Universidad de Oviedo

DEPARTAMENTO DE INFORMÁTICA

Estrategias Metaheurísticas para Scheduling bajo Incertidumbre

(Metaheuristic Strategies for Scheduling under Uncertainty)

Tesis Doctoral

Programa de Doctorado en Ingeniería Informática

Autor: Juan José Palacios Alonso

Directoras:
María Camino Rodríguez Vela
Inés González Rodríguez

Mayo, 2015

A los que siempre han estado ahí

Agradecimientos (Aknowledgements)

A mis directoras de tesis, María Camino Rodríguez Vela e Inés González Rodríguez, y a Jorge Puente Peinador, por su confianza, su paciencia y toda su ayuda durante estos años, y por haberme enseñado todo lo que sé. A todos los miembros del grupo iScOp, por su amistad y por hacerme sentir como uno más desde el primer momento.

Al profesor El-Ghazali Talbi por acogerme en su grupo de investigación y permitirme enriquecer mi experiencia profesional y al profesor Bilel Derbel por toda su ayuda e interés durante mi estancia. A los miembros del grupo DOLPHIN, por hacer tan agradable mi tiempo fuera de mi ciudad.

To professor El-Ghazali Talbi for hosting me in his research team and allowing me to improve my career and to professor Bilel Derbel for his help and interest. To the members of team DOLPHIN, for making so nice my stay far from my hometown.

A mi familia y amigos, en especial a mis padres María Esther y Andrés y a mi hermano Javier. A ellos les debo todo.

Resumen

Los problemas de scheduling son problemas de tipo combinatorio con gran presencia en la literatura durante las últimas décadas. Estos problemas son NP-completos, lo que supone un gran reto para los investigadores en Inteligencia Artificial. En su definición clásica, se asume que todos los datos son conocidos y precisos. Sin embargo, este no es el caso en situaciones reales, especialmente cuando hay factores humanos involucrados. Por ejemplo, es común que los tiempos exactos de procesamiento de cada tarea no se conozcan a-priori. La incertidumbre presente en el mundo real puede afectar drásticamente a los resultados que se obtienen asumiendo condiciones ideales. Esto ha llevado a muchos investigadores a considerar la incertidumbre como parte del problema. Entre las diversas maneras de modelar esta incertidumbre, los conjuntos difusos plantean una muy buena alternativa, lo que hace que su uso se esté extendiendo cada vez más. Cuando la incertidumbre se modela mediante números fuzzy, nos referimos a los problemas de scheduling como fuzzy scheduling. Concretamente, esta tesis se centra en el uso de los números fuzzy más comunes en este tipo de problemas: los números fuzzy triangulares o TFNs. Además, los problemas de scheduling presentan en muchas ocasiones fechas de entrega o due-dates. Estas fechas son generalmente estrictas, pero lo más común en escenarios reales es que sean flexibles. En esta tesis se considera el uso de fechas de entrega flexibles y tiempos de procesamiento fuzzy para tres problemas: fuzzy open shop, fuzzy job shop y fuzzy flexible job shop.

De entre los distintos métodos de resolución que se pueden encontrar en la literatura, esta tesis se centra en las metaheurísticas, las cuáles son capaces de encontrar soluciones de gran calidad en muy poco tiempo. Para que su comportamiento sea óptimo, es muy importante acotar el conjunto de soluciones entre las que buscar la óptima. Si bien estos conjuntos han sido muy estudiados en el problema clásico, no ocurre lo mismo en los problemas de fuzzy scheduling. En esta tesis se propone la primera definición formal de categorías de schedules y esquemas de generación de schedules en el entorno fuzzy. En base a estos esquemas, proponemos varios métodos de resolución para los mencionados problemas. En concreto, estos métodos se centran en la optimización de la función objetivo más común: el makespan. En base a los resultados obtenidos con cada método se plantean algunas guías sobre cómo resolver de forma eficiente problemas de fuzzy scheduling. Además de lo anterior, se proponen también métodos de optimización multi-objetivo que, además del makespan, optimicen el agreement index, que es el grado de cumplimiento de las fechas de entrega flexibles. Se tienen en cuenta tanto el caso en el que las dos funciones tienen la misma prioridad, como el caso en el que una es más importante que la otra.

Cuando hay incertidumbre en un problema, la robustez es un concepto de gran importancia. En muchos casos puede ser preferible tener soluciones robustas con una calidad razonable, a tener soluciones de gran calidad cuyo resultado se ve gravemente afectado cuando hay una mínima variación en los datos de entrada. Aunque esto es algo de vital importancia, no es sencillo de medir. En la literatura se puede encontrar una definición de semántica en problemas de fuzzy scheduling. En base a esta definición, en esta tesis se propone un nuevo marco en el que poder medir la robustez, distinguiendo entre dos tipos de medidas: las medidas a-priori, que pueden ser tomadas antes de la implantación de la solución en un entorno real, y las medidas a-posteriori, que no pueden ser evaluadas

antes de su implantación. En este marco, proponemos distintas medidas para la robustez, las cuáles nos permiten hacer comparaciones que no eran posibles con otro tipo de medidas. Finalmente, proporcionamos varios métodos para optimizar estas medidas, solas, o conjuntamente con una medida de calidad como el makespan.

Summary

Scheduling problems are a kind of combinatorial problems that pose a great challenge to Artificial Intelligence researchers, being very hard to solve. They are well-known NP-complete problems which have had a great presence in the literature during the last decades. However, in the classical definition of these problems, well-defined information is assumed, which is not the case in most of real-world scenarios, especially when human factors are involved. For instance, the exact processing times of operations may be unknown in advance. The uncertainty that is present in real situations can drastically deteriorate the performance of a solution obtained assuming deterministic conditions. This has led many researchers to consider this uncertainty as part of the problem to solve. Among the different approaches to model the uncertainty, fuzzy sets have emerged as a very interesting tool and have been extensively used in different manners. When uncertainty is modelled by means of fuzzy numbers, we refer to fuzzy scheduling problems. In particular, for this PhD thesis we use the most extended representation in fuzzy scheduling, which is triangular fuzzy numbers or TFNs. Scheduling problems may have also due date constraints, which represent deadlines for each job. These are usually taken to be hard constraints, but in real scenarios, it is more common for them to be flexible. In this thesis we consider the use of flexible due dates together with uncertainty in processing times in three different scheduling problems: fuzzy open shop, fuzzy job shop and fuzzy flexible job shop.

Among the different solving methods in the literature, we focus on metaheuristics, which are known to find very good solutions in a short amount of time. For these algorithms to perform well, it is important to define good sets of solutions in which look for the optimal solution. These are well defined for classical scheduling problems, but surprisingly, this is not the case in fuzzy scheduling. Here we provide a first formal definition of schedule categories and schedule generation schemes for fuzzy scheduling problems. This allows us to propose efficient solving methods for these problems. In particular, we consider the most common optimisation criteria, which is the makespan minimisation. Based on the ideas behind the best-known algorithms for the classical scheduling problems, we design new strategies for fuzzy scheduling and provide some insights and hints about the most promising ways of solving these problems. Furthermore, we also propose multi-criteria optimisation methods to minimise the makespan and maximise the agreement index, which is the degree to which the flexible due dates of jobs are fulfilled. We consider both the case in which one objective is given more relevance than the other and the case in which both objectives are equally relevant.

An issue of great importance when scheduling under uncertainty is the robustness of solution quality. Indeed, solutions which are robust and reasonably good in quality may be preferred to solutions which in principle are optimal but perform poorly when slight variations in the input data occur. This is a factor that must be taken into account, but which is difficult to translate into a well-defined measure. Using as starting point a definition of semantics for fuzzy scheduling from the literature, we propose a new framework to measure robustness which distinguishes between two kinds of measures: a-priori measures, which are obtained before implementing a solution in a real environment, and a-posteriori measures, that are obtained after implementing the solution. Based on this framework, we propose different robustness measures for fuzzy scheduling. We also propose optimisa-

tion algorithms that allow optimising these robustness measures alone or together with a performance measure such as makespan in order to obtain high quality robust solutions.

Contents

Part I

Report

Chapter 1

Introduction

1.1 Scheduling

Scheduling problems are a kind of combinatorial problems that pose a great challenge to Artificial Intelligence researchers, being very hard to solve [87]. They are well known NP-complete problems which have had a great presence in the literature during the last decades [16, 87]. Roughly speaking, a scheduling problem consists in assigning a set of resources to a set of tasks or operations fulfilling some constraints and optimising one or more objective functions. Seeing this generic definition, it is easy to guess that these problems are present in many real situations ranging from day-to-day ones to scenarios with a high economical impact. Some typical examples are assigning work shifts and activities at work centres, optimising the production of factories or optimising the performance of machines [87].

In the following, we will assume that a scheduling problem consists in scheduling a set of n jobs $J = \{J_1, \ldots, J_n\}$ on a set of m physical resources or machines $M = \{M_1, \ldots, M_m\}$, subject to a set of constraints. Each job J_i is composed by a set of n_i tasks θ_{ij} requiring the use of a machine for its whole processing time p_{ij}. A feasible schedule is an allocation of starting times S_{ij} for each task θ_{ij} such that a set of constraints hold. The objective is to find a schedule which is *optimal* according to some criterion.

Depending on the set of constraints considered, different scheduling problems may be defined. In this document we shall focus in two of the so-called shop scheduling problems, which are among the most popular ones in the literature:

- Job Shop Scheduling Problem (JSP):
  - Capacity constraints: Each task θ_{ij} requires the uninterrupted and exclusive use of one fixed machine $\eta_{ij} \in M$.

    $$S_{ij} \geq C_{rs} \vee C_{ij} \leq S_{rs}, \qquad \forall r, s\, r \neq i \vee s \neq j : \eta_{ij} = \eta_{rs} \tag{1.1}$$

    where $C_{ij} = S_{ij} + pij$, $1 \leq i \leq n, 1 < j \leq n_i$ denotes the completion time of task θ_{ij}.
  - Precedence constraints: The tasks θ_{ij} belonging to job J_i are to be sequentially scheduled following a fixed order.

    $$\begin{aligned} &S_{ij} \geq C_{ij-1}, && 1 \leq i \leq n, 1 < j \leq n_i \\ &S_{i1} \geq 0, && 1 \leq i \leq n \end{aligned} \tag{1.2}$$

- Open Shop Scheduling Problem (OSP):

– Precedence constraint: Two operations belonging to the same job cannot overlap their execution in time:

$$S_{ij} \geq C_{ir} \vee C_{ij} \leq S_{ir}, \qquad 1 \leq j, r \leq n_i \, r \neq j \tag{1.3}$$

– Capacity constraint: It has the same constraints as in JSP (Const. 1.1):

Several variants of these problems can be defined. In particular, we will consider the Flexible Job Shop Scheduling Problem (fJSP), as its popularity has been increasing during the last years. This is a generalization of the JSP, in which each task θ_{ij} can be scheduled in a set of machines $M_{ij} \subset M$ instead of having to be processed in one fixed machine. Furthermore, the processing time of task θ_{ij} will vary depending on the machine $\eta \in M_{ij}$ where it is scheduled, denoted p_{ij}^{η}.

Although there are many optimisation criterion that can be taken into account in scheduling problems, the most common one in the literature is the minimisation of the makespan C_{max}, which is the time it takes to finish all jobs:

$$C_{max} = \max_{i=1\ldots n} \{C_i\} \tag{1.4}$$

where $C_i = \max_{j=1\ldots n_i}\{C_{ij}\}$ denotes the completion time of job J_i.

In real situations it is quite common to find due dates, that is, each job J_i has a deadline D_i. When these constraints are present, other optimisation criteria can be considered. In this case, a common one is the minimisation of the maximum tardiness (T_{max}), which measures the maximum delay that a job has with respect to its due date:

$$T_{max} = \max_{i=1\ldots n} \{0, C_i - D_i\} \tag{1.5}$$

Let us illustrate these concepts with an example. Here we have a job shop scheduling problem with $n = 3$ jobs and $m = 2$ machines with processing times, machine assignment and due dates as follows:

$$p = \begin{pmatrix} 4 & 4 \\ 5 & 3 \\ 2 & 4 \end{pmatrix} \qquad \eta = \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 12 \\ 6 \\ 9 \end{pmatrix} \tag{1.6}$$

Now we schedule the operations following the order $\theta_{11}, \theta_{21}, \theta_{31}, \theta_{22}, \theta_{32}, \theta_{12}$. Operation θ_{11} can be scheduled with $S_{11} = 0$ and $C_{11} = S_{11} + p_{11} = 0 + 4 = 4$, and similarly operation θ_{21} is scheduled with $S_{21} = 0$ and $C_{21} = 5$. Now operation θ_{31} cannot be scheduled before C_{21} because of the capacity constraints (Const. 1.1), so $S_{31} = C_{21} = 5$ and $C_{31} = 5 + 2 = 7$. The next operation to be scheduled is θ_{22} which cannot begin before θ_{21} because of the precedence constraint (Const. 1.2) nor before θ_{11} because of capacity constraints, therefore $S_{22} = \max\{C_{11}, C_{21}\} = \max\{4, 5\} = 5$. If we continue, we obtain the completion times for each job $C_1 = C_{12} = 11$, $C_2 = C_{22} = 8$, $C_3 = C_{32} = 12$ and we can compute different objective functions. For instance, the makespan in this case is 12, and the maximum tardiness is 3.

The most natural way to represent a schedule is a Gantt Chart. In Figure 1.1 we see the Gantt Chart for the previous example. Red dotted lines represent the due dates and the colours of the tasks represent the machine to which they are assigned. However, the most useful representation to work with schedules are graphs. Figure 1.2 contains the directed graph corresponding to the previous example. Each node represent a task of the problem, with the exception of the dummy nodes start or s and end or e, representing tasks with null processing times. The black arcs represent the precedence constraints and the coloured ones the capacity constraints. Each arc is weighted with the processing time of the task at its source node, so for instance the makespan can be computed as the longest path from s to e.

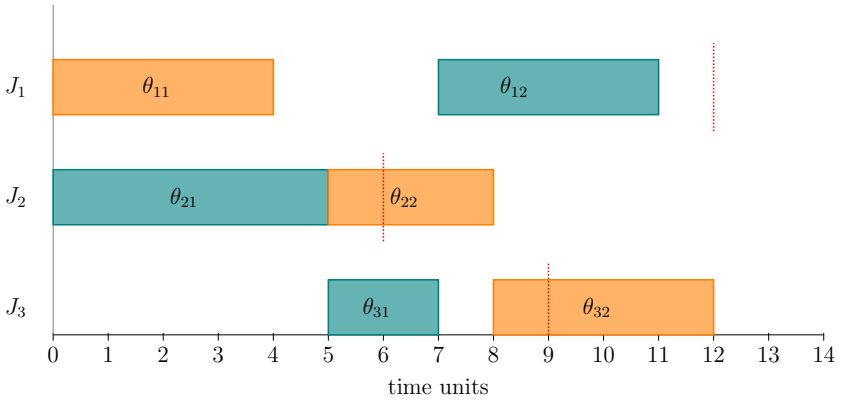


Figure 1.1: Solution of a scheduling problem (Gantt chart).

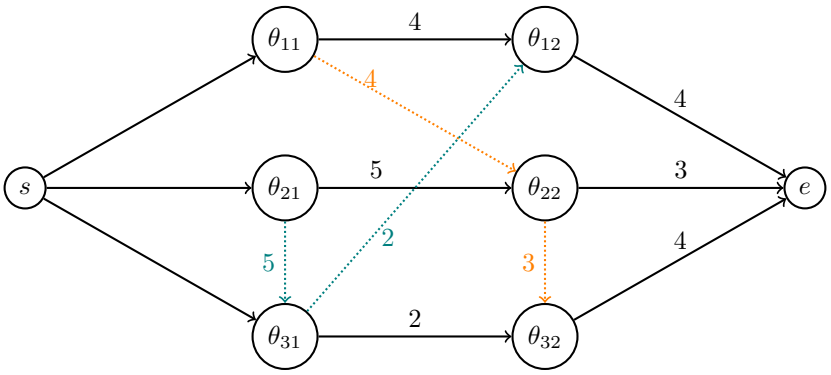


Figure 1.2: Solution of a scheduling problem (Graph).

1.2 Uncertainty and robustness

Notice that in the classical definition of the problem, well-defined information is assumed, which is not the case in most of real-world scenarios, specially when human factors are involved. The uncertainty that is present in real situations can drastically deteriorate the performance of a solution obtained assuming deterministic conditions. This has led many researchers to consider this uncertainty as part of the problem to solve [50]. Among the different ways of dealing with the uncertainty, two approaches have become more popular: stochastic scheduling and fuzzy scheduling. Stochastic scheduling is the best known approach to dealing with uncertainty in scheduling. It uses probability distribution functions, usually minimising the expected costs, the deviation w.r.t. the best expected cost or maximising the probability of the cost to be under a given threshold [10, 50, 87]. The main disadvantages of this approach are that it requires a great amount of information in advance to model the data correctly and it yields very complex computations in practice. On the other hand, the fuzzy approach has emerged as a very interesting tool and has been extensively used in different manners, ranging from representing incomplete or vague states of information to using fuzzy priority rules with linguistic qualifiers or preference modelling [31, 96, 115]. Fuzzy scheduling states a promising alternative to the stochastic approach and it is being used by an increasing number of authors. Instead of probability distribution functions, this approach models the uncertainty using fuzzy sets, which represent possibility distribution functions. As a counterpart to stochastic scheduling, this may be a less accurate approach, but has the advantage that it can be applied when having only some vague information, which is the most common situation. Furthermore, the simplicity of the fuzzy model allows to build solving methods that are less computationally demanding than the methods required for stochastic scheduling. According to [31], "the fuzzy set and possibility theory may help building a trade-off between the expressive power and the computational difficulties of stochastic scheduling techniques while tackling uncertainty and accounting for local specifications of preferences".

An issue of great importance is the robustness of solution quality when scheduling under uncertainty. Indeed, solutions which are robust and reasonably good in quality may be preferred to solutions which in principle are optimal but perform poorly when slight variations in the input data occur. Take for instance the case where the manager of a corporation is given two alternative business plans. The first one offers an expected profit of 10,000€, but depending on the actual conditions that are present when the plan is executed, the final profit may range between 3,000€ and 17,000€. Instead, the second alternative has an expected profit of 9,500€ but it ensures that, whatever the conditions during its execution, the profit will always be between 9,000€ and 10,000€. Depending on the risks the manager is willing to take, he/she may prefer the second plan to the first "optimal" one. In general, the trade-off between the expected performance quality and solution robustness should be taken into account. It is however difficult to translate the idea of robustness into a well-defined measure. Indeed, it is possible to find in the literature numerous interpretations and definitions of related measures when it comes to modelling the robustness of a schedule [3, 5, 10, 116].

1.3 Metaheuristics

Due to the complexity of scheduling problems, they are usually tackled by means of Artificial Intelligence techniques. We can roughly split these techniques in two main groups: exact techniques and metaheuristics. The former are focused on finding the optimal solution to the problem according to some criteria. These approaches are however computationally expensive, either in terms of computation time or memory usage. On the other hand,

metaheuristics are optimisation algorithms that allow to find very good solutions in a short amount of time. Even though they do not guarantee finding the optimal solution for the problem, their good performance has made them very popular, and many of them have been developed during the last decades [37, 101]. Probably the most extended metaheuristics are genetic algorithms [51], which keep a set of potential solutions to the problem, and through recombination and replacement strategies are able to make those solutions move towards better ones. This is a so-called population-based metaheuristic, but there are many others as for example Particle Swarm Optimisation [88], Scatter Search [72] or Ant Colony [23, 29]. There also exist metaheuristics that instead of using a pool of solutions, take one solution and follow an iterative process to improve its quality. This is done for instance in local search algorithms. These are based on the use of neighbourhoods, which are sets of solutions that can be obtained after performing a small change in an original solution or starting point. Once a neighbourhood is defined, the algorithm will examine it and take a new solution therein as starting point for the next iteration until some stopping criterion is met. As in the case of population-based strategies, depending on how we define the steps to follow we can find many approaches, being the most popular ones Hill Climbing and Tabu Search [39].

In summary, a wide range of metaheuristic algorithms can be found in the literature, each having its strong and weak points. Additionally, during the last years it is becoming popular to combine or hybridise different metaheuristics to exploit their strong points while minimising their shortcomings [13, 12, 102]. A good example is memetic algorithms [22, 108], which are a hybrid between a genetic algorithm and a local search strategy. The former is a very good strategy to explore different solutions but quite poor at exploiting them, whereas the latter has a great level of exploitation but lower exploration capability. The combination of both often results in a much better performance than any of its components used separately.

Metaheuristics can also be used for solving multiobjective optimisation problems [101, 103]. These are problems where the algorithm tries to find solutions which are optimal according to more than one criterion. If the criteria are perfectly correlated this is not a problem, but usually it is impossible to find a solution that is optimal for more than one criteria at the same time. Many approaches can be found in the literature to solve multi-criteria optimisation problems. For instance, if the different criteria are not equally relevant, then the multiobjective problem can be solved as a single-criteria problem in which the objective function is a weighted sum of the different objectives, or a hierarchy can be established between the different criteria following a so-called lexicographical approach. On the other hand, when there is no difference in the relevance of the criteria, the most common approach is to compute a set of non-dominated solutions instead of just one solution. Roughly speaking, this is a set of solutions in which we cannot clearly determine which one is the best. This situation appears when one solution is better than another regarding one criteria, but it is worse regarding another one.

Chapter 2

Objectives

The main target of this thesis is to solve scheduling problems under uncertain circumstances by means of metaheuristic techniques. Scheduling optimisation problems, both real problems and academic ones, have such a great complexity that even using classical metaheuristic techniques we need large budgets of time to find high-quality solutions. Furthermore, when we model the uncertainty that is present in real situations and take it into account during the optimisation process, the complexity of the problem increases. Because of this, it is needed to develop new solving methods to deal with the uncertainty and find good solutions in reasonable amounts of time. Additionally, the robustness appears as an important factor to take into account. This factor can be optimised alone, but it is not really useful to have very robust solutions with a very low quality. Therefore we shall try to find solutions that are robust but also have a high level of quality.

All the above motivates the following list of more detailed goals for this thesis:

1. Study and analyse different approaches to model uncertainty through the use of fuzzy numbers. Adopt new analytic measures of optimality for expected quality and robustness in uncertain environments, paying special attention to the relation between robustness and expected performance.

2. Study the effects of adding uncertainty. Asses if previous studies for classical scheduling problems are still valid in the case that uncertainty is modelled.

3. Design (hybrid) metaheuristics for fuzzy scheduling problems to optimise classical objective functions (e.g. makespan, tardiness, etc.), taking into account the particularities of the uncertainty model.

4. Adapt multiobjective algorithms to solve fuzzy scheduling problems where robustness shall be considered as an objective function together with the performance metric.

5. Evaluate the performance of the developed methods on already published benchmark instances, to compare them with the state-of-the-art methods.

Chapter 3

Uncertainty in scheduling problems

In this thesis we consider the most common source of uncertainty in the literature, which is the uncertainty in the processing times of the operations. In addition, we shall consider flexibility in the due-date constraints. These are usually hard constraints in classical scheduling problems and when they are present, it is common to try to optimise objective functions such as already mentioned maximum tardiness. However, flexible due dates allow to model the satisfaction degree of a customer depending on the delay of the job. This is a natural approach, as customers are usually flexible regarding delays in the delivery of a job.

3.1 Fuzzy processing times

In real-world situations, it is often the case that some factors are unknown a-priori by the analyst. This usually leads to the exact processing times of the operations not being known in advance. However, an expert may be able to roughly approximate them based on his major experience. For instance, he/she may establish an interval of possible values for the processing time and even assess whether some values in the interval appear to be more plausible than others. When there is little knowledge available, the crudest representation for uncertain processing times would be a human-originated confidence interval. If some values appear to be more plausible than others, a natural extension is a fuzzy interval or fuzzy number, which have been extensively studied in the literature (cf. [34]). The simplest model is a *triangular fuzzy number* or *TFN*, using an interval $[a^1, a^3]$ of possible values and a modal value a^2. That is, for a TFN A denoted $A = (a^1, a^2, a^3)$, the membership function takes a triangular shape as shown in Figure 3.1 completely determined by the three real numbers as follows:

$$\mu_A(x) = \begin{cases} \frac{x-a^1}{a^2-a^1} & : a^1 < x \leq a^2 \\ \frac{x-a^3}{a^2-a^3} & : a^2 < x < a^3 \\ 0 & : x \leq a^1 \text{ or } a^3 \leq x \end{cases} \tag{3.1}$$

For $\alpha \in (0, 1]$, its α-cut $A_\alpha = \{x : \mu_A(x) \geq \alpha\}$ is a closed interval $[\underline{a}_\alpha, \overline{a}_\alpha]$; we shall abuse notation slightly and denote its support as A_0.

TFNs are to date the most widely used model for uncertain durations in the fuzzy scheduling literature [1]. Furthermore, notice that any real number $r \in \mathbb{R}$ can be seen as a special case of TFN where $A = (r, r, r)$; this allows us to deal with problems where tasks have both uncertain and precise processing times. However, when TFNs are to be used to extend scheduling problems to handle uncertainty, two issues must be addressed:

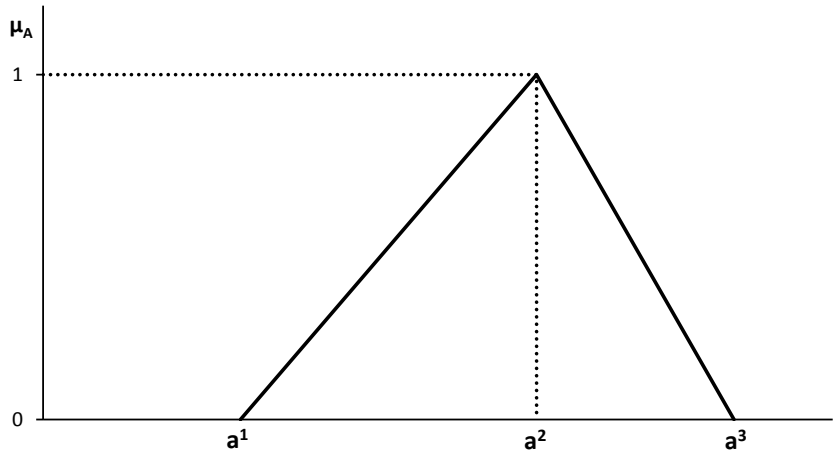


Figure 3.1: Membership function of a triangular fuzzy number.

the means of extending the arithmetic operations of addition and maximum to work with TFNs (to compute starting and completion times) and the establishment of an order relation between TFNs.

