# Genetic Tabu Search for the Fuzzy Flexible Job Shop Problem

Juan José Palacios[1], Miguel A. González[1], Camino R. Vela[1], Inés González-Rodríguez[b], Jorge Puente[1]

[a]*Department of Computing, University of Oviedo, (Spain)*
[b]*Dept. of Mathematics, Statistics and Computing, University of Cantabria, (Spain)*

**Abstract**

This paper tackles the flexible job-shop scheduling problem with uncertain processing times. The uncertainty in processing times is represented by means of fuzzy numbers, hence the name fuzzy flexible job-shop scheduling. We propose an effective genetic algorithm hybridised with tabu search and heuristic seeding to minimise the total time needed to complete all jobs, known as makespan. To build a high-quality and diverse set of initial solutions we introduce a heuristic method which benefits from the flexible nature of the problem. This initial population will be the starting point for the genetic algorithm, which then applies tabu search to every generated chromosome. The tabu search algorithm relies on a neighbourhood structure that is proposed and analysed in this paper; in particular, some interesting properties are proved, such as feasibility and connectivity. Additionally, we incorporate a filtering mechanism to reduce the neighbourhood size and a method that allows to speed-up the evaluation of new chromosomes. To assess the performance of the resulting method and compare it with the state-of-the-art, we present an extensive computational study on a benchmark with 205 instances, considering both deterministic and fuzzy instances to enhance the significance of the study. The results of these experiments clearly show that not only does the hybrid algorithm benefit from the synergy among its components but it is also quite competitive with the state-of-the-art when solving both crisp and fuzzy instances, providing new best-known solutions for a number of these test instances.

*Keywords:* Genetic Algorithms, Neighbourhood Structure, Local Search, Heuristics, Flexible Job Shop Scheduling, Fuzzy Processing Times

## 1. Introduction

Scheduling operations is one of the most critical issues in manufacturing and production systems as well as in information processing environments [1]. The Job-shop Scheduling Problem (*JSP*) is a simplified model of many problems in this class which has interested researchers for decades due to its simple formulation but, at the same time, high difficulty, being NP-hard [2]. In the classical *JSP* a set of jobs have to be processed on a set of machines, each job consisting of a sequence of consecutive operations and each operation requiring the exclusive use of exactly one machine during all its processing time, which is perfectly known in advance. A typical performance indicator is the makespan, i.e., the time required to complete all jobs.

Unfortunately, the classical *JSP* cannot model many practical situations due to the fact that project decisions usually have to be made in advance, when activity durations are still highly uncertain. A great variety of approaches have been considered to deal with these real-life situations, as can be seen in the review of fundamental approaches for scheduling under uncertainty from [3]. Maybe, the best-known approach is stochastic scheduling, where uncertain processing times are taken to be stochastic variables. A recent example can be found in [4], where a stochastic programming approach for the project scheduling is proposed. Here, the uncertainty of the durations is represented using a set of discrete scenarios in which each scenario has a probability of occurrence. The durations of activities are random variables which are supposed to be independent and for which the individual distributions can be estimated. More recently, in [5] a method for solving the resource constrained project scheduling problem with uncertain activity durations is given, where uncertain durations are described by independent random variables with a known probability distribution function. However, it is sometimes the case that probability distributions underlying durations are unknown and there is a lack of statistical data to validate the choice of duration distributions. It may even be argued that probability distributions allow us to model the variability of repetitive tasks, but not uncertainty due to a lack of information [6]. Even when durations are independent random variables it is admitted that estimating the makespan distribution is, in general, intractable [7]. An alternative and complementary approach to modelling ill-known processing times is to use fuzzy numbers or, more generally, fuzzy intervals in the setting of possibility theory. Fuzzy intervals share some of the disadvantages of probability theory, in particular the need of providing the possibility distribution that represents ill-known durations. However for, say, triangular fuzzy numbers the expert need only provide an interval of possible values and the most typical value, which is usually easier than accurately defining a probability distribution. Quantitative

*Email addresses:* palaciosjuan@uniovi.es (Juan José Palacios), mig@uniovi.es (Miguel A. González), crvela@uniovi.es (Camino R. Vela), gonzalezri@unican.es (Inés González-Rodríguez), puente@uniovi.es (Jorge Puente)

possibility theory is said to provide a natural framework, simpler and less data-demanding than probability theory, for handling incomplete knowledge about scheduling data. The fuzzy approach has been around for more than two decades and has received the attention of several researchers (c.f. [8],[9]). In particular, considerable effort has been made to solve the *fuzzy JSP* (*FJSP*), where task durations are modelled as fuzzy numbers (most commonly, triangular fuzzy numbers). Some of the existing approaches will be reviewed in Section 2.

Another characteristic of real-world problems is flexibility, which is contemplated in the *flexible JSP* (*fJSP* in short), a variant of the *JSP* where multiple machines can perform the same operation (possibly with different processing times). This flexibility allows the system to absorb changes in the demand of work or in the performance of the machines. On the other hand, it also increases the difficulty of the problem, since a solution must also consider the assignment of jobs to machines (job routings) in addition to scheduling operations on the machines.

Fuzzy processing times and flexibility on the machines can be considered simultaneously, as done for example in [10]. When this is the case, we have the *fuzzy flexible job-shop scheduling problem* (*FfJSP*). This will be the problem considered in this paper, with the objective of minimising the makespan.

As solving method, we propose to design a hybrid algorithm combining a genetic algorithm with a local search strategy. This is motivated by the success of this hybridisation not just for solving JSP ([11]) but also for solving several extensions of it such as *JSP* with setup times [12], *FJSP* [13] or *fJSP* [14]. It is not possible however to directly apply these existing methods to *FfJSP*, because the addition of both flexibility and fuzzy processing times to the problem changes its nature, and therefore well-known results, both theoretical and empirical, regarding existing neighbourhood structures are not longer applicable in the new setting of *FfJSP*. We need new neighbourhood structures specific for this problem, with the corresponding study of their properties. The benefit of having well-founded neighbourhood structures is beyond their use in our local search strategy, since this allows to incorporate them to any search method based on neighbourhoods or, if connectivity holds, they could also be used, for instance, as a branching scheme in an exact search method. Finally, although the use of heuristic strategies to generate the initial population is less frequent in the literature, there are also authors that have proved its efficacy in *fJSP* [15].

We shall propose an efficient hybrid algorithm which combines a memetic algorithm with a heuristic strategy to generate initial solutions. The initialisation strategy exploits the flexibility on the machine assignment to build a varied set of high-quality solutions. The memetic algorithm itself combines a genetic algorithm with tabu search, inspired in the method presented in [14] to solve the flexible job-shop scheduling problem with setup times. The tabu search relies on exploring both moves in machine assignments and in processing orders of critical operations. We propose two new neighbourhood structures for the local search. For the first structure, we shall prove that it verifies both feasibility and connectivity properties, the latter ensuring asymptotic convergence in probability to a global optimal solution. The second neighbourhood is obtained by incorporating a filtering mechanism that trims the first structure by discarding non-improving neighbours, keeping feasibility and considerably reducing the size of the set of neighbours at the cost of losing connectivity. Additionally, a method based on constraint propagation is introduced that allows to speed-up the evaluation of new chromosomes. An extensive computational study will show that our algorithm outperforms existing methods from the literature for the same problem, while it gives results comparable to those of the best available algorithms for the flexible job shop with deterministic processing times.

The remainder of this paper is organised as follows: Section 2 reviews the literature on job-shop scheduling with flexibility and with uncertainty in operation processing times. Section 3 is devoted to the problem formulation while Section 4 describes the proposed algorithm, including formal proofs of the properties of the neighbourhood structure. In Section 5, we report and analyse the results of the experimental study. Finally, in Section 6, some conclusions are given.

## 2. Related work

Hybrid metaheuristics are classical methods for solving combinatorial optimisation problems due to the fact that they allow algorithm designers to combine different search techniques and benefit from their synergy. In particular, they have a long track of success with scheduling problems. Even for the classical *JSP*, researchers continually propose new algorithms designed from different metaheuristics which outperform or at least are comparable to previous ones. Indeed, the algorithms proposed in [16] and [17] are probably the most efficient approaches to the *JSP* with makespan minimisation and both combine the $i-$TSAB algorithm from [18] with other existing methods: a simulated annealing algorithm in the first case and the solution-guided search method in the second. More recently, a hybrid genetic tabu search "with innovative initial solutions" is proposed in [19] which not only solves several benchmark problems optimally but also demonstrates to be capable of solving real-life job shop problems.

Regarding the *FJSP*, several metaheuristics have been proposed since the 1990s, starting with the simulated annealing method from [20]. In [21], the authors develop a GA to maximise several objectives in a fuzzy decision making framework. This GA is later improved in [22] using random keys. In [23], a particle swarm optimisation algorithm is combined with some genetic operators. In [24], a GA that searches in the so-called space of possibly active schedules is proposed and a semantics for fuzzy schedules is provided. In [13], we find a hybrid algorithm which combines a GA with a very efficient local search method. More recently, we find a great variety of nature-inspired methods for makespan minimisation: a swarm based neighbourhood search algorithm [25], a hybrid algorithm, combining particle swarm optimisation with tabu search [26] and an artificial bee colony algorithm [27].

It is also in the 1990s that flexibility in *JSP* was first addressed by researchers, after the seminal paper [28], and has

ever since been the object of intensive research. From the first works, such as [29], where the machine assignment and the scheduling of operations are studied separately, until now, many are the approaches proposed for the *fJSP*. Among others, a tabu search algorithm is proposed in [30] and is later improved with two neighbourhood structures in [31]. [15] presents a GA that incorporates different strategies for generating the initial population while a hybrid genetic algorithm combined with a variable neighbourhood descent search is given in [11]. More recently, approaches such as the discrepancy search proposed in [32], the hybrid harmony search and large neighbourhood search from [33] or the genetic algorithm combined with tabu search from [14] obtain the best results so far for many problem instances.

Compared to the *FJSP* and the *fJSP*, the combination of flexibility and uncertainty in *FfJSP* has received limited albeit increasing attention. Among the most representative proposals, we mention the genetic algorithm from [34], the hybrid artificial bee colony algorithm given in [35], the estimation distribution algorithm from [36], the co-evolutionary algorithm from [37] or the swarm-based neighbourhood search algorithm from [38]. These last four algorithms are, to the best to our knowledge, the most competitive methods in the literature for this problem.

## 3. Problem formulation

In the job shop scheduling problem, there is 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. There are also *capacity constraints*, whereby each operation $o_{ij}$ requires the uninterrupted and exclusive use of one of the machines for its whole processing time.

When flexibility is added to the *JSP*, an operation $o_{ij}$ is allowed to be executed on one machine out a given set $M(o_{ij})$. The processing time of the operation $o_{ij}$ on machine $M_k \in M(o_{ij})$ is denoted $p_{o_{ijk}} \in \mathbb{N}$. 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. A solution to this problem consists of an assignment to machines of all $N = \sum_{i=1}^{n} N_i$ operations in the set $O = \cup_{1 \leq i \leq n} O_i$ together with a schedule, i.e., an allocation of starting times for each operation on the assigned machine, which is *feasible* (i.e. all constraints hold). The objective is to find an optimal solution according to some criterion, most commonly that the *makespan*, which is the completion time of the last task to finish, is minimal.

Finally, in the fuzzy flexible job shop, processing times $p_{o_{ijk}}$ are allowed to be fuzzy numbers (a particular case of which are natural numbers), modelling the existing uncertainty regarding the exact duration of an operation on a particular machine.

### 3.1. Uncertain durations

In real-life applications, it is often the case that the exact processing time of operations is not known in advance. However, based on previous experience, an expert may be able to provide some qualitative information about what duration is more plausible than another, estimating for instance an interval of possible values for the processing time or its most typical value, and he/she may even be able to assess whether some values in the interval appear to be more plausible than others. This naturally leads to modelling such durations using fuzzy intervals or fuzzy numbers (see [20] and references therein for practical ways of eliciting fuzzy intervals). Fuzzy intervals have been extensively studied in the literature (cf. [39]). A *fuzzy interval* $A$ is a fuzzy set on the reals (with membership function $\mu_A : \mathbb{R} \to [0,1]$) such that its $\alpha$-*cuts* $A_\alpha = \{u \in \mathbb{R} : \mu_A(u) \geq \alpha\}$, $\alpha \in (0,1]$, are intervals. A fuzzy interval is a *fuzzy number* if its $\alpha$-cuts (denoted $[\underline{n}_\alpha, \overline{n}_\alpha]$) are closed, its *support* $A_0 = \{u \in \mathbb{R} : \mu_A(u) > 0\}$ is compact (closed and bounded) and there is a unique modal value $u*$, $\mu_A(u*) = 1$. Clearly, real numbers can be seen as a particular case of fuzzy ones.

The simplest model of fuzzy interval is a *triangular fuzzy number* or *TFN*, using an interval $[a^1, a^3]$ of possible values and a modal value $a^2$ in it. For a TFN $A$, denoted $A = (a^1, a^2, a^3)$, the membership function takes the following triangular shape:

$$\mu_A(x) = \begin{cases} \frac{x-a^1}{a^2-a^1} & : a^1 \leq x \leq a^2 \\ \frac{x-a^3}{a^2-a^3} & : a^2 < x \leq a^3 \\ 0 & : x < a^1 \text{ or } a^3 < x \end{cases} \quad (1)$$

If TFNs are to be used to extend the flexible job shop to handle uncertainty, two issues must be addressed: the precise meaning of "minimal makespan" when such makespan is a TFN as well as the means of extending the arithmetic operations of addition and maximum to work with TFNs.

The fact that there is no natural total ordering in the set of TFNs makes the concept "minimal makespan" ambiguous. In the literature on fuzzy job shop two main approaches to defining a total ordering co-exist.

The membership function $\mu_Q$ of a fuzzy quantity $Q$ can be interpreted as a possibility distribution on the real numbers; this allows to define the *expected value* of a fuzzy quantity [40], given for a TFN $A$ by

$$E[A] = \frac{1}{4}(a^1 + 2a^2 + a^3). \quad (2)$$

It induces a total ordering $\leq_E$ in the set of fuzzy intervals [20], where for any two fuzzy intervals $M, N$ $M \leq_E N$ if and only if $E[M] \leq E[N]$. The expected value coincides with the *neutral scalar substitute* of a fuzzy interval and can also be obtained as the centre of gravity of its *mean value* or using the *area compensation* method, which calculates areas under the membership function with an interpretation in terms of imprecise probabilities [8]. Clearly, for any two TFNs $A$ and $B$, if $\forall i, a^i \leq b^i$, then $A \leq_E B$.

