solutions fulfilling the above condition, then the remaining ones are selected at random.

4.3. Subset generation method and diversification phase

For subset generation, each pair of solutions in $RefSet$ is combined to obtain a new solution (as explained in Section 4.4), unless they have not changed since the last time they were selected together. TS is then applied to the new solution and $RefSet$ is updated in accordance with the reference set updating method (see Section 4.6). If no new solution is added to $RefSet$ after combining all possible pairs of solutions, the diversification process is applied: the set P is rebuilt starting with the best solution so far, and new $PSize - 1$ solutions are generated at random. These schedules are then improved by TS, and finally a new $RefSet$ is obtained from P as described in Section 4.2.

4.4. Solution-combination method

As solution-combination method we use PR. This metaheuristic has been successfully applied to the classical job shop scheduling problem. For example, Nowicki and Smutnicki (2005) improve their famous TSAB metaheuristic by introducing a new initial solution generator based on PR. Also, Nasiri and Kianfar (2012) combine TS and PR using two different neighborhoods.

PR combines two solutions, referred to as initial (S_{ini}) and guiding (S_{end}) solutions, to obtain a new solution. Starting from S_{ini}, it repeatedly applies moves so that each single move produces a solution which is closer to S_{end} than the current one. In our algorithm, S_{ini} is always the best of the two solutions taken from $RefSet$. In principle, the moves are those of $N^{\alpha\pi}$ (see Definition 6).

Let S be a solution and S_{end} be the guiding solution of the PR algorithm. The following Propositions 4 and 5 detail how the dissimilarity measures change between two consecutive solutions of the trajectory, depending on the neighborhood used.

Proposition 4. *Let $S_\pi \in N^\pi(S)$. Then, $d^\alpha(S_\pi, S_{end}) = d^\alpha(S, S_{end})$*

Proof. It is trivial as the neighborhood N^π does not change any machine assignment. □

On the other hand, a single move of N^π may reverse several processing orders of operations, hence $d^\pi(S_\pi, S_{end})$ may be several units lower or higher than, or even equal to, $d^\pi(S, S_{end})$.

Proposition 5. *Let $S_\alpha \in N^\alpha(S)$. Then, one and only one of the following conditions holds:*

- $d^\alpha(S_\alpha, S_{end}) = d^\alpha(S, S_{end}) - 1 \wedge d^\pi(S_\alpha, S_{end}) \geq d^\pi(S, S_{end})$
- $d^\alpha(S_\alpha, S_{end}) = d^\alpha(S, S_{end}) \wedge d^\pi(S_\alpha, S_{end}) = d^\pi(S, S_{end})$
- $d^\alpha(S_\alpha, S_{end}) = d^\alpha(S, S_{end}) + 1 \wedge d^\pi(S_\alpha, S_{end}) \leq d^\pi(S, S_{end})$

[2] The distance of a solution S to a set of solutions is given by the minimum distance from S to a solution of the set.

Require: A FJSP instance, an initial schedule S_{ini} and a guiding schedule S_{end}
$S \leftarrow S_{ini}$; $disMin, disCur \leftarrow d(S, S_{end})$; $numFails \leftarrow 0$; $TL \leftarrow \emptyset$;
while $S \neq S_{end}$ **do**
// Compute the candidate neighbors from each neighborhood
if $numFails < maxFails$ **then** $N_S = N^{\pi}$; $N_A = N^{\alpha}$; **else** $N_S = N_2^{\pi}$; $N_A = N_2^{\alpha}$; **end if**
$CN^{\alpha} \leftarrow \{S' \in N_A(S) : d(S', S_{end}) \prec disMin \vee \neg Tabu(S', TL)\}$;
$CN^{\pi} \leftarrow \{S' \in N_S(S) : d(S', S_{end}) \prec disMin \vee \neg Tabu(S', TL)\}$;
// Select two neighbors: one for each candidate set, tie-breaking C^e_{max}
$S^*_{\alpha} \leftarrow argmin\{d^{\alpha}(S', S_{end}) : S' \in CN^{\alpha}\}$;
$S^*_{\pi} \leftarrow argmin\{d^{\pi}(S', S_{end}) : S' \in CN^{\pi}\}$;
// Choose one of the two selected neighbors
if $d^{\pi}(S^*_{\pi}, S_{end}) < d^{\pi}(S, S_{end}) \wedge d^{\alpha}(S^*_{\alpha}, S_{end}) \geq d^{\alpha}(S, S_{end})$ **then** $S^* \leftarrow S^*_{\pi}$;
else if $d^{\pi}(S^*_{\pi}, S_{end}) \geq d^{\pi}(S, S_{end}) \wedge d^{\alpha}(S^*_{\alpha}, S_{end}) < d^{\alpha}(S, S_{end})$ **then** $S^* \leftarrow S^*_{\alpha}$;
else $S^* \leftarrow argmin\{C^e_{max}(S^*_{\alpha}), C^e_{max}(S^*_{\pi})\}$; **end if**
end if
// Finally, update the variables
if $(d(S^*, S_{end}) \prec disMin)$ **then** $disMin \leftarrow d(S^*, S_{end})$; **end if**
if $(d(S^*, S_{end}) \succ disCur)$ **then** $numFails \leftarrow numFails + 1$; **end if**
$disCur \leftarrow d(S^*, S_{end})$;
$S \leftarrow S^*$;
Update TL accordingly;
end while
return the best solution between 1/4 and 3/4 of the created trajectory, or one outside this range if it has the best makespan found so far;

Algorithm 2: Path relinking.