3.1.1 Order relations for fuzzy numbers

The fact that there is no natural total ordering in the set of TFNs makes concepts like "minimal makespan" ambiguous. Hence, in order to compare different TFNs, several ranking methods have been and keep being proposed in the literature (cf. [18, 107, 114]). Furthermore, quoting Brunelli and Mezei [18],

> It is impossible to give a final answer to the question on what ranking method is the best. Most of the time, choosing a method rather than another is a matter of preference or is context dependent.

Let $\mathcal{F}$ denote the set of fuzzy numbers. Ranking methods in $\mathcal{F}$ can be roughly divided in two types: those based on "defuzzification" and those based on fuzzy binary relations. In the first case, a mapping $M : \mathcal{F} \to \mathbb{R}$ is defined which associates each fuzzy number A with a real number and then the natural ordering on the real line is used, most commonly, $A \leq_M B$ iff $M(A) \leq M(B)$. In the second case, a relation $M : \mathcal{F} \times \mathcal{F} \to [0, 1]$ is defined such that $M(A, B)$ is the degree to which A is greater than B and, consequently, if $M(A, B) \geq M(B, A)$, then $A \geq_M B$. When it comes to comparing fuzzy quantities in the context of fuzzy scheduling (e.g. compare makespan values), the most common approach in the literature is the former.

The membership function μ_A of a fuzzy quantity A can be interpreted as a possibility distribution on the real numbers; this allows to define the *expected value* of a fuzzy quantity [49], given for a TFN A by

$$E[A] = \frac{1}{4}(a^1 + 2a^2 + a^3). \tag{3.2}$$

This mapping induces a total ordering $\leq_E$ in the set of fuzzy intervals [36], where for any two fuzzy intervals A, B:

$$A \leq_E B \leftrightarrow E[A] \leq E[B] \tag{3.3}$$

Clearly, for any two TFNs A and B, this property holds:

$$\forall i, a^i \leq b^i \rightarrow A \leq_E B. \tag{3.4}$$

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 [31].

Related to this, there is a ranking method widely used in the fuzzy scheduling literature following the seminal papers [93] and [94]. In this ranking method, the criterion for dominance is one of the following three in the order given below:

- Criterion 1 (c_1): The greatest associate ordinary number (expected value) is used as a first criterion.
- Criterion 2 (c_2): If c_1 does not rank the two TFNs, those which have the best maximal presumption (the mode) will be chosen as a second criterion.
- Criterion 3 (c_3): If c_1 and c_2 do not rank the TFNs, the difference of the spreads will be used as a third criterion.

So we can summarise this ranking method as follows:

$$A \leq_R B \leftrightarrow \begin{cases} E[A] < E[B] \\ E[A] = E[B] \wedge a^2 < b^2 \\ E[A] = E[B] \wedge a^2 = b^2 \wedge (a^3 - a^1) \leq (b^3 - b^1) \end{cases} \tag{3.5}$$

Even though these are the most common ranking methods in fuzzy scheduling, other methods can be found. For example in [6] the authors propose a new dominance criterion between TFNs that ranks fuzzy quantities depending on the overlapping existing between their membership functions, or in [74] the authors adapt a different ranking for solving the fuzzy job shop. Furthermore, different ranking methods have been considered to solve a flexible job shop problem with uncertainty in the paper [80] that is part of this thesis (see Sections 4.3.2 and 7.1).

3.1.2 Arithmetic with fuzzy numbers

In scheduling problems, we essentially need two operations on fuzzy numbers, the addition and the maximum. These are obtained by extending the corresponding operations on real numbers using the *Extension Principle*. In the case of the addition, it turns out that for any pair of TFNs A and B, the resulting expression is:

$$A + B = (a^1 + b^1, a^2 + b^2, a^3 + b^3). \tag{3.6}$$

Unfortunately, computing the resulting expression of the maximum is not that straightforward and, most importantly, the set of TFNs is not closed under this operation. For the sake of simplicity and tractability of numerical calculations, it is fairly common in the literature, following [36], to approximate the maximum by a TFN, evaluating only the operation on the three defining points, that is, for every A, B TFNs:

$$\max(A, B) \approx \max_I(A, B) = (\max(a^1, b^1), \max(a^2, b^2), \max(a^3, b^3)) \tag{3.7}$$

Some arguments can be given to support this approximation. First, for any two fuzzy numbers A and B, if f is a bivariate continuous isotonic function, then $F = f(A, B)$ is another fuzzy number such that

$$\forall \alpha \in [0, 1], F_\alpha = [f(\underline{a}_\alpha, \underline{b}_\alpha), f(\overline{a}_\alpha, \overline{b}_\alpha)]. \tag{3.8}$$

Computing $f(A, B)$ is then equivalent to computing f on every α-cut. In particular, the maximum is a continuous isotonic function, so it can be calculated by evaluating two

maxima of real numbers for every value $\alpha \in [0,1]$. If seems then natural to approximate the maximum by the TFN that results from using linear interpolation, evaluating equation (3.8) only for certain values of α (this is proposed for 6-point fuzzy numbers in [36]). Given that the defining values (a^1, a^2, a^3) of a TFN A are such that $A_0 = [a^1, a^3]$ and $A_1 = [a^2, a^2]$, the approximated maximum as in (3.7) corresponds to such an interpolation for $\alpha = 0$ and $\alpha = 1$. Secondly, if $F = \max(A, B)$ denotes the maximum of two TFNs A and B and $G = \max_I(A, B)$ the approximated value by interpolation, then $F = G$ if A and B do not overlap and, in any case, it holds that

$$\forall \alpha \in [0,1], \quad \underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha. \tag{3.9}$$

The approximated maximum G is thus a TFN which artificially increases the value of the actual maximum F, but maintaining the support and modal value, that is, $F_0 = G_0$ and $F_1 = G_1$. This approximation can be trivially extended to the case of more than two TFNs. It has been widely used in the scheduling literature, among others, in [20, 36, 44, 60, 93] or [110].

More recently, it has been proposed in [61] to approximate the maximum of two TFNs A and B by means of the above ranking method, so:

$$\max(A, B) \approx \max_R(A, B) = \begin{cases} A & : A <_R B \\ B & : B <_R A \end{cases} \tag{3.10}$$

Notice that with this approximation it is not guaranteed that the approximated maximum maintains the support nor the modal value. In other words, if we consider two crisp values within the supports of the two TFNs, the maximum of these crisp values is not guaranteed to be in the support of the approximate maximum. However, in [61] and [62] some examples are considered which lead the author to conclude that "the approximate max obtained by the new criterion approaches the real max better than that obtained from" $\max_I$. Since then, this alternative approximation for the maximum has been adopted among others in [64, 112, 121] and [122].

Using an approximation that keeps the support of the real maximum has some desirable properties in the case of scheduling, as we shall see for instance in Section 3.3. Based on this, we adopt the use of $\max_I(A, B)$ for the development of the thesis and refer to as $\max(A, B)$ (for the sake of simplicity), unless the opposite is explicitly stated. It is important to notice that, for any two TFNs A and B, if $M_I = \max_I(A, B)$ is the maximum approximated by interpolation and $M_R = \max_R(A, B)$ is the maximum approximated by the ranking method, it is always the case that $m_R^i \leq m_I^i$ for $i = 1, 2, 3$ and, hence, $M_R \leq_E M_I$ and $M_R \leq_R M_I$. This will be important when comparing results with authors that use the $\max_R(A, B)$ approximation.

3.2 Flexible due dates

In the case in which due-date constraints are present in real situations, they are often flexible. For instance, a customer may have a preferred delivery date d^1 for his job, but he will allow some delay until a later date d^2, after which he will be completely unsatisfied and will potentially cancel the order. The satisfaction of a due-date constraint becomes a matter of degree, our degree of satisfaction that a job is finished on a certain date. A natural approach to modelling such satisfaction levels is to use a fuzzy set $D = (d^1, d^2)$ with linear decreasing membership function as follows:

$$\mu_D(x) = \begin{cases} 1 & : x \leq d^1 \\ \frac{x - d^2}{d^1 - d^2} & : d^1 < x < d^2 \\ 0 & : d^2 \leq x \end{cases} \tag{3.11}$$

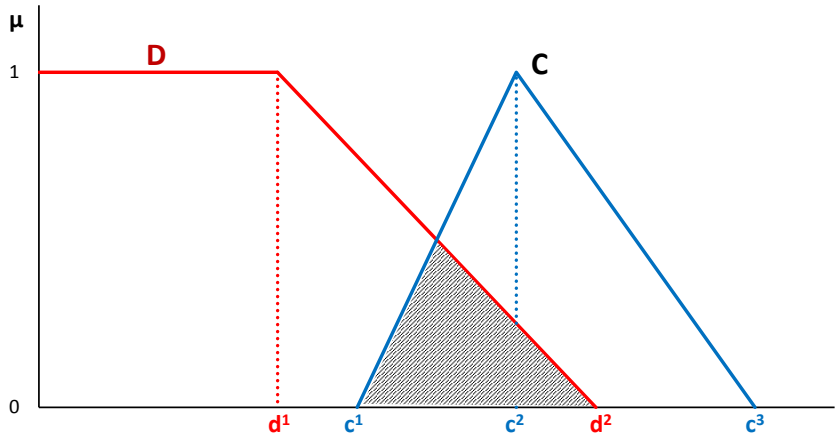


Figure 3.2: Degree to which a TFN C satisfies the flexible due date D.

A different definition is used for instance in [66, 117] and [118], where the authors propose that finishing a job too early may be as undesirable as finishing it too late, so a trapezoidal membership function is used. For this thesis, we follow the ideas from [31], where it is said that the above membership function expresses a flexible threshold "less than" representing the satisfaction degree of the customer for the job finishing at time x. However, notice that in our case the completion time of the jobs may be not a real number, but a fuzzy quantity. When the completion time of a job J_i is a TFN C_i, the degree to which C_i satisfies its due-date constraint D_i is usually measured by the *agreement index* (AI) [20, 58, 70, 93]. This degree is illustrated in figure 3.2 and is computed as follows:

$$AI(C, D) = \frac{Area(C \cap D)}{Area(C)} \tag{3.12}$$

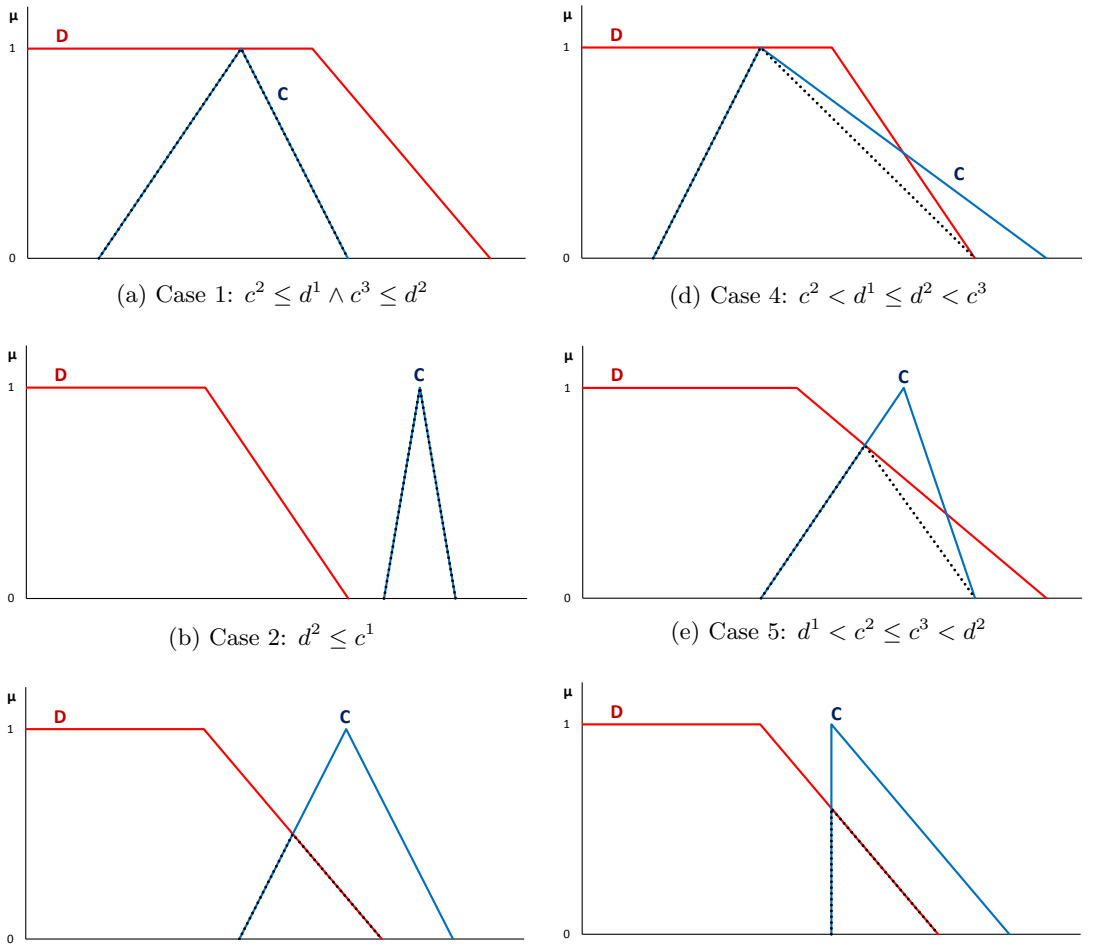
where $Area(D \cap C)$ and $Area(C)$ denote the areas under the membership functions of $(D \cap C)$ and C respectively. The intuition behind this definition is to measure the degree to which C is contained in D (the degree of subsethood). Notice that the flexible due date D is completely satisfied when $AI = 1$ (that is, $Area(C \cap D) = Area(C)$) and unsatisfied when $AI = 0$ ($C \cap D = \emptyset$). In the case that C is a TFN, the above formula is equivalent to:

$$AI(C, D) = \frac{2\, Area(C \cap D)}{c^3 - c^1} \tag{3.13}$$

Obviously, if $c^1 = c^3$ then C is a real value and the satisfaction degree is obtained as $\mu_D(C)$. Let us now consider the case where C is not a real number ($c^1 < c^3$). The main concern in this case is the computation of $Area(C \cap D)$, which corresponds to the following expression:

$$Area(C \cap D) = \int_{-\infty}^{\infty} \min\{\mu_C(x), \mu_D(x)\} dx \tag{3.14}$$

Notice that the computation of this formula changes depending on how C and D intersect. To overcome this issue, when $(C \cap D)$ is not a triangle, the area is usually approximated by the area of the maximum triangle inscribed in this plane area, obtaining an approximation $\widetilde{AI}$ to the agreement index. If the actual area is to be computed, we may distinguish 5 main scenarios, yielding different ways to compute the AI value. This is illustrated in Figure 3.3, where we show with a dotted line the area that is computed with the described approximation.



(a) Case 1: $c^2 \leq d^1 \wedge c^3 \leq d^2$

(d) Case 4: $c^2 < d^1 \leq d^2 < c^3$

(b) Case 2: $d^2 \leq c^1$

(e) Case 5: $d^1 < c^2 \leq c^3 < d^2$

(c) Case 3: $d^1 \leq c^2 \wedge c^1 < d^2 \leq c^3$

(f) Particularity on case 3: $d^1 \leq c^2 = c^1 < d^2 \leq c^3$

Figure 3.3: Different scenarios to compute the Agreement index.

Case 1: $c^2 \leq d^1 \wedge c^3 \leq d^2$
In this case the TFN C is completely included in D:

$$Area(C \cap D) = \frac{c^3 - c^1}{2}, \text{ hence } AI = 1 \tag{3.15}$$

Case 2: $d^2 \leq c^1$
This is the simplest case, since C and D do not overlap, thus the intersection is 0:

$$Area(C \cap D) = 0, \text{ hence } AI = 0 \tag{3.16}$$

Case 3: $d^1 \leq c^2 \wedge c^1 < d^2 \leq c^3$
In this case, AI coincides with its approximation $\widetilde{AI}$. The expression for AI depends on the x-coordinate ip^1 of the point where C and D intersect:

$$ip^1 = \frac{c^2 d^2 - c^1 d^1}{c^2 - c^1 + d^2 - d^1} \tag{3.17}$$

Clearly, $c^1 \leq ip^1 \leq c^2$ and $d^1 \leq ip^1 \leq d^2$. If $c^1 \neq c^2$, then AI is given by the following expression:

$$AI = \frac{(ip^1 - c^1)(d^2 - c^1)}{(c^2 - c^1)(c^3 - c^1)} \tag{3.18}$$

In the special case when $c^1 = c^2$, then $c^1 = ip^1$ and AI is given by:

$$AI = \frac{(c^1 - d^2)^2}{(d^2 - d^1)(c^3 - c^1)} \tag{3.19}$$

Notice that when the due date is strict and $d^1 = d^2$, the expression for AI does not change.

Case 4: $c^2 < d^1 \leq d^2 < c^3$
Here the x-coordinate ip^2 of the point where C and D intersect is such that $c^2 < d^1 \leq ip^2 \leq d^2 < c^3$ and is given by:

$$ip^2 = \frac{c^3 d^1 - c^2 d^2}{c^3 - c^2 + d^1 - d^2} \tag{3.20}$$

and the expression for AI is as follows:

$$AI = \frac{c^3(ip^2 - c^2) + d^2(c^3 - ip^2) - c^1(c^3 - c^2)}{(c^3 - c^2)(c^3 - c^1)} \tag{3.21}$$

Notice that, whereas in the previous cases $AI = \widetilde{AI}$, this no longer holds here, with $\widetilde{AI} = \frac{d^2 - c^1}{c^3 - c^1}$.

Case 5: $d^1 < c^2 \leq c^3 < d^2$
In this case, C intersects twice with D and AI depends both on ip^1 and ip^2 as follows:

$$AI = \frac{ip^1(c^1 - d^2) + ip^2(d^2 - c^3) + d^2(c^3 - c^1)}{(d^2 - d^1)(c^3 - c^1)} \tag{3.22}$$

As in the previous case, AI does not coincide with its approximation $\widetilde{AI} = \frac{ip^1 - d^2}{d^1 - d^2}$.

3.3 Fuzzy scheduling

The fact that now uncertainty is taken into account through the use of fuzzy numbers affects the model of the scheduling problems seen in Chapter 1. For instance, now processing times are not real values but TFNs $p_{ij} = (p^1_{ij}, p^2_{ij}, p^3_{ij})$, thus starting and completion times are TFNs as well. Therefore, the constraints need to be adapted to the new framework. In the scheduling problems we have seen, precedence and capacity constraints state that an operation θ_{ij} cannot overlap its execution with another task θ_{iq} belonging to its job, or a task θ_{rs} requiring its same machine. If θ_{iq} and θ_{rs} are already scheduled, this will generally lead to schedule the new task with a starting time such that $S_{ij} \geq \max\{C_{iq}, C_{rs}\}$, so no constraint is violated. Having this in mind, it is easy to see that depending on the maximum approximation we choose for the TFNs, the constraints in the fuzzy framework may be interpreted in different ways. For instance, when using $\max_I$ the constraints for this fuzzy framework are defined as follows:

- Capacity constraint (Const. 1.1): Now the constraint (1.1) is defined as:

$$\begin{aligned} &S^k_{ij} \geq C^k_{rs} \vee C^k_{ij} \leq S^k_{rs}, && \forall r, s\, r \neq i \vee s \neq j :\ \eta_{ij} = \eta_{rs}\, k = 1,2,3 \\ &C_{ij} = S_{ij} + pij, && 1 \leq i \leq n, 1 < j \leq n_i \end{aligned} \tag{3.23}$$

- Precedence constraint (Const. 1.2): The constraint (1.2) is also verified in each component of the TFN:

$$\begin{aligned} &S^k_{i0} \geq 0, && 1 \leq i \leq n\, k = 1,2,3 \\ &S^k_{ij} \geq C^k_{ij-1}, && 1 \leq i \leq n, 1 < j \leq n_i\, k = 1,2,3 \end{aligned} \tag{3.24}$$

- Precedence constraint (Const. 1.3): The precedence constraint for the OSP is now defined as:

$$S_{ij}^k \geq C_{ir}^k \vee C_{ij}^k \leq S_{ir}^k, \qquad 1 \leq j, r \leq n_i \, r \neq j \, k = 1, 2, 3 \tag{3.25}$$

This approach to extending precedence and capacity constraints to the fuzzy framework is used for instance in [20, 42, 47, 93, 110], and will be the one adopted in this thesis. As we have seen, the approximation $\max_I$ keeps the support and modal value of the actual maximum operation. Thanks to this property, given a schedule that is feasible for this set of constraints, if the real duration of every task lies in the support of its fuzzy duration, then the real makespan will be contained within the support of the fuzzy makespan. On the other hand, when the approximation to the maximum is taken to be $\max_R$ the constraints are not defined on each defining point. Instead, comparisons are made by means of the ranking methods $\leq_E$ and $\leq_R$. This approach can be found in different papers, for instance in [63, 64, 68, 69, 112]. Because of the properties of this approximation $\max_R$, we **cannot** guarantee that if the real duration of every task lies in the support of its fuzzy duration, then the real makespan is a value in the support of the fuzzy makespan. In the end, this can cause the fuzzy makespan of the schedule to be meaningless.

3.3.1 An illustrative example

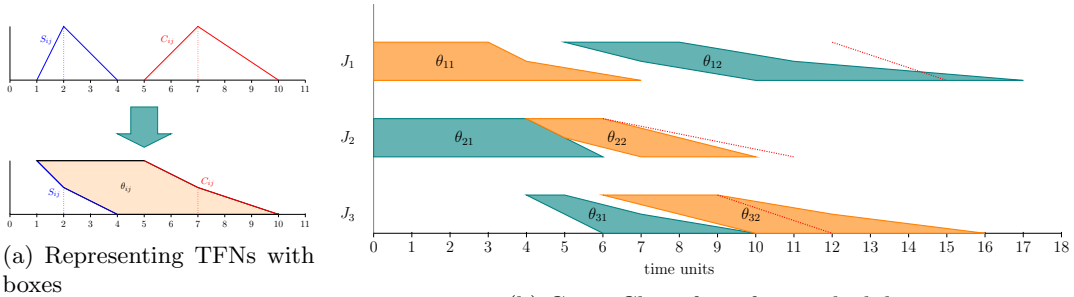
Let us illustrate these ideas with an example. Here we have a fuzzy job shop scheduling problem with n =3 jobs and $m = 2$ machines with processing times, machine assignment and due dates as follows:

$$p = \begin{pmatrix} (3,4,7) & (3,4,7) \\ (4,5,6) & (2,3,3) \\ (1,2,4) & (3,4,6) \end{pmatrix} \qquad \eta = \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} (12,15) \\ (6,11) \\ (9,12) \end{pmatrix} \tag{3.26}$$

Now we schedule the operations following the order $\theta_{11}, \theta_{21}, \theta_{31}, \theta_{22}, \theta_{32}, \theta_{12}$. Operation θ_{11} is scheduled with $S_{11} = (0,0,0)$ and $C_{11} = S_{11} + p_{11} = (0,0,0) + (3,4,7) = (3,4,7)$, and similarly operation θ_{21} is scheduled with $S_{21} = (0,0,0)$ and $C_{21} = (4,5,6)$. Now operation θ_{31} cannot be scheduled before C_{21} because of the constraint (3.23), so $S_{31} = C_{21} = (4,5,6)$ and $C_{31} = (4,5,6) + (1,2,4) = (5,7,10)$. The next operation to be scheduled is θ_{22} which cannot begin before θ_{21} because of constraint (3.24) nor before θ_{11} because of constraint (3.23), therefore $S_{22} = \max\{C_{11}, C_{21}\} = \max\{(3,4,7),(4,5,6)\} = (4,5,7)$. If we continue, we obtain the completion times for each job $C_1 = C_{12} = (8,11,17)$, $C_2 = C_{22} = (6,8,10)$, $C_3 = C_{32} = (9,12,16)$ and we can even compute the agreement indexes $AI_1 = 0.85$, $AI_2 = 0.81$, $AI_3 = 0.21$. Finally, we can compute objective functions such as the makespan $C_{max} = \max_i\{C_i\} = (9,12,17)$.

The fuzzy schedule can be represented again with a Gantt chart, following [36], as it is shown in figure 3.4. The red dotted lines are the flexible due dates and the colours of the tasks represent the machine to which they are assigned.

Schedules can be also represented using graphs, as it happened in the deterministic case. However, in this case the labels of the edges are TFNs instead of real values. An example is given in Figure 3.5. In this case, the graph can be also decomposed as proposed in [48], where the authors represent the schedules using three parallel graphs G^i, each one representing one of the defining points of the TFNs. This is especially useful to look for critical paths and optimise objective functions as the makespan C_{max}.



(a) Representing TFNs with boxes

(b) Gantt Chart for a fuzzy schedule

Figure 3.4: Solution of a fuzzy scheduling problem (Graph).

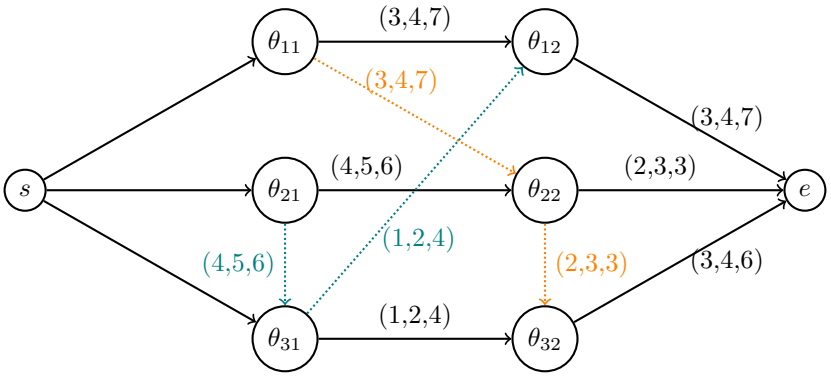


Figure 3.5: Solution of a fuzzy scheduling problem (Graph).

Chapter 4

Development and results

This thesis concentrates on studying the impact of having uncertainty in three of the numerous existing scheduling problems:

- FJSP: Fuzzy Job Shop Scheduling Problem
- FOSP: Fuzzy Open Shop Scheduling Problem
- FfJSP: Fuzzy Flexible Job Shop Scheduling Problem

This chapter describes the research carried out on these problems, including the study of robustness, with references to the publications that are present in the compendium.