Related to this is a ranking method widely used in the fuzzy scheduling literature following the seminal papers of Sakawa et al. [41],[21]. It is based on using multiple numerical indices for ranking fuzzy numbers, as suggested in [42]. In particular, for any TFN $A$, three indices are considered: $c_1(A) = E[A]$, $c_2(A) = a^2$ and $c_3(A) = a_3 - a_1$. Then, $A <_R B$ if $c_1(A) < c_1(B)$ or else

if $c_1(A) = c_1(B)$ and $c_2(A) < c_2(B)$ or else if $c_1(A) = c_1(B)$, $c_2(A) = c_2(B)$ and $c_3(A) < c_3(B)$. Obviously, if $A <_E B$, it is also the case $A <_R B$.

In the fuzzy flexible job shop, we essentially need two operations on fuzzy numbers, the sum and the maximum. These are obtained by extending the corresponding operations on real numbers using the *Extension Principle*. However, computing the resulting expression is cumbersome, if not intractable. For the sake of simplicity and tractability of numerical calculations, it is fairly common in the literature, following [20], to approximate the results of these operations by interpolation, evaluating only the operation on the three defining points of each TFN. It turns out that the sum and its approximation coincide, so for any pair of TFNs $A$ and $B$:

$$A + B = (a^1 + b^1, a^2 + b^2, a^3 + b^3). \tag{3}$$

Regarding the maximum, for any two TFNs $A, B$, if $F$ denotes their maximum and $G = max_I(A,B) = (\max\{a^1,b^1\}, \max\{a^2,b^2\}, \max\{a^3,b^3\})$ its approximated value, it holds that:

$$\forall \alpha \in [0,1], \quad \underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha. \tag{4}$$

where $[\underline{f}_\alpha, \overline{f}_\alpha]$ is the $\alpha$-cut of $F$. In particular, $F$ and $G$ have identical support and modal value, that is, $F_0 = G_0$ and $F_1 = G_1$. This approximation has been widely used in the scheduling literature, among others, in [43], [44],[20],[24],[45],[46],[47], [23],[48], [21] or [49].

More recently, it has been proposed in [50] to approximate the maximum using the above ranking method, so $\max(A,B) \approx \max_R(A,B)$ where $\max_R = A$ if $A <_R B$ and $\max_R(A,B) = B$ otherwise. Notice that $\max_R(A,B) \leq_E \max_I(A,B), \forall A,B$ (and therefore $\max_R(A,B) \leq_R \max_I(A,B)$ too). Notice as well that it is not guaranteed that $\max_R$ maintains the support nor the modal value of the actual maximum and, more generally, it is not coherent with the max operation it is meant to approximate. As we shall see in the following, this is of key importance for fuzzy scheduling and it is one of the reasons why we choose to use $\max_I$ instead of $\max_R$. Unless otherwise stated and for the sake of a simpler notation, we shall simply write max when referring to the interpolated maximum $\max_I$.

### *3.2. Solution graph and criticality*

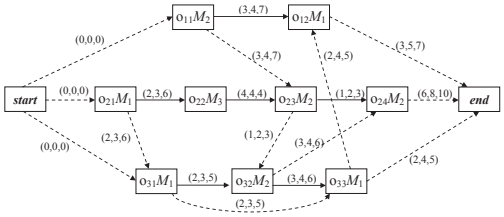
A solution to the *FfJSP* may be alternatively viewed as a pair $(\alpha,\pi)$ where $\alpha$ is a feasible assignment of each operation $o_{ij} \in O$ to a machine $M_k \in M(o_{ij})$, denoted $\alpha(o_{ij}) = k$, and $\pi$ is a processing order of the operations on all the machines in $M$ (i.e., a machine sequence) compatible with the job and the machine sequences that may be represented by a topological ordering. For an operation $o_{ij} \in O$ 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 $(\alpha,\pi)$. The starting and completion times of $o_{ij}$, denoted $S_{o_{ij}}$ and $C_{o_{ij}}$ respectively, can be calculated as $S_{o_{ij}} = \max(C_{PJ_{o_{ij}}}, C_{PM_{o_{ij}}})$ and $C_{o_{ij}} = S_{o_{ij}} + p_{o_{ij}k}$, where $k = \alpha(o_{ij})$. The objective is to find a solution $(\alpha,\pi)$ that minimises the makespan, i.e., the completion time of the last operation to end, denoted as $C_{max}(\alpha,\pi) = \max_{o_{ij} \in O} C_{o_{ij}}$.

Since operation processing times are fuzzy intervals, the addition and maximum operations used to propagate constraints are taken to be the corresponding operations on fuzzy intervals, approximated for the particular case of TFNs as explained above. The obtained schedule will be a fuzzy schedule in the sense that the starting and completion times of all operations and the makespan are fuzzy intervals, interpreted as possibility distributions on the values that the times may take. However, the machine assignment $\alpha$ and the operation processing ordering $\pi$ that determine the schedule are crisp; there is no uncertainty regarding the order and the machines in which operations are to be processed.

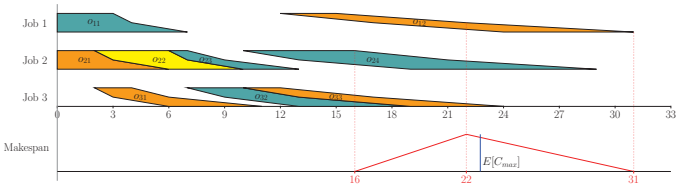
The fuzzy schedule is therefore a predictive schedule, given before the actual project execution. When the schedule is actually executed and operations are processed according to the ordering and machine assignment given by $(\alpha,\pi)$, their processing times will no longer be uncertain and will take precise values in the interval given by the original TFNs. Thanks to the coherence of the approximated maximum $\max_I$, we can be sure that all starting and completion times (in particular, the makespan) will lie within the support of the predicted fuzzy times. In particular, we can be sure that if the fuzzy makespan is $C_{max} = (C^1_{max}, C^2_{max}, C^3_{max})$, all possible executions of $(\alpha,\pi)$ will have a crisp makespan in the interval $[C^1_{max}, C^3_{max}]$ (being more likely those values around $C^2_{max}$). This is not the case with $\max_R$: a fuzzy schedule computed using $\max_R$ may predict a fuzzy makespan which has no correspondence with the crisp makespan obtained when the schedule is later executed.

Based on the above, we propose a solution graph model which extends that from [51] to incorporate machine flexibility. According to this model, a machine assignment $\alpha$ and a feasible operation processing order $\pi$ can be represented by an acyclic directed graph $G(\alpha,\pi) = (V, A \cup R(\alpha,\pi))$ where each node $x$ in $V$ represents either an operation of the problem, labelled with the machine to which it has been assigned $M_k$, $k = \alpha(x)$, or one of the dummy nodes *start* and *end*, which are fictitious operations with processing time 0. Arcs in $A$ represent job processing orders and the set $R(\alpha,\pi)$ is partitioned into subsets $R_k$, where $R_k$ is a minimal set of arcs representing the processing order given by $\pi$ for all operations assigned by $\alpha$ to the machine $M_k$. Each arc is weighted with the processing time (a TFN in our case) of the operation at the source node in the machine where it will be processed.

To illustrate previous concepts, Figure 1 shows a solution graph and a Gantt chart (adapted to TFNs following [20]) where $\pi = \{o_{21}, o_{11}, o_{22}, o_{31}, o_{23}, o_{32}, o_{33}, o_{24}, o_{12}\}$ and the assignment $\alpha$ is explicit in the label of each node. In accordance with graph model, there is a node for each operation together with the dummy nodes *start* and *end*. Solid arcs represent job processing orders while dotted arcs represent machine processing orders. Each arc is labelled with the processing time (the TFN) of the source node. In this example, the processing order for operations in $M_1$ is $o_{21}$, $o_{31}$, $o_{33}$; the processing order for operations in machine $M_2$ is $o_{11}, o_{23}, o_{32}, o_{34}$ and, finally, only operation $o_{22}$ is to be processed in machine $M_3$. On the right-hand




(a) Solution graph




(b) Gantt chart (Job-oriented)

Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines. The makespan is (16, 22, 31).

side of Figure 1, the Gantt chart represents the corresponding partial schedules on each job. For instance, the fuzzy time gap when operation $o_{23}$ is being processed corresponds to the green coloured polygon labelled $o_{23}$. This polygon is delimited on the left by the starting time (6, 7, 10) and on the right by the completion time (7, 9, 13); notice that in the case that starting and completion times were real numbers, the polygon would become a rectangle, which is the standard way of representing operation execution times in deterministic Gantt charts. Additionally, the fuzzy makespan and its expected value is depicted below the job partial schedules, making it possible to appreciate which operation contributes to each component of the makespan.

The way to confront criticality in the fuzzy framework is it not unique. Here we adopt the definition from [51] where, given a solution graph $G(\alpha,\pi)$, three *parallel solution graphs* $G^i(\alpha,\pi)$, $i = 1,2,3$, are defined with identical structure to $G(\alpha,\pi)$ but where the cost of any arc $(x,y)$ is $p^i_{xk}$, the $i$-th component of $p_{xk}$, for $k = \alpha(x)$. Since durations in each parallel graph $G^i(\alpha,\pi)$ are deterministic, a critical path in $G^i(\alpha,\pi)$ is the longest path from node *start* to node *end*. The set of *critical* paths in $G(\alpha,\pi)$ is then defined as the union of critical paths in $G^i(\alpha,\pi)$, $i = 1,2,3$. Nodes and arcs in a critical path are also termed critical. A critical path is naturally decomposed into critical blocks $B_1,\ldots,B_r$, where a *critical block* is a maximal subsequence of tasks in a critical path requiring the same machine and such that two consecutive operations of the block do not belong to the same job. Notice that the makespan of the schedule is not necessarily the cost of a critical path, but each component $C^i_{max}(\alpha,\pi)$ is the cost of a critical path in the corresponding solution parallel graph $G^i(\alpha,\pi)$. This will prove an important point when defining the neighbourhood structure in Section 4.4.

## 4. The hybrid algorithm

The long record of good results obtained with hybrid methods that combine genetic algorithms (GA) and different local search methods, in particular Tabu Search (TS), supports the choice of this kind of metaheuristic [11],[13],[10]. It is well-known that a key component for the success of a TS algorithm is the neighbourhood structure used in it. Here we propose a neighbourhood structure for the *FfJSP* that fulfills two important properties: feasibility and connectivity. We also incorporate new mechanisms to speed-up the evaluation of the individuals and a neighbour filter that allows the algorithm to discard a great number of non-improving ones, thus reducing the size of the neighbourhood and increasing the chance for improvement. Filtering is particularly important in the fuzzy framework as the number of feasible neighbours of a solution is considerably larger than in the crisp case. Regarding the initial population, a wise generation of heuristic solutions can help to improve the convergence of the memetic algorithm compared to starting from an initial population composed by only random solutions [15]. This is especially interesting when dealing with large instances ([52]).

The main steps of this hybrid algorithm are the following. In the first step the initial population is generated by the heuristic algorithm *(FfInsertion)* described below. Then the genetic algorithm iterates over a number of generations. In each iteration, a new generation is built from the previous one by applying the usual genetic operators. Tabu search is applied to every schedule produced either by the initialisation heuristic algorithm or by the GA; the corresponding chromosome is rebuilt from the improved schedule obtained by TS so its characteristics can be transferred to the subsequent offsprings (effect known as Lamarckian evolution). The flow chart of the resulting hybrid algorithm can be seen in Figure 2.

In the following, we describe in more detail these components: the genetic algorithm, the heuristic seeding strategy, and the tabu search algorithm, including the neighbourhood structures and the makespan estimation procedure.

### 4.1. Genetic algorithm

We consider here a GA previously used in [14] for tackling other variants of the *JSP* and extend it to the *FfJSP*. The main characteristics of this GA are the following. In the first step, the initial population (obtained either randomly or by some heuristic procedure) is evaluated. Then the GA iterates over a number of generations. In each iteration a new generation is built from the previous one by applying the genetic operators of selection, recombination and replacement. In the selection phase all chromosomes are randomly grouped into pairs, and then each one of these pairs is mated to obtain two offspring. Finally, the replacement is carried out as a tournament selection from each pair of parents and their two offspring. This algorithm differs

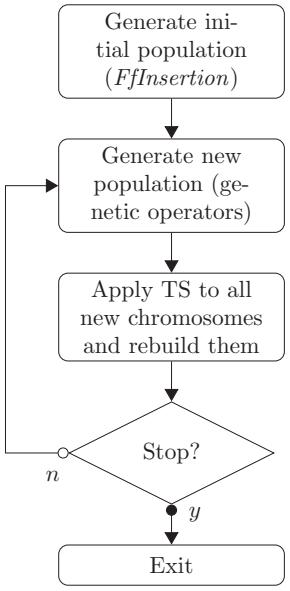



Figure 2: Flow chart of the hybrid algorithm.

slightly from the classic genetic algorithms in that the selective pressure is introduced in the replacement instead of in the selection phase.

To codify chromosomes we have chosen the two-vector representation [11], which is widely used in the flexible job-shop problem and its fuzzy version with slight differences. This encoding is quite natural because the *fJSP* is a combination of machine assignment and operation scheduling decisions, so a solution can be expressed by two vectors $v_1$ and $v_2$, $v_1$ representing the machines assigned to the operations and $v_2$ representing the processing sequences of operations on the machines.

The operation-sequence vector is based on permutations with repetition for the *JSP* [53]. It is a permutation of the set of operations, each being represented by its job number. For example, if we have a problem with 3 jobs: $J_1 = \{o_{11}, o_{12}\}$, $J_2 = \{o_{21}, o_{22}, o_{23}, o_{24}\}$, $J_3 = \{o_{31}, o_{32}, o_{33}\}$, then the sequence $v_2 =$ (2 1 2 3 2 3 3 2 1) is a valid vector that represents the topological order $\pi = \{o_{21}, o_{11}, o_{22}, o_{31}, o_{23}, o_{32}, o_{33}, o_{24}, o_{12}\}$. With this encoding, every permutation produces a feasible processing order.

Regarding the machine-assignment vector, at a given position it has the number of the machine assigned to the operation located at the same position in the operation-sequence vector. For example, if we consider the sequence vector above, then the machine vector $v_1 =$ (1 2 3 1 2 2 1 2 1), indicates that the operations $o_{21}$, $o_{31}$, $o_{33}$ and $o_{12}$ use the machine $M_1$, the operations $o_{11}$, $o_{23}$, $o_{32}$ and $o_{24}$ use the machine $M_2$, and only the operation $o_{22}$ uses the machine $M_3$ (that is, $\alpha(o_{21}) = \alpha(o_{31}) = \alpha(o_{33}) = \alpha(o_{12}) = 1$, $\alpha(o_{11}) = \alpha(o_{23}) = \alpha(o_{32}) = \alpha(o_{24}) = 2$ and $\alpha(o_{22}) = 3$).