Proof. As N^{α} only changes the machine assignment of one operation, then $d^{\alpha}(S_{\alpha}, S_{end}) - d^{\alpha}(S, S_{end})$ is 1, 0 or −1, if the operation has the same machine assigned in S and S_{end}, or it has different assignment in S_{end} than in both S and S_{α}, or it has the same assignment in S_{α} and S_{end}, respectively.

If $d^{\alpha}(S_{\alpha}, S_{end}) - d^{\alpha}(S, S_{end}) = -1$, then $d^{\pi}(S_{\alpha}, S_{end}) \geq d^{\pi}(S, S_{end})$ due to the fact that any pair of operations processed in the same order and the same machine in S and S_{end} remain in S_{α} and new pairs may be processed in S_{α} in different order than in S_{end} due to the new machine assignment. Analogous reasoning can be done if the value of the expression above is 1 or 0. □

From all the above, to select the next move in the PR algorithm, we start exploiting $N_{\alpha\pi}$ in the following way: the best solutions in each of $N^{\alpha}(S)$ and $N^{\pi}(S)$ in terms of distance to the guiding solution S_{end} are considered. Then, the best of these two solutions, S^*_{α} and S^*_{π}, is selected considering the estimated makespan and the proximity to S_{end}. In order to escape from local optima, similarly to TS, a neighbor obtained by reversing recently reversed arcs or by changing recent assignments is discarded, unless it is the closest to the guiding solution found so far.

Even with this mechanism, local optima may be so deep that $N^{\alpha\pi}$ may have difficulties to obtain an improving move. For this reason, we have considered the less restrictive neighborhood structure $N_2^{\alpha\pi}$ (see Definition 7). Thus, as soon as $N^{\alpha\pi}$ fails to reach a solution closer to the guiding solution *maxFails* times, the neighborhood structure changes to $N_2^{\alpha\pi}$ for the remaining of the path relinking procedure[3]. An alternative to the above would be using only $N_2^{\alpha\pi}$ in the path relinking algorithm. However, it is better to start using $N^{\alpha\pi}$, since it is a more refined structure which guides the path through promising neighbors, and use $N_2^{\alpha\pi}$ only to complete the path when $N^{\alpha\pi}$ gets stuck into local optima in terms of distance.

The search finishes when the guiding solution is reached. Then, as proposed by Nowicki and Smutnicki (2006), the algorithm returns the solution with the best makespan that is located between 1/4 and 3/4 of the created trajectory, unless a solution improving the best solution so far is found out of this range.

This way of building a path from the initial to the guiding solution is also quite different from that proposed in Jia and Hu (2014) and it generally builds a longer path. This can be seen in the example given in that paper where the path has only two intermediate solutions whereas our method would generate at least six.

The proposed PR algorithm is detailed in Algorithm 2, where TL denotes the Tabu List, and $\neg Tabu(S', TL)$ means that the move from S to S' is not in TL.

4.5. Improvement method

As improvement method we use TS (Tabu Search). This is an advanced local search technique with a solid record of good empirical performance in problem solving, in particular in scheduling (González, Vela, González-Rodríguez, & Varela, 2013; Nowicki & Smutnicki, 2005).

The general scheme of our TS is similar to that proposed in Amico and Trubian (1993). In the first step the initial solution is evaluated. Then, it iterates over a number of steps. In each iteration, the neighborhood of the current solution is built using $N_r^{\alpha\pi}$ (see Definition 5) and one of the neighbors is selected for the next iteration. The selection rule chooses the neighbor with the best estimated makespan, discarding suspect-of-cycle and tabu neighbors. The tabu search finishes after a number of *maxImproveIter* iterations without improvement, returning the best solution reached so far.

Instead of storing actual solutions in the tabu list, those arcs which have been reversed or the assignments that have been changed to generate a neighbor are stored. Thus a new neighbor is marked as tabu if it requires reversing at least one arc or changing a machine assignment included in the tabu list, unless its estimated makespan is better than the best makespan found so far.

The length of the tabu list is usually of critical importance, since it allows for an equilibrium between intensification and diversification. All TS algorithms try to manage this equilibrium with different proposals based on controlling the number of iterations for which a solution can keep its tabu status. We use here the dynamic length schema and the cycle checking mechanism based on witness arcs used, among others, by Amico and Trubian (1993), and extended it by adding the information about changes in the machine assignment of operations.

The time given to each tabu search must be short enough for the overall computational time of the scatter search algorithm not to be prohibitive. For this reason, we do not use any more diversification techniques in the TS procedure proposed here.