The chapter is organised as follows. First we introduce search spaces and schedule generation schemes and describe the work that has been done to define them in the fuzzy scheduling framework, including the proposal of new schedule categories. This is followed by a description of the different methods proposed in the thesis for solving the fuzzy scheduling problems mentioned above, including both single objective and multiobjective algorithms. Finally, we discuss robustness in scheduling problems by studying different robustness metrics, proposing new methods to measure it and providing optimisation methods to obtain robust solutions.

4.1 Search spaces and schedule generation schemes

A key issue in scheduling is the definition of subsets of feasible solutions and the study of their properties. We define a *search space* as a set of solutions to a problem. In general, if a search space is too large, it is potentially more difficult to find the optimal solution than when the set of solutions is small. However, if it is too small it may not be guaranteed to contain at least one optimal solution. The ideal search space would be the smallest one containing at least one optimal solution.

Definition 4.1.1 *A search space is said to be* dominant *if it contains at least one optimal solution to the problem.*

This is illustrated in Figure 4.1, where black dots represent non-optimal solutions and red dots represent the optimal ones. On the left we see the set of all feasible solutions of the problem. On the right we can see different search spaces which are subsets of the previous one, thus making the search for optimal solutions easier. Notice however, that the smallest set (the red one) does not contain any optimal solution, i.e. it is not dominant. In this case, it may be preferable to use the grey set, which is larger but dominant.

In general, depending on the situation, we may be interested in different search spaces. For instance, solving methods such as exact search methods may benefit from considering

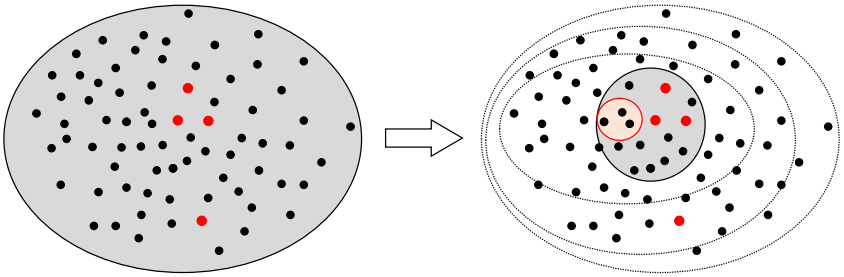


Figure 4.1: Different sets of solutions (red dots are optimal solutions).

small sets of solutions while other solving methods benefit from having large diversity in the sets of solutions that they consider, so it seems appropriate that they use slightly larger search spaces.

In classical scheduling problems, there are well known search spaces which define different categories of schedules [98]. Among them, the most extended ones for the JSP and OSP are the following:

- Feasible solutions: Set of solutions such that the constraints of the problem hold. Obviously, this search space is dominant.
- Semi-active schedules: Set of feasible solutions such that no task can be scheduled earlier without changing the relative order of at least, two tasks. It is dominant.
- Active schedules: Set of semi-active schedules such that no task can be scheduled earlier without delaying the starting time of at least another task. This search space is dominant.
- Non-delay schedules: A machine cannot be idle at time t if there is a task available to be performed at that same instant. This search space is **not** dominant.

Notice that the smallest dominant search space in this list is the set of active schedules. Furthermore, every semi-active schedule can be mapped in the set of the active schedules. On the other hand, the set of non-delay schedules is not dominant, but it is a very small search space in which the average quality of the solutions is very high (regarding many objective functions). This makes this search space especially useful when it is required to find good solutions very quickly (e.g. to generate a initial population or when the number of potential solutions to the problem is extremely large).

To build a schedule such that all constraints hold or such that it belongs to a specific category, the most usual way is to use a task processing order (or a priority array).

Definition 4.1.2 *A Schedule Generation Scheme (SGS) is an algorithm that is able to build a schedule from a task processing order.*

It is essential to have proper SGSs and study which is the set of schedules obtainable with a given SGS. Moreover, it is important to know how a SGS relates with the schedule categories and study its theoretical ability to reach the optimum.

Definition 4.1.3 *A SGS is* complete *for a given search space if it can be used to generate all the schedules in it.*

Ideally, for a given search space we want to have a SGS that is complete in that search space and also does not build solutions outside this space. In general, any SGS for scheduling problems can be defined following the generic schema provided in Algorithm 1. Here, the set A is determined by the constraints of the problem, but depending on how

while there are unscheduled tasks **do**

1. Compute a set A of operations that are available to be scheduled.
2. Compute a subset $E \subset A$ of eligible operations.
3. Choose the operation $\theta_{ij} \in E$ that is in the most left position in the task processing order.
4. Schedule θ_{ij} in its earliest possible starting time ES_{ij}.

Algorithm 1: Generic Schema of a SGS

the set E is defined and how the value ES_{ij} is computed, different sets of solutions can be obtained.

In the case of classical scheduling, there are well-defined SGSs which generate solutions in the search spaces introduced before. Perhaps the best known SGS is the Giffler&Thompson algorithm (G&T) [38], which generates active schedules for the JSP and is complete in that search space. Surprisingly enough, although we can find some ad-hoc extensions of deterministic SGSs to the fuzzy framework, no effort has been made to give precise definitions for categories of schedules when fuzzy times are involved, nor have SGSs been defined and studied systematically in this framework. The first step of this thesis is intended to fill this existing gap in the literature. This will (hopefully) allow to define more efficient solving methods for fuzzy scheduling problems. Inspired by the work of [4, 98, 99] for different deterministic scheduling problems, we have provided formal definitions for scheduling categories in the fuzzy framework as well as several SGSs for the FJSP and the FOSP. Moreover, we have studied the relationship between different types of schedules and the sets of solutions generated by the proposed SGSs, as well as investigating whether such sets necessarily contain one optimal schedule. Due to the FfJSP being a generalisation of the FJSP, the studies made for FJSP can then be easily extended to the FfJSP.

4.1.1 Schedule categories and SGS for the FJSP

The study of schedule categories and SGSs for the FJSP is the core of one of the publications that are part of this thesis [85]. In the following, we introduce the contribution we have made for the FJSP. However, we refer the reader to Section 8.1 for more detail on these concepts. Following the schema in Algorithm 1 we may build different SGSs for the FJSP. Due to the precedence constraints of the FJSP, the set of available tasks A contains at each step the first non scheduled task of each job. Regarding the ES_{ij} values, we propose two different strategies to compute it: an insertion strategy and an appending strategy. In the insertion strategy, the earliest possible starting time is computed as the earliest starting time ESI_{ij} we can assign to task θ_{ij} without delaying any already scheduled task. On the other hand, if θ_{xy} is the last operation scheduled in machine η_{ij} and θ_{iz} is the last scheduled operation of job J_i, then the appending strategy computes the earliest possible starting time ESA_{ij} as:

$$ESA = \max\{C_{xy}, C_{iz}\} \tag{4.1}$$

These strategies are easy to interpret in the deterministic case. However, it is not so straight forward in the fuzzy case. In the already mentioned paper we provide a definition for each strategy which is consistent with the constraints we defined in Section 3.3 for fuzzy scheduling.

In the classical JSP, schedule categories are defined based on the concepts of "left shift", "local left shift" and "global left shift" [98]. Therefore, it seems reasonable to extend these concepts to fuzzy scheduling so to define scheduling categories in this framework.

Definition 4.1.4 *Let* $\boldsymbol{t}$ *be a feasible schedule, then a* left shift *of an operation* θ_{ij} *in* $\boldsymbol{t}$ *is a move giving another feasible schedule* $\boldsymbol{s}$ *where:*

$$\begin{aligned} \exists k \in \{1,2,3\} : s_{ij}^k < t_{ij}^k \wedge \forall l \neq k \; s_{ij}^l \leq t_{ij}^l \\ s_{xy} = t_{xy} \; \forall x, y \; x \neq i \vee y \neq j \end{aligned} \tag{4.2}$$

Definition 4.1.5 *Let* $\boldsymbol{t}$ *be a feasible schedule (i.e. an assignment of starting times), then a* local left shift *of a task* θ_{ij} *in* $\boldsymbol{t}$ *is a move giving another feasible schedule* $\boldsymbol{s}$ *where*

$$\begin{aligned} \exists k \in \{1,2,3\} : s_{ij}^k = t_{ij}^k - 1 \wedge \forall l \neq k \; s_{ij}^l = t_{ij}^l \\ s_{xy} = t_{xy} \; \forall x, y \; x \neq i \vee y \neq j \end{aligned} \tag{4.3}$$

Definition 4.1.6 *Let* $\boldsymbol{t}$ *be a feasible schedule, then a* global left shift *of a task* θ_{ij} *in* $\boldsymbol{t}$ *is a left shift of* θ_{ij} *that is not obtainable by a sequence of local left shifts.*

Based on these extensions, we provide the first formal definitions and studies for both semi-active and active schedules for the FJSP paying special attention to the dominance of the categories.

Definition 4.1.7 *A semi-active schedule is a feasible schedule in which none of the tasks can be locally left-shifted.*

Semi-active schedules can be obtained by defining a SGS such that $E = A$ at each step and the the earliest possible starting times are computed following an appending strategy ($ES = ESA$). We have proven that this SGS, we denote *SemiActiveSGS*, always generates semi-active schedules and is complete in that search space.

Definition 4.1.8 *An active schedule is a feasible schedule where no global or local left shift lead to a feasible schedule.*

The definition of a SGS for building active schedules is not so straight forward. We propose to extend to the fuzzy case the well-known G&T algorithm, which is based on an appending strategy ($ES = ESA$). At each step, this algorithm looks for the operation θ^* which has the earliest possible completion time C^*. The set E is then built with all the operations θ_{ij} requiring the same machine as θ^* and such that $ESA_{ij} < C^\star$. Notice that depending on how we perform the comparison between each ESA_{ij} value and $C^\star$, the set E can change. For instance, we have seen that dominance and completeness are lost when considering a simple extension of this algorithm (we call *fG&T-SGS1*), while an insertion based SGS (*ActiveSGS*) that takes $E = A$ and $ES = ESI$ generates active schedules and is complete on that search space.

Definition 4.1.9 *The* fG&T-SGS1 *algorithm is an appending SGS where the eligible set* E *is computed as follows:*

$$\begin{aligned} C^\star = \min\{ESA_{ij} + p_{ij} : \theta_{ij} \in A\} \\ E = \{\theta_{ij} \in A : \exists k \; ESA_{ij}^k < (C^\star)^k\} \end{aligned} \tag{4.4}$$

After performing a more detailed analysis on the causes of this behaviour, we design a more sophisticated extension of the G&T (we call *fG&T-SGS2*) which completely changes the way in which the subset E is built.

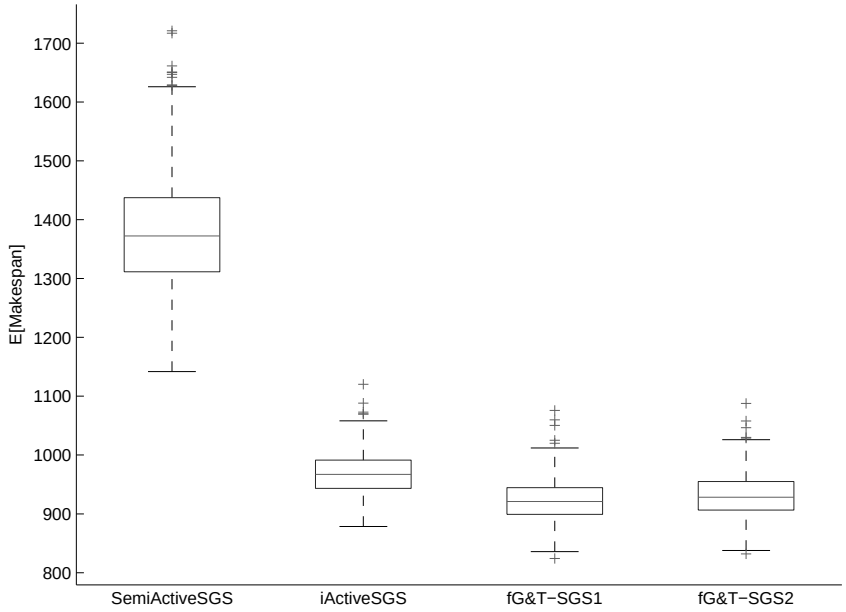


Figure 4.2: Comparison of solutions (in terms of C_{max}) obtained with different SGSs for FJSP.

Definition 4.1.10 *The* fG&T-SGS2 *algorithm is an appending SGS where the eligible set* E *is computed as follows.*

$$\begin{aligned} C^\star &= \min\{ESA_{ij} + p_{ij} : \theta_{ij} \in A\} \\ A^\star &= \{\theta_{ij} \in A : \exists k\ ESA^k_{ij} + p^k_{ij} = (C^\star)^k\} \\ E &= \{\theta_{ij} \in A : \forall \theta_{xy} \in A^\star\ x \neq i \vee y \neq j\ \exists k\ ESA^k_{ij} < ESA^k_{xy} + p^k_{xy}\} \end{aligned} \tag{4.5}$$

We have proven that this new SGS is able to keep the completeness and dominance properties. In order to get more information about the schedules generated by each of the proposed SGS, we conduct a set of experiments in which we generate a pool of random task orderings and evaluate them using the different SGSs. The results are illustrated in figure 4.2, where the comparison is made in terms of makespan, which is the most common objective function. They confirm the differences between semi-active and active subspaces. Furthermore, the best quality is obtained with *fG&T-SGS1*, which has the smallest associated search space, so it seems that narrowing the search space can improve the average quality of schedules even if dominance is lost. In addition, we see that using an appending strategy (*fG&T-SGS2*) leads to a slightly better mapping (i.e. average quality of obtained solutions is better), but is more computationally expensive. We believe that these results are a very good starting point for this research on fuzzy scheduling, and they can also provide a guide for designing new SGS and incorporate them both into metaheuristic and exact search methods.

4.1.2 Schedule categories and SGS for the FOSP

The definition of good search spaces is a critical factor for solving the FOSP, as it has many more feasible solutions than for instance, the FJSP. This is because the most characteristic constraint of this problem (const. 3.25) is actually a relaxation of the precedence constraint of the FJSP (const. 3.24). We can find in the literature some ad-hoc definitions of schedule categories for the FOSP, as for instance in [84] where an extension of the G&T algorithm is used, but no formal definitions are provided. When defining a SGS for the FOSP following the generic schema in algorithm 1, the main difference w.r.t. to the FJSP is that, in the

FOSP the set A contains at each step all the non scheduled tasks. In addition, even though the computation of ESA_{ij} is done as in the FJSP, this is no the case of ESI_{ij}. Unlike the FJSP, in a FOSP problem a task θ_{ij} can be scheduled before C_{iz}, where θ_{iz} is the last scheduled task of job J_i. Fortunately, the definitions we have provided of "left shift", "local left shift" and "global left shift" for fuzzy scheduling are valid for the FOSP, which allows to define the same categories in this problem: semi-active and active schedules. Moreover, the *SemiActiveSGS* we have previously defined can be applied to generate semi-active schedules for the FOSP and is complete in that search space. Remember however, that the set A, and therefore the set E are different than in the FJSP. If we consider the FOSP constraints when computing ESI_{ij}, then the *ActiveSGS* can also be used to generate active schedules in FOSP and is complete in that search space. Unfortunately, the study made for the FJSP is not valid for the FOSP when designing appending based SGSs to build active schedules. Following a similar process than before, the extension of the G&T algorithm for the deterministic OSP can be considered. In this extension, the subset E is built with the operations requiring the same machine as, or belonging to the same job as θ^* that can begin before $C^\star$.

Extending G&T to the fuzzy case:

The same definition of the *fG&T-SGS1* can be applied to the FOSP by taking into account that the set A is different. Nevertheless, the formal studies provided for the FJSP are not valid in this case. It is easy to prove that the SGS is not complete by finding an example of active schedule that cannot be generated with this SGS. However, we have not been able (yet) to provide any formal demonstration about its (non)dominance.

When extending the G&T algorithm to the fuzzy framework, two main issues must be addressed: how to compute $C^\star$ and how to compare each ESA_{ij} value with $C^\star$. For the former, a "minimum" operation between TFNs is required. In the previous approach, this operation was made based on the approximation for the maximum of two TFNs $\max_I$ (see Section 3.1.2). However, we can also use the $\max_R$ approach, thus $C^\star$ is computed as the completion time with the minimum **expected** value. By doing this, it can be proven than the resulting SGS is complete in the set of active schedules, and therefore it is dominant, but it does also generate solutions that are not active. Let's illustrate this with an example: let $\theta_{ix}, \theta_{iy}, \theta_{jz} \in A$ be three tasks such that θ_{ix}, θ_{iy} belong to the same job, but θ_{jz} belongs to a different one and requires a different machine than θ_{ix} and θ_{iy}. It may be possible to find at one iteration of the algorithm that ES_{ix}=(11,12,13), C_{iy}=(10,11,12) and C_{jz}=(6,10,14). Clearly, by scheduling task θ_{ix} before θ_{iy} we obtain a schedule that is not active (θ_{iy} could be scheduled without delaying θ_{ix}), thus θ_{ix} must not be scheduled before θ_{iy}. However, notice that $E[C_{iy}]$=11 and $E[C_{jz}]$=10, thus C^* =(6,10,14). This leads to task θ_{ix} being part of the set E due to $ES^3_{ix} < (C^\star)^3$ and it could be scheduled before θ_{iy}, leading to a not active schedule.

Finally, by keeping the computation of $C^\star$ as in the first approach and adapting the (*fG&T-SGS2*) to the FOSP, we obtain an SGS that apparently overcomes the previous issues. Even though we strongly believe that this SGS generates only active schedules and is complete in that search space, we have not been able (yet) to formally prove it.

E-active schedules:

In the previous approaches, the comparison between the ESA_{ij} values and $C^\star$ has been made by checking if $\exists i, ESA^i < (C^\star)^i$. However, this comparison can be done by means of the ranking methods $<_E$ and $<_R$ defined in section 3.1.1. To be consistent with this approach, the $C^\star$ value should be computed again as the completion time with the minimum **expected** value. By doing this, we obtain a SGS that is actually the same as the G&T

for the deterministic problem in which the durations of the tasks are taken to be their expected values. This approach is used for instance in [84]. Even though this approach does not make use of all the available information about the uncertainty, we can prove that this SGS always generates active schedules, but it is not complete in that search space. To distinguish the schedules that are generated with this approach, we call them "e-active" schedules (expectation based active schedules). As it happened with *fG&T-SGS1* in the FJSP, we have seen empirically that the average C_{max} of the e-active schedules is better than the average quality of active schedules. However, we have not been able to formally prove the dominance of this search space.

4.2 Metaheuristic strategies for fuzzy scheduling

4.2.1 Fuzzy Open Shop

Among the three problems considered in this thesis, the FOSP is maybe the one with the least presence in the literature. Nevertheless, a genetic algorithm is proposed in [84] to solve it; and in [43] the authors define a specific neighbourhood by following the ideas from [48] to define critical paths based on the use of parallel graphs. This neighbourhood is used to implement a Hill Climbing algorithm which is then combined with a genetic algorithm.

Due to the scarce literature for this problem, it seems reasonable to look for the most successful methods for solving the deterministic problem in order to have some preliminary insights. As far as we know, the best results for the classic OSP are published in [11] and [95], where the latter provides slightly better results overall. In [95], a particle swarm optimisation algorithm (PSO) is proposed which uses different decoding schemas or SGS, most of them based on the well known G&T algorithm. The main ideas proposed in that paper can be adapted to the case of the FOSP with makespan minimisation. **This is the basis for the paper [79] that is part of the compendium of publications of this thesis (see Section 7.4)**. The paper does not only describe the proposed method, but is also devoted to the study of robustness. This section focuses on the solving method, and the part about robustness will be detailed in section 4.3. The method we propose is a PSO in which solutions are codified by means of priority arrays, which allows to assign the solution to a position in the space. The mechanism to move the particles in the algorithm is not the standard one. For instance, the velocity of each particle is given by an array of values in the set {-1,0,1} and it is updated following a stochastic strategy. This strategy guides the particle towards the best global solution found so far by the algorithm, or towards the best position found by the particle itself. This choice is done by an *inertia* parameter, which defines the probability of moving towards one or the other, and whose value can vary during the evolution of the algorithm. Additionally, a mutation and diversification strategies are included to avoid getting stuck in local optima. Due to the large number of solutions for the FOSP, the most relevant factor is the decoding strategy. In this proposal, we decode the particle position by using the SGS introduced in the previous section for building e-active schedules. Furthermore, we introduce a parameter $\delta \in [0, 1]$ which is based on the ideas from [40]. This allows to narrow the size of the search space -the lesser the δ value, the smaller the search space-. After studying the effect that this parameter has in the performance of our PSO based algorithm, we determine that the best option is to use the SGS with $\delta = 0.25$. In fact, using smaller values causes the algorithm to lose too much diversity thus making it easier to get stuck in local optima, whereas having larger δ values makes the algorithm get lost quite easily in such a vast search space thus not finding very good solutions. Finally, we perform a detailed parametric analysis to find the best configuration for the algorithm, and compare its performance with the state-of-the-art method, which is the memetic algorithm published

1. Initialize with en empty state.
while there are unscheduled tasks **do**
  for each node in the current level **do**
    2. Computes a set of candidate operations to expand.
    3. Computes a heuristic value h for the new nodes.
    4. Discard all nodes with $h > UB$, where UB is an upper bound.
    5. Apply *ReduceToRelated* method proposed in [11].
    6. Expand the node by scheduling the ext nodes with the highest priority in the array.
  6. Keep the bw nodes with the best h value.

Algorithm 2: Beam Search for FOSP

in [43]. Even though a PSO based algorithm may seem less sophisticated than a genetic algorithm with local search, it has been able to greatly outperform the results obtained in [43]. We think this is thanks to our proposal focusing on a better search space. This confirms our suspects about the size of the search space being a key factor when solving the FOSP.

In addition to reduce the size of the search space as we have done, we may find good solutions for the FOSP by proposing more intensive search algorithms. This is done for instance in [11], where a beam search (BS) algorithm is combined with a local search and then embedded into an ant colony optimisation algorithm (ACO) to solve the deterministic OSP. Moreover, the PSO from [95] includes a beam search as part of the decoding strategy. Roughly speaking, BS are breadth-first search algorithms where the maximum number of nodes per level is limited by a "beam width" (bw) parameter. There is also a constant value ext that limits the maximum number of children that a node is allowed to generate. These two parameters control the trade-off between exploration and exploitation. In addition, a BS may be enhanced by including an upper bound to prune the tree, which guides the search to more promising areas, or a local search to improve its exploitation at the cost of increasing the computational effort. In this thesis we propose a beam search strategy that follows the scheme given in algorithm 2 and is guided by a task priority array.

The set of candidate tasks in step 2 is the set E that results from using the SGS that we have introduced for building e-active schedules. In step 3, for a given partial schedule, the heuristic function h is defined as follows:

$$\max\{\max_{i=1...n}\{CJ_i + \sum p_{ij}\}, \max_{k=1...m}\{CM_k + \sum p_{ij}\}\} \tag{4.6}$$

where θ_{ij} is an unscheduled task, CJ_i is the completion time of job J_i and CM_k is the completion time of the last task scheduled in machine M_k.

The priority array has the role of guiding the search towards different areas of the search space. Therefore, if the BS is run several times using different priority arrays, it will explore different areas of the search space. In order to keep the best guiding priority arrays and build better ones, we propose to evolve them through the use of a genetic algorithm. We call Genetic Beam Search (GBS) to this hybrid of a genetic algorithm with a beam search strategy. Similarly to memetic algorithms, the genetic component provides exploration capabilities whereas the beam search focuses on intensification. The larger the bw and ext, the larger the intensification. We run the algorithm with low exploration capabilities (ext=2 and bw=2n) in order to have reasonable computational times. The results obtained with this strategy outperform all the previous results published for the FOSP, including the ones we have obtained with the PSO based algorithm. Moreover, even though the algorithm is designed for the FOSP, it performs quite well in the deterministic problem, being close to the best known results. **This approach has been presented**

in the META conference in 2014 [1].

Multiobjective approaches

In the case in which due dates are present, we can focus on optimising additional objective functions such as the maximum tardiness. **This is actually done in [81], which is part of the compendium of this thesis (see Section 7.3)**. As in previous cases, this section will focus only on the solving method proposed in this paper, and the work about robustness will be introduced in section 4.3. Here we tackle a bi-objective FOSP with hard due dates in which the optimisation criteria is to minimise the makespan C_{max} and the fuzzy maximum tardiness T_{max}, that is defined as:

$$T_{max} = \max_{i=1...n} \{(0,0,0), T_i\} \tag{4.7}$$

where $T_i^k = \max\{0, C_i^k - d_i\}$ for all $k = \{1,2,3\}$.

In addition, we consider different goals for the objective functions, which model the desirable targets given by a decision maker for each objective. When this happens, it is often the case that some are achievable only at the expense of others. A well established approach to dealing with multiple and possibly conflicting objectives is lexicographic goal programming. It assumes that there exists a hierarchy of importance for these goals so as to satisfy as many as possible in the specified order. This is a common situation in real world problems, where the decision maker has a clear priority between the objective functions but is satisfied once the objective with the highest priority reaches a certain quality level. A formal model for this problem is provided in the paper. We tackle this multiobjective problem by adapting the PSO based algorithm previously described in this section to work as a lexicographical goal programming method. This is done by changing the way in which different solutions are compared: let A and B be two solutions, $f_1, f_2, \ldots, f_t$ different objective functions sorted by their priority, and $Goal(f_i)$ the desirable target for function f_i, then A is better than B (in a minimisation context) if and only if one of the following hold:

- $\exists i \in [1,t] \ : \ f_i(B) > Goal(f_i)$ such that:
  - $f_i(A) < f_i(B) \wedge f_j(A), f_j(B) \leq Goal(f_j) \forall j \in [1,i)$
- $\forall i \in [1,t] f_i(A), f_i(B) \leq Goal(f_i)$ and $\exists j \leq t$ such that:
  - $f_j(A) < f_j(B) \wedge f_k(A) = f_k(B) \forall k \in [1,j)$

The main issue to assess the behaviour of our solving method is the lack of instances for this problem in the literature. Even though we are able to find hard instances for makespan minimisation, there are not available test beds considering due dates or desirable targets as far as we know. Because of this, we propose and make available online[2] a new set of problems based on the common use instances of Brucker [17] for the OSP. Using this new test bed, we perform an extensive parametric analysis to optimise the performance of our algorithm in this multiobjective scenario taking into account the two possible hierarchies between the objective functions. When evaluating our lexicographical goal programming approach, we see that the goal for the most prioritised objective functions is always reached. Furthermore, if we run the algorithm optimising just one objective function, then the value we obtain in the secondary objective function is much worse than when we use the multiobjective approach. For instance, when using the lexicographical

[1] There are no proceedings yet for this conference. The extended abstract of this work is available online at http://meta2014.sciencesconf.org/37981

[2] Repository section at http://di.uniovi.es/iscop

approach having C_{max} as the most relevant objective function, the values obtained for the second function T_{max}, are a 26% better than a single-objective approach optimising only C_{max}. For the sake of completeness, we also compare our method with a Pareto-like approach, more appropriate for the case in which no hierarchy can be established for the objectives. We observe that the solutions obtained with the lexicographic approach complement very well those obtained by the Pareto based approach, by focusing on the "extreme ends" of the set of non-dominated solutions.