For chromosome mating, the genetic algorithm uses an extension of the well known Job Order Crossover (JOX). Given two parents, JOX selects a random subset of jobs and copies their genes to one offspring in the same positions as in the first parent, then the remaining genes are taken from the second parent so that they maintain their relative ordering. To create the second offspring, the parents change their roles. In order to extend this operator to the flexible case, we also need to consider the machine-assignment vector. We propose to choose for every operation the corresponding assignment in the parent it comes from. For instance, let us consider the following two parents:

| | Assignment | Sequence |
|---|---|---|
| Parent1 | (**1** 2 **3** 1 **2** 2 1 **2** 1) | (**2** 1 **2** 3 **2** 1 3 **2** 3) |
| Parent2 | (3 2 3 1 3 2 1 3 3) | (1 1 2 2 3 3 2 2 3) |

Assuming that the selected subset of jobs (in bold) includes only job 2, then

| | Assignment | Sequence |
|---|---|---|
| Offspring1 | (**1** 3 **3** 2 **2** 3 2 **2** 3) | (**2** 1 **2** 1 **2** 3 3 **2** 3) |
| Offspring2 | (2 1 3 1 2 1 1 3 1) | (1 3 2 2 1 3 2 2 3) |

The operator JOX may swap any two operations requiring the same machine; this is an implicit mutation effect. This is the reason we have chosen not to use any explicit mutation operator. In consequence, parameter setting in the experimental study is considerably simplified, because crossover probability is set to 1 and mutation probability needs not be specified. With this setting, we have obtained results similar to those obtained with a lower crossover probability and a low probability of applying mutation operators. Also, some authors, for example [54] or [55], have already noticed that a mutation operator does not play a relevant role in a genetic algorithm hybridised with local search.

To evaluate chromosomes, we need to generate schedules, obtaining their makespan and compute the expected value thereof (this constitutes the fitness value). To do so, we have used a simple decoding algorithm: operations are scheduled in the machines given by the machine-assignment vector at the earliest possible instant that maintains the order in which they appear in the operation-sequence vector of the chromosome. In other words, we produce a possibly semiactive schedule, which means that the possibility of that no operation can start earlier without altering the operation sequence for a given machine assignment is 1.

### *4.2. Heuristic seeding*

To generate initial solutions we schedule operations in an insertion mode but taking advantage of the flexibility. Let $\Omega$ denote the set of operations that can be scheduled at the current stage (initially, this set contains the first operation from each job). We select a random operation $o_{ij}$ in $\Omega$ and compute its earliest completion time, $C^\star$, considering all the machines where it can be processed. Then, we randomly select a machine $M_k \in M(o_{ij})$ in which $o_{ij}$ may finish at $C^\star$ and schedule $o_{ij}$ in machine $M_k$ at its earliest starting time, given by $C^\star - p_{o_{ij}k}$. $o_{ij}$ is removed from $\Omega$ and its successor in the job sequence is added to $\Omega$, provided that it exists. The process finishes when $\Omega$ becomes empty.

**Input** A *FfJSP* instance
**Output** An operation processing order and a machine assignment which determines a schedule
$\Omega \leftarrow \{o_{i1}, 1 \leq i \leq n\}$;
**while** $\Omega \neq \emptyset$ **do**
$o_{ij} \leftarrow$ an operation selected at random from $\Omega$;
**for** each $M_k \in M(o_{ij})$ **do**
Compute $EST_{o_{ijk}}$;
$C^\star \leftarrow \min\{EST_{o_{ijk}} + p_{o_{ijk}}, M_k \in M(o_{ij})\}$;
$K \leftarrow \{M_k \in M(o_{ij}), EST_{o_{ijk}} + p_{o_{ijk}} = C^\star\}$;
Choose a machine $M_{k^\star}$ from $K$ at random;
Schedule the operation $o_{ij}$ in machine $M_{k^\star}$; {fix the value of $S_{o_{ij}} = EST_{o_{ijk^\star}}$};
$\Omega \leftarrow \Omega - \{o_{ij}\}$
**if** $j$ is not the last operation of job $i$ **then**
$\Omega \leftarrow \Omega \cup \{o_{i(j+1)}\}$;
Build the sequence and the assignment vectors according to the created schedule;
**return** The schedule $S$ given by $\{S_{o_{ij}} : 1 \leq i \leq n, 1 \leq j \leq N_i\}$ and the sequence and assignment vectors

**Alg. 1:** The *FfInsertion*

To understand what is an insertion mode and indeed how $C^\star$ is computed, we define a *feasible insertion interval* for a operation $o_{ij}$ in a machine $M_k \in M(o_{ij})$ to be a time interval $[t_k^S, t_k^E]$ in which machine $M_k$ is idle and such that $o_{ij}$ can be processed within that time interval without violating precedence constraints, that is, $t_k^S + p_{o_{ijk}} \leq t_k^E$, and $t_k^S \geq C_{o_{i(j-1)}}$ (if $j = 0$, $C_{o_{i(j-1)}}$ is taken to be 0). Then, the *earliest starting time* for operation $o_{ij}$ in machine $M_k$, denoted $EST_{o_{ijk}}$, is the smallest $t_k^S$ that can be found. Thus, the earliest completion time for $o_{ij}$ is $C^\star = \min\{EST_{o_{ijk}} + p_{o_{ijk}}, M_k \in M(o_{ij})\}$. We schedule $o_{ij}$ in any machine $M_k$ such that $EST_{o_{ijk}} + p_{o_{ijk}} = C^\star$.

The pseudocode description of this fuzzy flexible insertion algorithm *(FfInsertion)* is shown in Algorithm 1.

### 4.3. Tabu search

Tabu search (TS) is an advanced local search technique, proposed in [56] and [57], which may select non-improving neighbours in order to escape from local optima. To avoid revisiting recently visited solutions and so to promote the exploration of new promising regions of the search space, it maintains a tabu list with a set of moves which are not allowed when generating new neighbourhoods. TS has a solid record of good empirical performance, often used in combination with other metaheuristics. In particular, as already mentioned in Section 2, the *i*-TSAB algorithm is the basis for two of the state-of-the-art approaches to *JSP*.

The general scheme of TS algorithm used herein is similar to other TS algorithms proposed in the literature, for instance in [58]. In the first step the initial solution, generated by the GA, is evaluated and it then iterates for a number of steps. At each iteration, the neighbourhood of the current solution is calculated and one of the neighbours is selected as new solution. Neighbours are evaluated using a makespan estimate, so the selection criterion is based on selecting the neighbour with lowest expected value of estimated makespan. A neighbour is tabu if it is generated by reversing a tabu arc or by assigning a tabu machine to an operation, unless its estimated expected makespan is better than that of the current best solution. Additionally, we use the dynamic length schema for the tabu list and the cycle checking mechanism as they were proposed in [58]. TS finishes after a number of iterations without improvement, returning the best solution found so far.

Two key points of this TS algorithm are the definition of the neighbourhood structure and the method used to estimate the neighbour's makespan. The next two subsections describe, respectively, new neighbourhood structures for *FfJSP* (including some properties thereof) and the procedure for makespan estimation.

### 4.4. Neighbourhood structure

Clearly, a central element in any local search procedure is the definition of neighbourhood. For the crisp job shop, a well-known neighbourhood, which relies on the concepts of critical path and critical block, is that proposed in [59], later extended to the fuzzy case in [51] using the given definition of criticality. In this structure, given a operation processing order, $\pi$, the neighbourhood of $\pi$ is the set of operation processing orders obtained from $\pi$ by reversing single critical arcs. Adding flexibility to the problem requires considering the assignment of machines to operations as well. We thus propose to extend the neighbourhood from [51] to consider also all the moves that result from changing the machine assignment of a single critical operation. The resulting neighbourhood is termed $N^{AP}$ and it is obtained as the union of other two, termed $N^A$ and $N^P$. $N^{AP}$ is quite similar in its motivation and definition to the structure proposed in [14] for the *SDST-fJSP*; however, we shall see that the different nature of the *FfJSP* results in significant differences, for instance, conditions to discard unfeasible neighbours are no longer necessary.

**Definition 1.** (Neighbourhood $N^A$) Let $\alpha$ be a machine assignment, $\pi$ a feasible operation processing order, $x$ an operation and $k' \in M(x)$ a machine such that $k' \neq \alpha(x)$. Let $\alpha_{<x,k'>}$ denote the assignment obtained from $\alpha$ after reassigning operation $x$ to machine $k'$. The neighbourhood structure $N^A$ obtained from $\alpha$ is defined as: $N^A(\alpha,\pi) = \{(\alpha_{<x,k'>}, \pi) : x \text{ is critical}, k' \in M(x), k' \neq \alpha(x)\}$.

**Definition 2.** (Neighbourhood $N^P$) Let $\alpha$ be a machine assignment and $\pi$ a feasible operation processing order. Given an arc $v = (x,y) \in R(\alpha,\pi)$, let $\pi_{(v)}$ denote the processing order obtained from $\pi$ after reversing arc $v$ in $G(\alpha,\pi)$. The neighbourhood structure obtained from $\pi$, $N^P$, is given by $N^P(\alpha,\pi) = \{(\alpha,\pi_{(v)}) : v \in R(\alpha,\pi) \text{ is in a critical block}\}$.

For a fixed assignment this neighbourhood coincides with that defined in [51] for the *FJSP*.

Notice that while $N^A$ concerns both an operation $x$ and a machine $k'$, $N^P$ concerns the arc formed by a pair of operations $v = (x,y)$.

**Definition 3.** $N^{AP}(\alpha,\pi) = N^A(\alpha,\pi) \cup N^P(\alpha,\pi)$.

According to the following property it makes sense in the above definitions to discard non-critical arcs or operations.

**Proposition 1.** *Let $\alpha$ be a machine assignment, $\pi$ a feasible processing order, $\beta = \alpha_{<x,k'>}$ and $\sigma = \pi_{(v)}$ where x and v are not critical in $G(\alpha,\pi)$. Then*

$$\begin{aligned} &\forall i, \quad C^i_{max}(\alpha,\pi) \leq C^i_{max}(\alpha,\sigma). \\ &\forall i, \quad C^i_{max}(\alpha,\pi) \leq C^i_{max}(\beta,\pi). \end{aligned} \tag{5}$$

This property follows immediately from the definition of critical arcs and activities.

In addition, the neighbourhood $N^{AP}$ has two highly desirable properties: feasibility and connectivity, which are stated in the following two theorems.

**Theorem 1.** *Let $\alpha$ be a machine assignment and let $\pi$ be a feasible operation processing order; then all elements in $N^{AP}(\alpha,\pi)$ are feasible.*

PROOF. Feasibility of neighbours obtained with $N^P$ is proved in [51]. Notice that neighbours in $N^A$ are defined so they maintain the processing order of operations $\pi$, and therefore they are feasible.

This result allows the algorithm to limit the local search to a subspace of feasible operation orders and so it avoids feasibility checking on the neighbours, hence reducing the computational load.

Now, in order to establish the connectivity property, we will follow a similar reasoning as [59] to prove this property for the structure $N$ defined for the classical *JSP*. In our case, the reasoning is more complicated as we have to deal with flexibility and uncertainty. In fact, in absence of these characteristics, $N^{AP}$ is the same as $N$. Hence, we start with the following Lemma.

**Lemma 1.** *Let $(\alpha,\pi)$ be a feasible solution, $G(\alpha,\pi) = (V, A \cup R(\alpha,\pi))$ its disjunctive graph and $(\alpha^*,\pi^*)$ an optimal solution. Let us define*

$$\begin{aligned} W^{(1)}_{\alpha,\pi}(\alpha^*,\pi^*) = \{&v = (x,y) \in R(\alpha,\pi) : \\ &v \text{ is critical in } G(\alpha,\pi), (y,x) \in \overline{R(\alpha^*,\pi^*)}\} \end{aligned} \tag{6}$$

*where $\overline{R(\alpha,\pi)}$ denotes the transitive closure of $R(\alpha,\pi)$,*

$$W^{(2)}_{\alpha,\pi}(\alpha^*,\pi^*) = \{x \in V : x \text{ is critical in } G(\alpha,\pi), \alpha(x) \neq \alpha^*(x)\}; \tag{7}$$

*and*

$$W_{\alpha,\pi}(\alpha^*,\pi^*) = W^{(1)}_{\alpha,\pi}(\alpha^*,\pi^*) \cup W^{(2)}_{\alpha,\pi}(\alpha^*,\pi^*) \tag{8}$$

*i.e., $W_{\alpha,\pi}(\alpha^*,\pi^*)$ is the set of critical arcs $(x,y)$ in $G(\alpha,\pi)$ such that there exists a path from y to x in $R(\alpha^*,\pi^*)$ together with the set of critical operations in $G(\alpha,\pi)$ assigned to a different machine in $\alpha^*$.*

*If holds that, if $(\alpha,\pi)$ is not optimal, then $W_{\alpha,\pi}(\alpha^*,\pi^*) \neq \emptyset$ or equivalently, if $W_{\alpha,\pi}(\alpha^*,\pi^*) = \emptyset$ then $(\alpha,\pi)$ is optimal.*

PROOF. We shall first prove that, if $(\alpha,\pi)$ is not optimal, there is at least a critical arc in $R(\alpha,\pi)$ or a critical operation $x$ such that $\alpha(x) \neq \alpha^*(x)$.

Suppose that there are no critical arcs in $R(\alpha,\pi)$ and that for every critical operation $x$ it holds that $\alpha(x) = \alpha^*(x)$, that means that all critical arcs in $G(\alpha,\pi)$ belong to $A$. Therefore, for all $i$, all critical paths in $G^i(\alpha,\pi)$ belong to $A$. Hence, in each $G^i(\alpha,\pi)$ there exists a critical path where all arcs belong to $A$ and such path is optimal in $G^i(\alpha,\pi)$ (for every $i$, a path where all arcs belong to the same job is a lower bound of $C^i_{max}$) for this assignment $\alpha$. In principle, in the presence of flexibility this does not guarantee the optimality of the solution, as the processing time of operations in the path depend on the machine assignment, which may not be the same as in the optimal solution. However, since we are also supposing that for every critical operation $x$ in $G^i(\alpha,\pi)$ $\alpha(x) = \alpha^*(x)$, we can conclude that $(\alpha,\pi)$ is optimal.

Secondly, we shall prove that if $(\alpha,\pi)$ is not optimal, then there exists some critical arc $v = (x,y)$ such that $(y,x) \in \overline{R(\alpha^*,\pi^*)}$ or there exists a critical operation $x$ such that $\alpha(x) \neq \alpha^*(x)$.