4.6. Reference set update

Let S_B and S_W be the best and worst solutions in *RefSet* respectively. Each solution returned by PR is improved by TS. Then, for the sake of quality and diversity, the resulting solution S replaces S_W either if $C_{max}(S) < C_{max}(S_B)$, or if $C_{max}(S) < C_{max}(S_W)$ and for all $S_{RS} \in RefSet$ it holds that $d^{\pi}(S, S_{RS}) > MinD^{\pi}$ or $d^{\alpha}(S, S_{RS}) > MinD^{\alpha}$.

4.7. Similarities and differences w.r.t. other proposals

As we have mentioned, metaheuristics have been widely applied to solve the FJSP. In this work, we borrowed some ideas from existing methods and include a good number of new ones. For example,

[3] Notice that $N_2^{\alpha\pi}(S)$ contains at least one solution closer to S_{end} than S. This guarantees that Algorithm 2 finishes in a finite number of iterations.

we used SS and TS templates similar to those proposed in Glover (1998) and Amico and Trubian (1993) respectively. On the other hand, the proposed neighborhood structures are quite different from other structures defined for the FJSP in Mastrolilli and Gambardella (2000), González et al. (2013), Yuan and Xu (2013). However, if we will restrict to the JSP, we could observe that some sequencing structures are in fact extensions of some classic structures: N^{π} extends the structure *NB* proposed in Amico and Trubian (1993) and N_2^{π} extends the structure H' defined in Van Laarhoven et al. (1992). At the same time, the structure H defined in Nowicki and Smutnicki (1996) is a restriction of H', in much the same way as N_r^{π} restricts N^{π} (removing moves for which it is known a priori that they will not improve the makespan). For the sequencing structures we established feasibility and non-improvement conditions, as well as a procedure to estimate the makespan of the neighbors, which are inspired in proposals given in Amico and Trubian (1993), Mattfeld (1995) for the classic JSP. Also, the proposed assignment structures N^{α} and N_2^{α} are simpler and less time consuming versions of the *k*-insertion structure proposed in Mastrolilli and Gambardella (2000). A key point of our algorithm is the new fine grain dissimilarity measure for solutions. With this measure and the neighborhood structures $N^{\alpha\pi}$ and $N_2^{\alpha\pi}$, the trajectories between two solutions are much larger than, for example, those obtained from the coarse grain measure defined in Jia and Hu (2014), which allows PR to explore more solutions in each run.

5. Experimental study

We have conducted an experimental study to evaluate our proposals and to compare the algorithm, termed SSPR, with the state of the art. In this study, we have considered the four benchmark sets most widely used in the literature, namely, *DPdata* (Dauzère-Pérès & Paulli, 1997), *BCdata* (Barnes & Chambers, 1996), *BRdata* (Brandimarte, 1993) and *HUdata* (Hurink, Jurisch, & Thole, 1994), making a total of 178 instances with different sizes and flexibility (the average number of possible machines per operation).

SSPR was implemented in C++ and the target machine was Intel Core 2 Duo at 2.66 gigahertz and 2 gigabytes RAM. We run SSPR 10 times for each instance and register the best and average makespan. To compare different methods, we indicate whether or not a method was able to reach the best known solution for each instance and report the Relative Percentage Deviation (RPD) defined as:

$$RPD = ((Alg_{sol} - LB)/LB) \times 100$$

where Alg_{sol} is the value of the objective function obtained by a given algorithm for a given instance, and *LB* is a lower bound on the optimal solution. We consider here the lower bounds reported in Mastrolilli and Gambardella (2000) for all 178 instances.

In the next sections we report and analyze the results of our experimental study. Firstly, we describe how we have done the parameter tuning. Then, we analyze the proposed neighborhood structures and compare SSPR with its TS component running alone. In the last section, we present an exhaustive and fair comparison of SSPR with the current best methods.

5.1. Parameter tuning

We have performed a preliminary study considering different values of the parameters and using similar run times for each configuration tested. Below, we summarize the parameters of the SSPR algorithm and the tested values for each one of them.

RSSize defines the number of solutions of *RefSet*. This number is usually lower than 20, as indicated by Glover in Glover (1998). Here we tried $RSSize = 8$ and $RSSize = 10$ because these are typical values in the literature (see Nowicki & Smutnicki, 2006 and Yamada & Nakano, 1996).

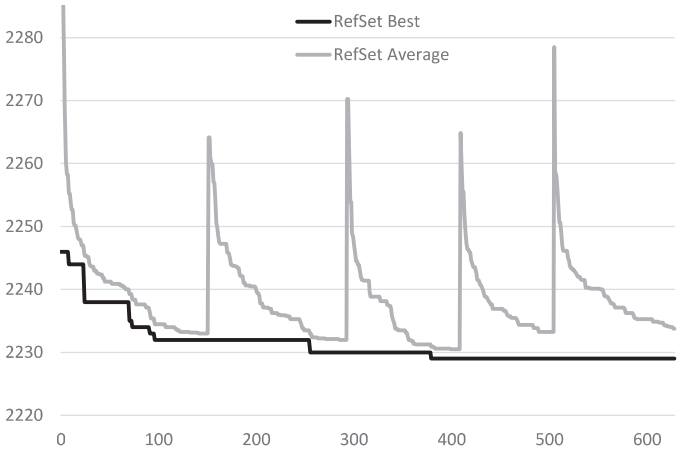

Fig. 5. Evolution of the best and average makespan of *RefSet* depending on the iteration number, for one run of the 02a instance of the *DPdata* benchmark.