A similar work is followed **in the paper [78] that is also part of the compendium included in this thesis (see Section 7.5)**, in which flexible due dates are defined instead of hard constraints (see Section 3.2). In this paper we model a multiobjective FOSP in which the optimisation criteria are the minimisation of makespan C_{max} and the maximisation of the agreement index AI. The AI measures the satisfaction degree to which a flexible due date is met, therefore each job J_i has a different AI_i value. The degree of overall due-date satisfaction for a given schedule is then obtained by aggregating all the satisfaction degrees AI_i, $i = 1, \ldots, n$. In particular, we consider two aggregation functions: the minimum and the average; previously used in the literature concerning shop scheduling with soft constraints, for instance in [45, 60, 93]:

$$AI_{av} = \frac{1}{n} \sum_{i=1}^{n} AI_i, \tag{4.8}$$

$$AI_{min} = \min_{i=1,\ldots,n} AI_i \tag{4.9}$$

The function AI_{av} can be interpreted as an overall performance we shall optimise in order to improve the average satisfaction degree of the costumers, while AI_{min} is a more conservative and restrictive measure in which the objective is to guarantee that all the customers are satisfied at a certain (optimal) degree. Again we assume that a decision maker has desirable targets for each objective, thus we have a lexicographical goal programming model. As we have done in the previous case, we propose a multiobjective optimisation algorithm considering the different hierarchies between the objectives. The results show a similar behaviour than in the case in which due dates were strict: the goal for the main objective function is always reached and when we run the algorithm optimising just one objective function, then the value we obtain in the secondary objective function is much worse than when we use the multiobjective approach. In addition to this analysis, we also perform a comparison between the use of the AI_{min} as the main objective function and the use of AI_{av}. It may seem that these functions are strongly correlated, but we have seen that the C_{max} values (secondary objective function) we obtain are slightly better when AI_{av} is considered. We think that this is because in the first iterations of the algorithm there are many solutions with $AI_{min} = 0$, thus this function is a worse guiding the algorithm to promising solutions in the first iterations.

In conclusion, we may state that the large search space of the fuzzy open shop scheduling problem makes very important the definition of good search spaces in which look for solutions. In general, this is more important that the choice of different generic metaheuristics. For instance, we have seen that a PSO working in a good search space is able to outperform a more sophisticated memetic algorithm searches for solutions in a larger search space. Another promising strategy to solve this problem is to find a really good trade-off between exploration and exploitation, and here the hybridisation of exact methods with metaheuristics is providing very good results. Moreover, the best results we have reached until now have been by using a beam search algorithm (which is based on an exact method) combined with a genetic algorithm. Finally, in the multiobjective case we have seen that hierarchical approaches like lexicographical goal programming techniques are a good approach for the case in which goals are established and a hierarchy among

the objective functions is present. Furthermore, in the case in which there is no hierarchy, they offer an interesting alternative to Pareto-like approaches, as they complement the set of solutions obtained by algorithms like the well-known NSGA-II.

4.2.2 Fuzzy Job Shop

On the opposite to the FOSP, the FJSP has been the fuzzy scheduling problem with more presence in the literature. An interesting review can be found in [1] citing most of the contributions about this problem. Due to the number of already existing methods to solve this problem, in this thesis we shall focus less in the solving methods for the FJSP and instead pay more attention to more theoretical ideas. In fact, we have already provided a formal study on search spaces for this problem, and it will be also the main scheduling problem we will use when studying robustness.

Regarding makespan minimisation, we can find several contributions in the literature. For instance, we can find among others a memetic algorithm in [91] or a shifting bottleneck hybridised with a genetic algorithm (SBP-GA) in [86]. Furthermore, a random-key based GA (RKGA) is proposed in [63], a hybrid discrete PSO (HDPSO) in [68], a multiobjective optimiser in [104], a simulated annealing method in [110] and a genetic algorithm (SMGA) in [93]. In addition to these methods, it may be interesting to know the metaheuristic methods that perform the best in the deterministic case. It seems reasonable that this methods can perform really well in the FJSP as well, as both problems share many features. In the case of the deterministic JSP, it appears that the most successful algorithms are based on local search strategies [9, 28, 75]. Defining good neighbourhood structures to guide those algorithms seems to be the key to solve this problem. Back in the FJSP, different neighbourhood structures for local search algorithms have been already proposed in the literature. In [36], the neighbourhood structure proposed in [106] for the JSP is extended to the fuzzy framework. This structure is based on the reversal of critical arcs in the graph representing the solution, and is extended later in [48], where the authors consider the use of three parallel graphs for representing a fuzzy schedule. A more efficient neighbourhood is then proposed in [46], and more recent one is described in [91]. We make use of this last neighbourhood structure to define a tabu search algorithm (TS) for the FJSP in [83]. Its main structure is based on the ideas from [28] and its exploitation capabilities are parametrised so its stopping criteria is reached after a fixed number of consecutive iterations without improvement. The algorithm is run several times from different initial solutions as to explore different areas. These solutions are provided by, and evolved through, a genetic algorithm. The details on both components (i.e. the TS and the GA) and how they are combined can be found in the paper. The results obtained with this hybrid are very promising. In the first place, we observe a synergy effect in figure 4.3, that is, the hybrid version is able to get better results than the GA or the TS by separate in similar running conditions. This shows that the combination of the exploration capabilities of the GA complement very well the exploitation ability of the TS. Finally, we compare our approach with the best method in the literature, that is the memetic algorithm from [91]. Having similar runtime, our approach outperforms the state-of-the-art results in almost a 34%.

One of the main issues we find when designing solving methods for the FJSP is the lack of a common framework to compare different algorithms. Unlike the deterministic case, no common test-bed is available for the FJSP that allows for fair and meaningful comparisons and assessment of different proposals. Moreover, only a portion of the instances used in the literature for experimental results are available to the research community. To fill this gap, we invest part of this thesis in reviewing and studying the level of difficulty of the available instances with regard to the most widely used objective function, the makespan. We have done an exhaustive compilation of all the instances used so far in the literature,

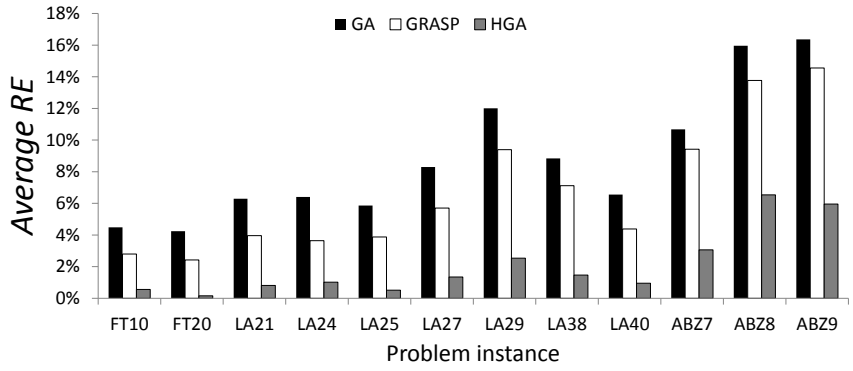


Figure 4.3: Synergy effect in a hybrid between GA and TS.

which can be divided in two big groups depending on whether they have been generated from scratch (e.g. [63, 93]) or by fuzzyfying well-known benchmark instances from the JSP (e.g. [46, 105]). We also find that some of the authors that fuzzyfy deterministic instances do not make available the resulting fuzzy instances, but the stochastic method they use to generate them (e.g. [74, 121]). In those cases, we follow the respective method to generate the fuzzy instances so to be able to study them. We make these instances available online in a common repository[3] together with all the available instances of the literature to make them more accessible to the scientific community. The analysis on the difficulty of the instances is done based on the state-of-the-art algorithms for makespan minimisation. We also consider the memetic algorithm from [91] that seems to be the most successful method and a GRASP algorithm that is based in the neighbourhood structure used by that memetic algorithm. The reason for using a GRASP is to compare the solutions achieved with a simple method with the solutions obtained with the state-of-the-art algorithms. The most common way of assessing the quality of solutions is by means of relative measures such as a relative error w.r.t. a lower bound. We define a lower bound to the FJSP by extending the method from [100] to the fuzzy framework, thus we can compute a lower bound as:

$$LB = \max\{\max_{i=1...n}\{\sum_{j=1...n_i} p_{ij}\}, \max_{k=1...m}\{\sum_{i,j:\eta_{ij}=k} p_{ij}\}\} \tag{4.10}$$

However, this LB is not as tight as desirable to have a proper reference point and assess the room for improvement that an instance may have. In the case of fuzzy instances, it can be proved that a lower bound for the expected makespan of the fuzzy instance is given by an optimal solution (or any lower bound) of the associated expected crisp problem, that is, the problem where durations are the expected value of the corresponding fuzzy ones. Based on this, we are able to provide-and make available for the community-new lower bounds for all the instances by using the IBM ILOG CPLEX CP Optimizer software [54] to find the optimal solution (when possible) of the associated expected crisp problem. Having tighter lower bounds, we follow an exhaustive analysis on the difficulty of each instance. What we see is that many of the proposed instances are already optimally solved or they offer little room for improvement, thus they are not suitable for assessing future solving methods for FJSP. The existing instances that may offer enough room for improvement to serve as future benchmarks are the original fuzzy instances $Lei01$,$Lei02$ [63] and $LP01$ [68], all the fuzzyfyed instances from $La21$,24,36,39,40 [46, 97, 121], $La27$,37,40 [46, 97] and ABZ_F7 [46] (which are not yet solved to optimality even if they do not seem specially hard) and finally instances ABZ_F8,9, La_S29 and La_F29 [46, 97], this last group representing

[3]Repository section at http://di.uniovi.es/iscop

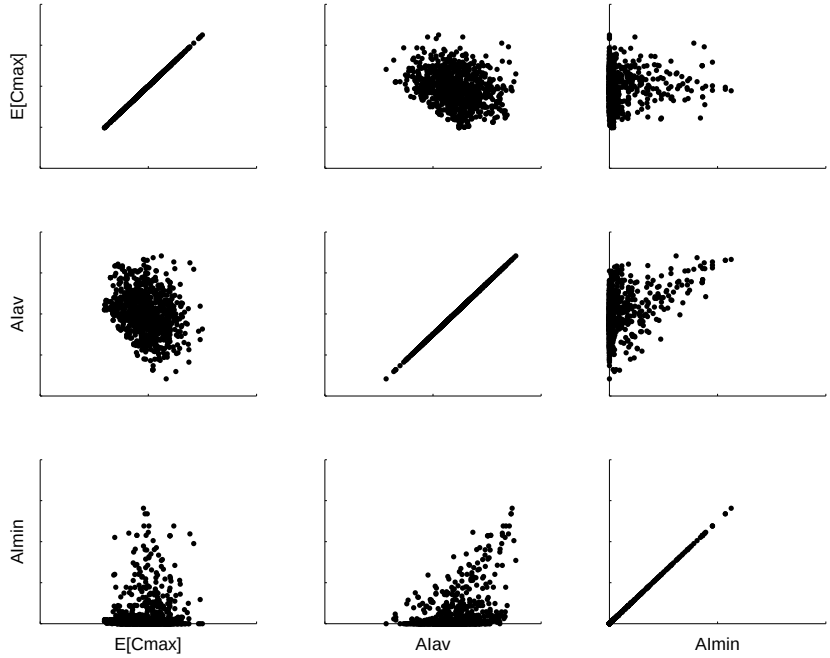


Figure 4.4: Correlation between $E[C_{max}]$, AI_{min} and AI_{av} in a 6×6 instance.

a real challenge. The obtained results for the existing FJSP instances suggest that it is necessary to have more challenging problem instances in order to test the potential of future proposals to solve the FJSP. Therefore, we propose and make available a new test bed based on the well-known Ta benchmark proposed in [100] for the JSP. An analysis on these instances shows that they are much more challenging than the ones already proposed in the literature, so we hope they may serve as future point of reference.

Multiobjective approaches

In addition to makespan minimisation, in the cases in which flexible due dates are present other objective functions such as the agreement index can be considered. Unlike the FOSP, there are several works optimising this objective function in the FJSP. For instance, a genetic algorithm is proposed in [93] to maximise the minimum AI, in [66] the authors maximise the minimum and average AI, and in [118] a co-evolutionary algorithm is proposed to maximise the average AI. We can find also different multiobjective approaches in which the AI is optimised together with the makespan: goal programming is used in [45, 93], in [118] the authors assign weights to each objective function and apply a GA to optimise it, a lexicographical approach can be found in [44] and a Pareto archive PSO is proposed in [60]. Here we shall focus on the case in which there is no preference between the objective functions. It may seem that AI_{min} and AI_{av} are strongly correlated, as it is obvious that having a large value in AI_{min} helps to improve the value of AI_{av}, and a short value in AI_{min} can affect negatively AI_{av}. To measure the possible correlation of these functions and also $E[C_{max}]$, we evaluate them in a pool of 1000 random solutions for three well-known FJSP instances (6×6 instances from [93]). The obtained results for one of the instances are shown in figure 4.4.

Clearly, $E[C_{max}]$ is not correlated with any of the AI functions. Moreover, its correlation index R^2 is in average 0.18 w.r.t AI_{av} and 0.01 w.r.t. AI_{min}. When comparing both AI measures, we see than in fact, there are a lot of solutions with $AI_{min} = 0$, which is not surprising taken into account that we are using random solutions. To have a more reliable study, we remove all the solutions having $AI_{min} < 0.1$. By doing so, we obtain

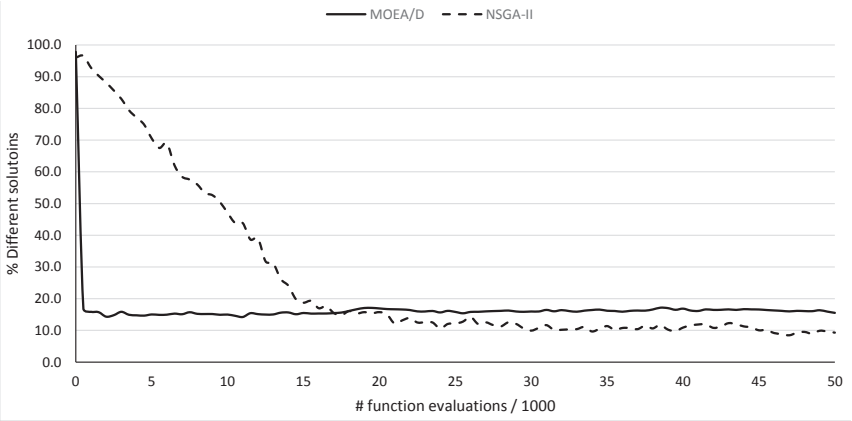


Figure 4.5: Evolution of the percentage of different objective vectors in the population.

a correlation index R^2 between both AI measures that is around 0.35. This small value indicates that it could make sense to consider both measures as objective functions. However, the fact that there a lot of solutions having $AI_{min} = 0$ leads us to think that it may be hard to guide a metaheuristic in its first stages, when most of the initial solutions have a 0 value for this function. Therefore, for our first approach we shall consider only $E[C_{max}]$ and AI_{av}. Besides the well-established class of Pareto-dominance based algorithms, e.g. NSGA-II [27], we can report a recent and growing interest in the so-called aggregation-based algorithms, and especially the MOEA/D (multiobjective evolutionary algorithm based on decomposition) framework [120]. This algorithm decomposes a given problem into a number of single-objective optimisation sub-problems, each of them defined by a scalarising function using a different weight vector. We there-by implement a MOEA/D algorithm for our bi-objective problem by using one of the most extended crossover operators for the JSP, the Generalized Order Crossover (GOX), and inversion as mutation strategy. Surprisingly, this approach performs much worse than a NSGA-II algorithm with the same crossover and mutation operators. After a detailed analysis, we observe that MOEA/D loses the diversity in its population in the early stages of the run. This is illustrated in figure 4.5, were we plot the evolution of the average number of different solutions (in the objective space) in the population maintained by both algorithms as a function of the number of function evaluations for one instance.

We clearly attribute this to the fact that as soon as a good solution is found in MOEA/D, its aggressive replacement strategy will immediately replace all solutions from the near sub-problems. This shows that even though this algorithm performs really good in continuous optimisation, it may be not appropriate for scheduling problems. Based on our observations, we provide improved variants of MOEA/D for the FJSP by addressing the raised diversity issue. Firstly we test already existing variants that are designed as to keep more diversity in the population: The MOEA/D-n_r variant from [67], and the MOEA/D-xy variant from [71]. As expected, this approaches perform much better than the standard MOEA/D and in most of the cases, better than NSGA-II. Moreover, if we perform a more detailed analysis to get some insights we can appreciate that MOEA/D-xy is slower than MOEA/D-n_r converging but keeps more diversity, which allows it to get better results in the long term (when we have large time budgets), whereas MOEA/D-n_r appear to be better in the short-mid term. Once we have addressed that the main issue that MOEA/D has to solve FJSP is the diversity, we design a completely new replacement strategy for this algorithm. First, we simply do not allow an offspring to replace a solution if there already exists a solution having the same objective values in the corresponding neighbourhood. Moreover, every time this condition is satisfied and the replacement is activated with respect to an offspring, say y, and a neighbouring sub-problem solution,

say x^j, we do the following. The offspring y becomes the new current solution for sub-problem j, but the previous solution x^j is not discarded if it can improve the solution of other neighbours. Hence, we recursively check whether there is an opportunity that solution x^j replaces a solution j' in the neighbourhood of sub-problem j. If such a solution is found, a new replacement is activated and so on until no improvement is observed. Notice that with this strategy we do maintain diversity but we also attempt to improve convergence since we heuristically check whether a solution can serve for some sub-problems before discarding it from the population. This new strategy provides better results than the already mentioned variants, having better results than both independently of the time budget.

In conclusion, the FJSP is the fuzzy scheduling problem with more presence in the literature. Many methods have been proposed to solve it, but it seems that the best strategy to tackle it is to adapt to the fuzzy case the most successful ideas for the deterministic problem. Specifically, the definition of good neighbourhood structures to design more sophisticated local search algorithms appears as a very relevant factor. Furthermore, the hybridisation of these local search strategies with population-based evolutionary algorithms have provided the best results known so far. Regarding the multiobjective approaches, it is clear that the MOEA/D algorithm is not very appropriate to solve this problem, even though it can be refined to improve its behaviour. Based on the obtained results for the single-objective case, we think it would be very interesting to design a multiobjective local search algorithm in the near future.

4.2.3 Fuzzy Flexible Job Shop

The fuzzy flexible job shop has grabbed the attention of several researchers during the last decade and different methods to solve this problem can be found. For instance, a genetic algorithm can be found in [62], a co-evolutionary algorithm in [64], a swarm-based neighbourhood search algorithm in [65], a hybrid artificial bee colony algorithm in [112], an EDA algorithm in [113], and more recently a hybrid biogeography-based method in [69].

In previous problems, solutions could be easily codified by means of priority arrays, which are then used as input for SGS (see Section 4.1) to build different schedules. In general, this is not enough for the case of the case FfJSP, where in addition to the priority array, a machine assignment η is required. As reminder, tasks θ_{ij} in FfJSP have not a fixed machine η_{ij} assigned, but it can be performed in a set of machines $M_{ij} \subset M$. Moreover, once a machine assignment η is established, the FfJSP becomes a FJSP. This clear division of the problem in two sub-problems (machine assignment and task processing order) leads to think that a co-evolutionary strategy may be the most natural way to tackle the FfJSP. **This is done in the paper [80] that is introduced in the compendium of this thesis (see Section 7.1)**, in which we propose a cooperative co-evolutionary algorithm [89, 90]. Cooperative co-evolutionary algorithms handle two or more populations, each with its own coding schemes and recombination operators, that cooperate to perform the evaluation of individuals. In our proposal, the algorithm handles two populations: the machine assignment population P^M and the task ordering population P^T. A general schema of our proposal is illustrated in Figure 4.6.

We propose a heuristic seeding method based on an insertion decoding algorithm. The idea is to use this decoding method as a production rule to generate a pool of full schedules for the FfJSP and then encode their task orderings as individual for P^T and their machine assignments as an individuals for P^M. Due to the different nature of both populations, crossover and mutation operators must be specific for each sub-problem. For instance, for population P^M we use a one-point crossover and a mutation strategy that consist on taking one task θ_{ij} at random, and assign it to a random machine $\eta'_{ij} \in M_{ij}$; for population P^T we use the JOX crossover operator [76] and and insertion based mutation operator.

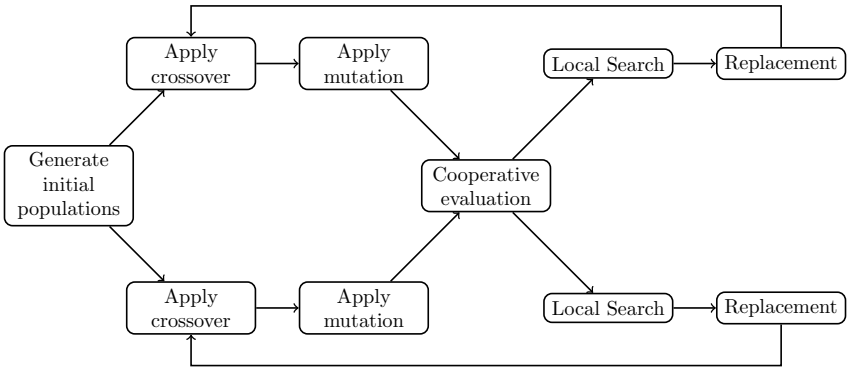


Figure 4.6: Generic schema of our hybrid cooperative co-evolutionary algorithm.

It is at the time of evaluation that populations need to cooperate: any individual only encodes part of a solution and needs to be complemented by an individual from the other population, the so-called cooperative partner, to conform a full solution which can be evaluated. If a machine assignment is taken from one individual in P^M, the task ordering given by an individual of the other population is evaluated using the insertion based SGS from Section 4.1.1. The choice of cooperative partners is done based on [52], where three cooperative partners are used to evaluate each individual. The three full schedules built in this evaluation process are then improved using a Hill Climbing strategy before selecting the best individuals to go into the next generation. This process is repeated until a stopping criterion is met, in our case, for 100 iterations. The most relevant contribution in our method is the design of specific local search strategies for each population. For population P^M, based on the works in [41, 73], we build a neighbour by taking a critical task θ_{ij} and assigning it to a new random machine $\eta_{ij} \in M_{ij}$. Regarding population P^T, aimed at finding good task orderings, the local search assumes a fixed machine assignment (provided by the cooperative partner). This allows to use the neighbourhood structures for fuzzy job shop explained in the previous section. A more detailed study on these neighbourhoods can be found in the paper [77] that is part of this thesis, and in which we propose and study them for the first time in the FfJSP. The results obtained with our proposal are promising. We see that the hybrid of the co-evolutionary algorithm with specialised local search strategies provides better results than using both methods by separate. Furthermore, the algorithm compares favourably with the methods mentioned at the beginning of this section, which are the state-of-the-art for this problem.

If the FfJSP is considered as a generalisation of the FJSP, it seems reasonable to think that if a hybrid of a genetic algorithm with local search strategies performed well for the FJSP, it may also be suitable for the FfJSP. This reasoning is the basis for **the paper [77], which is part of the compendium in this thesis (see Section 7.2)**, and in which a hybrid between a genetic algorithm and a tabu search is proposed. The neighbourhood structures we have considered for the FJSP are based on the use of on finding critical paths in three parallel graphs. Roughly speaking, these graphs join the nodes belonging to the same job or needing the same machine to be performed. However, the machine in which each task is going to be performed is not known in advance in the FfJSP, thus the graph representation must be adapted to this case. In the paper, we propose a new graph-based representation for solutions of this problem in which the nodes are labelled not only with the task θ_{ij}, but also with the machine $\eta_{ij} \in M_{ij}$ to which the task is assigned in the solution. This is illustrated with an example in figure 4.7.

This graph can be easily extended to the three parallel graph representation following [48]. Based on this representation we define two new neighbourhood structures for the FfJSP:

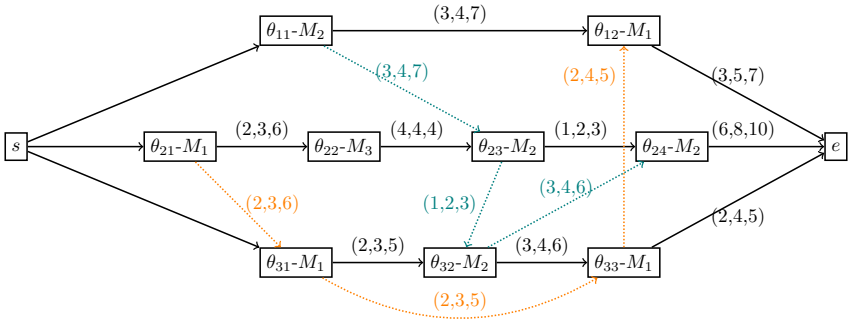


Figure 4.7: Graph representation of a solution for a FfJSP problem.

Definition 4.2.1 *(Neighbourhood N^A) Let a solution be denoted as (η, π), where η is a machine assignment and π is a feasible task processing order. The neighbourhood structure N^A is defined by the set of solutions (η', π) such that:*

$$N^A(\eta, \pi) = \{(\eta', \pi) \ : \ \eta'_{ij} \neq \eta_{ij}, \forall (r,s) \neq (i,j)\, \eta_{rs} = \eta'_{rs}, \theta_{ij} \text{ is critical}, \eta'_{ij} \in M_{ij}\} \quad (4.11)$$

where criticality is defined according to [48].