Let us now assume that all critical arcs $v = (x,y) \in R(\alpha,\pi)$ verify that $(x,y) \in \overline{R(\alpha^*,\pi^*)}$ and that all critical operations $x$ in $G(\alpha,\pi)$ it holds that $\alpha(x) = \alpha^*(x)$. The set of critical arcs in $G(\alpha,\pi)$ is the union of the set of critical arcs across all parallel disjunctive graphs. Therefore, the assumption means that for all $i$ all critical arcs in $R^i(\alpha,\pi)$ belong to the transitive closure $\overline{R^i(\alpha^*,\pi^*)}$. Hence, a critical path $P^i$ in $G^i(\alpha,\pi)$ is also a path in $\overline{G^i(\alpha^*,\pi^*)} = (V, A \cup \overline{R^i(\alpha^*,\pi^*)})$. As above, unlike the non-flexible case, the length of $P^i$ in $G^i(\alpha,\pi)$ may be different from its length in $\overline{G^i(\alpha^*,\pi^*)}$ because it depends on the machine assignment, but since we are supposing that for all operations in $P^i$ $\alpha(x) = \alpha^*(x)$, then the length of $P^i$ in $\overline{G^i(\alpha^*,\pi^*)}$ is the same as in $G^i(\alpha,\pi)$. By definition of transitive closure, there is a path in $G^i(\alpha^*,\pi^*)$ with length greater or equal than it and the length of that path is obviously less or equal than the length of a critical path in $G^i(\alpha^*,\pi^*)$. Let $Q^i$ denote an arbitrary critical path in $G^i(\alpha^*,\pi^*)$ and let $\|Q^i\|$ denote its length. It holds that:

$$\forall i, C^i_{max}(\alpha,\pi) = \|P^i\| \leq \|Q^i\| = C^i_{max}(\alpha^*,\pi^*)$$

In consequence, $E[C_{max}(\alpha,\pi)] \leq E[C_{max}(\alpha^*,\pi^*)]$. But, since $(\alpha^*,\pi^*)$ is optimal, it must be the case that $E[C_{max}(\alpha^*,\pi^*)] \leq E[C_{max}(\alpha,\pi)]$, therefore $E[C_{max}(\alpha^*,\pi^*)] = E[C_{max}(\sigma)]$ and $(\alpha,\pi)$ is optimal.

Then, if $(\alpha,\pi)$ is not optimal, either there exists a critical arcs $v = (x,y) \in R(\alpha,\pi)$ verifying that $(x,y) \notin \overline{R(\alpha^*,\pi^*)}$ or there exists a critical operation $x$ such that $\alpha(x) \neq \alpha^*(x)$. In the first case, either $(y,x) \in \overline{R(\alpha^*,\pi^*)}$ or $x$ and $y$ are not related in $\overline{R(\alpha^*,\pi^*)}$, i.e. $\alpha^*(x) \neq \alpha^*(y)$, but given that $\alpha(x) = \alpha(y)$, at least one of them, suppose it is $x$, verifies that $\alpha^*(x) \neq \alpha(x)$.

**Theorem 2.** *$N^{AP}$ verifies the connectivity property, that is, for every non-optimal solution $(\alpha,\pi)$ we may build a finite sequence of transitions of $N^{AP}$ leading from $(\alpha,\pi)$ to a globally optimal solution.*

PROOF. Let $(\alpha^*,\pi^*)$ be any optimal solution and let $\{\lambda_k\}_{k\geq 0}$ be the sequence of solutions defined recursively as follows:

$\lambda_0 = (\alpha, \pi)$

$\lambda_{k+1}$ is obtained from $\lambda_k$ by reversing an arc $v \in W^{(1)}_{\lambda_k}(\alpha^*, \pi^*)$ or by assigning $\alpha^*(x)$ to an operation $x \in W^{(2)}_{\lambda_k}(\alpha^*, \pi^*)$

Notice that $\lambda_{k+1}$ is obtained from $\lambda_k$ using a move from $N^{AP}$ so, by Theorem 1, $\forall k\ \lambda_k$ is a feasible solution. Let us prove that the above sequence is finite. For any feasible solution $(\alpha, \pi)$, we define the following sets:

$$M^{(1)}_{\alpha,\pi}(\alpha^*, \pi^*) = \{v = (x,y) \in R(\alpha, \pi) : (y,x) \in \overline{R(\alpha^*, \pi^*)}\},$$
$$\overline{M^{(1)}_{\alpha,\pi}(\alpha^*, \pi^*)} = \{v = (x,y) \in \overline{R(\alpha, \pi)} : (y,x) \in \overline{R(\alpha^*, \pi^*)}\},$$
$$M^{(2)}_{\alpha,\pi}(\alpha^*, \pi^*) = \{x \in V : \alpha(x) \neq \alpha^*(x)\},$$
$$M^{(3)}_{\alpha,\pi}(\alpha^*, \pi^*) = \{\{x,y\} : \alpha^*(x) = \alpha^*(y), \alpha(x) \neq \alpha^*(x)\},$$

$$M_{\alpha,\pi}(\alpha^*, \pi^*) = M^{(1)}_{\alpha,\pi}(\alpha^*, \pi^*) \cup M^{(2)}_{\alpha,\pi}(\alpha^*, \pi^*) \cup M^{(3)}_{\alpha,\pi}(\alpha^*, \pi^*),$$

and

$$\overline{M_{\alpha,\pi}(\alpha^*, \pi^*)} = \overline{M^{(1)}_{\alpha,\pi}(\alpha^*, \pi^*)} \cup M^{(2)}_{\alpha,\pi}(\alpha^*, \pi^*) \cup M^{(3)}_{\alpha,\pi}(\alpha^*, \pi^*).$$

The relation between $M^{(1)}_{\alpha,\pi}$ and $W^{(1)}_{\alpha,\pi}$ above and between $M^{(2)}_{\alpha,\pi}$ and $W^{(2)}_{\alpha,\pi}$ is clear. As for $M^{(3)}_{\alpha,\pi}$, it is the set of non-directed arcs between operations which are processed in the same machine in the optimal solution $(\alpha^*, \pi^*)$ and such that at least one of them is processed in other machine in the current assignment $\alpha$; $\overline{R(\alpha*, \pi*)}$ contains one arc for every element in $M^{(3)}_{\alpha,\pi}$. Notice that $M^{(3)}_{\alpha,\pi}$ includes a non-directed arc for every arc that might appear in $\overline{M^{(1)}_{\alpha,\pi}}$ when moving from $\lambda_k$ to $\lambda_{k+1}$. Indeed, if an operation $x$ is assigned to $\alpha*(x)$, its relative position with respect to the rest of the operations in that machine may not be the same as in $\pi*$, in which case the size of $M^{(1)}_{\alpha,\pi}$ increases (and at the same time $M^{(3)}_{\alpha,\pi}$ decreases in at least the same amount).

Clearly, $W_{\alpha,\pi}(\alpha^*, \pi^*) \subset M_{\alpha,\pi}(\alpha^*, \pi^*) \subset \overline{M_{\alpha,\pi}(\alpha^*, \pi^*)}$. Let $\|M_{\alpha,\pi}(\alpha^*, \pi^*)\|$ and $\|\overline{M_{\alpha,\pi}(\alpha^*, \pi^*)}\|$ denote their cardinals. If $\lambda_k$ is not optimal, by lemma 1, there exists a critical arc $v = (x,y)$ such that $(y,x) \in \overline{R(\alpha^*, \pi^*)}$ or there exists a critical operation $x$ such that $\alpha(x) \neq \alpha^*(x)$ thus making it possible to obtain $\lambda_{k+1}$. If $\lambda_{k+1}$ is obtained from $\lambda_k$ by reversing an arc $v \in W^{(1)}_{\lambda_k}(\alpha^*, \pi^*)$, then

$$\|\overline{M^{(1)}_{\lambda_{k+1}}(\alpha^*, \pi^*)}\| = \|\overline{M^{(1)}_{\lambda_k}(\alpha^*, \pi^*)}\| - 1,$$

and, since no machine assignment has changed,

$$\|\overline{M_{\lambda_{k+1}}(\alpha^*, \pi^*)}\| = \|\overline{M_{\lambda_k}(\alpha^*, \pi^*)}\| - 1.$$

On the other hand, if $\lambda_{k+1}$ is obtained from $\lambda_k$ by assigning $\alpha^*(x)$ to a critical operation $x \in M^{(2)}_{\lambda_k}(\alpha^*, \pi^*)$, then

$$\|M^{(2)}_{\lambda_{k+1}}(\alpha^*, \pi^*)\| = \|M^{(2)}_{\lambda_{k+1}}(\alpha^*, \pi^*)| - 1,$$

and from $M^{(3)}_{\lambda_{k+1}}(\alpha^*, \pi^*)$ disappear all pairs $\{x,y\}$ such that the machine assigned to $y$ in $\lambda_k$ is $\alpha^*(x)$. If any of these pairs corresponds to a new arc $v = (x,y)$ in $\overline{M_{\lambda_{k+1}}(\alpha^*, \pi^*)}$ such that $(y,x) \in \overline{R(\alpha^*, \pi^*)}$, the arc will be added to $\overline{M^{(1)}_{\lambda_{k+1}}(\alpha^*, \pi^*)}$. However, for every new element in $\overline{M^{(1)}}$, the corresponding one will disappear from $M^{(3)}$. Else, if no element is added to $\overline{M^{(1)}}$, its cardinal will remain the same, while $M^{(3)}$ may loose some of its elements. In consequence,

$$\|\overline{M_{\lambda_{k+1}}(\alpha^*, \pi^*)}\| \leq \|\overline{M_{\lambda_k}(\alpha^*, \pi^*)}\| - 1.$$

Therefore, in the worst case, for $k^\star = \|\overline{M_{\alpha,\pi}(\alpha^*, \pi^*)}\|$, we have an optimal solution.

As mentioned above, connectivity is an important property for any neighbourhood used in local search. It ensures the non-existence of starting points from which the local search cannot reach a global optimum. It also ensures asymptotic convergence in probability to a globally optimal order. Additionally, although the neighbourhood structure is used in a heuristic procedure in this paper, the connectivity property would allow to design exact methods for fuzzy flexible job shop.

Preliminary experimental results with $N^P$ have endorsed the good theoretical behaviour, obtaining good expected makespan values; however, the large size of the neighbourhood structure for the fuzzy case results in an extremely high computational load. The fact that, for a fixed assignment $\alpha$, $N^P$ is the neighbourhood structure proposed in [51] for *FJSP* allows us to profit from the following result, which can be seen as a filtering mechanism that trims the $N^{AP}$ structure by discarding non-improving neighbours:

**Proposition 2.** *For a given solution $(\alpha, \pi)$ reversing a critical arc which is not at the extreme of a critical block does not improve the expected makespan*

PROOF. See Theorem 2 in [51]: since reversing an arc does not change any machine assignment, the same reasoning applies and, in consequence, the length of the critical paths that existed before the move and whose arc has been reversed, remain unchanged after the move, so the expected makespan cannot improve.

This suggests defining the following reduced neighbourhoods:

**Definition 4.** $N^P_r = \{v \in N^P : v$ is in an extreme of a critical block $\}$.

**Definition 5.** $N^{AP}_r = N^A \cup N^P_r$.

Clearly, $N^{AP}_r \subseteq N^{AP}$ and hence it contains only feasible neighbours. According to Proposition 2, the discarded neighbours in $N^{AP} - N^{AP}_r$ are always non-improving ones; however, connectivity no longer holds. In [45] an analogous reduction for the *FJSP* was shown to be much more efficient than the original structure and, in [60], similar criteria are used to omit some moves provided that they do not generate better solutions than the current ones for *fJSP*.

### 4.5. Makespan estimate

The most time-consuming part of evolutionary algorithms is usually the fitness evaluation. The use of approximate fitness functions in order to gain in efficiency is not new; for example in [61], the authors propose to use surrogate functions to this end. In scheduling problems, it is often possible to accurately estimate the makespan after a move, even without resorting to surrogate functions. In this section, we show how this can be done for the *FfJSP*.

In order to simplify expressions, we extend to the fuzzy and flexible framework, the well-known concepts of head and tail of an operation. For a solution graph $G(\alpha,\pi)$ and an operation $x$, the *head* of $x$, denoted $r_x$, is the starting time of $x$, a TFN given by $r_x = \max\{r_{PJ_x} + p_{PJ_xk_1}, r_{PM_x} + p_{PM_xk}\}$, being $k = \alpha(x)$ and $k_1 = \alpha(PJ_x)$. At the same time, the *tail* of $x$, denoted $q_x$, is the time lag between the moment when $x$ is finished until the completion time of all tasks that must be processed after $x$, a TFN given by $q_x = \max\{q_{SJ_x} + p_{SJ_xk_2}, q_{SM_x} + p_{SM_xk}\}$, where $k_2 = \alpha(SM_x)$.

Clearly, the makespan coincides with both the head of the last operation and the tail of the first operation: $C_{max} = r_{end} = q_{start}$. There are also other basic properties that hold for each parallel graph $G^i(\alpha,\pi)$: $r^i_x$ is the length of the longest path from node *start* to node $x$; $q^i_x + p^i_x$ is the length of the longest path from node $x$ to node *end*; and $r^i_x + p^i_x + q^i_x$ is the length of the longest path from node *start* to node *end* through node $x$, i.e., it is a lower bound on $C^i_{max}(\alpha,\pi)$, being equal if $x$ belongs to a critical path in $G^i(\alpha,\pi)$.

Let us start by considering moves in $N^P$. If $(\alpha,\pi)$ is a solution and $v = (x,y)$ is a critical arc in $G(\alpha,\pi)$, reversing arc $v$ produces a feasible solution $(\alpha,\sigma)$ with $\sigma = \pi_{(v)}$. Let $r$ and $q$ denote the heads and tails in $G(\alpha,\pi)$ (before the move) and let $r'$ and $q'$ denote the heads and tails in $G(\alpha,\sigma)$ (after the move). For every operation $a$ previous to $x$ in $\pi$, $r_a = r'_a$ and for every operation $b$ posterior to $y$ in $\pi$, $q_b = q'_b$. For $x$ and $y$, the heads and tails after the move are calculated as follows:

$$\begin{aligned} r'_y &= \max\{r_{PJ_y} + p_{PJ_yk_3}, r_{PM_x} + p_{PM_xk}\},\\ r'_x &= \max\{r_{PJ_x} + p_{PJ_xk_1}, r'_y + p_{yk}\},\\ q'_x &= \max\{q_{SJ_x} + p_{SJ_xk_2}, q_{SM_y} + p_{SM_yk}\},\\ q'_y &= \max\{q_{SJ_y} + p_{SJ_yk_4}, q'_x + p_{xk}\}, \end{aligned} \tag{9}$$

where $k = \alpha(x) = \alpha(y)$, $k_1 = \alpha(PJ_x)$, $k_2 = \alpha(SJ_x)$, $k_3 = \alpha(PJ_y)$ and $k_4 = \alpha(SJ_y)$. Given this, the estimate of the makespan after the move is $C^e_{max}(\alpha,\sigma) = \max\{r'_x + p_{xk} + q'_x, r'_y + p_{yk} + q'_y\}$. This is a lower bound on the makespan of $(\alpha,\sigma)$.