PSize defines the number of solutions in the initial set *P*. In Glover, Laguna, and Martí (2003) it is suggested that $PSize = max(100, 5 * RSSize)$. However, the diversification phase that we have proposed will need to rebuild the set *P* several times during an execution. As this process is computationally expensive, we considered a smaller size for *P*. In particular we tested sizes 20 and 100.

maxFails is a parameter that defines the number of times PR chooses a non-improving neighbor with $N^{\alpha\pi}$ before switching to $N_2^{\alpha\pi}$. We considered the values 1, 5 and 20.

maxImproveIter is the maximum number of iterations without improvement in each run of TS and establishes its stopping condition. TS is embedded in the scatter search core and so it is issued many times during an execution of SSPR, for this reason we cannot choose values as high as in a TS algorithm running alone. We considered values of 100, 500, 2000 and 10,000.

$MinD^{\alpha}$ and $MinD^{\pi}$ are two parameters that control the minimum distance allowed between solutions in *RefSet*, and hence they should ensure diversity in the set. We tested values of 5, 20 and 50 for $MinD^{\pi}$, and values of 1, 3 and 5 for $MinD^{\alpha}$.

From this preliminary study, we have chosen $RSSize = 8$, $PSize = 20$, $maxFails = 5$, $maxImproveIter = 2000$, $MinD^{\pi} = 20$ and $MinD^{\alpha} = 3$, as this was the combination that reached the best results in average. As stopping criterion we choose a number of 250 iterations of SSPR without improvement. This value results in reasonable convergence patterns, as shown in Fig. 5, which details the evolution of the best and average makespan using the chosen configuration for one run on the 02a instance of the *DPdata* benchmark. Notice the four times the diversification phase is issued with a sudden increase in the average makespan of the solutions in *RefSet*.

5.2. Analysis of the neighborhood structures

To analyze the neighborhood structures we considered all instances in the *HUdata* benchmark. Firstly, we have evaluated the percentage of neighbors created by N_r^{π} and N^{α} in the TS procedure. Table 1 illustrates that this percentage strongly depends on the flexibility of the instance; which is reasonable as the number of neighbors created with N^{α} is in direct ratio with the number of machines suitable for each operation on the critical path.

Table 1
Percentage of neighbors created by N_r^{π} and N^{α} in the TS procedure.

Ins. (flex)	*edata* (1.15)	*rdata* (2)	*vdata* (5–7.5)
Percent Neigh by N^{α}	8	47	73

Table 2
Average length of the critical blocks and number of neighbors created by N_r^π.

Size	10 × 5	15 × 5	20 × 5	10 × 10	15 × 10	20 × 10	30 × 10	15 × 15
#Op. in CB	6.39	11.57	16.45	2.83	4.83	7.85	16.72	2.98
#Neigh per CB	9.60	17.69	25.68	2.83	6.44	10.02	17.31	3.08

Table 3
Accuracy of the estimates.

	N_r^π		N^α	
	Percent of Est	Rel dif	Percent of Est	Rel dif
$C^e_{max} = C_{max}$	83.3		96.9	
$C^e_{max} < C_{max}$	3.0	1.4	3.1	3.1
$C^e_{max} > C_{max}$	13.7	2.1	–	

Table 4
Comparison of average performance of tabu search running alone versus SSPR.

	Tabu search		SSPR	
	Best	Avg	Best	Avg
RPD	2.26	2.52	1.57	1.79

Then, we analyzed the influence of the size and shape of the instances on the length and the number of neighbors of the critical blocks. The results summarized in Table 2 show clearly that these values depend on the instance size but mainly on the quotient n/m. This is reasonable as for instances with few machines and many operations, critical paths most certainly go through many disjunctive arcs. We also made some experiments to compare our definition of critical block with that considered in Mastrolilli and Gambardella (2000) and in González et al. (2013), i.e., not allowing several operations of the same job in the same critical block. The results (not reported here) were slightly better with our definition as the number of generated neighbors was lower, and this fact allowed our algorithm to obtain better results in slightly less time.

Finally, in order to assess the makespan estimation procedure, we have registered the actual and estimated makespan of about 300 million neighbors. Table 3 summarizes the percentages of coincidences and discrepancies, in this last case showing the relative errors. As we can observe, the accuracy of the estimation is high, especially for N^α. In these experiments we have also observed that the accuracy is in inverse ratio with flexibility.

5.3. Comparing SSPR with tabu search running alone

It is well known that tabu search is a very efficient metaheuristic at solving scheduling problems (see for example Nowicki & Smutnicki, 2005 or González et al., 2013). Here we have carried out some experiments to assess if the scatter search with path relinking shell is capable of improving the performance of the tabu search running alone. To this end, we compared the results of SSPR with those from tabu search, giving both algorithms the same time. In the experiments, we have opted to set the same stopping criterion as in SSPR (2000 iterations without improvement), and to launch tabu search starting from random solutions as many times as possible in the time taken by SSPR. We have tried other settings with worse results for TS.