Definition 4.2.2 *(Neighbourhood N^P) Let a solution be denoted as (η, π), where η is a machine assignment and π is a feasible task processing order. Given an arc $v = (x, y)$ in the associated graph, let π' denote the processing order obtained after reversing arc v. The neighbourhood structure N^P is defined by the set of solutions (η, π') such that:*

$$N^P(\eta, \pi) = \{(\eta, \pi') \ : \ v \text{ is in a critical block}\} \quad (4.12)$$

where criticality is defined according to [48].

These neighbourhoods can be joined in one structure $N^{AP} = N^A \cup N^P$. A formal study on the properties of this union is provided in the paper. For instance, it is proved that it always produces feasible solutions and verifies the connectivity property, that is, for every non-optimal solution it is possible to build a finite sequence of transitions of N^{AP} leading to a globally optimal solution. When designing a local search algorithm, the most consuming part is the evaluation of the neighbours to choose a solution for the next iteration. We propose a method to estimate the makespan of each neighbour by extending to the fuzzy and flexible framework, the well-known concepts of head and tail of a task. This allow to quickly evaluate the potential of the neighbours and choose one for the next iteration. Notice that this estimation is made differently depending on the neighbourhood we are using: N^A or N^P. Furthermore, the proposed estimation is a lower bound for the actual makespan obtained after the change is performed, which allows to easily discard non-improving neighbours if needed. The defined neighbourhoods together with this mechanism are used to implement a tabu search that follows the ideas from [28]. This algorithm is then combined with a genetic algorithm to increase its exploration capabilities. The genetic algorithm encodes the solutions by means of two arrays: one for the machine assignment and other for the task processing order. It uses an extension of the well-known JOX crossover operator [76] for the mating, which already includes an implicit mutation effect, thus we do not include an explicit mutation strategy. Another contribution of this work is the way in which the initial population for the hybrid algorithm is generated. We propose a heuristic seeding based on an insertion SGS but taking advantage of the flexibility. Let A be the set of available operations in the SGS at the current stage (initially, this set contains the first operation from each job). Instead of using a priority array, we

select a random operation $\theta_{ij} \in A$ and compute its earliest completion time, $C^\star$, considering all the machines where it can be processed. Then, we randomly select a machine $M_k \in M_{ij}$ in which θ_{ij} may finish at $C^\star$ and schedule θ_{ij} in machine M_k at its earliest starting time (ESI), given by $C^\star - p_{ij}^k$. The resulting algorithm, named heuristic genetic tabu search (HGTS in short), is studied through an extensive experimental analysis. We see that each of the components that conform the algorithm (heuristics seeding, genetic component and tabu search) contribute to its overall performance, which is impossible to reach if we remove any of the components. We also compare our method with the state-of-the-art methods mentioned above using the instances that are available in the literature. The results show a great improvement with respect to the already published results and the results obtained with our previous co-evolutionary algorithm. However, only 6 instances are available in the literature. Moreover, we believe that the results obtained with our method for these instances are close to the optimal solutions, thus we feel in the need of proposing a new test bed. In order to design proper instances, we make a compilation with the hardest instances for the deterministic fJSP, which can be found in [7, 14, 26, 53]. We select the most challenging ones (13 in total) and use them as base to build fuzzy instances by following the method proposed by [121]. We have made these instances available online[4] together with the results obtained with our HGTS method so to encourage competition. Finally, and for the sake of completeness, we run our method over the published instances for the deterministic fJSP. Surprisingly, even though the proposed methods is not designed for this problem, the obtained results are very close to the state-of-the-art results.

In conclusion, we may consider the FfJSP as a hot topic in the fuzzy scheduling environment, due to its increasing popularity in the last years. The fact that tasks can be scheduled in different machines and that the problem is easy to split in two sub-problems, makes the FfJSP really interesting and challenging. However, from the point of view of including uncertainty in its formulation, which is the main target of this thesis, it does not present a behaviour very different to the FJSP. In both cases, the hybridisation of local search strategies with population based evolutionary algorithms provides very good results. Nevertheless, we have provided several formal studies on the FfJSP, which include the definition of new neighbourhood structures and heuristic methods for designing initial solutions. We hope this may serve for future researchers to better understand the particularities that appear in the FfJSP when including uncertainty.

4.3 Robustness

A fuzzy schedule does not provide exact starting times for each task. Instead, it gives a fuzzy interval of possible values for each starting time, provided that tasks are executed in the order determined by the schedule. In fact, it is impossible to predict what the exact time-schedule will be, because it depends on the realisation of the tasks durations, which is not known in advance. When uncertainty is present, robustness becomes a very relevant factor. Roughly speaking, a schedule is said to be robust if it minimises the effect of executional uncertainties on its performance [5]. This straightforward definition may, however, be subject to many different interpretations when it comes to specifying robustness measures [5, 92].

4.3.1 Semantics for fuzzy scheduling

In [45], a semantics for fuzzy schedules was proposed and then used to measure the performance of fuzzy schedules in real environments. According to this semantics, solutions

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

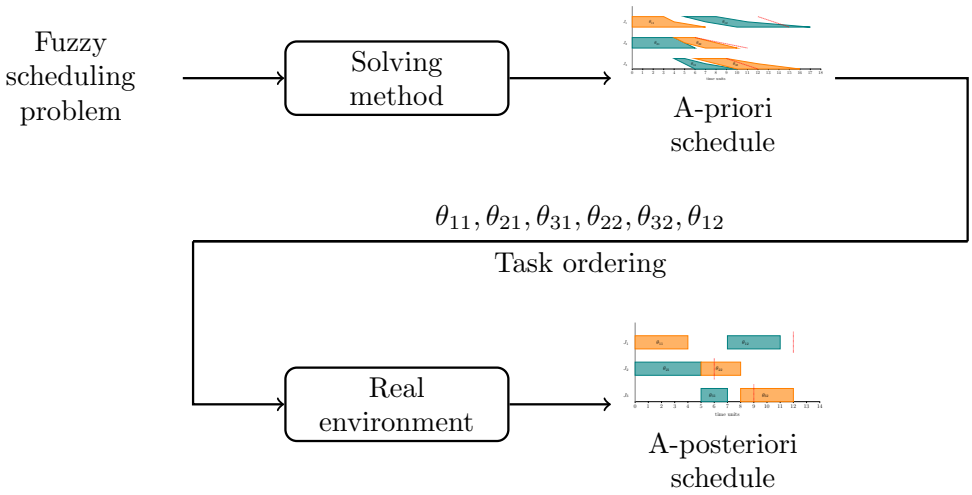


Figure 4.8: Semantics for fuzzy scheduling.

to fuzzy scheduling problems should be understood as a-priori solutions, also called baseline or predictive schedules [50]. These solutions are found when the duration of tasks is not exactly known and a set of possible scenarios must be taken into account. Each fuzzy schedule corresponds to an ordering of tasks; it is not until tasks are executed in a real environment according to this ordering that we know their real duration and, hence, obtain a real schedule, the a-posteriori solution with deterministic times. This process is illustrated in figure 4.8 for a better understanding.

This definition requires an actual execution of the problem which may not always be available. For instance, most of the problems that are available in the literature are synthetic. In this case, we propose to run a Monte-Carlo simulation. Given a fuzzy instance, we generate a sample of K possible realisations of that instance by assigning an exact duration to each task. A crucial factor in this method is the way in which we sample deterministic durations for the tasks based on their fuzzy values. We do this by simulating exact durations for tasks following a probability distribution that is consistent with the possibility distribution μ_A defined by each fuzzy duration A.

This first approach to the study of robustness is published in the paper [81] that is part of the compendium of this thesis (see Section 7.3). There, the main interest is to find a-priori solutions that yield to good schedules in the moment of their practical use, that is, to good a-posteriori solutions. To perform the Monte-Carlo simulation, four different scenarios are proposed to cover a wide range of possible situations.

- Scenario I: Given the fuzzy duration p_{ij}, the crisp duration is sampled following a probability distribution which is computed by dividing the membership function of p_{ij} by its surface.

- Scenario II: From an optimistic point of view, given the fuzzy duration p_{ij}, the crisp duration is sampled following a uniform probability distribution in the interval $[p^1_{ji}, p^1_{ij} + 0.25(p^3_{ij} - p^1_{ij})]$.

- Scenario III: From a pessimistic point of view, given the fuzzy duration p_{ij}, the crisp duration is sampled following a uniform probability distribution in the interval $[p^3_{ij} - 0.25(p^3_{ij} - p^1_{ij}), p^3_{ji}]$.

- Scenario IV: In a particularly adverse scenario characterized by a wrong prediction,

given the fuzzy duration p_{ij}, the crisp duration is sampled following a uniform probability distribution in the interval $[p^1_{ji}, p^1_{ij} + 0.25(p^3_{ij} - p^1_{ij})] \cup [p^3_{ij} - 0.25(p^3_{ij} - p^1_{ij}), p^3_{ji}]$

The idea of evaluating different alternatives according to various scenarios as a way of dealing with imprecise or poorly defined data is not new of this thesis. Indeed, a "robust" solution is, intuitively, a solution that performs "well" or "not too bad" in all scenarios [56], and an approach based on finding robust solutions should prevent from taking decisions with disastrous consequences in the case that a particularly adverse scenario should prevail at the end. In the paper, this approach is used to assess the advantages of taking into account the uncertainty during the optimisation process. -This is done by running the algorithm twice: one to solve the fuzzy instance, and one to solve the instance without taking into account the uncertainty (taking the expected duration of each task). Being the algorithm a lexicographical goal programming approach (see Section 4.2.1), we measure the percentage of real executions (simulated through the Monte-Carlo method) in which the obtained solutions reach the established target. We observe that reach the target more often when we use the solutions obtained with the algorithm when the uncertainty is modelled in the instance. In addition, we measure the performance of the solutions in terms of their objective functions. Even though both approaches behave similarly in Scenario I, in Scenario III, which represents the most pessimistic scenario, the quality of the solution obtained when modelling the uncertainty is not so deteriorated as the solution obtained without using TFNs.

4.3.2 Robustness measures

Even though the previous method allows to compare different approaches in terms of robustness to some extent, we shall need a well-defined metric to assess the robustness of a single solution. Based on the previous semantics for fuzzy scheduling problems and to be coherent with it, we adopt the definition of ϵ-robustness given in [10] for stochastic scheduling, and extend it to the fuzzy scheduling case.

Definition 4.3.1 *A predictive schedule with objective value f_{pred} is ϵ-robust for a given ϵ, if the objective value f_{exec} of the eventually executed schedule is such that:*

$$(1 - \epsilon) f_{pred} \leq f_{exec} \leq (1 + \epsilon) f_{pred} \tag{4.13}$$

which is equivalent to:

$$\frac{|f_{pred} - f_{exec}|}{f_{pred}} \leq \epsilon \tag{4.14}$$

That is, the relative error of the estimation made by the predictive schedule is bounded by ϵ. Obviously, the smaller ϵ is, the more robust is the schedule. It is worth noticing that this approach is different from the better-known approach from combinatorial optimisation, based on min-max or min-max regret criteria, which aims at constructing solutions having the best possible performance in the worst case [2]. The study of such criteria is motivated by practical applications where an anticipation of the worst case is crucial and has already been translated to the fuzzy framework [57, 111]. However, the min-max approach may be deemed as too conservative in some cases where the worst case is not that critical and an overall acceptable performance is preferred. It is in these situations where an approach such as the one proposed here might be more adequate. This definition is coherent with the semantics for fuzzy scheduling described before: the f_{pred} value is actually the estimated objective value given by the a-priori schedule, whereas the f_{exec} value is the actual objective value obtained in the a-posteriori schedule. Notice that the ϵ robustness is measured after the actual realisation of the schedule, thus in the framework

of the previous semantics, we call this an "a-posteriori robustness measure". In the case in which we need to perform a Monte-Carlo simulation to obtain possible actual environments, we shall obtain K different f_{exec} values: $f^1_{exec}, \ldots, f^K_{exec}$. If that is the case, we measure the ϵ-robustness as:

$$\overline{\epsilon} = \frac{1}{K} \sum_{i=1}^{K} \frac{\left| f_{pred} - f^i_{exec} \right|}{f_{pred}} \tag{4.15}$$

In the paper [79] that can be found in this thesis (see Section 7.4), we propose these ideas and use the $\overline{\epsilon}$ measure to solve the same question as before: is it worthy to model the uncertainty?. In this case the objective function is the makespan C_{max}, so the predicted value f_{pred} is the expected makespan of the a-priori solution, $f_{pred} = E[C_{max}]$, and the f_{exec} is the actual makespan obtained in the real scenario. We confirm the behaviour we have found in the previous section in which the solutions obtained modelling the uncertainty during the optimisation process are much robust than those obtained when considering only the expected durations. Being more concrete, the $\overline{\epsilon}$ obtained by the latter are in average a 85% worse than those obtained by the former.

The definition of this robustness measure allows to perform new comparisons and analysis in fuzzy scheduling problems. **That is the case of the paper [80] that can be found in this thesis (see Section 7.1)**. When TFNs are introduced in scheduling problems, many ordering relations can be defined to rank them (see section 3.1.1). These ranking methods have a great influence on the optimisation algorithms. For instance, every time the algorithm needs to compare two solutions to guide the search, a ranking method is required. We propose to asses which ranking method is better for fuzzy scheduling problems. However, when comparing the results obtained with an algorithm using a ranking method A and the results obtained using a ranking method B, we would need another ranking method to establish the comparison. Knowing that the performance measure cannot be used to asses the advantages of one ranking method over another, we propose to compare them in terms of robustness using the $\overline{\epsilon}$ metric. As we have seen, ranking methods can be roughly divided in two types: those based on defuzzification and those based on fuzzy binary relations. In [19] it is proposed to summarise a fuzzy set A by the value:

$$E_\beta(A) = \int_0^1 (\beta \underline{a}_\alpha + (1-\beta)\overline{a}_\alpha) d\alpha \tag{4.16}$$

where $\beta \in [0,1]$ is a pessimism value. Obviously, this value can be used in a ranking method of the first type. When we translate this formula into the specific case of TFNs, we observe than many of the already defined defuzzyfycation methods for fuzzy sets correspond actually to the cases in which $\beta \in \{0, 0.5, 1\}$. A brief summary of these ranking methods and their relation to this general formula is provided in the paper. To asses the differences between the use of one or another ranking method, three index based methods are considered by taking the three mentioned values of β in the previous formula and the co-evolutionary algorithm described in 4.2.3 is run.

The solutions obtained with each method are then compared by means of the $\overline{\epsilon}$-robustness measure. A more detailed study of the literature reveals that the Scenario I described for sampling crisp durations from the fuzzy one, can be objected to; according to [30], it is arbitrary and the obtained probability may fail to belong to $\mathcal{P}(\mu_A)$, the set of probability measures dominated by μ_A. Therefore we adopt two different approaches:

- Scenario V: Consists in considering the uniform probability distribution that is bounded by the support of the TFN. This possibility-probability transformation is motivated by several results from the literature (see [8, 32]) that justify the use

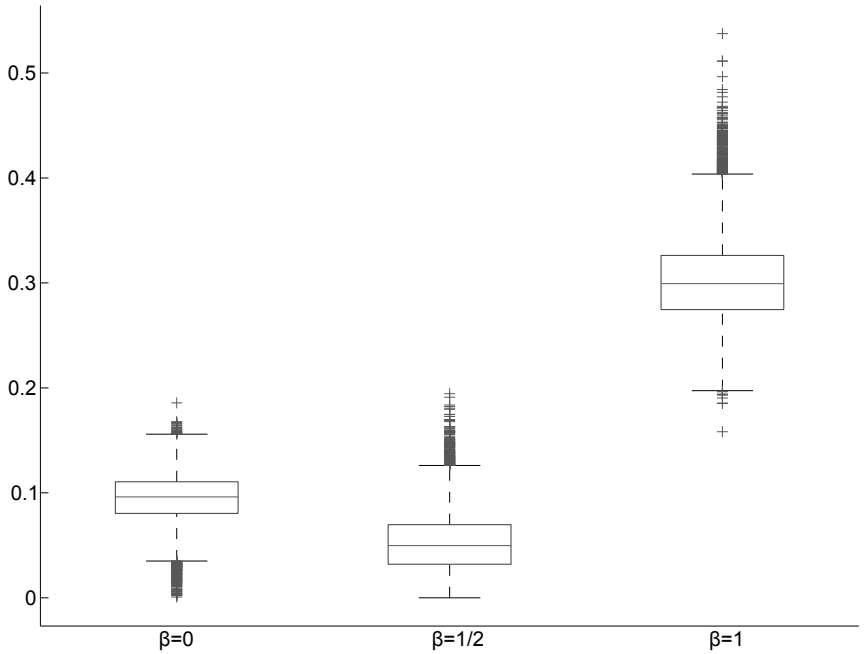


Figure 4.9: $\bar{\epsilon}$ values for solutions obtained with different $E_\beta(A)$ ranking methods.

of TFNs as fuzzy counterparts to uniform probability distributions and model-free approximations of probability distributions with bounded support.

- Scenario VI: We consider the probability distribution obtained from each fuzzy duration p_{ij} after applying the pignistic transformation obtained by considering cuts as uniformly distributed probabilities [35]. This is the probability one would obtain from the membership function of a fuzzy duration applying a generalised version of the Insufficient Reason Principle by Laplace.

In figure 4.9 we see the comparison between the use of the three ranking methods. Thanks to this metric, we can see that ranking the fuzzy numbers with $E_{\beta=0.5}(A)$, which is actually the same as index as the expected value $E[C_{max}]$, appears to be the best option in terms of robustness. Moreover, the robustness analysis allows to identify curious behaviours. For instance, using $E_{\beta=1}(A)$, which is the most optimistic case (i.e. is based on the idea that the tasks would finish earlier than expected) provides the least robust schedules. This behaviour has a natural explanation if we look at the solution graph: in scheduling, every single delay in a critical task increases in the same quantity the makespan, whereas a shorter processing time of a critical task is likely to derive in this task being critical no more, thus having a small or non-existing impact in the makespan which might be determined by a new critical path.

In addition to the $\bar{\epsilon}$ robustness measure, in this thesis we tackle robustness from a different point of view so to have different metrics. **In the paper [82] included in this thesis (see Section 8.2)**, we propose to find the equivalent to what has been called in the stochastic framework β-robust schedules [24, 116], schedules with a certain confidence level that the performance will be within a given threshold.

The membership function μ_A of a fuzzy duration A may be interpreted as a possibility distribution on the real numbers [33, 119], representing the set of more or less plausible, mutually exclusive values of a variable x (for instance, the underlying uncertain duration). Since a degree of possibility can be viewed as an upper bound of a degree of probability, μ_A also encodes a whole family of probability distributions. It is well known that for a given interval $I \subseteq \mathbb{R}$, the *possibility* and *necessity measure* that $A \in I$ are respectively

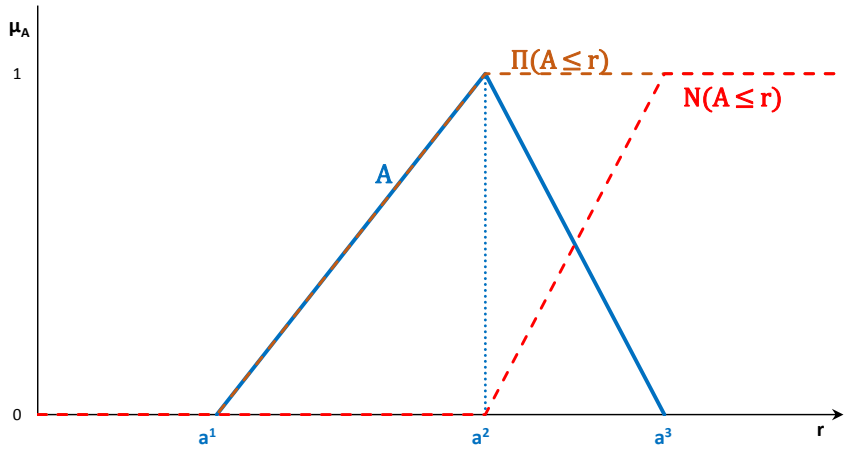


Figure 4.10: Possibility and necessity that a TFN A is less than a threshold r.

given by $\Pi(A \in I) = \sup_{x \in I} \mu_A(x)$ and $N(A \in I) = \inf_{x \in I} 1 - \mu_A(x) = 1 - \sup_{x \notin I} \mu_A(x) = 1 - \Pi(A \notin I)$, so necessity and possibility are dual measures which provide lower and upper bounds for the probability that x is in I given the information 'x is A': $N(A \in I) \leq Pr(A \in I) \leq \Pi(A \in I)$. In particular, for a TFN A, the *necessity* and the *possibility* that A is less than a given real number r are given by:

$$N(A \leq r) = \begin{cases} 0, & r \leq a^2, \\ \frac{r-a^2}{a^3-a^2}, & a^2 \leq r \leq a^3, \\ 1, & a^3 < r \end{cases} \Pi(A \leq r) = \begin{cases} 0, & r \leq a^1, \\ \frac{x-a^1}{a^2-a^1}, & a^1 \leq r \leq a^2, \\ 1, & a^2 < r \end{cases} \tag{4.17}$$

Clearly, for any value r, $N(A \leq r) \leq \Pi(A \leq r)$. Figure 4.10 illustrates both measures.

Assuming we have a fuzzy objective function f and a target or threshold $f^\star$ for it, we want to maximise the confidence that the value we obtain for f will "for sure" be less than this threshold. In our setting, this means to maximise the necessity degree that f is less than $f^\star$.

Definition 4.3.2 *A schedule with a fuzzy objective value f is said to be* necessarily β_*-robust *w.r.t. a threshold $f^\star$ if and only if $\beta_* = N(f \leq f^\star)$. Analogously, the schedule is said to be* possibly β^*-robust *w.r.t. $f^\star$ iff $\beta^* = \Pi(f \leq f^\star)$. β_* and β^* are respectively the* degrees of necessary and possible robustness *w.r.t. the threshold $f^\star$.*

Clearly, if a schedule is β^*-robust and β_*-robust w.r.t. the same threshold, and $\beta = Pr(f \leq f^\star)$, we have that $\beta_* \leq \beta \leq \beta^*$.

The degree of necessary robustness represents the degree of confidence that the value f will certainly be less than the threshold. In the following, we will consider that the objective will be to find a schedule maximising this confidence level. Obviously, by maximising the degree of necessary robustness we are also maximising the possible robustness of the schedule. Notice that for this metric we do not need an actual realisation of the schedule, thus according to the semantics we are using, we consider this as an "a-priori robustness measure". Furthermore, we can make use of the Monte-Carlo simulation described before as to have an "a-posteriori robustness" measure related to this. Once a schedule is performed in a real environment, we obtain the actual value f_{exec} of its objective function, which may be under or above the threshold $f^\star$. Due to the Monte-Carlo method simulating K different possible situations or possible realisation of the schedule, we consider the proportion κ of those values which are actually below the threshold $f^\star$. This κ value is an

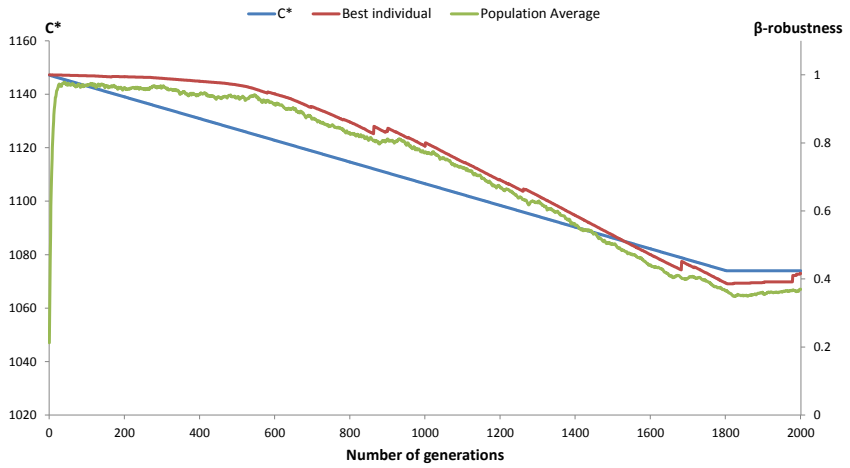


Figure 4.11: Evolution of a GA optimising β_*-robustness by using adaptive thresholds.

a-posteriori measure of the robustness of the schedule. Ideally, if a schedule has a good β_*-robustness, then it should correspond to a high κ.

$$\kappa = \frac{1}{K} \left| \{ i = 1, \ldots, K : \ f^i_{exec} < f^\star \} \right| \tag{4.18}$$

4.3.3 Robustness optimisation

The main advantage of having robustness metrics is that they allow to obtain robust schedules by finding solutions that are optimal according to those metrics. This is the case of the already mentioned paper [82], in which apart from adapting the idea of β-robustness to the case of fuzzy scheduling, we assess the quality of that measure and propose a method to optimise it. In this paper, the FOSP is considered and a GA from the literature [84] is adapted to optimise the β_*-robustness of the makespan. That is, given a threshold $C^\star$ for the makespan, we want to maximise the degree of confidence that the actual makespan of the schedule C_{max} will certainly be less than the threshold $C^\star$. In principle, to do so it would only be necessary to substitute the fitness function therein (C_{max}) by the robustness metric. However, such a straightforward approach has a serious drawback: the initial population, generated at random, consists of poor schedules, with high makespan values which, most likely, will lead to all solutions having $\beta_* = 0$ for any reasonable threshold $C^\star$. This makes it impossible for the GA to evolve until it finds a promising solution by chance. In order to overcome this drawback, we propose to adapt the GA to use an "adaptive" threshold, with successive approximations $C^\star_0, C^\star_1, \ldots$ until $C^\star$ is reached. We propose to obtain $C^\star_0$ as the most pessimistic value of the best makespan in this population, so to ensure that all the individuals on the initial population will have non-zero fitness values (in fact, the individual with the best makespan will have fitness 1), thus allowing the GA to evolve. The successive $C^\star_i$ values can be computed in many different ways. For instance, we propose to linearly decrease the threshold at each iteration until $C^\star$ is reached. Once $C^\star$ is reached, we allow the GA to evolve a little bit more so the solutions can be effectively optimised with respect to $C^\star$. This is illustrated in figure 4.11, in which we see the different $C^\star_i$ values and how the GA evolves in time. A more detailed analysis on these results is provided in the paper.