Regarding moves in $N^A$, let $x$ be a critical operation in $(\alpha,\pi)$ and $k = \alpha(x)$, assigning $x$ to another machine $M_{k'} \in M(x)$ maintaining the processing order $\pi$ produces a new feasible solution $(\beta,\pi)$, where $\beta = \alpha_{<x,k'>}$, i.e., $\beta(y) = \alpha(y)$ for all $y \neq x$ and $\beta(x) = k'$. Again, the head of the operations before $x$ and the tail of the operations after $x$ do not change; while the head and tail for $x$ after the move are given by:

$$\begin{aligned} r'_x &= \max\{r_{PJ_x} + p_{PJ_xk_1}, r_{PM_x} + p_{PM_xk'}\},\\ q'_x &= \max\{q_{SJ_x} + p_{SJ_xk_2}, q_{SM_x} + p_{SM_xk'}\}. \end{aligned} \tag{10}$$

Therefore, the makespan of $(\beta,\pi)$ can be estimated as $C^e_{max}(\beta,\pi) = r'_x + p_{xk'} + q'_x$, which is also a lower bound on the makespan of $(\beta,\pi)$.

## 5. Experimental study

The purpose of this experimental study is twofold: first, to analyse the behaviour of the proposed Hybrid Genetic Tabu Search (HGTS) algorithm and, second, to compare it with the state-of-the-art. For the first purpose we consider the components of HGTS (the Heuristic Initial Population (HIP) generator, the GA and the TS) both separately and in combination. To compare HGTS with the state-of-the-art, we start by considering the best approaches for the *FfJSP*, which to our knowledge are those proposed in [37], [38], [35] and [36]. HGTS is further tested on *fJSP* instances (with no uncertainty); this allows us to establish comparisons with a large number of algorithms from the literature on a varied set of instances. We conclude this empirical study proposing a new benchmark for the *FfJSP* with larger and harder instances than those of the current benchmarks.

After a series of preliminary experiments, the following setting for HGTS has been chosen: the population size is 100 chromosomes and the stopping criterion for the GA, is 20 generations without improving the best solution, while for the TS, the number of iterations without improvement depends on the average size of the instances of the different benchmarks (details are given later). HGTS has been implemented in C++ and the target machine is a PC with a Xeon E5520 processor and 24GB RAM. For each problem instance, we have launched 30 runs and considered as performance metric the Mean Relative Error (MRE) with respect to a lower bound of the makespan, calculated for the *fJSP* instances as

$$MRE = (C_{max} - LB)/LB \times 100 \tag{11}$$

where $LB$ is the instance's makespan lower bound, and for the *FfJSP* instances as

$$MRE = (E[C_{max}] - LB_F)/LB_F \times 100 \tag{12}$$

where $LB_F$ is a lower bound of the expected makespan. The lower bounds for most of the *fJSP* instances are those reported in [31]. To obtain lower bounds for fuzzy instances, we adapt the lower bound proposed in [62] for *JSP* to the fuzzy and flexible setting as follows:

$$LB_F = E[\max_i\{\sum_{j=1}^{N_i} pm_{o_{ij}}\}] \tag{13}$$

where $pm_{o_{ij}} = \min\{p_{o_{ijk}}, M_k \in M(o_{ij})\}$.

Regarding the benchmarks for the *FfJSP*, we consider here those proposed in [34], [37] and [38] with six instances altogether. Instances 01 and 02 have 10 jobs, 10 machines and 40 operations each $(10 \times 10 \times 40)$. Instances 03 and 04 are $10 \times 10 \times 50$ and instances 05 and 06 are $15 \times 10 \times 80$. All instances have total flexibility, i.e., any operation can be executed

on any machine. For these instances, the stopping criterion for the TS is 50 iterations without improvement.

In [38], the authors report results on six more *FfJSP* instances which are obtained as fuzzified versions of *fJSP* instances from [63],[64], and [29]. In this case, the fuzzy processing times of operations are not explicitly reported but a method to generate the TFNs is given instead, so the modal value is the original crisp duration, and the lower and upper defining points are randomly chosen from some intervals. In consequence, it is impossible to work with exactly the same instances. Additionally, the solutions reported in [38] are not fully consistent with the described fuzzifying method. For example, for the instance with 10 jobs and 6 machines from [63], even if the TFNs are formed by assigning to each defining point the smallest possible value (the lower endpoint of each interval as proposed in [38]), the lower bound of the expected makespan obtained from equation (13) is 312; however, the expected values reported in [38] for the average and the best solutions are 273.53 and 167.25 respectively. For these reasons, we have not considered these instances on our experimental study for the *FfJSP*. We do however consider the original crisp *fJSP* instances from [63],[64] and [29] in Section 5.3.

### *5.1. Analysis of the HGTS*

We start the analysis of HGTS by considering the effect of the initial population. To this end, we obtain two different initial populations, one generated using the heuristic from Section 4.2 and the other one, randomly. The best and average values (the latter between brackets) of the expected makespan in both populations can be seen in the third and fourth columns of Table 1, labelled Rd.IP and HIP respectively. These clearly show the higher quality of the heuristic population. The table also contains results for the GA both with random and heuristic initial populations (fifth and sixth columns respectively, labelled RGA and HGA), results for the heuristic strategy used as production rule (seventh column, labelled H), results for the TS both with random and heuristic initial population (eighth and ninth columns, labelled RTS and HTS) and results for the combination of GA and TS with heuristic population, i.e., HGTS (last column). Each row in the table corresponds to a problem instance, with the instance identifier in the first column and the previously best known solution in the second column; additionally, the last row reports the average MRE across all instances w.r.t. the lower bound as explained above.

The results for RGA, HGA, H, RTS, HTS and HGTS in Table 1 correspond to the case where all methods are given the same running time: HGTS stops following the criterion given above, while HGA and RGA stop after they have been running for the same time as HGTS, H iteratively applies the heuristic scheduling schema *FfInsertion* to random orderings until the same time as HTGS is consumed and HTS and RTS are launched iteratively from different solutions generated with the heuristic algorithm (or by random in the case of RTS) until the same time is used.

Notice that, despite the difference in quality of the initial populations, this difference does not always translate into different results after the search, with the three algorithms under consideration presenting quite different sensitivity to this initial population. The GA is the method that benefits most from the heuristic seeding (57% improvement w.r.t. starting from a random population). This improvement is reduced to 15% for the TS, being negligible in all instances except the two largest ones (05 and 06). Finally, the benefit becomes insignificant for the hybrid algorithm HGTS in this benchmark (this is the reason that no column is added for the combination of GA and TS with random initial population). We believe that this may be explained by the fact that these instances are not challenging enough to appreciate the contribution of the heuristic seeding to the overall performance of the HGTS. In Section 5.4, we will introduce larger instances and use them to further analyse the influence of the different initial populations. In the remaining of the experimental study, we consider heuristic initial populations.

In addition, we can assess the potential of the proposed heuristic strategy, with H being able to yield quite competitive solutions. This is due both to the large number of solutions that can be generated in the running time given to H and to the diversity among them. However, even these solutions are far from the quality of those obtained with HTS (40% worse), and even further from the solutions provided by HGTS (68% worse).

We must however be cautious when comparing HTS or H with HGA based on the results from Table 1, since HGA is somewhat hindered by the parameterisation used. Indeed, HGA has the same population size as HGTS but a different stopping criterion (same time as HGTS). This parameterisation has been chosen to assess as fairly as possible the contribution of the TS in terms of intensification to the final algorithm HGTS w.r.t. HGA, but it has the downside effect of not showing the full potential of HGA on its own. On the other hand, we can appreciate how the genetic algorithm (RGA or HGA) manages to evolve the initial population (Rd.IP or HIP respectively). When it starts from random individuals, it has a drastic effect in the MRE values and it provides a noticeable reduction when it starts from heuristic individuals.

Finally, the results illustrate the synergy that exists between the search strategies that are combined in HGTS, with HGTS reducing the average MRE nearly 17% w.r.t. HTS, more than 40% w.r.t. H and 54% w.r.t. HGA, showing that this combination obtains better results than either GA and TS when run separately. In summary, HGTS provides a good symbiosis between a good starting point (provided by H) and a good combination of exploration, thanks to the GA, and exploitation, thanks to the iterative improvement of the TS.

In the next section we analyse these results in more detail in the context of comparison with other existing methods.

### *5.2. Comparison with the state-of-the-art in the* FfJSP

In order to compare HGTS with the state-of-the-art, we consider the best methods proposed so far for the *FfJSP*: the Co-evolutionary Genetic Algorithm (CGA) proposed by D.Lei in [37], the Swarm-based Neighbourhood Search Algorithm (SNSA) proposed by D. Lei and X. Guo in [38], the Hybrid Artificial Bee Colony Algorithm (hABC) proposed by L. Wang *et al* in [35] and the Estimation Distribution Algorithm (EDA)

Table 1: Analysis of the components of HGTS for *FfJSP*

| Ins | pBKS | Rd.IP | HIP | RGA | HGA | H | RTS | HTS | HGTS |
|---|---|---|---|---|---|---|---|---|---|
| 01 | 30.25 | 66.28 (97.85) | 32.03 (37.19) | 38.75 (40.45) | 30.50 (31.15) | 29.00 (29.60) | 28.50 (28.70) | 28.50 (28.53) | 28.50 (28.50) |
| 02 | 45.25 | 87.60 (127.7) | 46.90 (54.58) | 53.75 (57.60) | 45.75 (46.80) | 45.25 (45.55) | 45.25 (45.25) | 45.25 (45.25) | 45.25 (45.25) |
| 03 | 47.75 | 98.23 (139.6) | 49.65 (56.93) | 61.75 (63.70) | 47.00 (48.18) | 45.50 (46.00) | 44.50 (44.78) | 44.50 (43.53) | 43.50 (43.68) |
| 04 | 38.00 | 82.05 (115.5) | 38.83 (44.72) | 49.00 (51.33) | 37.75 (38.25) | 35.75 (36.18) | 35.25 (35.38) | 35.00 (35.08) | 34.25 (34.28) |
| 05 | 62.00 | 126.1 (167.3) | 63.23 (69.58) | 73.50 (76.13) | 60.75 (61.63) | 58.50 (59.18) | 56.50 (56.95) | 53.75 (54.30) | 51.50 (52.15) |
| 06 | 63.75 | 117.4 (156.7) | 61.73 (67.96) | 72.25 (74.25) | 59.00 (60.78) | 57.00 (57.53) | 55.50 (56.18) | 53.25 (53.58) | 51.25 (51.90) |
| *MRE* | *25.60* | *154.3 (254.7)* | *28.10 (45.09)* | *53.19 (59.51)* | *23.00 (25.59)* | *18.54 (19.92)* | *16.11 (16.87)* | *13.83 (14.27)* | *11.24 (11.88)* |

proposed by S. Wang *et al* in [36]. CGA is implemented in Microsoft Visual C++ 6.0 and run on a 512MB RAM 1.7GHz PC, it uses a population of 150 chromosomes and a number of 1000 generations and the time taken ranges from 8 to 11 seconds for a single run. SNSA is coded in Microsoft Visual C++ 6.0 and run on a 2GB RAM 2.2GHz PC, the swarm size is 100, the number of iterations is limited to 500 and the time taken varies from 9 to 14 seconds a single run. hABC is implemented in C++ and run on a 3.2GB RAM 2.83GHz PC with a population of $2 \times n \times m$ chromosomes, $n \times m$ steps for local search and a limit of 20 trials without improving a source of food; the time taken varies between 11 and 15 seconds per single run. Finally, EDA is coded in C++ and run on Thinkpad T420 2GB RAM 2.3GHz; the parameters are set as follows: population size of 150, percentage of superior sub-population from population $\nu$ = 20, and learning rates $\alpha$ = 0.3 and $\beta$ = 0.1; the time taken ranges between 4 and 10 seconds per single run. In all cases, the reported results correspond to the best and average solutions in 20 runs.

Table 2 shows the results obtained by CGA, SNSA, hABC, EDA and HGTS on the six instances 01–06 provided these data are available. Unfortunately, some of the references do not report results for at least one of the largest instances 05, 06; when this is the case, the corresponding cell in Table 2 is left empty. For each method and instance, the table reports the best and average makespan (the latter between brackets). Additionally, the second column contains the lower bound for the expected makespan calculated according to (13) and the last column shows the average time taken by HGTS in a single run. It is worth noting that the results for HGTS correspond to using the reduced neighbourhood $N_r^{AP}$ instead of $N^{AP}$, since they provide very similar results (compare with Table 1). Rows 7 to 9 include average MRE values (across the first 4, 5 and all 6 instances respectively) for all methods for which we have available data. Finally, the last row in Table 2, labelled #best, indicates the number of instances where each method obtains the best known solution. In bold are the best known solutions, with a superindex "*a*" in the case that our method improves the previous best known solution, and with a superindex "*b*" in the case it is the optimal solution.

Clearly, HGTS outperforms all the other four methods across the six instances in best and average expected makespan. It obtains the best-so-far solutions in all instances, improving the previously best-known solutions in 5 of them. In fact, even the average makespan value of HGTS improves the best value obtained with the other methods in all but one instance (02). Moreover, for instances 01 and 03, HGTS obtains the optimal solution, since its expected makespan coincides with the lower bound $LB_f$. With respect to run times, it is worth mentioning that EDA requires considerably less time than HGTS, even CPU times are not directly comparable due to differences in target machines. The reason may be that HGTS is a more complex metaheuristic and needs more time to converge.

It is important to remark that all the available results for the three algorithms CGA, SNSA, hABC and EDA have been obtained using the maximum approximation $\max_R$, while HGTS uses $\max_I$[1]. This however should not be a problem in this case, since $\max_R(A,B) \leq \max_I(A,B)$ for every pair of TFNs $A$ and $B$, meaning that if all algorithms were to use the same maximum approximation the difference in favour of HGTS would either be the same or even greater. Just for the sake of completeness, we have evaluated the solutions obtained by HGTS (the operation processing order together with the machine assignment) using $\max_R$ instead of $\max_I$. Obviously, the resulting expected makespan does not get worse in any case. More interestingly, the results are very similar: the best MRE obtained with $\max_R$ is 10.18% versus 10.61% with $\max_I$ and the average is 11.20% versus 11.54%. We may conclude that the comparison between HGTS and the state-of-the-art algorithms CGA, SNSA, hABC and EDA is not affected by the maximum operation, being in all cases favourable to the new method.

### *5.3. Comparison with the state-of-the-art in the* fJSP

To enhance the significance of the experimental study, we have conducted experiments to compare HGTS with the state-of-the-art approaches for the *fJSP*. The motivation is that the deterministic version of the problem has been considered in a large number of research works over the last two decades, so we can expect the best approaches proposed so far to be really refined, making it a challenge to improve or even match their results. Therefore, if HGTS (designed for *FfJSP*) were at least similar to some of the best approaches for the *fJSP*, this fact would be another strong evidence of the good performance of HGTS.