We have used the *DPdata* benchmark for this comparison. To obtain similar run times we have had to launch tabu search from 300 to 600 times, depending on the particular instance. In 16 of the 18 instances the best makespan reached by tabu search was worse than that reached by SSPR, and in the remaining 2 instances the results were the same. The mean values are summarized in Table 4. From these results it may be fair to consider that SSPR outperforms tabu search running alone.

5.4. Comparison with the state of the art

Finally, we have conducted experiments to compare SSPR with the best algorithms we have found in the literature, namely: TS, hGA, HHS/LNS, CDDS, GA + TS, TSBM^{2}h and MGARH from Mastrolilli and Gambardella (2000), Gao et al. (2008), Yuan and Xu (2013), Hmida et al. (2010), González et al. (2013), Bozejko et al. (2010), Gutierrez and Garcia-Magario (2011) respectively.

Tables 5–7 show the results of the experiments in *DPdata*, *BCdata* and *BRdata* benchmarks respectively. We indicate for each instance its flexibility and the lower bound reported in Mastrolilli and Gambardella (2000). Then, for each method we report the best and average (between parentheses) makespan. We also report the average runtime in seconds of a single run of SSPR. At the bottom of each table we show the average of the best and the mean RPD for each algorithm, the number of instances for which a method reaches the best known solution, and the sum of average Computer-Independent CPU time. CI-CPU time values have been obtained from the normalization coefficients of Dongarra (2013). As indicated in Yuan and Xu (2013) "it should be noted that the comparison between CPU time is meant to be indicative, because we do not have access to other information that influences the computation time, such as the operating systems, software engineering decisions, and coding skills of the programmer."

In the *DPdata* benchmark (Table 5) SSPR performs the best in all metrics considered, improving previous best known solutions in 13 out of the 18 instances. Notice that the mean RPD of SSPR is better than the best RPD of the remaining methods in average.

The results on the *BCdata* set are shown in Table 6. In this case, we include the results from TSBM^{2}h reported in Bozejko et al. (2010) and omit results from HHS/LNS, as only the average RPD for the best solutions (22.43) is given in Yuan and Xu (2013) for this data set. Overall, SSPR is the best of the six algorithms in all metrics, and improves the previous best known solution in 4 of the 21 instances.

Table 7 shows the results on *BRdata*. This is the only table that includes results from MGARH, as this is the only set considered in Gutierrez and Garcia-Magario (2011). In this case, SSPR obtains the best known solution in all instances, it is the second best considering the average makespan and the best one considering the best makespan, as indicated by RPD values.

From the CI-CPU times reported in Tables 5–7, we can observe that SSPR is the best on *DPdata* and *BCdata*. However, it appears as the worst one for *BRdata* due to the parameters chosen, which make SSPR to run for too long time in most of the cases. This is clear as SSPR reaches the best solution in the first iteration for most of these instances with negligible time. In Hmida et al. (2010) the authors did not provide the time taken by HHS/LNS on these instances as they consider these times not relevant for similar reasons.

For *HUdata* set, we only report summary results of RPD (detailed results from SSPR on all benchmarks considered here are openly available on the web[4]). Table 8 reports results reached by TS, hGA, CDDS, HHS/LNS and SSPR. We give the average value of the RPD of the best makespan and average makespan across each of the three subsets of 43 instances: *edata*, *rdata* and *vdata*. It is remarkable that the RPD values of the best and average solutions obtained by SSPR are better than those obtained by all other algorithms.

[4] Repository section in http://www.di.uniovi.es/iscop.

Table 5
Summary of results: ***DPdata***.