The obtained results are promising. We appreciate that even for the worst solutions $\beta_* > 0$, so by the given definitions, in all solutions the possible β^*-robustness is 1. Finally, we assess the quality of the β_* measure by following the Monte-Carlo simulation described before to obtain its a-posteriori counterpart (κ). We observe that the obtained κ values

are very close to 1. We think this can be explained by the conservative character of the necessary robustness. In fact, in all cases where the fuzzy schedule has $\beta_* > 0.6$, the makespan values for all deterministic simulations are below the threshold $C^\star$, that is $\kappa = 1$.

In most of the real situations, an expert may want to obtain solutions with a high performance quality but also robust enough. Indeed, an optimal solution may be of little or no use when it is executed if changes in the input data drastically affect its real performance. Therefore, our aim in the sequel is to optimise both, a performance or quality function and the robustness of the solution with respect to that function. Clearly, robustness measures such as the $\bar{\epsilon}$-robustness are dependent on the performance function. Therefore we opt for an optimisation strategy that allows to optimise both functions simultaneously. This is actually proposed in different papers, as for instance in [25] where the authors optimise the makespan C_{max} and its robustness. In the sequel, we focus in the FJSP in which the objective function related to performance is the makespan C_{max}, and the objective function related to robustness is $\bar{\epsilon}$. When a-posteriori robustness measure such as $\bar{\epsilon}$ are used, we may find a shortcoming. Indeed, when we evaluate the performance quality of a potential solution we do it in the fuzzy framework, but to evaluate its robustness we shall need to perform a Monte-Carlo simulation and evaluate the schedule in K different situations. This multiplies by K the computational cost of the evaluation function, thus leading to a high computational cost in the cases in which K is high. To overcome this situation, one approach may be to find an a-priori robustness measure that can be easily computed. Given that an a-priori schedule provides a most plausible value for the makespan C^2_{max}, we propose to interpret the a-priori robustness as the maximum deviation that the makespan of the executed schedule may suffer with respect to this value. We denote this measure as Rob^D. In the case in which we approximate the maximum of TFNs by $\max_I$ (see section 3.1.2), this is the maximum possible difference between the modal value and the bounds of the support of the fuzzy interval C_{max}:

$$Rob^D = \max\{C^2_{max} - C^1_{max}, C^3_{max} - C^2_{max}\} \tag{4.19}$$

Obviously, the smaller this difference, the better the robustness. It is easy to see that Rob^D thus defined measures the maximum possible difference between the makespan of a real execution and the most likely estimated makespan value. In the following, we design a metaheuristic strategy to solve the bi-criteria FJSP with C_{max} and Rob^D minimisation. We intend to minimise both objectives together, thus we adopt a Pareto-based algorithm to solve the problem. In particular, our algorithm is based on the well-known NSGA-II algorithm [27] and uses the *ActiveSGS* algorithm described in Section 4.1.1. A parametric analysis shows that the best performance is achieved by using a GOX crossover operator with probability 1.0 and an inversion based mutation with probability 0.1, as to keep diversity in the population. Diversity is a key issue when solving FJSP problems, thus we slightly change the replacement strategy of NSGA-II. Specifically, we propose to start by removing the individuals that are repeated, in the sense that there already exists at least another individual having identical objective values. Only after this elimination is the NSGA-II replacement strategy applied. In the unlikely case that such elimination causes that we do not have enough individuals for the next generation, all the non-repeated individuals pass onto the next generation, which is later completed with the best repeated individuals according to their rank level and crowding distance. Based on the good results obtained in Section 4.2.2 when hybrid algorithms were considered, we include a dominance-based tabu search in the NSGA-II. This tabu search can be seen as an improvement strategy which is applied to every new individual after evaluating it so to improve it in terms of dominance. This improving mechanism has a similar philosophy than the PAES search algorithm [59]. In our case, given a current solution S, the algorithm looks for its

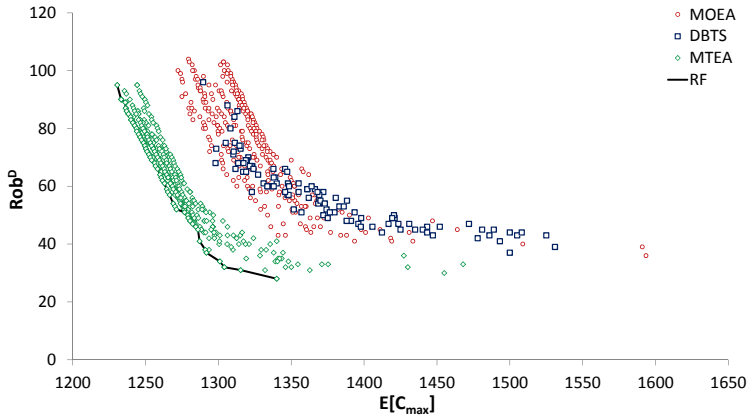


Figure 4.12: Results of the approach for optimising Rob^D and c_{max} in a FJSP instance.

neighbours using the neighbourhood structure for the FJSP from [48]. The neighbours are then classified in three different categories depending on if they dominate S, they are dominated by S or none of these cases hold. According to this division, the tabu search tries to pick one solution for the next iterations by choosing a solution that dominates S. If there is not, then take a solution that is non-dominated by S, and a solution that is dominated by S is taken only as the last option. In case of a tie, it chooses the solution with the lesser (normalised) euclidean distance to the origin (0,0). The tabu search shall stop after a fixed number of consecutive iterations without finding a solution that dominates the best solution found so far. An experimental analysis is performed using the same framework and test beds we have used in Section 4.2.2 for assessing other approaches. As we had guessed, in figure 4.12 we see that the hybrid (MTEA) between the dominance-based tabu search (DBTS) and the NSGA-II based algorithm (MOEA) provides much better results that both strategies by separate.

Finally, to assess if this a-priori robustness measure accurately predicts the a-posteriori robustness, we perform a series of experiments to check the degree of correlation between both, Rob^D and $\bar{\epsilon}$, using the Scenarios V and VI previously described. Firstly, we randomly generate a set of non-dominated solutions and measure both the average a-priori Rob^D and a-posteriori $\bar{\epsilon}$ robustness, where the latter is obtained under Scenario V. The results show a moderate correlation index R^2 in the largest tested instances (between 0.55 and 0.70) and a high positive correlation in the other ones, with an average correlation index across all instances of 0.71. Based on this, the proposed a-priori measure Rob^D may be considered to be a good predictive approach to the actual executed robustness. The same experiment is repeated, but this time using the sets of non-dominated solutions obtained after running the hybrid approach. In this case the correlation between both measures is much stronger, with an average correlation index of 0.83. The obtained results suggest that the simultaneous optimisation of the a-priori robustness and the expected makespan enhances the quality of this a-priori measure as an estimator of the real robustness. This supports the idea of optimising both objectives simultaneously, as we have done here. Finally, the same experiments are run under Scenario VI. In this case, the obtained average correlation indices were 0.83 and 0.85 respectively. This small gap shows that the optimisation proposed in this work has a greater impact in terms of robustness when there is a greater likelihood of deviations from the most probable durations, which is the case of Scenario V. Notice nonetheless that the correlation index after optimisation is quite high under both simulation scenarios, which illustrates the high reliability of Rob^D as a

predictive estimate of the schedule's executed robustness.

The definition of an a-priori robustness measure is not the only way to overcome the issue of the high computational cost of computing $\bar{\epsilon}$. Fitness approximation has been addressed from different areas, as can be seen for instance in [15, 55]. Techniques to manage surrogates for fitness evaluation include evaluating the fitness function only in some of the generations or in some individuals within a generation based on a wide range of criteria. Here we propose to use an approximation to the $\bar{\epsilon}$ robustness by using data sampling techniques. In particular, we implement this idea in an optimisation algorithm based again on NSGA-II. When a solution is to be evaluated in the algorithm, we use the *ActiveSGS*, which shall provide the C_{max} value. Then we approximate the $\bar{\epsilon}$-robustness of the solution with that of the most similar solution in the set for which $\bar{\epsilon}$ values have been previously computed, provided that this similarity exceeds a given threshold. In this surrogate model, we shall define the similarity measure and the update strategy for this set of solutions we call cache. For the former, we propose to use a similarity measure based on the fuzzy makespan values due to the $\bar{\epsilon}$ depending on the expected value of the fuzzy makespan and, the fact that the makespan of every actual realisation lies in the support of the fuzzy makespan. In the literature we can find numerous proposals to quantify the degree of similarity between two fuzzy numbers [21, 109]. However, most similarity functions are not adequate for our framework. Among all of them, we adopt a measure based on the so-called shared area between the fuzzy numbers, as we consider it is a simple but still representative measure. The shared area between fuzzy numbers with respect to the total area of these fuzzy numbers has been incorporated as a component of the measure of similarity of generalised fuzzy trapezoidal numbers in [109]. In our case, given two TFNs A and B, the degree of similarity $S(A, B)$ defined as:

$$S(A, B) = \frac{Area(A \cap B)}{Area(A \cup B)} \tag{4.20}$$

to simplify computations, when $A \cap B$ is not a triangle, we approximate the area by the maximum triangle inscribed in this plane area. We shall say that A and B are approximately equal given a small non-negative number δ iff $S(A, B) \leq \delta$. Regarding the cache updating, it is motivated by the fact that, as the algorithm converges, the chance that a new solution lies in areas of the search space with bad solutions becomes smaller. A new solution is thus added to the cache only if it is not similar enough to any of the elements already in the list, in the sense that they are not approximately equal as defined above. When this is the case, the solution is fully evaluated to obtain its actual $\bar{\epsilon}$ value and added to the list. If this is full, the new solution replaces the one in the list that has not been used for longer to estimate the value of the robustness for another individual. We perform an experimental study to assess the influence of the size of the cache and the use of more or less restrictive similarity thresholds.

We can observe that, in general, increasing the similarity threshold improves the performance of the algorithm at the cost of increasing the computational cost. This behaviour is quite natural, as having a stricter threshold causes the algorithm to fully evaluate more solutions, thus having more accurate information, at the cost of having a longer runtime. Moreover, when the similarity threshold is high and the cache size is small, the processing time is even larger than in the case in which we do not use the surrogate and fully evaluate $\bar{\epsilon}$ for every solution. This is explained by the fact that it is unlikely to find an approximately equal individual in a small-sized cache, so most of individuals are fully evaluated by the algorithm and, additionally, it loses time comparing solutions and updating the cache. On the other hand, if we use a large cache size and a low similarity threshold, the algorithm speeds up but loses quality on its performance. We could say that these parameters control the trade-off between computational cost and performance. From a thorough experimental study, we have found that for a population of size 100, using a

threshold between 0.9 and 0.95 and a cache of size 100 provides a reasonable trade-off, which allows to reduce running times in more than 20% w.r.t. not using this surrogate, without a significant loss of solution quality.

Chapter 5

Conclusions and future work

5.1 Conclusions

Along this thesis we have tackled different scheduling problems with uncertainty in the processing times. We have modelled this uncertainty by means of fuzzy numbers; in particular, we have focused on the use of triangular fuzzy numbers. To incorporate this model to scheduling problems, we have seen that two major concerns must be addressed: arithmetic and ranking. For the former, the addition can be easily computed by using the Extension Principle, but the case of the maximum is different. Given its complexity, the maximum is usually approximated in fuzzy scheduling. From the two main approximations in the literature, we have seen that the use of one of them, denoted $\max_I$ in this document, offers some desirable properties, especially for the case in which we intend to study robustness. This strongly supports using this approximation. Regarding the ranking, many different methods can be considered to order fuzzy numbers. Among them, the ones based on the expected value have shown to be very consistent. In the case of TFNs, the use of the expected value as ranking method coincides with many other popular ranking methods for fuzzy numbers. Furthermore, we have compared this method with other well-known ranking methods and the use of the expected value appears to be the one that leads to more robust solutions. In addition to uncertainty on task durations, we have also considered the use of flexible due dates, quite common in real-world situations, and measured their satisfaction by means of the agreement index, that can be seen in terms of scheduling as the degree to which a fuzzy completion time satisfies a flexible due date.

We have tackled three different scheduling problems: the Fuzzy Open Shop (FOSP), the Fuzzy Job Shop (FJSP) and the Fuzzy Flexible Job Shop (FfJSP). Surprisingly enough, we have seen that there are in the literature no formal studies about the different search spaces that can be used to look for solutions in these problems when uncertainty is considered. Due to the importance of this aspect, we have provided the first formal definition and study of types of feasible fuzzy schedules and related schedule generation schemes (SGS) for the FJSP and the FOSP. We have seen that it is not straight forward to extend well-known SGSs such as the G&T algorithm to the fuzzy case. Actually, simple extensions of this algorithm lead to losing completeness and dominance properties. We believe both the theoretical and experimental results in this work can provide a guide for designing SGS and incorporate them both into metaheuristic and exact search methods in the future. The defined categories and SGS have allowed us to design more efficient metaheuristics and study the effects of uncertainty in scheduling problems. These solving methods have been initially designed based on the best known solving methods for the respective deterministic problem. In the case of the *FOSP*, we can conclude that the large search space of this problem, makes the definition of good search spaces the key to solve it. We have seen for instance, that when we design a PSO working in a good

search space, it is able to outperform a more sophisticated memetic algorithm that looks for solutions in a wider search space. Another promising way to solve this problem seems to be to find a really good trade-off exploration-exploitation, and here the hybridisation of exact methods with metaheuristics is providing very good results. The best results we have reached until now have been by using a beam search algorithm (which is based on an exact method) combined with a genetic algorithm. Regarding the multi-criteria approaches we have tried, we have seen that hierarchical approaches like lexicographical goal programming techniques can complement the results obtained with Pareto-like approaches such as the well-known NSGA-II. Regarding the *FJSP*, many solving methods can be already found in the literature, but it seems that the best way to tackle it is to follow similar strategies than to solve the deterministic problem. Specifically, the definition of good neighbourhood structures to design more sophisticated local search algorithms appears as a key factor. Furthermore, we have proposed an hybridisation of these local search strategies with population-based evolutionary algorithms which have provided the best results known so far. The main issue we have found for this problem is the lack of a common framework in the literature to compare different solving methods. Therefore we made an exhaustive analysis of all the existing test beds and propose a new more challenging one with the hope that it can be used for other researchers to assess their methods and encourage the competition. Regarding the multiobjective approaches, it is clear that the MOEA/D algorithm is not very appropriate to solve this problem, even though it can be refined to improve its behaviour. Finally, for the case of the *FfJSP*, the fact that a task can be scheduled in different machines and that it can be easily split in two sub-problems makes this problem really interesting and challenging. We have provided several formal studies on this problem, which include the definition of new neighbourhood structures and a heuristic method for designing initial solutions. Afterwards, we have proposed a co-evolutionary algorithm with local search to exploit the nature of this problem and we obtained very promising results. However, we have also designed a hybrid algorithm that combines a genetic algorithm with the new defined neighbourhoods and a heuristic seeding, and we have obtained the best known results so far for this problem. We conclude then, that even though this is an interesting problem, it does not present a behaviour very different to what we have seen in the FJSP. As in that case, the hybridisation of local search strategies with population based evolutionary algorithms provides very good results. Finally, as it happened with the FJSP, we found ourselves in the need of proposing a new challenging test bed so to assess the quality of our methods and encourage future competition.

We have highlighted the relevance of robustness when dealing with scheduling problems under uncertain conditions. Its study is not only important for getting robust solutions, which is already a desirable target, but also allows for further studies that could no been done in terms of quality measures. Indeed, the use of robustness measures has allowed us to propose fair comparison between the use of different ranking methods for TFNs in scheduling problems. We have observed that there are many different definition of robustness when it comes to make a clear definition in terms of scheduling and that this is yet a topic with much work to do in. We have opted for adopting a preliminary work from the literature where semantics where propose for fuzzy scheduling problems. From that starting point, we began defining a framework for the study of robustness based on two different measures: the ϵ-robustness and the necessary/possible robustness. We have been developing this framework and applying it to different problems. For instance, we have been able to see that modelling the uncertainty through fuzzy numbers provides much more robust solutions than solving the associated deterministic problem in which we consider only the most plausible durations for the tasks. By means of the a-priori and a-posterior robustness concepts derived from the use of the previously mentioned semantics,

we have proposed different optimisation techniques to optimise robustness alone, as well as optimising it together with a performance measure like the makespan. However, the high computational cost of computing the ϵ-robustness, which includes a Monte-Carlo simulation, has put us on the need of proposing surrogates for this robustness measure. Even so, the results obtained so far until now are very promising. Both, the use of the necessary robustness measure and the surrogate Rob^D we have proposed, has lead us to obtain much more robust solutions than the case in which we do not optimise the robustness explicitly. We hope this research line about robustness can be helpful for other researchers and may encourage them to work in this line in the near future. We believe this is actually one of the main concerns when solving problems under uncertainty.

5.2 Future work

During the development of this thesis we have provided new definitions for many aspects of scheduling problems under uncertainty. However, we think this is a wide field of research in which there are still many thing to do. Actually, following the research lines established in this document, we may point out several topics which we think are really interesting to tackle in the near future.

In order to put bounds to this research we have been using exclusively triangular fuzzy numbers to model uncertainty in task durations. However many different fuzzy models can be found in the literature (e.g. trapezoidal fuzzy numbers or 6-point fuzzy numbers) which can be used to model uncertain processing times. These approaches are not so present in the literature of fuzzy scheduling, but we think it would be interesting to make a comparison, maybe in terms of robustness, between different models. Moreover, a comparison in terms of robustness between fuzzy approaches and stochastic approaches could be really enlightening, as there are currently many debates in the scientific community about which approach is better for dealing with uncertainty in this kind of problems.

We have provided the first formal definition of schedule categories and SGS for fuzzy scheduling problems. From this starting point and using the framework we provide here, we may look for smaller search spaces that are complete and dominant. This would be of special use in the particular case of the FOSP, in which this is a very relevant factor. Different definitions for semi-active, active and non-delay schedules can be provided in the case of the fuzzy scheduling. As we have seen, depending on the decisions we take when designing a SGS, we may many different search spaces. Furthermore, using different definitions of "left shifts" we can also define new categories. Maybe the most open topic is the design of solving methods for fuzzy scheduling problems. We have pointed out the most promising techniques for solving each of the three problems we tackled in this document. However, these techniques can be for sure improved, and new techniques can be found that outperformed the previous ones.

Finally, we think that the most important topic that must be addressed in fuzzy scheduling is robustness. We think that the main goal of modelling uncertainty is to have optimisation methods that take into account that lack of knowledge and provide solutions that can perform well in most of the situations, which is the definition of robustness. We have taken the first steps in defining a framework in which solutions can be compared in terms of robustness, and we have also proposed methods to optimise it together with performance measures. However, more research is required in this line. For instance, until now we have been focused on makespan robustness, but notice that any performance measure is subject to studying its robustness. Also many other measures for the robustness can be defined and applied, and the solving methods can be improved so they can tackle the robustness more effectively. This is, from our point of view, an emerging topic that must be really taken into account in future researches for scheduling problems with uncertainty.

Part II

Compendium of publications

Chapter 6

List of publications

In this part of the document we provide a list of all the publications that have been done as a result of the development of this thesis. For each one we shall provide the bibliographic reference and the status of the publication. In addition, for each journal paper we also report its **impact factor**. Finally, we list the sections of the report in which these papers are referred to, so to find more details on their content. The publications are divided into journal and conference papers and are sorted by publication date. We also include a list of the papers written during the development of the thesis that are under review in a journal or conference.

Notice that not all of the papers mentioned here will be present in this compendium. In particular, this compendium contains 7 papers in total: all the journal papers (5 papers) and the conference paper that were presented in a conference ranked in the ERA or CORE conference rankings (2 papers).

6.1 Journal papers

1. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop. **Fuzzy Sets and Systems**. In press (2015). Doi: 10.1016/j.fss.2014.12.003.

    - **Status:** In press. Published online (2014).
    - **Mentioned in:** Sections 3.1.1, 4.2.3 and 4.3.2.
    - Impact Factor (JCR 2013): 1.880
    - Impact Factor (5-year): 2.263
    - Journal Ranking:
        - Mathematics, Applied: 20/251 **Q1 (T1)**
        - Statistics & Probability: 14/119 **Q1 (T1)**
        - Computer Science, Theory and Methods: 16/102 **Q1 (T1)**

2. Palacios, J.J., González, M.A., Vela, C.R., González-Rodríguez, I., Puente, J.: Genetic tabu search for the fuzzy flexible job shop problem. **Computers & Operations Research** 54, 74-89 (2015). Doi: 10.1016/j.cor.2014.08.023.

    - **Status:** Published in 2015.
    - **Mentioned in:** Section 4.2.3.
    - Impact Factor (JCR 2013): 1.718
    - Impact Factor (5-year): 2.335

- Journal Ranking:
  - Engineering, Industrial: 10/43 **Q1 (T1)**
  - Operations Research & Management Science: 19/79 **Q1 (T1)**
  - Computer Science, Interdisciplinary Applications: 33/102 **Q2 (T1)**

3. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem. **AI Communications** 28(2), 239-257 (2015). Doi: 10.3233/AIC-140637.

   - **Status:** Published in 2015.
   - **Mentioned in:** Sections 4.2.1 and 4.3.1.
   - Impact Factor (JCR 2013): 0.466
   - Impact Factor (5-year): 0.582
   - Journal Ranking:
     - Computer Science, Artificial Intelligence: 105/121 **Q4 (T3)**

4. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Robust swarm optimisation for fuzzy open shop scheduling. **Natural Computing** 13(2), 145-156 (2014). Doi: 10.1007/s11047-014-9413-1.

   - **Status:** Published in 2014.
   - **Mentioned in:** Sections 4.2.1 and 4.3.2.
   - Impact Factor (JCR 2013): 0.539
   - Impact Factor (5-year): Not available
   - Journal Ranking:
     - Computer Science, Theory and Methods: 74/102 **Q3 (T3)**
     - Computer Science, Artificial Intelligence: 97/121 **Q4 (T3)**

5. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Swarm lexicographic goal programming for fuzzy open shop scheduling. **Journal of Intelligent Manufacturing**. In press (2013). Doi: 10.1007/s10845-013-0850-y.

   - **Status:** In press. Published online (2013).
   - **Mentioned in:** Section 4.2.1.
   - Impact Factor (JCR 2013): 1.142
   - Impact Factor (5-year): 1.658
   - Journal Ranking:
     - Computer Science, Artificial Intelligence: 64/121 **Q3 (T2)**
     - Engineering, Manufacturing: 22/39 **Q3 (T2)**

6.2 Conference papers

1. On Maintaining Diversity in MOEA/D: Application to a biobjective combinatorial FJSP.

   - **Conference:** GECCO 2015: Genetic and Evolutionary Computation Conference.
   - **Status:** Accepted. To be presented in July 2015. Madrid (Spain).
   - **Mentioned in:** It describes a piece of work from Section 4.2.2.
   - Conference Ranking:
     - ERA 2010: **Rank A**
     - CORE: **Rank A**

2. Surrogate-Assisted Multi-Objective Evolutionary Algorithm for Fuzzy Job Shop Problems.

   - **Conference:** MIC 2015. 11th Metaheuristics International Conference.
   - **Status:** Accepted. To be presented in June 2015. Agadir (Morocco).
   - **Mentioned in:** It describes a piece of work from Section 4.3.3.
   - Conference Ranking:
     - ERA 2010: Not ranked
     - CORE: Not ranked

3. Genetic Beam Search for Fuzzy Open Shop Problems.

   - **Conference:** META 2014. International Conference on Metaheuristics and Nature Inspired Computing. Marrakech (Morocco).
   - **Status:** Presented. Available online (http://meta2014.sciencesconf.org/37981).
   - **Mentioned in:** It describes a piece of work from Section 4.2.1.
   - Conference Ranking:
     - ERA 2010: Not ranked
     - CORE: Not ranked

4. Palacios, J.J., Vela, C.R., González-Rodríguez, I., Puente, J.: Schedule generation schemes for job shop problems with fuzziness. In: Proceedings of ECAI 2014. Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 687-692. IOS Press (2014). Doi: 10.3233/978-1-61499-419-0-687.

   - **Conference:** ECAI 2014. 21st European Conference on Artificial Intelligence. Prague (Czech Republic).
   - **Status:** Presented and published.
   - **Mentioned in:** Section 4.1.1.
   - Conference Ranking:
     - ERA 2010: **Rank A**
     - CORE 2014: **Rank A**

5. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente Peinador, J.: β-robust solutions for the fuzzy open shop scheduling. In: Information Processing and Management of Uncertainty in Knowledge-Based Systems. Communications in Computer and Information Science, vol. 442, pp. 447-456. Springer (2014). Doi: 10.1007/978-3-319-08795-5_46.

    - **Conference:** IPMU 2014. 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. Montpellier (France).
    - **Status:** Presented and published.
    - **Mentioned in:** Sections 4.3.2 and 4.3.3.
    - Conference Ranking:
        - ERA 2010: **Rank C**
        - CORE 2014: **Rank C**

6. Palacios, J.J., Vela, C.R., Puente Peinador, J., González-Rodríguez, I.: Hybrid Cooperative Coevolution for Fuzzy Flexible Job Shop Scheduling Problems. In: Proceedings of EUROFUSE 2013, pp. 199-206. University of Oviedo (2013).

    - **Conference:** EUROFUSE 2013. Workshop on Uncertainty and Imprecission Modelling in Decision Making. Oviedo (Spain).
    - **Status:** Presented and published.
    - **Mentioned in:** It describes a piece of work from Section 4.2.3.
    - Conference Ranking:
        - ERA 2010: Not ranked
        - CORE 2014: Not ranked

7. Palacios, J.J., Puente, J., González-Rodríguez, I., Vela, C.R.: Hybrid tabu search for fuzzy job shop. In: Natural and Artificial Models in Computation and Biology. Lecture Notes in Computer Science, vol. 7930, pp. 376-385. Springer (2013)

    - **Conference:** IWINAC 2013. 5th International Work-Conference on the Interplay between Natural and Artificial Computation. Mallorca (Spain).
    - **Status:** Presented and published.
    - **Mentioned in:** Section 4.2.2.
    - Conference Ranking:
        - ERA 2010: Not ranked
        - CORE 2014: Not ranked

8. Palacios, J.J., Vela, C.R., González-Rodríguez, I., Puente, J.: A Particle Swarm Solution based on Lexicographical Goal Programming for a Multiobjective Fuzzy Open Shop Problem. Proceedings of the 19th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA 2012)

    - **Conference:** RCRA 2012. 19th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion. Rome (Italy).