We have considered six benchmark sets: the *XWdata* from [64] and [65], the set of instances from [63], the *BRdata* from [29], the *BCdata* from [66], the *DPdata* from [30] and the

[1] We have already motivated in Section 3.1 our choice of the $\max_I$ operator for HGTS

Table 2: Summary of results in the *FfJSP*

| Ins | $LB_f$ | CGA | SNSA | hABC | EDA | HGTS | $T_{HGTS}$(s.) |
|---|---|---|---|---|---|---|---|
| 01 | 28.50 | 30.00 (30.18) | 30.25 (31.68) | 30.50 (32.15) | 30.00 (33.18) | **28.50**$^{a,b}$ (28.50) | 5.8 |
| 02 | 45.00 | 45.75 (47.45) | **45.25** (47.05) | 45.75 (47.70) | 45.75 (46.35) | **45.25** (45.25) | 3.4 |
| 03 | 43.50 | 47.75 (51.00) | 47.50 (51.25) | 47.75 (50.70) | 45.75 (47.53) | **43.50**$^{a,b}$ (43.64) | 11.7 |
| 04 | 33.50 | 37.75 (40.80) | 39.25 (40.80) | 38.00 (40.45) | 35.75 (37.78) | **34.25**$^{a}$ (34.29) | 12.7 |
| 05 | 37.50 | 62.00 (65.95) | 65.75 (68.53) | | 54.75 (57.68) | **51.00**$^{a}$ (51.83) | 51.6 |
| 06 | 40.25 | | 63.75 (65.65) | | | **50.25**$^{a}$ (51.50) | 53.3 |
| *MRE(01-04)* | | *7.35 (15.22)* | *8.26 (14.31)* | *7.97 (14.03)* | *4.70 (10.35)* | *0.70 (0.81)* | |
| *MRE(01-05)* | | *18.94 (27.35)* | *21.68 (28.00)* | | *12.96 (19.04)* | *7.76 (8.29)* | |
| *MRE(01-06)* | | | *27.80 (33.85)* | | | *10.61 (11.54)* | |
| #best | | 0 | 1 | 0 | 0 | 6 | |

*HUdata* from [67], making a total of 186 instances (we refer the interested reader to the original references for further detail on these test beds). For every test bed, HGTS is compared with the best available results in the literature. To reduce the computational load, our method uses the reduced neighbourhood.

For *XWdata*, our method is compared with the Knowledge-Based Ant Colony Optimization method (KBACO) by Xing *et al.* from [68], the Tabu Search with an efficient Public Critical Block neighbourhood structure (TSPCB) by Li *et al.* from [69], the Artificial Bee Colony (ABC) by Wang *et al.* from [70] and the bi-population based estimation of distribution algorithm (BEDA) by Wang *et al.* from [71]. The results reported in the literature together with those obtained with HGTS can be seen in Table 3: each row corresponds to an instance in the test bed, with its identifier in the first column and the makespan lower bound in the second column. In absence of other information, we have calculated lower bounds for these instances using equation 13 (without expected values as it corresponds to crisp instances). The next 5 columns correspond each to one of the methods above, showing the best and average makespan values (the latter between brackets) obtained on that instance. Finally, the last column shows the average time (in seconds) taken by a single run of HGTS on that instance; these are included for the sake of completeness, even though we are aware that these times may not be fairly comparable to the times reported in the above references due to the differences on the running environments and the target machines. As we can observe, for three instances HGTS obtains the optimal solution in every single run, while for the remaining instances it reaches the best-known solution, as it is also the case with the other methods.

The results on the Thomalla's benchmark [63] are compared with the results reported for the PSO by Girish and Jawahar in [72] together with those obtained by ILOG OPL Studio, also reported in [72]. In this paper best-known solution (BKS) values are also given. All these data, together with the data regarding HGTS can be seen in Table 4, following the same format as Table 3 above. Again, the LBs for these instances are computed using equation 13. We can see that HGTS obtains the previously-known BKS both in average and best values for the first two instances and establishes a new BKS for the third instance. In fact, in two cases it yields the optimal solution; this optimal solution was already known for instance EX1 but, more interestingly, it is established for the first time by HGTS for instance EX3.

Table 4: Summary of results in the *fJSP*: *Thomalla* Benchmark

| Ins | LB | PSO | ILOG | HGTS | T(s.) |
|---|---|---|---|---|---|
| EX1 | 117 | **117** | **117** | **117**$^{b}$ (117) | 0.11 |
| EX2 | 95 | **109** | **109** | **109** (109) | 0.62 |
| EX3 | 316 | 328 | 675 | **316**$^{a,b}$ (316) | 4.33 |
| *MRE* | | *6.18* | *42.78* | *4.91 (4.91)* | |
| #best | | 2 | 2 | 3 | |

For the following three test beds (the most widely used in the literature), unless otherwise stated, we compare HGTS with the tabu search (TS) by Mastrolilli and Gambardella from [31], the hybrid genetic algorithm (hGA) by Gao, Sun and Gen from [11], the climbing depth-bounded discrepancy search (CDDS) by Hmida *et al.* from [32], the hybrid harmony search and large-scale neighbourhood search algorithm (HHS/LNS) by Yuan and Xu from [33], and the genetic algorithm hybridised with tabu search (GA+TS) by González *et al.* from [14].

Considering the size of the instances, the stopping criterion for TS is 200 iterations without improvement for the *BCdata* and *BRdata* families and 400 iterations without improvement for the *DPdata* dataset. As above, for the sake of completeness, we report the time taken by a single run of HGTS to solve each instance. It is however worth mentioning that HGTS has been designed for fuzzy instances so, to run it on deterministic problems, every instance has been converted into a fuzzy one (given that every real number $r$ can be represented as the TFN $(r,r,r)$). In consequence, CPU times may be expected to be about 3 times longer than those required by a simplified algorithm specifically designed for the *fJSP*. Despite of this, CPU times with HGTS are not significantly longer than those reported in the references above for the other algorithms. A detailed comparison in [33] states that the computational effort of HHS/LNS is comparable with that of hGA but it is much longer than those of TS and CDDS. Taking into account that the target machines are very similar, following the comparison framework explained in [33], we could consider that HHS/LNS takes 20% longer than HGTS. However, as there are factors other than the language and the machine that influence the computational time, this comparison is only indicative.

Tables 5, 6 and 7 show the results of the experiments on the *BRdata*, the *BCdata*, and *DPdata* benchmarks respectively. The format is analogous to the tables above, however, in this case, the lower bound is the value reported in [31]. We also include

Table 3: Summary of results in the *fJSP*: ***XWdata***

| Ins | LB | KBACO | TSPCB | ABC | BEDA | HGTS | T(s.) |
|---|---|---|---|---|---|---|---|
| Case 1 | 11 | **11** (11.0) | **11** (11.0) | **11** (11.0) | **11** (11.0) | **11**$^b$ (11.0) | 0.3 |
| Case 2 | 12 | **14** (14.3) | **14** (14.2) | **14** (14.0) | **14** (14.0) | **14** (14.0) | 2.2 |
| Case 3 | 11 | **11** (11.0) | **11** (11.0) | **11** (11.0) | **11** (11.0) | **11**$^b$ (11.0) | 2.1 |
| Case 4 | 7 | **7** (7.4) | **7** (7.1) | **7** (7.0) | **7** (7.0) | **7**$^b$ (7.0) | 7.3 |
| Case 5 | 10 | **11** (11.3) | **11** (11.7) | **11** (11.0) | **11** (11.0) | **11** (11.0) | 1.3 |
| *MRE* | | *5.33 (7.58)* | *5.33 (7.35)* | *5.33 (5.33)* | *5.33 (5.33)* | *5.33 (5.33)* | |
| #best | | 5 | 5 | 5 | 5 | 5 | |

two more rows which serve as summary, containing MRE values and the number of instances for which a method reaches the best-known solution.

For the *BRdata* benchmark, results in Table 5 shows that HGTS improves the previously-best-known solution in one instance (*MK06*). and obtains the best-known solution in 9 of the 10 instances, something done only by hGA. In terms of MRE, HGTS is the best method if we consider the best makespan and the second best if we consider average makespan instead. Figure 3 shows the job-oriented Gantt chart of the new best solution encountered for the instance *MK06*.

The results for the *BCdata* in Table 6 incorporate best makespan values for yet another method from the literature, the parallel double-level metaheuristic approach (TSBM$^2$h) proposed by Bozejko *et al.* in [60]. The reason is that it reports detailed good results on this benchmark but not for the remaining test beds. Also, HHS/LNS is omitted in this table because for this benchmark only the MRE average value for the best solutions (22.43) is reported in [33]. As we can see, for the *BCdata* HGTS obtains the best-known solution in 19 of the 21 instances. Moreover, in one of these instances (*seti5c12*), HGTS improves the previously-best-known solution. Additionally, HGTS obtains the best MRE values for both the best and the average makespan among all the algorithms considered, even if differences are small.

On the *DPdata* benchmark, HGTS improves the previously best-known solution in three instances (10a, 13a and 16a) and in other three instances (01a, 03a and 04a) it obtains the optimal solution. Additionally, HGTS yields the best the MRE values both for the best and average makespan from all the six algorithms considered.

Finally, for the *HUdata* set, Figure 4 provides a summary of the results (detailed results for HGTS on these data are openly available on the web[2]). In this case we only report results for TS, hGA, CDDS and HGTS, since results on this benchmark are not available for the other methods. Also, the figure portrays MRE values for the average makespan across groups of instances; the reason is that the makespan results reported in [11] and [32] for this benchmark are averages on the same groups. We can observe that HGTS performs slightly better than the other approaches on the instances of the *edata* subset, which seems to be the hardest of the set. Moreover, HGTS reaches the optimal solution in 77 instances (27 *edata*, 19 *rdata* and 31 *vdata*), improves the best known solution given in [31] in 26 instances (12 *edata*, 12 *rdata* and 2 *vdata*) and reaches the best known solution in other 90 instances (29 *edata*, 28 *rdata* and 33 *vdata*) of the 129 instances of the test bed.

Overall, for the crisp version of the problem, we have tested 186 instances, reaching the best-known solution for 160 of them. Furthermore, 87 out of these 160 solutions are optimal. Also, HGTS improves the previously-best-known solution in 31 instances.

To further avail the quality of the proposed method, we have done some statistical tests to analyse differences between HGTS and other algorithms from the literature. Following [73], since we have multiple-problem analysis, we have used non-parametric statistical tests. First, we have run a Shapiro-Wilk test to confirm the non-normality of the data. Then we have used paired Wilcoxon signed rank tests to compare the medians of the MRE values between HGTS and each of the other methods, provided that results for single instances are available, that is, we have considered the sets $BRdata \cup BCdata \cup DPdata$ and the methods TS, hGA, CDDS and GA+TS. In all these tests, the level of confidence used was 95% and the alternative hypothesis was "the difference between the errors of the HGTS and the method tested is smaller than 0". The $p$-values obtained with these tests (TS: 0.0001086, hGA: 0.03996, CDDS: 0.000526, GA+TS: 0.001188) show that there exist statistically significant differences between HGTS and some of the methods of the state-of-the-art in *fJSP* on these benchmarks.

In summary, we can conclude that HGTS clearly outperforms the previous results for the *FfJSP* and, in the deterministic setting of *fJSP*, it is slightly but significantly better than the state-of-the-art.

### *5.4. A new benchmark for the* FfJSP

As we have seen that there are but a few *FfJSP* instances openly available in the literature and, in addition, the optimal solution for some of these instances has already been found and proven. This motivates us to propose here a new set of more challenging problems and provide preliminary results of our algorithm on them for future comparisons.

We base the new instances on well-known crisp *fJSP* problems and add uncertainty to the durations based on the ideas from [25] but using a wider interval for the third defining point, as it is more likely (and critical) for an operation to take longer time than expected instead of a shorter one. Let $p$ be the real duration of a task in some machine, from this we build a TFN $P = (p^1, p^2, p^3)$ where $p^2 = p$ and $p^1$ and $p^3$ are random positive integer values verifying that $p^1 \in [0.85p, p)$ and

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

Table 5: Summary of results in the *fJSP*: ***BRdata***

| Ins | LB | TS | hGA | CDDS | HHS/LNS | GA+TS | HGTS | T(s.) |
|---|---|---|---|---|---|---|---|---|
| Mk01 | 36 | **40** (40) | **40** (40) | **40** (40) | **40** (-) | **40** (40) | **40** (40) | 5 |
| Mk02 | 24 | **26** (26) | **26** (26) | **26** (26) | **26** (-) | **26** (26) | **26** (26) | 15 |
| Mk03 | 204 | **204** (204) | **204** (204) | **204** (204) | **204** (-) | **204** (204) | **204**$^b$ (204) | 2 |
| Mk04 | 48 | **60** (60) | **60** (60) | **60** (60) | **60** (-) | **60** (60) | **60** (60) | 10 |
| Mk05 | 168 | 173 (173) | **172** (172) | 173 (174) | **172** (-) | **172** (172) | **172** (172) | 18 |
| Mk06 | 33 | 58 (58) | 58 (58) | 58 (59) | 58 (-) | 58 (58) | **57**$^a$ (58) | 63 |
| Mk07 | 133 | 144 (147) | **139** (139) | **139** (139) | **139** (-) | **139** (139) | **139** (139) | 33 |
| Mk08 | 523 | **523** (523) | **523** (523) | **523** (523) | **523** (-) | **523** (523) | **523**$^b$ (523) | 3 |
| Mk09 | 299 | **307** (307) | **307** (307) | **307** (307) | **307** (-) | **307** (307) | **307** (307) | 24 |
| Mk10 | 165 | 198 (199) | **197** (197) | **197** (198) | 198 (-) | 199 (200) | 198 (199) | 104 |
| *MRE* | | *15.41 (15.83)* | *14.92 (14.92)* | *14.98 (15.35)* | *14.98 (-)* | *15.04 (15.13)* | *14.67 (15.02)* | |
| #best | | 6 | 9 | 8 | 8 | 8 | 9 | |

- means that the corresponding data is not available.

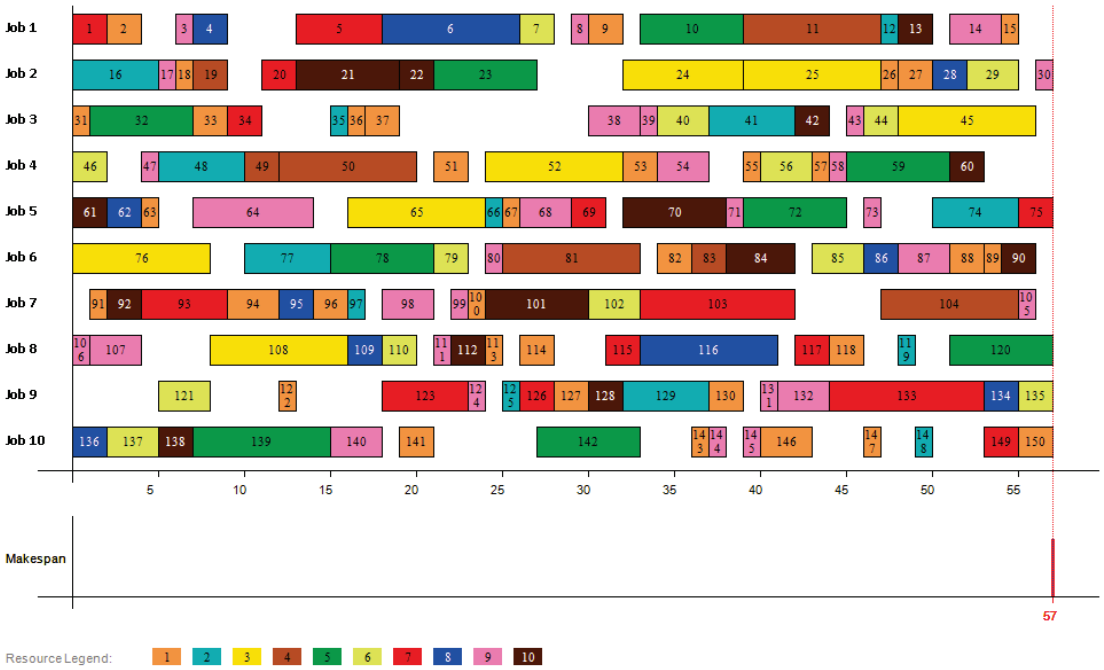



Figure 3: Gantt chart, job-oriented, of new best solution of instance *MK06* from ***BRdata***.