Ins	flex.	LB	TS	hGA	CDDS	HHS/LNS	GA + TS	SSPR	T(s.)
01a	1.13	2505	2518 (2528)	2518 (2518)	2518 (2525)	**2505** (2513)	**2505** (2511)	**2505** (2508)	68
02a	1.69	2228	2231 (2234)	2231 (2231)	2231 (2235)	2230 (2231)	2232 (2234)	**2229**(*) (2230)	100
03a	2.56	2228	2229 (2230)	2229 (2229)	2229 (2232)	**2228** (2229)	2229 (2230)	**2228**(*) (2228)	110
04a	1.13	2503	**2503** (2516)	2515 (2518)	**2503** (2510)	2506 (2506)	**2503** (2504)	**2503** (2504)	57
05a	1.69	2189	2216 (2220)	2217 (2218)	2216 (2218)	2212 (2215)	2219 (2221)	**2211**(*) (2215)	112
06a	2.56	2162	2203 (2206)	2196 (2198)	2196 (2203)	2187 (2192)	2200 (2204)	**2183**(*) (2192)	181
07a	1.24	2187	2283 (2298)	2307 (2310)	2283 (2296)	2288 (2303)	**2266** (2286)	2274 (2285)	139
08a	2.42	2061	2069 (2071)	2073 (2076)	2069 (2069)	2067 (2074)	2072 (2075)	**2064**(*) (2066)	181
09a	4.03	2061	2066 (2067)	2066 (2067)	2066 (2067)	2069 (2073)	2066 (2067)	**2062**(*) (2063)	213
10a	1.24	2178	2291 (2306)	2315 (2315)	2291 (2303)	2297 (2302)	**2267** (2273)	2269 (2287)	120
11a	2.42	2017	2063 (2066)	2071 (2072)	2063 (2072)	2061 (2067)	2068 (2071)	**2051**(*) (2058)	193
12a	4.03	1969	2034 (2038)	2030 (2031)	2031 (2034)	2027 (2036)	2037 (2041)	**2018**(*) (2020)	280
13a	1.34	2161	2260 (2266)	2257 (2260)	2257 (2260)	2263 (2269)	2271 (2276)	**2248**(*) (2257)	119
14a	2.99	2161	2167 (2168)	2167 (2168)	2167 (2179)	2164 (2168)	2169 (2171)	**2163**(*) (2164)	269
15a	5.02	2161	2167 (2167)	2165 (2165)	2165 (2170)	2163 (2166)	2166 (2166)	**2162**(*) (2163)	376
16a	1.34	2148	2255 (2259)	2256 (2258)	2256 (2258)	2259 (2266)	2266 (2271)	**2244**(*) (2253)	131
17a	2.99	2088	2141 (2144)	2140 (2142)	2140 (2146)	2137 (2141)	2147 (2150)	**2130**(*) (2134)	299
18a	5.02	2057	2137 (2140)	2127 (2131)	2127 (2132)	2124 (2128)	2138 (2141)	**2119**(*) (2123)	409
RPD			2.01 (2.24)	2.12 (2.19)	1.94 (2.19)	1.89 (2.13)	1.99 (2.17)	1.57 (1.79)	
#best			1	0	1	2	4	16	
CI-CPU			2467	6206	2890	6279	3397	1934	

Values in bold are best known solutions, (*) improves previous best known solution.

Table 6
Summary of results: ***BCdata***.

Ins	flex.	LB	TS	hGA	CDDS	TSBM^{2}h	GA + TS	SSPR	T(s.)
mt10c1	1.10	655	928 (928)	**927** (927)	928 (929)	**927** (–)	**927** (927)	**927** (928)	26
mt10cc	1.20	655	910 (910)	910 (910)	910 (911)	**908** (–)	**908** (909)	**908** (908)	20
mt10x	1.10	655	**918** (918)	**918** (918)	**918** (918)	922 (–)	**918** (922)	**918** (918)	23
mt10xx	1.20	655	**918** (918)	**918** (918)	**918** (918)	**918** (–)	**918** (918)	**918** (918)	19
mt10xxx	1.30	655	**918** (918)	**918** (918)	**918** (918)	**918** (–)	**918** (918)	**918** (918)	20
mt10xy	1.20	655	906 (906)	**905** (905)	906 (906)	**905** (–)	**905** (905)	**905** (906)	21
mt10xyz	1.30	655	**847** (850)	849 (849)	849 (851)	849 (–)	849 (850)	**847** (847)	20
setb4c9	1.10	857	919 (919)	**914** (914)	919 (919)	**914** (–)	**914** (914)	**914** (916)	28
setb4cc	1.20	857	909 (912)	914 (914)	909 (911)	**907** (–)	**907** (907)	**907** (907)	21
setb4x	1.10	846	**925** (925)	**925** (931)	**925** (925)	**925** (–)	**925** (925)	**925** (925)	19
setb4xx	1.20	846	**925** (926)	**925** (925)	**925** (925)	**925** (–)	**925** (925)	**925** (925)	21
setb4xxx	1.30	846	**925** (925)	**925** (925)	**925** (925)	**925** (–)	**925** (925)	**925** (925)	22
setb4xy	1.20	845	916 (916)	916 (916)	916 (916)	**910** (–)	**910** (910)	**910** (912)	32
setb4xyz	1.30	838	905 (908)	905 (905)	905 (907)	**903** (–)	905 (905)	905 (905)	21
seti5c12	1.07	1027	1174 (1174)	1175 (1175)	1174 (1175)	1174 (–)	1171 (1173)	**1170**(*) (1173)	25
seti5cc	1.13	955	1136 (1136)	1138 (1138)	1136 (1137)	1136 (–)	1136 (1137)	**1135**(*) (1136)	29
seti5x	1.07	955	1201 (1204)	1204 (1204)	1201 (1202)	**1198** (–)	1199 (1200)	**1198** (1199)	41
seti5xx	1.13	955	1199 (1201)	1202 (1203)	1199 (1199)	**1197** (–)	**1197** (1198)	**1197** (1199)	37
seti5xxx	1.20	955	1197 (1198)	1204 (1204)	1197 (1198)	1197 (–)	1197 (1197)	**1194**(*) (1198)	38
seti5xy	1.13	955	1136 (1136)	1136 (1137)	1136 (1138)	1136 (–)	1136 (1137)	**1135**(*) (1136)	29
seti5xyz	1.20	955	**1125** (1127)	1126 (1126)	**1125** (1125)	1128 (–)	1127 (1128)	**1125** (1126)	35
RPD			22.53 (22.63)	22.61 (22.66)	22.54 (22.60)	22.45 (–)	22.42 (22.49)	22.36 (22.44)	
#best			8	9	7	14	13	20	
CI-CPU			356	821	253	1068	437	248	

Values in bold are best known solutions, (*) improves previous best known solution. – means that the corresponding data is not available.