- **Status:** Presented and published.
- **Mentioned in:** It describes a piece of work from Section 4.2.1.
- Conference Ranking:
  - ERA 2010: Not ranked
  - CORE 2014: Not ranked

6.3 Under review

1. Benchmarks for Fuzzy Job Shop Problems.

   - **Status:** Under second review. Major changes were required.
   - **Related to:** The work done in Section 4.2.2.
   - **Sent to:** Information Sciences (Journal).
   - Impact Factor (JCR 2013): 3.893
   - Impact Factor (5-year): 3.969
   - Journal Ranking:
     - Computer Science, Information Systems: 8/135 **Q1 (T1)**

2. Robust Multiobjective Optimisation for Fuzzy Job Shop Problems.

   - **Status:** Under first review.
   - **Related to:** The work done in Section 4.3.3.
   - **Sent to:** European Journal of Operational Research (Journal).
   - Impact Factor (JCR 2013): 1.843
   - Impact Factor (5-year): 2.625
   - Journal Ranking:
     - Operations Research & Management Science: 15/79 **Q1 (T1)**

Chapter 7

Journal papers

In this chapter we include all the journal papers previously mentioned. For the sake of clarity, before each paper we shall include again its data.

7.1 Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop

In this section, we include the following publication.

- **Title:** Surrogate-Assisted Multi-Objective Evolutionary Algorithm for Fuzzy Job Shop Problems.
- **Journal:** Fuzzy Sets and Systems.
- **Year:** In press. Published online in 2014.
- Impact Factor (JCR 2013): 1.880
- Impact Factor (5-year): 2.263
- Journal Ranking:
  - Mathematics, Applied: 20/251 **Q1 (T1)**
  - Statistics & Probability: 14/119 **Q1 (T1)**
  - Computer Science, Theory and Methods: 16/102 **Q1 (T1)**

This publications contains pieces of work described in Sections 3.1.1, 4.2.3 and 4.3.2.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop. Fuzzy Sets and Systems. In press (2015). Doi: 10.1016/j.fss.2014.12.003.

Available online at www.sciencedirect.com

ScienceDirect

Fuzzy Sets and Systems ••• (••••) •••–•••

FUZZY sets and systems

www.elsevier.com/locate/fss

Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop

Juan José Palacios [a], Inés González-Rodríguez [b,*], Camino R. Vela [a], Jorge Puente [a]

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

Received 9 May 2014; received in revised form 6 October 2014; accepted 2 December 2014

Abstract

In this paper we tackle a variant of the flexible job shop scheduling problem with uncertain task durations modelled as fuzzy numbers, the fuzzy flexible job shop scheduling problem or FfJSP in short. To minimise the schedule's fuzzy makespan, we consider different ranking methods for fuzzy numbers. We then propose a cooperative coevolutionary algorithm with two different populations evolving the two components of a solution: machine assignment and task relative order. Additionally, we incorporate a specific local search method for each population. The resulting hybrid algorithm is then evaluated on existing benchmark instances, comparing favourably with the state-of-the-art methods. The experimental results also serve to analyse the influence in the robustness of the resulting schedules of the chosen ranking method.

Keywords: Flexible job shop scheduling; Robustness; Local search; Coevolutionary algorithm; Ranking of fuzzy numbers; Fuzzy processing times

1. Introduction

The importance of scheduling as a research topic is undeniable, both as a source of interesting complex combinatorial optimisation problems and as a field with multiple real applications in industry, finance, welfare, etc. In particular, shop problems in their multiple variants—for instance, incorporating flexibility or operators—can model many situations which naturally arise in manufacturing environments [1].

Fuzzy sets have contributed to enhancing the applicability of scheduling, helping to bridge the gap between classical techniques and real-world user needs. They have been used both for handling flexible constraints and uncertain data [2–5]. They are also emerging as an interesting tool for improving solution robustness, a much-desired property in real-life applications [6–8].

* Corresponding author.
E-mail addresses: palaciosjuan@uniovi.es (J.J. Palacios), gonzalezri@unican.es (I. González-Rodríguez), crvela@uniovi.es (C.R. Vela), puente@uniovi.es (J. Puente).

http://dx.doi.org/10.1016/j.fss.2014.12.003
0165-0114/

Incorporating fuzzy sets to scheduling is, however, far from trivial, and they usually require a reformulation of the problem under consideration and the development of new solving techniques. In that sense, one important issue (shared with many other applications of fuzzy sets) is how to rank different solutions when their quality is given as a fuzzy quantity. Many ranking methods have been proposed in the literature [9,10] and their use in fuzzy scheduling has been quite heterogeneous (cf. [3]). In consequence, comparisons between different proposals become difficult, if not meaningless. Furthermore, this introduces a new difficulty for the practitioner, who may be at a loss when deciding for the use a particular ranking method. In this work, we intend to shed new light into this matter, by interpreting different ranking methods in terms of optimism/pessimism of the decision maker and also by studying their influence in the robustness of the solutions. This, to our knowledge, represents a novel approach to ranking methods in the context of fuzzy scheduling.

In deterministic scheduling, the complexity of problems such as job shop means that practical approaches to solving them usually involve metaheuristic strategies [11]. Some attempts have been made to extend such methods, mostly evolutionary algorithms, to the case where uncertain durations are modelled via fuzzy intervals. In particular, the fuzzy flexible job shop problem is receiving an increasing attention, with proposals including a genetic algorithm [12], a hybrid artificial bee colony algorithm [13], an estimation distribution algorithm [14], a swarm-based neighbourhood search algorithm [15] and a coevolutionary algorithm [16].

Indeed, coevolutionary algorithms [17,18] are a special case of evolutionary algorithms which are proving to be very successful in solving complex problems [19–21]. They have been also applied for solving different scheduling problems and, in particular, for several variants of job shop. For example in [22] they are used to solve the integrated problem of process planning and scheduling, in [23], to solve the classical job shop, in [24], to the stochastic version of the problem, in [25], to the dynamic job shop and, finally, to the fuzzy flexible job shop in the above cited [16] and in [26], the latter maximising due-date satisfaction.

In the following we tackle the fuzzy flexible job shop problem, where uncertainty in task durations is modelled using fuzzy numbers. After introducing the problem, we consider different ranking methods to minimise the resulting fuzzy makespan and give a definition of schedule robustness based on average behaviour across all possible cases. We will see how the problem naturally lends itself to cooperative coevolution and we shall also propose neighbourhood structures for each population, so local search can be embedded in the coevolutionary algorithm. The experimental results will illustrate the synergy between the coevolution and the local search, as well as the competitiveness of our approach when compared to the state of the art. The results will also allow for an empirical assessment of different ranking methods in terms of solution robustness.

2. The fuzzy flexible job shop scheduling problem

The *flexible job shop scheduling problem*, fJSP in short, consists in scheduling a set of jobs $J = \{J_1, \ldots, J_n\}$ on a set of physical machines $M = \{M_1, \ldots, M_m\}$, subject to a set of constraints. There are *precedence constraints*, so each job $J_i, i = 1, \ldots, n$ consists of a sequence of n_i tasks $\Theta_i = \{\theta_{i1}, \ldots, \theta_{in_i}\}$ that must be sequentially scheduled. There are also *capacity constraints*, whereby each task θ_{ij} requires the uninterrupted and exclusive use of one machine from a subset $R_{ij} \subset M$, so the task's processing time p_{ijk} depends on the machine $M_k \in R_{ij}$.

A *feasible schedule* or solution consists of an assignment to machines of all $N = \sum_{i=1}^{n} n_i$ tasks in the set $\Theta = \bigcup_{1 \leq i \leq n} \Theta_i$ together with an allocation of starting times for each task such that all constraints hold. Alternatively, a solution can be represented as a feasible assignment of each task $\theta_{ij} \in \Theta$ to a machine $M_k \in R_{ij}$ and a task processing order for each machine in M. Indeed, given these two pieces of information, the starting time of θ_{ij}, denoted S_{ij}, is easily computed as the maximum between the completion times of the predecessor of θ_{ij} in its job and the predecessor of θ_{ij} in the machine M_k where it has been allocated, and the completion time is given by $C_{ij} = S_{ij} + p_{ijk}$. The objective is to find an *optimal* solution according to some criterion, in our case, minimise the *makespan*, which is the completion time of the last task to be executed, denoted $C_{max} = \max_{\theta_{ij} \in \Theta} C_{ij}$.

2.1. Uncertain processing times

In real-life applications, it is often the case that the exact processing time of tasks is not known in advance. However, based on previous experience, an expert may be able to estimate, for instance, an interval for the possible processing time or its most typical value. When there is little knowledge available, the crudest representation for uncertain

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

processing times would be a human-originated confidence interval. If some values appear to be more plausible than others, a natural extension is a fuzzy interval or fuzzy number. The simplest model is a *triangular fuzzy number* or *TFN*, using an interval $[a^1, a^3]$ of possible values and a modal value a^2, and with the membership function taking a triangular shape as follows:

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

We shall denote such TFN as $A = (a^1, a^2, a^3)$. For $\alpha \in (0, 1]$, its α-cut $A_\alpha = \{x : \mu_A(x) \ge \alpha\}$ is a closed interval $[\underline{a}_\alpha, \overline{a}_\alpha]$; we shall abuse notation slightly and denote its support as A_0. TFNs are to date the most widely used model for uncertain durations in the fuzzy scheduling literature.

In the 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*. In the case of the addition, it turns out that for any pair of TFNs A and B:

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

Unfortunately, computing the maximum is not that straightforward and, most importantly, the set of TFNs is not closed under this operation. For the sake of simplicity and tractability of numerical calculations, a common approach is to approximate the maximum by the TFN that results from evaluating this operation on the three defining points of each TFN, that is, for every A, B TFNs:

$$\max(A, B) \approx \max_I(A, B) = \left(\max\left(a^1, b^1\right), \max\left(a^2, b^2\right), \max\left(a^3, b^3\right)\right). \tag{3}$$

This approximation has been widely used in the scheduling literature, among others, in [27–31] or [32].

Some arguments can be given to support this approximation. First, for any two fuzzy numbers A and B, if f is a bivariate continuous isotonic function, that is, $f : \mathbb{R}^2 \to \mathbb{R}$ such that for any $a \ge a'$ and $b \ge b'$ it holds that $f(a, b) \ge f(a', b')$, then $F = f(A, B)$ is another fuzzy number such that

$$F_\alpha = \left[f(\underline{a}_\alpha, \underline{b}_\alpha), f(\overline{a}_\alpha, \overline{b}_\alpha)\right]. \tag{4}$$

Computing $f(A, B)$ is then equivalent to computing f on every α-cut. In particular, the maximum is a continuous isotonic function, so it can be calculated by evaluating two maxima of real numbers for every value $\alpha \in [0, 1]$. If seems then natural to approximate the maximum by the TFN that results from using linear interpolation, evaluating Eq. (4) only for certain values of α (this is proposed for 6-point fuzzy numbers in [28]). Given that the defining values (a^1, a^2, a^3) of a TFN A are such that $A_0 = [a^1, a^3]$ and $A_1 = [a^2, a^2]$, the approximated maximum as in (3) corresponds to such an interpolation for $\alpha = 0$ and $\alpha = 1$. Secondly, if $F = \max(A, B)$ denotes the maximum of two TFNs A and B and $G = \max_I(A, B)$ the approximated value by interpolation, then $F = G$ if A and B do not overlap and, in any case, it holds that

$$\forall \alpha \in [0, 1], \quad \underline{f}_\alpha \le \underline{g}_\alpha, \qquad \overline{f}_\alpha \le \overline{g}_\alpha. \tag{5}$$

The approximated maximum G is thus a TFN which artificially increases the value of the actual maximum F, while maintaining the support and modal value, that is, $F_0 = G_0$ and $F_1 = G_1$. This approximation can be trivially extended to the case of more than two TFNs.

2.2. Fuzzy flexible job shop

When task durations in a flexible job shop problem are given as TFNs, the resulting problem is a *fuzzy flexible job shop problem*, FfJSP in short. The objective is to minimise the makespan C_{max} according to one of the ranking methods described in Section 3.

Notice that a solution to the FfJSP is fuzzy in the sense that starting, processing and completion times of each task are fuzzy numbers, seen as possibility distributions on the actual values they may take. However, there is no uncertainty regarding the machine assignment nor the order in which tasks must be processed.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

3. Ranking methods of fuzzy numbers

For a given schedule, the makespan C_{max}, the completion time of the last task to be executed, is obtained by performing addition and maximum operations on fuzzy durations and, hence, is a TFN. If several schedules are available, the "best" one would be the one with minimal makespan, which requires comparing fuzzy numbers.

In general, fuzzy scheduling problems involve ordering or ranking fuzzy numbers representing solution performance. However, no natural total order exists in the set of fuzzy numbers and several ranking methods have been and keep being proposed in the literature (cf. [10,33,34]). Furthermore, quoting Brunelli and Mezei [10],

> It is impossible to give a final answer to the question on what ranking method is the best. Most of the time, choosing a method rather than another is a matter of preference or is context dependent.

We intend to consider some ranking methods for fuzzy numbers and their influence in the robustness of solutions of fuzzy flexible job shop scheduling problems.

Let $\mathcal{F}$ denote the set of fuzzy numbers. Ranking methods in $\mathcal{F}$ can be roughly divided in two types: those based on "defuzzification", also known as index-based methods, and those based on fuzzy binary relations. In the first case, a mapping $M : \mathcal{F} \to \mathbb{R}$ is defined which associates each fuzzy number X with a real number and then the natural ordering on the real line is used, most commonly, $X \leq_M Y$ iff $M(X) \leq M(Y)$. In the second case, a relation $M : \mathcal{F} \times \mathcal{F} \to [0, 1]$ is defined such that $M(X, Y)$ is the degree to which X is greater than Y and, consequently, if $M(X, Y) \geq M(Y, X)$, then $X \geq_M Y$.

In [35] it is proposed to summarise a fuzzy set X by the value:

$$E_\beta(X) = \int_0^1 \left(\beta \underline{x}_\alpha + (1-\beta)\bar{x}_\alpha\right) d\alpha, \tag{6}$$

where $\beta \in [0, 1]$ is a "pessimism" value. This proposal can also be found in [36]. Obviously, this value can be used in a ranking method of the first type.

For the special case of $\beta = \frac{1}{2}$, $E_{\frac{1}{2}}$ has been proposed by many authors, among others, as the neutral scalar substitute of a fuzzy interval in [37], as the expected value of a fuzzy number in [38], using the area compensation method in [39], as the generative expected value induced by the evidence X [40], as the credibilistic expectation of a fuzzy variable [41] or as the middle point of a fuzzy number defined for ranking by distance minimisation [42]. It is also the centre of the mean value of a fuzzy number as defined in [43] and the expected value of the so-called pignistic probability distribution, which is found as the centroid of the set of probabilities dominated by the possibility measure associated with X, $\mathcal{P}(\Pi_X)$ (cf. [44]).

When TFNs are considered, the special case $\beta = 0$ coincides with the index A_α suggested by [45], which simply evaluates the fuzzy number based on the rightmost point of the α-cut for a given α, in this case, $\alpha = 0.5$. According to [10], this approach is the only one which satisfies all the reasonable properties proposed in [46,47] for ordering fuzzy quantities.

3.1. Relationship with classical interval comparison and interpretation

In [48], the authors study the relationship between some well-known criteria for classical interval comparison and fuzzy ranking methods in the light of imprecise probabilities, extending some preliminary ideas which can already be found in [49]. In particular, they consider four interval comparisons:

- Weak ordering: $[\underline{x}, \bar{x}] \leq [\underline{y}, \bar{y}]$ iff $\underline{x} \leq \bar{y}$;
- Maximin: $[\underline{x}, \bar{x}] \leq [\underline{y}, \bar{y}]$ iff $\underline{x} \leq \underline{y}$;
- Maximax: $[\underline{x}, \bar{x}] \leq [\underline{y}, \bar{y}]$ iff $\bar{x} \leq \bar{y}$;
- Hurwicz: $[\underline{x}, \bar{x}] \leq_{H(\gamma)} [\underline{y}, \bar{y}]$ iff $\gamma \underline{x} + (1-\gamma)\bar{x} \leq \gamma \underline{y} + (1-\gamma)\bar{y}$.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Obviously, the Hurwicz comparison subsumes both the maximim ($\gamma = 1$) and the maximax ($\gamma = 0$). Interestingly, E_β comes down to using Hurwicz criterion on the expectation values with pessimism value β:

$$E_\beta(X) = \beta \underline{E}(X) + (1-\beta)\overline{E}(X), \tag{7}$$

where $\underline{E}$ and $\overline{E}$ denote the upper and lower expectations derived from X [48,49].

This provides us with a nice interpretation for comparing TFNs based on E_β:

- if $\beta = 0$, comparing TFNs based on E_0 would correspond to a pessimistic decision maker;
- if $\beta = 1$, comparing TFNs based on E_1 would correspond to an optimistic decision maker;
- if $\beta = \frac{1}{2}$, comparing TFNs based on $E_{\frac{1}{2}}$ would correspond to an in-between decision maker, with an equilibrium between pessimism and optimism.

3.2. Relationship with other fuzzy ranking methods

A recent numerical study in [10] suggests that several ranking methods represent very similar (referred to by the authors as *compatible*) points of view. In practice, this means that the ordering they induce in a sample of fuzzy numbers is strongly correlated. In particular, for TFNs, the ranking based on $E_{\frac{1}{2}}$ is identical to Yager's ranking based on the neutral scalar substitute from [37] and the credibilistic mean from [41] and the ranking based on these two indices is grouped as compatible with seven more ranking methods (see [10] for further detail):

- $N^{0.5}$, the parametric method defined in [50] based on a fuzzy binary relation;
- *CoM*, the method based on the centre of maxima or mean of maxima [33];
- E_p, the method based on the possibilistic mean value [51];
- CH^1, the method based on a ranking index using the concepts of fuzzy maximising and minimising sets from [52];
- *CoG*, the method based on the centre of gravity of a fuzzy set [33];
- *Med*, the method based on the median of a fuzzy number [53];
- *PD*, the method based on the *PD* relation introduced in [54].

Interestingly, the latter method extends the weak ordering of intervals, with $PD(X, Y)$ corresponding to the upper probability of the event $X \geq Y$ under the monotonic dependence assumption [48].

As for $\beta = 0$, we have already noted that E_0 coincides with Adamo's index $A_{0.5}$. According to [10], the ranking based on this index is not strongly correlated to the one based on $E_{\frac{1}{2}}$ and is therefore to produce significantly different orderings in the set of TFNs.

In conclusion, if we consider three ranking possibilities, based on E_β with $\beta = 0, \frac{1}{2}, 1$, we are in fact modelling three different behaviours of the decision maker according to his level of pessimism. But it is also the case that, by considering these three possibilities, we are taking into account many other ranking methods from the literature, either because they are based on a defuzzification index which coincides with some E_β for TFNs or because, according to [10], they yield very similar orderings to $E_{\frac{1}{2}}$ or E_0 (in the case of $A_{0.5}$).

4. Robust schedules

A fuzzy schedule does not provide exact starting times for each task. Instead, it gives a fuzzy interval of possible values for each starting time, provided that tasks are executed in the machine and in the order determined by the schedule. In fact, it is impossible to predict what the exact time-schedule will be, because it depends on the realisation of the task's durations, which is not known yet. This idea is the basis for a semantics for fuzzy schedules from [30] by which solutions to the fuzzy job shop should be understood as a-priori solutions, also called baseline or predictive schedules in the literature [55]. These solutions are found when the duration of tasks is not exactly known and a set of possible scenarios must be taken into account. When tasks are executed according to machine assignment and the ordering provided by the fuzzy schedule we shall know their real duration and, hence, obtain a real (executed) schedule, the a-posteriori solution with deterministic times.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Clearly, a fuzzy solution should yield reasonably good executed schedules in the moment of its practical use. Also, the estimates for starting and completion times and, in particular, for the makespan, should be reasonably accurate for each possible scenario of task durations. This leads us to the concept of solution robustness. As [56] puts it, "Intuitively, a solution can be considered as robust if it behaves "well" or "not too bad" in all the scenarios.". This is the idea underlying a definition of ϵ-robustness given in [57] for stochastic scheduling which can be adapted to the fuzzy flexible job shop as follows.

A predictive schedule is considered to be *robust* if the quality of the eventually executed schedule is close to the quality of the predictive schedule. In particular, a predictive schedule with objective value f^{pred} (a TFN) is ϵ-*robust* for a given ϵ if the objective value f^{exec} of the eventually executed schedule (a real value) is such that:

$$(1-\epsilon) \leq \frac{f^{exec}}{E_\beta(f^{pred})} \leq (1+\epsilon), \tag{8}$$

or, equivalently,

$$\frac{|f^{exec} - E_\beta(f^{pred})|}{E_\beta(f^{pred})} \leq \epsilon. \tag{9}$$

That is, the relative error of the estimation made by the predictive schedule is bounded by ϵ. Obviously, the smaller ϵ is, the better.

According to this interpretation, the robustness of a solution can only be measured once we have a real execution of the problem. However, it is very common in the literature to use synthetic problems instead of real ones, so no real execution is available. For those cases we propose to run a Monte-Carlo simulation to provide a surrogate of the ϵ-robustness measure. Given a fuzzy instance, we may generate a sample of K possible realisations of that instance by assigning an exact duration to each task, that is K deterministic instances in which we can evaluate the robustness of the solution. Now for each realisation $k = 1, \ldots, K$, let $C_{max,k}$ denote the makespan obtained by executing tasks according to the ordering and machine assignment provided by the predictive schedule. Then, the average ϵ-robustness of the predictive schedule, denoted $\bar{\epsilon}$, is calculated as:

$$\bar{\epsilon} = \frac{1}{K}\sum_{k=1}^{K} \frac{|C_{max,k} - E_\beta(C_{max,pred})|}{E_\beta(C_{max,pred})}, \tag{10}$$

where $C_{max,pred}$ is the makespan estimated by the predictive schedule.

Notice that a crucial factor in this method is the way in which we simulate real durations for the tasks. This is actually done by generating real durations for tasks following a probability distribution that is consistent with the possibility distribution defined by each fuzzy duration. Originally, in [30] the authors use the renormalisation technique (dividing the membership function μ_M by its surface). However, this technique can be objected to; according to [44], it is arbitrary and the obtained probability may fail to belong to $\mathcal{P}(\mu_M)$, the set of probability measures dominated by μ_M. Here we will consider instead the probability distribution obtained from each fuzzy duration after applying the pignistic transformation obtained by considering cuts as uniformly distributed probabilities [58]. This is the probability one would obtain from the membership function of a fuzzy duration applying a generalised version of the Insufficient Reason Principle by Laplace.

Our approach to robustness is different from the better-known approach from combinatorial optimisation, based on min–max or min–max regret criteria, which aims at constructing solutions having the best possible performance in the worst case [59]. The study of such criteria is motivated by practical applications where an anticipation of the worst case is crucial and has already been translated to the fuzzy framework [6,7]. However, the min–max approach may be deemed as too conservative in some cases where the worst case is not that critical and an overall acceptable performance is preferred. It is in these situations where an approach such as ϵ-robustness might be more adequate.

5. Cooperative coevolutionary algorithm for the FfJSP

Coevolutionary algorithms are advanced evolutionary techniques specially suited to solve complex problems which are decomposable. They handle two or more populations, each with its own coding schemes and recombination operators, that interact through evaluation. When all populations cooperate to build the problem solution, we talk about cooperative algorithms [17].

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Input A FfJSP instance
Output A solution
Generate a pool P of initial solutions.
$Best \leftarrow \arg\min_{S \in P}\{C_{max}(S)\}$;
Split P into populations P_0^T and P_0^M;
$i \leftarrow 1$;
while stop condition not satisfied **do**
//Evolve one iteration for population P_{i-1}^T
$P_i^T \leftarrow$ Paired individuals from P_{i-1}^T;
for each pair of individuals **do**
Apply crossover and mutation with probabilities $prob_c$ and $prob_m$;
Evaluate P_i^T using partners from P_{i-1}^M;*//Best is updated if necessary*
$P_i^T \leftarrow$ Apply 4:2 parent–children tournament between P_i^T and P_{i-1}^T;
//Evolve one iteration for population P_{i-1}^M
$P_i^M \leftarrow$ Paired individuals from P_{i-1}^M;
for each pair of individuals **do**
Apply crossover and mutation with probabilities $prob_c$ and $prob_m$;
Evaluate P_i^M using partners from P_{i-1}^T;*//Best is updated if necessary*
$P_i^M \leftarrow$ Apply 4:2 parent–children tournament between P_i^M and P_{i-1}^M;
//Apply elitism
Replace worst individual in P_i^T and in P_i^M with respective partial solutions from *Best*;
return *Best*

Algorithm 1: Main steps of the coevolutionary algorithm.

Cooperative coevolutionary algorithms have been extensively used in a varied sets of scheduling problems [22–25, 60–62]. Specifically for the FfJSP, [16] proposes a cooperative coevolutionary algorithm to minimise the makespan and [26] combines a genetic algorithm with a dynamic particle swarm optimisation method in a kind of cooperative coevolutionary algorithm. However, both proposals differ greatly from our coevolutionary algorithm, among others, in the coding schema and consequent recombination operators, as well as in the selection of cooperative partners for evaluation.

The nature of solutions to the FfJSP, with two separate components, suggests that cooperative coevolution may be specially suited for this problem. The first subproblem we face when searching for a solution to the FfJSP is to assign the processing of each task θ_{ij} to a machine $M_k \in R_{ij}$. Once this has been done, we obtain a classical job shop problem where we need to establish the order in which tasks are to be processed in each machine. We propose to separately evolve those two components in a coevolutionary framework, with a "machine assignment population" P^M in charge of evolving the machine assignment and a "task ordering population" P^T in charge of finding the processing order for tasks.

Algorithm 1 summarises the main steps of our coevolutionary algorithm. It first builds a pool of initial solutions, which are then split to form the initial populations P_0^T and P_0^M. A variable *Best* will record the best full initial solution and will then be updated throughout the evolution so it always keeps record of the best solution found so far, thus incorporating elitism. After the initialisation phase, the algorithm iterates until a stopping criterion is met. At each iteration, the individuals of each population are paired and crossover and mutation operators are applied to each pair with probability $prob_c$ and $prob_m$ respectively; each individual is then evaluated using some partners from the other population in order to have a complete solution and, finally, a replacement strategy is applied.