$p^3 \in [2p - p^1, 1.2p]$. In the case that no integer value exists in the interval $[0.85p, p)$, we set $p^1 = \max\{1, p-1\}$, and if there is no integer value in $[2p - p^1, 1.2p]$, $p^3$ takes a random value in $[p+1, p+2]$. In this way, unlike the instances in [25], the generated TFNs always verify that $p^2 - p^1 \leq p^3 - p^2$ and therefore $E[P] \geq p$. It is easy to prove that, thanks to this inequality, for these instances the optimal solution of the original crisp problem (or any lower bound thereof, usually better than the raw LB defined in equation 13) provides a lower bound for the expected makespan of the fuzzy solution, allowing to measure relative errors more accurately. The crisp problems we take to build the benchmark are the largest ones, in number of operations, from the common-use benchmarks *DPdata*, *BRdata* and *BCdata*, as we conjecture they provide bigger room for improvement. More precisely, we fuzzify instances $07a$ to $18a$ from *DPdata* and instance *Mk10* from *BRdata*, all of them having more than 240 operations. Instance *MK09*, which has the same size, has been discarded because all authors obtain the same results, both in best and average makespan values, which leads us to think that no further improvement is possible on this problem. The new fuzzy instances thus generated are openly available on the web[3]

This set of larger and harder *FfJSP* instances allow us to better evaluate the behaviour of our algorithm using the neighbourhood $N^{AP}$, for which connectivity holds, instead of using the reduced structure $N_r^{AP}$, which contains less neighbours and therefore has been used across all of the experimental analysis above. As already mentioned, the loss of the connectivity is relatively unimportant in our case, given that the neighbourhood is used within a metaheuristic, whereas the reduction in neighbourhood size obtained by discarding non-improving neighbours augments the chance of obtaining improving ones.

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

Table 6: Summary of results in the *fJSP*: ***BCdata***

| Ins | LB | TS | hGA | CDDS | TSBM$^2$h | GA+TS | HGTS | T(s.) |
|---|---|---|---|---|---|---|---|---|
| mt10c1 | 655 | 928 (928) | **927** (927) | 928 (929) | **927** (-) | **927** (927) | **927** (927) | 13 |
| mt10cc | 655 | 910 (910) | 910 (910) | 910 (911) | **908** (-) | **908** (909) | **908** (910) | 13 |
| mt10x | 655 | **918** (918) | **918** (918) | **918** (918) | 922 (-) | **918** (922) | **918** (918) | 15 |
| mt10xx | 655 | **918** (918) | **918** (918) | **918** (918) | **918** (-) | **918** (918) | **918** (918) | 12 |
| mt10xxx | 655 | **918** (918) | **918** (918) | **918** (918) | **918** (-) | **918** (918) | **918** (918) | 12 |
| mt10xy | 655 | 906 (906) | **905** (905) | 906 (906) | **905** (-) | **905** (905) | **905** (905) | 13 |
| mt10xyz | 655 | **847** (850) | 849 (849) | 849 (851) | 849 (-) | 849 (850) | **847** (850) | 18 |
| setb4c9 | 857 | 919 (919) | **914** (914) | 919 (919) | **914** (-) | **914** (914) | **914** (914) | 16 |
| setb4cc | 857 | 909 (912) | 914 (914) | 909 (911) | **907** (-) | **907** (907) | **907** (908) | 15 |
| setb4x | 846 | **925** (925) | **925** (931) | **925** (925) | **925** (-) | **925** (925) | **925** (925) | 15 |
| setb4xx | 846 | **925** (926) | **925** (925) | **925** (925) | **925** (-) | **925** (925) | **925** (925) | 14 |
| setb4xxx | 846 | **925** (925) | **925** (925) | **925** (925) | **925** (-) | **925** (925) | **925** (925) | 15 |
| setb4xy | 845 | 916 (916) | 916 (916) | 916 (916) | **910** (-) | **910** (910) | **910** (910) | 19 |
| setb4xyz | 838 | 905 (908) | 905 (905) | 905 (907) | **903** (-) | 905 (905) | 905 (905) | 15 |
| seti5c12 | 1027 | 1174 (1174) | 1175 (1175) | 1174 (1175) | 1174 (-) | 1171 (1173) | **1170**$^a$ (1171) | 41 |
| seti5cc | 955 | **1136** (1136) | 1138 (1138) | **1136** (1137) | **1136** (-) | **1136** (1137) | **1136** (1137) | 34 |
| seti5x | 955 | 1201 (1204) | 1204 (1204) | 1201 (1202) | **1198** (-) | 1199 (1200) | 1199 (1201) | 38 |
| seti5xx | 955 | 1199 (1201) | 1202 (1203) | 1199 (1199) | **1197** (-) | **1197** (1198) | **1197** (1198) | 34 |
| seti5xxx | 955 | **1197** (1198) | 1204 (1204) | **1197** (1198) | **1197** (-) | **1197** (1197) | **1197** (1198) | 31 |
| seti5xy | 955 | **1136** (1136) | **1136** (1137) | **1136** (1138) | **1136** (-) | **1136** (1137) | **1136** (1137) | 34 |
| seti5xyz | 955 | **1125** (1127) | 1126 (1126) | **1125** (1125) | 1128 (-) | 1127 (1128) | **1125** (1126) | 43 |
| *MRE* | | *22.53 (22.63)* | *22.61 (22.66)* | *22.54 (22.60)* | *22.45 (-)* | *22.42 (22.49)* | *22.39 (22.46)* | |
| #best | | 11 | 10 | 10 | 17 | 16 | 19 | |

This is confirmed by our experimental results, which show that the *MRE* for the best and average expected makespan obtained with $N_r^{AP}$ in each instance are slightly better (7% MRE improvement). However, the most relevant fact is that, since $N^{AP}$ generates more neighbours, the runtime of HGTS using $N^{AP}$ is 21% longer than the runtime using $N_r^{AP}$.

As advanced in Section 5.1, this more challenging benchmark also allows to better evaluate the effect of the heuristic seeding in the HGTS. Figure 5 depicts the average MRE values and CPU times obtained with the hybrid algorithm GA+TS both using heuristic seeding and starting with a random population given three different stopping criteria for the tabu search: 50, 100 and 400 maximum number of iterations without improvement. It is clear that MRE values are slightly smaller when using heuristic seeding and, moreover, the CPU time spent in generating this heuristic initial population is compensated when the local search takes longer. This can be interpreted as a result of the fact that the heuristic seeding helps the local search to find good solutions quickly not only in the first iterations but along the whole evolutive process.

Finally, Table 8 contains the results of 30 runs of our algorithm HGTS on every instance of the new benchmark. Each row corresponds to one of these instances, with the identifier in the first column (Ins). The LB values in the second column are the available lower bounds of the original crisp problem (which are also valid LBs for the expected makespan). The next four columns correspond to makespan results: the columns with the header "Best" contain the best makespan (a TFN) together with its expected value obtained across the 30 runs and the next two columns, with headers "Avg" and "Std. Dev." contain the average and standard deviation values for the expected makespan across these 30 runs. Finally, the last column with header "T(s.)" reports the average time per run of HGTS in seconds. Notice that due to the fuzzification method used to generate the problems (with asymmetric TFNs stretched to the right), the expected makespan values are necessarily greater that the values obtained by the same algorithm on the corresponding original crisp problem. Furthermore, we can see that the runtime required for solving the fuzzy instances is in average a 13% longer than the time required for the original crisp problems, illustrating the increased difficulty of handling uncertainty.

## 6. Conclusions

We have considered the *FfJSP*, a variant of the job shop problem which incorporates both flexibility in machine assignment and uncertainty in operation durations, in an attempt to reduce the gap between academic and real-world problems. We have proposed a new hybrid algorithm which combines a GA with TS and heuristic seeding. The new heuristic method to generate initial solutions benefits from the flexible nature of the problem and generates high-quality and diverse initial solutions which provide a starting point for the GA, enhancing its exploitation ability. The designed TS algorithm is then applied to every newly generated chromosome in the GA. A key point for the TS is the neighbourhood structure. We have proposed here two new structures. For the first one, we have proved that it verifies both feasibility and connectivity, the latter ensuring asymptotic convergence in probability to a global optimal solution. The second neighbourhood is obtained by incorporating a filtering mechanism that trims the first structure by discarding non-improving neighbours; this second neighbourhood keeps the feasibility property and considerably reduces the size of the first one at the cost of losing connectivity. Finally, a method based on constraint propagation has been introduced that allows to speed-up the evaluation of new chromosomes. We have

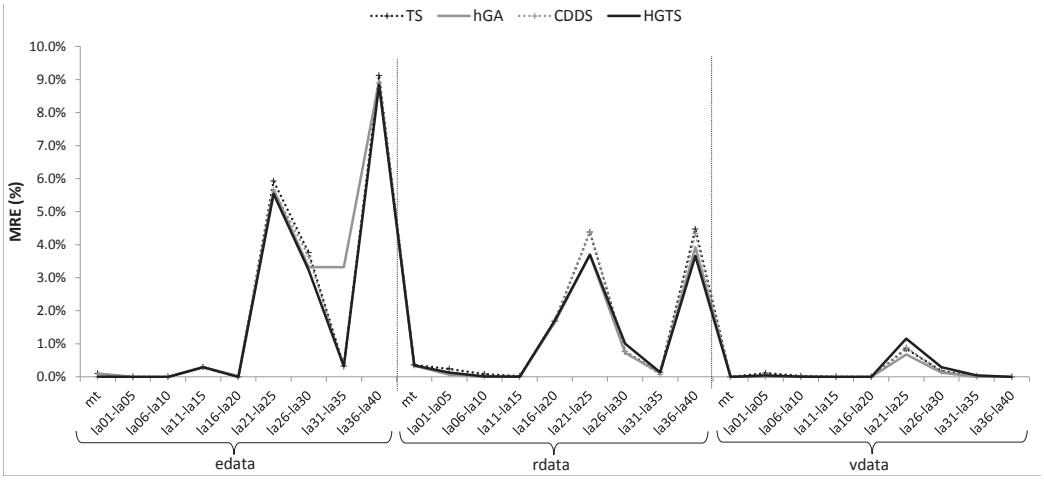
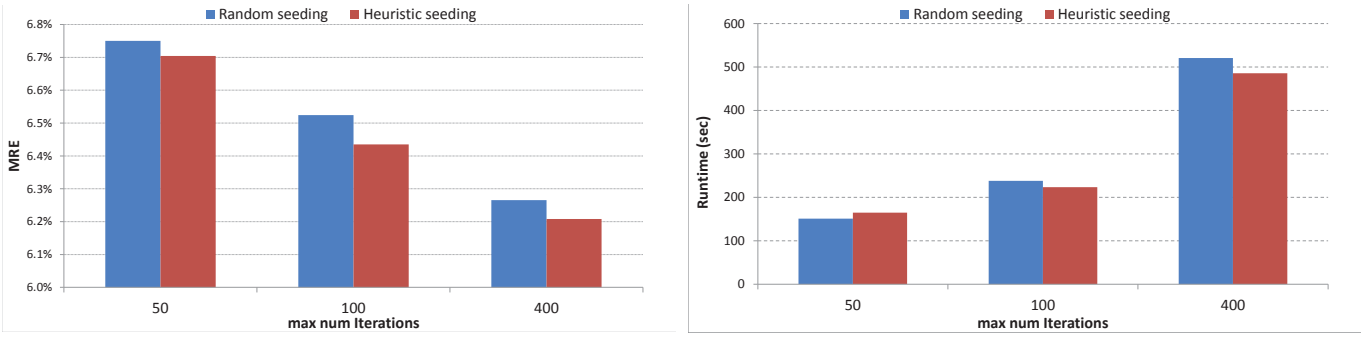



Figure 4: Main Relative Errors on the ***HUdata***.




(a) MRE average values

(b) Average computational time

Figure 5: Comparison between random and heuristic seeding in the new *FfJSP* benchmark.