Table 7
Summary of results: ***BRdata***.

Ins	flex.	LB	TS	hGA	CDDS	HHS/LNS	MGARH	GA + TS	SSPR	T(s.)
Mk01	2.09	36	**40** (40)	**40** (40)	**40** (40)	**40** (–)	**40** (–)	**40** (40)	**40** (40)	11
Mk02	4.10	24	**26** (26)	**26** (26)	**26** (26)	**26** (–)	**26** (–)	**26** (26)	**26** (26)	15
Mk03	3.01	204	**204** (204)	**204** (204)	**204** (204)	**204** (–)	**204** (–)	**204** (204)	**204** (204)	24
Mk04	1.91	48	**60** (60)	**60** (60)	**60** (60)	**60** (–)	**60** (–)	**60** (60)	**60** (60)	19
Mk05	1.71	168	173 (173)	**172** (172)	173 (174)	**172** (–)	**172** (–)	**172** (172)	**172** (172)	57
Mk06	3.27	33	58 (58)	58 (58)	58 (59)	58 (–)	**57** (–)	58 (58)	**57** (58)	40
Mk07	2.83	133	144 (147)	**139** (139)	**139** (139)	**139** (–)	**139** (–)	**139** (139)	**139** (141)	84
Mk08	1.43	523	**523** (523)	**523** (523)	**523** (523)	**523** (–)	**523** (–)	**523** (523)	**523** (523)	83
Mk09	2.53	299	**307** (307)	**307** (307)	**307** (307)	**307** (–)	308 (–)	**307** (307)	**307** (307)	52
Mk10	2.98	165	198 (199)	197 (197)	197 (198)	198 (–)	**196** (–)	199 (200)	**196** (197)	94
RPD			15.41 (15.83)	14.92 (14.92)	15.41 (15.55)	14.98 (–)	14.59 (–)	15.04 (15.13)	14.55 (15.03)	
#best			6	8	7	8	9	8	10	
CI-CPU			74	91	96			250	**383**	

Values in bold are best known solutions, – means that the corresponding data is not available.

Table 8
Summary of results (in RPD): ***HUdata***.

Set	flex.	TS	hGA	CDDS	HHS/LNS	SSPR
edata	1.15	2.17 (2.18)	2.13 (2.40)	2.32 (2.78)	2.11 (–)	2.03 (2.10)
rdata	2	1.24 (1.36)	1.19 (1.21)	1.34 (1.54)	1.18 (–)	1.03 (1.12)
vdata	2.5 to 7.5	0.10 (0.13)	0.08 (0.09)	0.12 (0.16)	0.11 (–)	0.04 (0.05)

– means that the corresponding data is not available.

Overall, for all 129 instances in *HUdata*, SSPR reaches a solution equal to or better than the solution given by TS in Mastrolilli and Gambardella (2000). Being the makespan of these solutions equal to the lower bound in 82 instances and lower than the upper bounds reported in Mastrolilli and Gambardella (2000) for 41 instances.

Results from other methods are not included in Table 8 for different reasons. No results were given for GA + TS and MGARH in González et al. (2013) and Gutierrez and Garcia-Magario (2011) respectively. In Bozejko et al. (2010) the authors only report results for instances la01 to la15 of the *rdata* set, and in all cases these results are equal or worse than those obtained by TS.

Summarizing, we have considered 178 instances of the FJSP and in 175 of them we have reached the best known solution. Moreover, we have found new upper bounds for 13 instances of *DPdata*, 4 instances of *BCdata* and 41 instances of *HUdata*.[5]

To further avail the quality of the proposed method, we have done some statistical tests to analyze differences between SSPR and other algorithms. As we have multiple-problem analysis, we used non-parametric statistical tests. First, we run a Shapiro–Wilk test to confirm the non-normality of the data. Then we used paired Wilcoxon signed rank tests to compare the medians of the RPD values between SSPR and each one of the other methods, provided that results for single instances are available, that is, we have considered the set *BRdata* ∪ *BCdata* ∪ *DPdata* and the methods TS, hGA, CDDS and GA + TS. In these tests, the level of confidence used was 95 percent and the alternative hypothesis was "the difference between the errors of SSPR and the method tested is smaller than 0." The *p*-values obtained with these tests (TS: 4.027e−08, hGA: 2.133e−05, CDDS: 1.404e−06, GA + TS: 7.364e−04) show that there exist statistically significant differences between SSPR and the other tested methods in these benchmarks. In summary, we can conclude that SSPR is significantly better than some methods of the state of the art.

6. Conclusions

We devised a new algorithm for the flexible job shop scheduling problem with makespan minimization. The algorithm, termed SSPR, combines scatter search and path relinking with tabu search. From a thorough analysis of the problem and the schedules, we have obtained useful insights that allowed us to define some new neighborhood structures and methods to accurately estimate the makespan of the neighbors. Also, we have defined a novel fine grain dissimilarity measure for schedules. All these elements were incorporated in different components of the SSPR algorithm, which was evaluated and compared with the state of the art on the most commonly used benchmarks, showing clear improvements over existing methods in the literature considering the time taken and the makespan of the schedules together. In particular, SSPR was able to improve the best known solution in 58 of the 178 FJSP instances considered.

We believe that the main reasons for the good performance of SSPR are the combination of the diversification provided by the scatter search and path relinking shell combined with the intensification provided by the tabu search, together with the fact that the neighborhoods are specifically tailored to deal with flexibility. Also, the proposed dissimilarity measure is another strong point of SSPR as it allows PR to search over paths of diverse and potentially good candidate solutions of the search space.

[5] Remember that the upper bounds we considered for *HUdata* are those from TS reported in Mastrolilli and Gambardella (2000) as, to our knowledge, detailed results from other methods on this benchmark set have not yet been published. So, we do not know whether or not a better upper bound was reached by other method for some of these instances.

Acknowledgments

All authors are supported by the Spanish Government under projects MEC-FEDER TIN2010-20976-C02-02 and TIN2013-46511-C2-2-P and by the Principality of Asturias under project FICYT-COF13-035.

References

Amico, M. D., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, *41*, 231–252.
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.
Bozejko, W., Uchronski, M., & Wodecki, M. (2010). Parallel hybrid metaheuristics for the flexible job shop problem. *Computers & Industrial Engineering*, *59*(2), 323–333.
Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, *41*, 157–183.
Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, *45*(4), 369–375.
Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, *70*(3), 281–306.
Dongarra, J. (2013). Performance of various computers using standard linear equations software (Tech. rep.). Computer Science Department, University of Tennessee, Knoxville, Tennessee.
Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, *35*, 2892–2907.
Garey, M., Johnson, D., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, *1*(2), 117–129.
Glover, F. (1998). A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Artificial evolution, LNCS 1363* (pp. 13–54). Springer.
Glover, F., Laguna, M., & Martí, R. (2003). In *Advances in evolutionary computation: Theory and applications* (pp. 519–537). Springer.
González, M. A., Vela, C., González-Rodríguez, I., & Varela, R. (2013). Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. *Journal of Intelligent Manufacturing*, *24*(4), 741–754.
González, M., Vela, C. R., & Varela, R. (2013). An efficient memetic algorithm for the flexible job shop with setup times. In *Proceedings of ICAPS-2013* (pp. 91–99).
González, M. A., Vela, C., Varela, R., & González-Rodríguez, I. (2015), An advanced scatter search algorithm for solving job shops with sequence dependent and non-anticipatory setups. *AI Communications*, *28*, 179–193.
Goshtasby, A. A. (2012). *Advances in computer vision and pattern recognition*. Springer-Verlag.
Gutierrez, C., & Garcia-Magario, I. (2011). Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem. *Knowledge-Based Systems*, *24*, 102–112.
Hmida, A., Haouari, M., Huguet, M., & Lopez, P. (2010). Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research*, *37*, 2192–2201.
Ho, N. B., Tay, J. C., & 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, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machine. *Operations Research Spektrum*, *15*, 205–215.
Jia, S., & Hu, Z.-H. (2014). Path-relinking tabu search for the multi-objective flexible job shop scheduling problem. *Computers & Operations Research*, *47*, 11–26.
Martí, R., Laguna, M., & Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, *169*, 359–372.
Mastrolilli, M., & Gambardella, L. (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, *3*(1), 3–20.
Mati, Y., Dauzere-Peres, S., & Lahlou, C. (2011). A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, *212*, 33–42.

Mattfeld, D. C. (1995). *Evolutionary search and the job shop investigations on genetic algorithms for production scheduling*. Springer-Verlag.
Meeran, S., & Morshed, M. (2014). Evaluation of a hybrid genetic tabu search framework on job shop scheduling benchmark problems. *International Journal of Production Research*, *52*(9), 5780–5798.
Nasiri, M., & Kianfar, F. (2012). A guided tabu search/path relinking algorithm for the job shop problem. *International Journal on Advanced Manufacturing Technologies*, *58*, 1105–1113.
Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop scheduling problem. *Management Science*, *42*, 797–813.
Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, *8*(2), 145–159.
Nowicki, E., & Smutnicki, C. (2006). Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, *169*, 654–666.
Sels, V., Craeymeersch, K., & Vanhoucke, M. (2011). A hybrid single and dual population search procedure for the job shop scheduling problem. *European Journal of Operational Research*, *215*, 512–523.
Van Laarhoven, P., Aarts, E., & Lenstra, K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*, 113–125.
Webb, A. R. (2002). *Statistical pattern recognition* (2nd ed.). John Wiley & Sons.
Yamada, T. & Nakano, R. (1996). Scheduling by genetic local search with multi-step crossover. In *Proceedings of fourth international conference on parallel problem solving from nature (PPSN IV)* (pp. 960–969).
Yuan, Y., & Xu, H. (2013). An integrated search heuristic for large-scale flexible jobshop scheduling problems. *Computers & Operations Research*, *40*, 2864–2877.