Table 7: Summary of results in the *fJSP*: ***DPdata***

| Ins | LB | TS | hGA | CDDS | HHS/LNS | GA+TS | HGTS | T(s.) |
|---|---|---|---|---|---|---|---|---|
| 01a | 2505 | 2518 (2528) | 2518 (2518) | 2518 (2525) | **2505** (2513) | **2505** (2511) | **2505**$^b$ (2505) | 122 |
| 02a | 2228 | 2231 (2234) | 2231 (2231) | 2231 (2235) | **2230** (2231) | 2232 (2234) | **2230** (2234) | 205 |
| 03a | 2228 | 2229 (2230) | 2229 (2229) | 2229 (2232) | **2228** (2229) | 2229 (2230) | **2228**$^b$ (2230) | 181 |
| 04a | 2503 | **2503** (2516) | 2515 (2518) | **2503** (2510) | 2506 (2506) | **2503** (2504) | **2503**$^b$ (2503) | 112 |
| 05a | 2189 | 2216 (2220) | 2217 (2218) | 2216 (2218) | **2212** (2215) | 2219 (2221) | 2214 (2218) | 208 |
| 06a | 2162 | 2203 (2206) | 2196 (2198) | 2196 (2203) | **2187** (2192) | 2200 (2204) | 2193 (2198) | 260 |
| 07a | 2187 | 2283 (2298) | 2307 (2310) | 2283 (2296) | 2288 (2303) | **2266** (2286) | 2270 (2280) | 344 |
| 08a | 2061 | **2069** (2071) | 2073 (2076) | 2069 (2069) | **2067** (2074) | 2072 (2075) | 2070 (2074) | 318 |
| 09a | 2061 | **2066** (2067) | **2066** (2067) | **2066** (2067) | 2069 (2073) | **2066** (2067) | 2067 (2069) | 376 |
| 10a | 2178 | 2291 (2306) | 2315 (2315) | 2291 (2303) | 2297 (2302) | 2267 (2273) | **2247**$^a$ (2266) | 369 |
| 11a | 2017 | **2063** (2066) | 2071 (2072) | **2063** (2072) | **2061** (2067) | 2068 (2071) | 2064 (2069) | 294 |
| 12a | 1969 | 2034 (2038) | 2030 (2031) | 2031 (2034) | **2027** (2036) | 2037 (2041) | **2027** (2033) | 486 |
| 13a | 2161 | 2260 (2266) | 2257 (2260) | 2257 (2260) | 2263 (2269) | 2271 (2276) | **2250**$^a$ (2264) | 416 |
| 14a | 2161 | **2167** (2168) | **2167** (2168) | **2167** (2179) | **2164** (2168) | 2169 (2171) | 2170 (2173) | 396 |
| 15a | 2161 | 2167 (2167) | **2165** (2165) | **2165** (2170) | **2163** (2166) | 2166 (2166) | 2168 (2169) | 523 |
| 16a | 2148 | 2255 (2259) | 2256 (2258) | 2256 (2258) | 2259 (2266) | 2266 (2271) | **2246**$^a$ (2257) | 384 |
| 17a | 2088 | 2141 (2144) | 2140 (2142) | 2140 (2146) | **2137** (2141) | 2147 (2150) | 2142 (2146) | 483 |
| 18a | 2057 | 2137 (2140) | 2127 (2131) | 2127 (2132) | **2124** (2128) | 2138 (2141) | 2129 (2133) | 650 |
| *MRE* | | *2.01 (2.24)* | *2.12 (2.19)* | *1.94 (2.19)* | *1.89 (2.13)* | *1.99 (2.17)* | *1.73 (1.98)* | |
| #best | | 5 | 3 | 5 | 12 | 3 | 8 | |

Table 8: Results on the new *FfJSP* benchmark

| Ins | LB | $C_{max}$ | $E[C_{max}]$ | | | T(s.) |
|---|---|---|---|---|---|---|
| | | Best | Best | Avg. | Std. Dev. | |
| fuzzMk10 | 165 | (175, 198 , 225) | 199 | 200.2 | 0.5 | 314 |
| fuzz07a | 2187 | (2105, 2276, 2604) | 2315 | 2330.5 | 8.2 | 440 |
| fuzz08a | 2061 | (1917, 2078, 2368) | 2110 | 2119.7 | 3.5 | 299 |
| fuzz09a | 2061 | (1913, 2071, 2363) | 2105 | 2108.8 | 2.5 | 420 |
| fuzz10a | 2178 | (2074, 2259, 2570) | 2291 | 2312.9 | 12.1 | 489 |
| fuzz11a | 2017 | (1926, 2075, 2360) | 2109 | 2118.7 | 4.3 | 284 |
| fuzz12a | 1969 | (1884, 2042, 2305) | 2068 | 2075.8 | 4.0 | 560 |
| fuzz13a | 2161 | (2111, 2279, 2588) | 2314 | 2329.7 | 9.2 | 365 |
| fuzz14a | 2161 | (2007, 2178, 2483) | 2212 | 2215.7 | 2.6 | 459 |
| fuzz15a | 2161 | (1996, 2177, 2472) | 2206 | 2208.8 | 2.1 | 642 |
| fuzz16a | 2148 | (2067, 2247, 2560) | 2280 | 2313.6 | 14.5 | 394 |
| fuzz17a | 2088 | (1993, 2156, 2443) | 2187 | 2193.1 | 3.2 | 456 |
| fuzz18a | 2057 | (1972, 2136, 2430) | 2169 | 2177.2 | 3.9 | 684 |
| *MRE* | | | *5.66* | *6.22* | | |

tested the resulting algorithm, HGTS, on a varied set of 205 instances, considering both deterministic and fuzzy instances of *fJSP* from the literature to enhance the significance of the study. The extensive experimental results clearly show that not only does the hybrid algorithm benefit from the synergy among its components, improving each of them when run separately for the same time, but it is also quite competitive with the state-of-the-art in solving both crisp and fuzzy instances, providing new best-known solutions for a number of these test instances. Finally, we have argued that the existing *FfJSP* benchmarks are not challenging enough and, in consequence, we have proposed a new more challenging benchmark and we have provided the first makespan results for the new instances with HGTS. We hope that these provide a basis for future research on the *FfJSP* problem.

## Acknowledgements

This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02, COF13-035 and MTM2010-16051 and by the Principality of Asturias (Spain) under grant Severo Ochoa BP13106.

## References

[1] M. L. Pinedo, Scheduling. Theory, Algorithms, and Systems., 3rd Edition, Springer, 2008.

[2] M. Garey, D. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research 1 (2) (1976) 117–129.

[3] W. Herroelen, R. Leus, Project scheduling under uncertainty: Survey and research potentials, European Journal of Operational Research 165 (2005) 289–306.

[4] E. Klerides, E. Hadjiconstantinou, A decomposition-based stochastic programming approach for the project scheduling problem under time/cost trade-off settings and uncertain durations, Computers & Operations Research 37 (2010) 2131–2140.

[5] M. Bruni, F. Beraldi, F. Guerriero, E. Pinto, A heuristic approach for resource constrained project scheduling with uncertain activity durations, Mathematical Methods of Operations Research 38 (2011) 1305–1318.

[6] D. Dubois, H. Prade, P. Smets, Representing partial ignorance, IEEE Transactions on Systems, Man and Cybernetics, Part A 26 (3) (1996) 361–377.

[7] A. Ludwig, R. H. Möhring, F. Stork, A computational study on bounding the makespan distribution in stochastic project networks, Annals of Operations Research 102 (2001) 49–64.
[8] D. Dubois, H. Fargier, P. Fortemps, Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge, European Journal of Operational Research 147 (2003) 231–252.
[9] R. Słowiński, M. Hapke (Eds.), Scheduling Under Fuzziness, Vol. 37 of Studies in Fuzziness and Soft Computing, Physica-Verlag, 2000.
[10] Q. Zhang, H. Manier, M.-A. Manier, A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times, Computers & Operations Research 39 (2012) 1713–1723.
[11] J. Gao, L. Sun, M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35 (2008) 2892–2907.
[12] C. R. Vela, R. Varela, M. A. González, Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times, Journal of Heuristics 16 (2010) 139–165.
[13] J. Puente, C. R. Vela, I. González-Rodríguez, Fast local search for fuzzy job shop scheduling, in: Proceedings of ECAI 2010, IOS Press, 2010, pp. 739–744.
[14] M. González, C. R. Vela, R. Varela, An efficient memetic algorithm for the flexible job shop with setup times, in: Proceedings of the 23th International Conference on Automated Planning and Scheduling (ICAPS-2013), 2013, pp. 91–99.
[15] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & Operations Research 35 (2008) 3202–3212.
[16] C. Y. Zhang, P. Li, Y. Rao, Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, Computers & Operations Research 35 (2008) 282–294.
[17] J. C. Beck, T. Feng, J.-P. Watson, Combining constraint programming and local search for job-shop scheduling, Informs Journal on Computing 23 (2011) 1–14.
[18] E. Nowicki, C. Smutnicki, An advanced tabu search algorithm for the job shop problem, Journal of Scheduling 8 (2005) 145–159.
[19] S. Meeran, M. Morshed, A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study, Journal of Intelligent Manufacturing 23 (2012) 1063–1078.
[20] P. Fortemps, Jobshop scheduling with imprecise durations: a fuzzy approach, IEEE Transactions of Fuzzy Systems 7 (1997) 557–569.
[21] M. Sakawa, R. Kubota, Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms, European Journal of Operational Research 120 (2000) 393–407.
[22] D. Lei, Solving fuzzy job shop scheduling problems using random key genetic algorithm, International Journal of Advanced Manufacturing Technologies 49 (2010) 253–262.
[23] Q. Niu, B. Jiao, X. Gu, Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time, Applied Mathematics and Computation 205 (2008) 148–158.
[24] I. González Rodríguez, J. Puente, C. R. Vela, R. Varela, Semantics of schedules for the fuzzy job shop problem, IEEE Transactions on Systems, Man and Cybernetics, Part A 38 (3) (2008) 655–666.
[25] Y. Zheng, Y. Li, D. Lei, Swarm-based neighbourhood search for fuzzy job shop scheduling, International Journal of Innovative Computing and Applications 3 (3) (2011) 144–151.
[26] J.-q. Li, Y.-x. Pan, A hybrid discrete particle swarm optimization algorithm for solving fuzzy job shop scheduling problem, International Journal of Advanced Manufacturing Technology online first (2012)
[27] Y.-L. Zheng, Y.-X. Li, Artificial bee colony algorithm for fuzzy job shop scheduling, International Journal of Computer Applications in Technology 44 (2) (2012) 124–129.
[28] P. Brucker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing 45(4) (1990) 369–375.
[29] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (1993) 157–183.
[30] S. Dauzère-Pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (3) (1997) 281–306.
[31] M. Mastrolilli, L. Gambardella, Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (2000) 3–20.
[32] A. Hmida, M. Haouari, M. Huguet, P. Lopez, Discrepancy search for the flexible job shop scheduling problem, Computers & Operations Research 37 (2010) 2192–2201.
[33] Y. Yuan, H. Xu, An integrated search heuristic for large-scale flexible job-shop scheduling problems, Computers & Operations Research 40 (2013) 2864–2877.
[34] D. Lei, A genetic algorithm for flexible job shop scheduling with fuzzy processing time, International Journal of Production Research 48 (10) (2010) 2995–3013.
[35] L. Wang, G. Zhou, Y. Xu, L. Min, A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem, International Journal of Production Research 51 (12) (2013) 3593–3608.
[36] S. Wang, L. Wang, Y. Xu, L. Min, An effective estimation of distribution algorithm for the flexible job-shop scheduling problem with fuzzy processing time, International Journal of Production Research 51 (12) (2013) 3779–3793.
[37] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Applied Soft Computing 12 (2012) 2237–2245.
[38] D. Lei, X. Guo, Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling, International Journal of Production Research 50 (6) (2012) 1639–1649.
[39] D. Dubois, H. Prade, Possibility Theory: An Approach to Computerized Processing of Uncertainty, Plenum Press, New York (USA), 1986.
[40] S. Heilpern, The expected value of a fuzzy number, Fuzzy Sets and Systems 47 (1992) 81–86.
[41] M. Sakawa, T. Mori, An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate, Computers & Industrial Engineering 36 (1999) 325–341.
[42] G. Bortolan, R. Degani, A review of some methods for ranking fuzzy subsets, in: D. Dubois, H. Prade, R. Yager (Eds.), Readings in Fuzzy Sets for Intelligence Systems, Morgan Kaufmann, Amsterdam (NL), 1993, pp. 149–158.
[43] G. Celano, A. Costa, S. Fichera, An evolutionary algorithm for pure fuzzy flowshop scheduling problems, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 11 (2003) 655–669.
[44] S.-M. Chen, T.-H. Chang, Finding multiple possible critical paths using fuzzy PERT, IEEE Transactions on Systems, Man, and Cybernetics–Part B: 31 (6) (2001) 930–937.
[45] I. González Rodríguez, C. R. Vela, A. Hernández-Arauzo, J. Puente, Improved local search for job shop scheduling with uncertain durations., in: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-2009), AAAI Press, Thesaloniki, 2009, pp. 154–161.
[46] M. Kuroda, Z. Wang, Fuzzy job shop scheduling, International Journal of Production Economics 44 (1996) 45–51.
[47] D. Lei, Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems, International Journal of Advanced Manufacturing Technology 37 (2008) 157–165.
[48] S. Petrovic, X. Song, A new approach to two-machine flow shop problem with uncertain processing times, Optimization and Engineering 7 (2006) 329–342.
[49] B. Wang, Q. Li, X. Yang, X. Wang, Robust and satisfactory job shop scheduling under fuzzy processing times and flexible due dates, in: Proc. of the 2010 IEEE International Conference on Automation and Logistics, 2010, pp. 575–580.
[50] D. Lei, Fuzzy job shop scheduling problem with availability constraints, Computers & Industrial Engineering 58 (2010) 610–617.
[51] I. González Rodríguez, C. R. Vela, J. Puente, R. Varela, A new local search for the job shop problem with uncertain durations, in: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008), AAAI Press, Sidney, 2008, pp. 124–131.
[52] R. Varela, C. R. Vela, J. Puente, A. Gómez, A knowledge-based evolutionary strategy for scheduling problems with bottlenecks, European Journal of Operational Research 145 (2003) 57–71.
[53] C. Bierwirth, A generalized permutation approach to jobshop scheduling with genetic algorithms, OR Spectrum 17 (1995) 87–92.
[54] I. Essafi, Y. Mati, S. Dauzère-Pérès, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, Computers & Operations Research 35 (2008) 2599–2616.
[55] M. González, C. R. Vela, R. Varela, A competent memetic algorithm for

complex scheduling, Natural Computing 11 (2012) 151–160.
[56] F. Glover, Tabu search–part I, ORSA Journal on Computing 1 (3) (1989) 190–206.
[57] F. Glover, Tabu search–part II, ORSA Journal on Computing 2 (1) (1990) 4–32.
[58] M. Dell' Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, Annals of Operational Research 41 (1993) 231–252.
[59] P. Van Laarhoven, E. Aarts, K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 40 (1992) 113–125.
[60] W. Bozejko, M. Uchronski, M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, Computers & Industrial Engineering 59 (2) (2010) 323–333.
[61] Y. Jin, Surrogate-assisted evolutionary computation: Recent advances and future challenges, Swarm and Evolutionary Computation 1(2) (2011) 61–70.
[62] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278–285.
[63] C. S. Thomalla, Job shop scheduling with alternative process plans, International Journal of Production Economics 74 (2001) 125–134.
[64] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems., IEEE Transactions on System, Man, and Cybernetics-Part C. Applications and Reviews 32 (1) (2002) 1–13.
[65] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic., Mathematics and Computers in Simulation 60 (2002) 245–276.
[66] J. Barnes, J. Chambers, 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.
[67] E. Hurink, B. Jurisch, M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machine, Operations Research Spektrum 15 (1994) 205–215.
[68] L. Xing, Y. Chen, P. Wang, Q. Zhao, J. Xion, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Applied Soft Computing 10 (3) (2010) 888–896.
[69] J. Li, Q. Pan, P. Suganthan, T. Chua, A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, International Journal of Advanced Manufacturing Technology 52 (5-8) (2011) 683–697.
[70] L. Wang, G. Zhou, Y. Xu, L. Min, An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, International Journal of Advance Manufacturing Technology 60 (2012) 303–315.
[71] L. Wang, S. Wang, G. Zhou, L. Min, A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem, Computers & Industrial Engineering 62 (2012) 917–926.
[72] B. S. Girish, N. Jawahar, A particle swarm optimization algorithm for flexible job shop scheduling problem, in: Proceedings of the 5th Annual IEEE Conference on Automation Science and Engineering, 2009, pp. 298–303.
[73] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power., Information Sciences 180 (2010) 2044–2064.