In the following, we describe in more detail the algorithm's components.

5.1. Genotype coding and decoding

Every individual from population P^M encodes a machine assignment as a vector $\alpha = \{\alpha_1, \ldots, \alpha_N\}$; task θ_{ij} is associated to the element in position $p = j + \sum_{l=1}^{i-1} n_l$, so $\alpha_p \in R_{ij}$ represents the machine assigned to θ_{ij}. On the other hand, an individual in P^T encodes a topological order of tasks as a permutation with repetition $\pi = \{\pi_1, \ldots, \pi_N\}$ such that $\forall l, 1 \leq \pi_l \leq n$ and $|\{\pi_l : \pi_l = i\}| = n_i, \forall i = 1, \ldots, n$. This is a permutation of the set of tasks as proposed in [63] for the JSP, where each task is represented by its job number. For example, the topological order θ_{21}, θ_{11}, θ_{22}, θ_{31}, θ_{32}, θ_{12} is encoded as (2 1 2 3 3 1).

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Input A FfJSP instance
Output An individual for each population
  $A \leftarrow \{o_{i1}, 1 \leq i \leq n\}$;
  while $A \neq \emptyset$ **do**
    $o_{ij} \leftarrow$ a task selected at random from A;
    for each $k \in M_{ij}$ **do**
      Compute EST_{ijk};
    $C^* \leftarrow \min\{EST_{ijk} + p_{ijk}, k \in M_{ij}\}$;
    $K \leftarrow \{k \in M_{ij}, EST_{ijk} + p_{ijk} = C^*\}$;
    $k^* \leftarrow$ a machine selected at random from K;
    Schedule the task θ_{ij} in machine k^* with $S_{ij} = EST_{ijk^*}$;
    $A \leftarrow A - \{o_{ij}\}$
    if $j < n_i$ **then**
      $A \leftarrow A \cup \{\theta_{ij+1}\}$;
  Split and encode the schedule.

Algorithm 2: The *Ff Insertion*.

Notice that the encoding of each population is completely independent of the other population, unlike the coevolutionary approach from [16] for the same problem. This independence allows both populations to evolve separately, interacting only at the evaluation phase. Indeed, to calculate a schedule we require a full solution, combining an individual α from P^M and an individual π from P^T. Then, a pair $(\alpha, \pi) \in P^M \times P^T$ will be decoded using the following insertion strategy. The task sequence is traversed in the order given by π and each task θ_{ij} is then scheduled in the machine M_k to which it is assigned by chromosome α. To assign a starting time to the task, it is necessary to compute a *feasible insertion interval*, that is a time interval $[t_{k,S}, t_{k,E}]$ in which machine M_k is idle and such that $t_{k,S} + p_{ijk} \leq t_{k,E}$ and $t_{k,S} \geq C_{i(j-1)}$ (if $j = 0$, $C_{i(j-1)}$ is taken to be 0); thus θ_{ij} can be processed within that time interval without violating precedence constraints. When $t_{k,S}$, $t_{k,E}$ and p_{ijk} are TFNs, we require that these inequalities hold in each of their three components (in accordance to the definition of maximum and addition). Then, the *earliest starting time* for operation θ_{ij} in machine M_k, denoted EST_{ijk}, is the smallest $t_{k,S}$ that can be found. We schedule operation θ_{ij} in machine M_k with starting time $S_{ij} = EST_{ijk}$.

5.2. Initial populations

The simplest way to generate both initial populations is to do it randomly, as done, for instance, in [16] and [26]. Alternatively, we propose a heuristic seeding method based on the insertion decoding algorithm. The idea is to use this algorithm as a production rule to generate a full schedule for the FfJSP and then encode its task ordering as an individual for P^T and its machine assignment as an individual for P^M.

The heuristic method is detailed in Algorithm 2. Let A denote the set of tasks that can be scheduled at a certain stage, initially the first task from each job. We iteratively select a random task $\theta_{ij} \in A$ and compute $C^\star = \min\{EST_{ijk} + p_{ijk} : M_k \in R_{ij}\}$, the earliest possible completion time for θ_{ij} in all machines where it can be processed. A machine M_{k*} is then randomly selected from the set $K = \{M_k : EST_{ijk} + p_{ijk} = C^\star\}$ of machines where this earliest completion time can be achieved, so θ_{ij} is scheduled in M_{k*} with starting time EST_{ijk*}. θ_{ij} is removed from A and its successor in the job is added to A, provided it exists. The process finishes when A becomes empty, i.e., all tasks have been scheduled.

5.3. Recombination operators

For the machine assignment population P^M, we use the one-point crossover: given two genotypes $\alpha^A = \{\alpha_1^A, \ldots, \alpha_N^A\}$ and $\alpha^B = \{\alpha_1^B, \ldots, \alpha_N^B\}$ the operator chooses a random point $p \in (1, N)$ and builds two offsprings α^C, α^D such that $\alpha_i^C = \alpha_i^A$ and $\alpha_i^D = \alpha_i^B$ for $i \leq p$ and $\alpha_i^C = \alpha_i^B$ and $\alpha_i^D = \alpha_i^A$ for $i > p$. A mutation strategy is also introduced, which takes a random gene α_q in the genotype associated to task θ_{ij} and changes its value to a random machine in R_{ij}.

In the case of P^T, individuals are combined using the JOX operator [64]. Given two genotypes π^A, π^B, JOX selects a random subset of jobs, copies their genes to one offspring in the same positions as in the first parent π^A, and fills

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Input A FfJSP instance, P^T, P^M, *Best*
Output Fitness values for P^T's individuals and *Best* updated
  for each individual $I_p^T \in P^T$ **do**
    //Decoding with Cooperative partners
    $S_1 \leftarrow$ Decode(I_p^T, Best solution from P^M);
    $S_2 \leftarrow$ Decode(I_p^T, Random solution from P^M);
    $S_3 \leftarrow$ Decode(I_p^T, $I_p^M \in P^M$);
    //Intensification phase
    Apply Local search to S_1, S_2 and S_3;
    $S \leftarrow \arg\min_{j=1,2,3}\{C_{max}(S_j)\}$;
    $fitness(I_p^T) \leftarrow C_{max}(S)$;
    Update I_p^T chromosome with S;
    if $C_{max}(S) < C_{max}(Best)$ **then**
      $Best \leftarrow S$;
  return $fitness(P^T)$, *Best*

Algorithm 3: Evaluation algorithm for P^T population using cooperative partners of P^M.

the remaining genes from the second parent π^B so that they maintain their relative ordering. The second offspring is formed exchanging the role of the parents. The well-known insertion operator is used for mutation. A random gene π_p in the genotype is chosen and changes its position to a random one, while keeping the relative order of the other tasks.

In both populations, all individuals are grouped in pairs for mating. Acceptance is carried out using tournament in each group of parents and offsprings, selecting the best two individuals from this group of four to pass to the next generation. Additionally, we introduce elitism, so the *Best* solution is split and the worst solutions from P^T and P^M are replaced by the task sequence and machine assignment of *Best* respectively.

5.4. Cooperative partners for evaluation

It is at the time of evaluation that populations need to cooperate: any individual only encodes part of a solution and needs to be complemented by an individual from the other population, the so-called *cooperative partner*, to conform a full solution which can be evaluated, using the decoding method above. Based on [23], we use three cooperative partners to evaluate each individual. Assuming all individuals in both populations are arbitrarily ordered, an individual in position p from one population has as cooperative partners from the other population the best individual in the previous generation, a random individual and the individual in the same position p. Finally, the individual's fitness value is the best makespan of the three obtained schedules.

As we will see in Section 6, the three full schedules built in this evaluation process may be improved using a local search strategy before selecting the best individual. We will propose a different neighbourhood structure for each population and we will explain how the specific local search procedures are embedded in the coevolutionary algorithm.

6. Local search

Evolutionary algorithms are often hybridised with local search to benefit from the synergy between both methods, i.e., between the exploitation of the local search and the exploration of the evolutionary approach. Here, we propose to apply local search to each individual after its evaluation. This means applying local search three times for each individual, one for each of its cooperative partners. In order for the computational effort to not increase excessively, we implement the local search following a simple hill climbing strategy, which is one of the fastest ones. The best solution found after the three local search processes per individual is selected and the chromosome is updated accordingly, thus introducing Lamarckism. Algorithm 3 illustrates the evaluation of chromosomes incorporating the selection of cooperative partners and the local search.

It is common in the literature to represent solutions to shop problems using acyclic graphs and define neighbourhood structures based on critical paths in these graphs. In this work we shall define different neighbourhood structures to be used on each population, taking into account their specific characteristics. We adapt the *solution graph* model

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

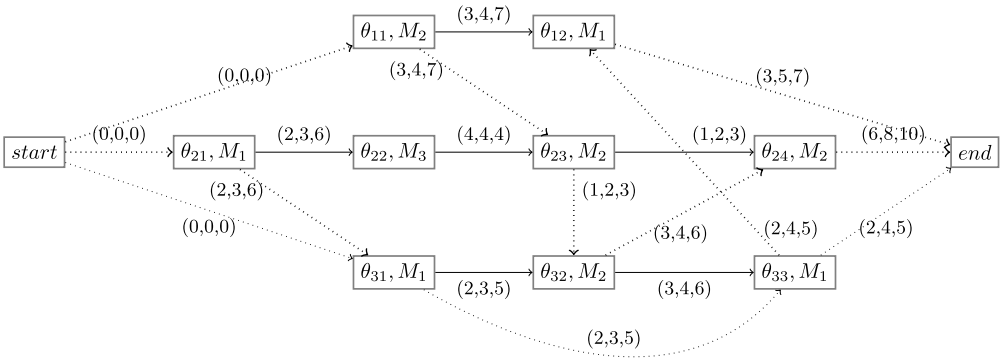


Fig. 1. Solution graph representing a feasible schedule to a problem with 3 jobs and 3 machines. The makespan is (16, 22, 31).

from [65] to incorporate machine flexibility. A solution can be represented by an acyclic directed graph G with a node for each task of the problem, labelled with the machine to which it has been assigned, plus two nodes representing fictitious tasks *start* and *end* with null processing times. There are *conjunctive arcs* representing job precedence constraints (including arcs from node *start* to the first task of each job and arcs from the last task of each job to node *end*) and *disjunctive arcs* representing machine processing orders. Each arc is weighted with the processing time of the task at the source node (a TFN in our case) in the machine where it is to be processed. This is illustrated in Fig. 1, which shows a solution graph obtained after decoding the cooperative partners: $\alpha = \{2, 1, 1, 3, 2, 2, 1, 2, 1\}$ as vector for the machine assignment and $\pi = \{2, 1, 2, 3, 2, 3, 2, 3, 1\}$ as the permutation with repetition which encodes the topological order of tasks $\theta_{21}\ \theta_{11}\ \theta_{22}\ \theta_{31}\ \theta_{23}\ \theta_{32}\ \theta_{24}\ \theta_{33}\ \theta_{12}$. The makespan of this solution is (16, 22, 31).

The starting and completion times of each task can be found by propagating constraints in the graph, and the makespan will be the completion time of task *end* (which may not coincide with the completion time of any job). In the crisp case, the makespan corresponds to the cost of a critical path, which is defined as the longest path in a solution graph from node *start* to node *end*. It is not trivial to extend concepts and algorithms related to criticality to the problem with uncertain durations (cf [3,28]). Here we adopt the definition from [65], where it is proposed that a solution graph G be decomposed into three *parallel solution graphs* G^i, $i = 1, 2, 3$, with identical structure to G but where the cost of any arc is the i-th component of the TFN labelling that arc in G. The union of all critical paths in G^i $i = 1, 2, 3$ will be the set of critical paths in G and *critical nodes and arcs* will be those within a critical path. Finally, a *critical block* is a maximal subsequence of tasks of a critical path assigned to the same machine. The makespan of the schedule is not necessarily the cost of a critical path in G, but it holds that each component C^i_{max} is the cost of a critical path in the corresponding solution parallel graph G^i.

For population P^M, representing machine assignments, based on the work of [66] and [67] for other variants of fJSP, we build a neighbour by taking a critical task θ_{ij} and assigning it to a new random machine $M_k \in R_{ij}$. For example, by assigning the machine M_3 to the critical task θ_{12} in the solution graph of Fig. 1, we obtain another solution with makespan (14, 20, 30). The resulting neighbours are always feasible, so no repair strategy is needed. The evaluation of neighbours and, hence, the cost of the local search procedure, is optimised by using makespan estimates in the line of [67], adapted to the fuzzy context.

Regarding population P^T, aimed at finding good task orderings, the local search assumes a fixed machine assignment (provided by the cooperative partner). This allows to use the structure for fuzzy job shop from [68], where a neighbour is built by reversing a critical arc at the extreme of a critical block. For instance, by reversing the critical arc $(\theta_{23}, \theta_{32})$ in the solution graph of Fig. 1, we obtain a neighbour with makespan (16, 21, 29). The motivation for this definition is that reversing critical arcs preserves feasibility and, additionally, reversing arcs inside critical blocks does not improve the makespan. Again, the evaluation of neighbours and consequent cost of the local search procedure is optimised using makespan estimates, as proposed in [68].

7. Experimental study

The goal of this experimental study is twofold. The first objective is to evaluate the hybrid coevolutionary algorithm proposed, analysing the contribution of each of its components and comparing its behaviour with the state-of-the-art methods. The second objective is to study the influence of the ranking method in the robustness of the solutions.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Table 1
Analysis of algorithm's components with best (average) expected makespan values obtained in each case.

Inst	*LB*	pBKS	*RP*	*HP*	CCEA	LS	CELS
01	28.50	30.00	62.48 (83.53)	32.30 (37.18)	30.25 (31.15)	28.75 (29.51)	28.50 (28.53)
02	45.00	45.25	81.98 (107.39)	47.80 (54.53)	45.75 (46.60)	45.25 (45.38)	45.25 (45.25)
03	43.50	47.50	90.88 (114.11)	50.23 (56.95)	47.00 (47.63)	45.25 (45.86)	43.50 (44.18)
04	33.50	37.75	76.25 (95.22)	39.25 (44.85)	37.25 (38.28)	36.00 (36.51)	34.25 (35.08)
05	37.50	62.00	110.18 (133.92)	63.33 (69.58)	58.50 (60.43)	57.75 (58.73)	53.25 (55.07)
06	40.25	63.75	103.89 (125.26)	61.63 (68.04)	57.00 (58.50)	55.75 (57.41)	52.75 (53.93)
MRE		*Best (Avg)*	*131.64 (191.10)*	*29.03 (45.17)*	*20.78 (23.84)*	*17.57 (19.67)*	*12.64 (14.63)*

Experiments are made on the instances that are available in the literature for the FfJSP which are, to our knowledge, the four instances proposed in [12] (denoted 01–04), and the two instances proposed in [16] (denoted 05, 06). All instances have $m = 10$ machines. Instances 01–04 have 10 jobs each, with a total of 40 tasks for the first two instances 01, 02 and 50 tasks for instances 03 and 04, while instances 05 and 06 have 15 jobs and 80 tasks each. Despite the relatively low number of tasks, the difficulty of the benchmark is considerable. The reason is that all instances have full flexibility, meaning that every task can be performed in any machine with varying processing time, which significantly increases the size of the search space.

Our hybrid algorithm (denoted CELS hereafter) has been implemented in C++ on a PC with a Xeon E5520 processor and 24 Gb RAM. After some preliminary testing, the parameters have been set as follows: 50 individuals per population and 100 generations as stopping criterion, crossover probability equal to 0.90 and mutation probability equal to 0.05 for both populations.

7.1. Analysis of CELS's performance

In the experiments devoted to evaluate the proposed algorithm, to keep the experimentation within reasonable bounds and to be in line with the existing works for FfJSP in the literature, we will restrict ourselves to the ranking method based on $E_{\frac{1}{2}}$. Even though the ranking methods considered in this work have been used as such by other authors for solving the FfJSP, the state-of-the-art methods for this problem do use a ranking method based on $E_{\frac{1}{2}}$, which is combined with other defuzzification indices to break ties, as originally proposed by [29]. In the remaining of this section, we might refer to $E_{\frac{1}{2}}(C_{max})$ as *expected makespan* and, since no confusion is possible, for the sake of a simpler notation we will drop the subindex and simply write $E(C_{max})$. As a reference for the quality of a solution of a given instance of FfJSP, we will use a lower bound of the expected makespan given by $LB = E(\max_j\{\sum_{i=1}^{n} p^*_{ij}\})$ where $p^*_{ij} = \min\{p_{ijk}, k \in R_{ij}\}$.

A first set of experiments is devoted to analysing the different components of our algorithm. To evaluate the heuristic seeding we generate the initial pool of solutions using two different methods, the first one applying the heuristic introduced in Section 5.2 and the second one generating the solutions at random. We then evaluate the quality of the resulting chromosomes in terms of expected makespan. A summary of the results can be seen in Table 1. The first column corresponds to the instance id and for reference the second column reports the lower bound *LB*. The best-known solution so far (pBKS) is included in the third column, while the fourth and fifth columns report the best (average) expected makespan for both pools of initial solutions. We can observe a considerable gain in quality for the heuristic solutions: the average makespan mean relative error (*MRE*) w.r.t. *LB* is reduced in average 76% across all instances when *HP* is considered instead of *RP*.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

We now evaluate the contribution of the cooperative coevolutionary algorithm (CCEA) and the local search procedure (LS). To this end, CCEA is run with no local search for the same time taken by CELS. Additionally, since CELS uses two populations of size 50 and evolve for 100 generations, we evaluate LS by generating two populations of 500 individuals and applying LS to the resulting populations (three searches per chromosome, one per cooperative partner). The last three columns in Table 1 report the best (average) expected makespan values obtained with the three methods CCEA, LS and CELS. We can see that CCEA improves the best expected makespan 18% w.r.t. the heuristic initial population, which means that the heuristic seeding provides a good starting point for the CCEA in terms of quality but also in terms of diversity, allowing for a proper evolution of the populations. In fact, the results of CCEA are already quite competitive w.r.t. the previously best-known solutions, especially as the problem size increases. LS obtains even better results than CCEA, with a MRE equal to 17.57% in the best case and equal to 19.67% in average. More importantly, CELS, combining both CCEA and LS, improves the best and average expected makespan in every instance, with the exception of 02, where the best makespan is equal for CELS and LS. As an added value for CELS, the runtime of LS is in average 134% greater than the runtime of CELS. We conclude that CELS benefits from the synergy among the good starting point provided by the HS, a good combination of exploration and exploitation in each subproblem provided by the CCEA and the LS respectively and a good cooperation between both subproblems provided by the coevolution.

We now proceed to evaluate CELS compared with the state of the art in the FfJSP. From the literature we gather that the most competitive approaches to FfJSP are the coevolutionary genetic algorithm (CGA) from [16], the swarm-based neighbourhood search algorithm (SNSA) from [15], and the hybrid artificial bee colony algorithm (hABC) from [13].

CGA is implemented in Microsoft Visual C++ 6.0 and run on a 512 RAM 1.7 G CPU PC. The population size of CGA is 150 and the maximum generation is 1000. With this configuration the CPU times ranges from 8 and 11 seconds for every one of the 20 executions. SNSA is coded in Microsoft Visual C++ 6.0 and run on a 2 G RAM 2.2 G CPU PC. The size of swarm is 100 and the number of iterations is limited to 500. With this configuration they report run times between 9 and 14 seconds, in average, in every one of the 20 runs. Finally, hABC is implemented in C++ and run on 2.83-GHz PC with 3.21-GB RAM. Parameters are set as follows: population size $2 \times n \times m$, steps for local search $n \times m$ and the number of trials after which a food source cannot be further improved (Limit) 20. For each instance, the algorithm is run 20 times and in average, every run takes between 11 and 15 seconds, but the two largest instances are not included in these results.

Table 2 shows the results of 30 runs of CELS on each instance compared to the methods above. For each method it includes the makespan of the best solution (with its expected value between brackets), the average expected makespan across all solutions found in several runs, the corresponding MRE values and the average runtime of a single run in seconds. The missing rows for instances 05 and 06 correspond to the cases when the original works do not report results on these instances. In bold we highlight the best solution from all methods, marked with "[a]" when it improves the previous best-known solution and with "[b]" when the solution is optimal, given that the lower bound is reached. We see that CELS improves the best and average values in all cases except for instance 02, where it obtains the same best expected value as SNSA. For instances 01–04 (for which all algorithms provide results), CELS reduces the MRE more than 77% in average. For instance 05, the reduction w.r.t. CGA and SNSA exceeds 38%, and on instance 06 it obtains a 46% reduction w.r.t. SNSA. Notice that solutions for instances 01 and 03 are indeed optimal, as they coincide with the lower bound.

Overall, CELS establishes new best solutions for all instances except for 02, where it obtains the same expected makespan as SNSA. Regarding the average expected makespan, not only is CELS significantly better than the other methods, but it also improves the previously known best values.

7.2. Influence of fuzziness and ranking methods in robustness

In this subsection we propose to evaluate in terms of ϵ-robustness the behaviour of the predictive schedules obtained using different ranking methods for fuzzy numbers. In our case, for each instance we obtain three predictive schedules, namely the schedules obtained after solving the fuzzy problem with CELS using each of the three fuzzy ranking methods from Section 3. The surrogate robustness of each predictive schedule is computed as explained in Section 4. The $\bar{\epsilon}$ value for each ranking method will be denoted $\bar{\epsilon}_0, \bar{\epsilon}_{\frac{1}{2}}$ and $\bar{\epsilon}_1$, corresponding respectively to $\beta = 0$, modelling the pessimistic decision maker, $\beta = \frac{1}{2}$, the compromising decision maker, and $\beta = 1$, the optimistic decision maker.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

Table 2
Summary of results in FfJSP instances with best-known solutions in **bold**.

Instance (*LB*)	Algor.	$Best(C_{max})$ ($E(Best(C_{max}))$)	*Avg E*	MRE		Time (s)
				Best	Avg	
01	CGA	21, 29, 41 (30.00)	33.18	5.26	16.40	8.3
(28.50)	SNSA	21, 29, 42 (30.25)	31.68	6.14	11.14	8.7
	hABC	19, 30, 43 (30.50)	32.15	7.02	12.81	9.9
	CELS	**21, 28, 37** (28.50)[a,b]	**28.53**	**0.00**	**0.09**	1.9
02	CGA	32, 47, 57 (45.75)	47.45	1.67	5.44	8.3
(45.00)	SNSA	**35, 43, 60 (45.25)**	47.05	**0.56**	4.56	8.9
	hABC	33, 46, 58 (45.75)	47.70	1.67	6.00	10.9
	CELS	**32, 46, 57 (45.25)**	**45.25**	**0.56**	**0.56**	2.3
03	CGA	34, 47, 63 (47.75)	51.00	9.77	17.24	10.7
(43.50)	SNSA	36, 46, 62 (47.50)	51.25	9.20	17.82	11.4
	hABC	33, 47, 64 (47.75)	50.70	9.77	16.55	14.8
	CELS	**31, 43, 57 (43.50)**[a,b]	**44.18**	**0.00**	**1.55**	3.0
04	CGA	26, 37, 51 (37.75)	40.80	12.69	21.79	10.8
(33.50)	SNSA	26, 39, 53 (39.25)	41.45	17.16	23.73	11.5
	hABC	23, 38, 53 (38.00)	40.45	13.43	20.75	13.9
	CELS	**24, 33, 47 (34.25)**[a]	**35.08**	**2.24**	**4.73**	2.7
05	CGA	42, 62, 82 (62.00)	65.95	65.33	75.87	23.9
(37.50)	SNSA	40, 65, 93 (65.75)	68.53	75.33	82.73	14.2
	CELS	**35, 53, 72 (53.25)**[a]	**55.07**	**42.00**	**46.84**	6.7
06	SNSA	46, 63, 83 (63.75)	65.65	58.39	63.11	14.4
(40.25)	CELS	**35, 52, 72 (52.75)**[a]	**53.93**	**31.06**	**34.00**	6.7

[a] Improves the previous best known solution.
[b] Optimal solution is reached.

We run CELS 30 times on each instance for each ranking method, thus obtaining 30 predictive schedules. Each predictive schedule is then evaluated via a Monte-Carlo simulation with $K = 1000$, yielding a total of 30 $\bar{\epsilon}_\beta$ values for each ranking method ($\beta = 0, \frac{1}{2}, 1$). Fig. 2 depicts for each instance, the average $\bar{\epsilon}_\beta$ value of each ranking method across the 30 predictive schedules, while the box plot in Fig. 3 illustrates in more detail the distribution of ϵ-values obtained across the 30 predictive schedules for instance 06.

The predictive schedules obtained from the fuzzy problem using a compromising approach to rank fuzzy numbers appears to be the best option in terms of robustness (with smaller prediction error ϵ), being the pessimistic approach also quite good and clearly much better than the schedule obtained from the optimistic one. This behaviour has a natural explanation: in scheduling, every single delay in a critical task increases in the same quantity the makespan, whereas a shorter processing time of a critical task is likely to derive in this task being critical no more, thus having a small or non-existing impact in the makespan which might be determined by a new critical path.

To enhance the conclusions of the experimental comparison based on ϵ-robustness among solutions obtained with different raking methods, we have conducted some statistical analysis on instance 06, which seems to be the hardest one. We take for each value of β, the ϵ-robustness values $\bar{\epsilon}_\beta$ given by 30 runs of CELS. In a preliminary analysis, the Kolmogorov–Smirnov test rejected the hypotheses of normality so we have used non-parametric statistical techniques. Specifically, we have performed a Friedman test to rank the three different sets of data (corresponding to $\bar{\epsilon}_\beta$ for $\beta = 0, \frac{1}{2}, 1$) and deduce whether there are significant differences among the robustness of solutions for varying β. With a p-value ≈ 0, the mean rank values are 1.873 for $\beta = 0$, 1.127 for $\beta = \frac{1}{2}$ and 3.000 for $\beta = 1$, which show that the ranking method used for TFNs has a significant influence on the robustness of the solutions. Additionally, to have a pairwise comparison, we have made a Mann–Whitney-U test over every pair of samples: $\beta = 0$ vs. $\beta = \frac{1}{2}$, $\beta = 0$ vs. $\beta = 1$, and $\beta = \frac{1}{2}$ vs. $\beta = 1$ with p-value ≈ 0 in all cases. It is clear that ranking TFNs with expected value ($\beta = \frac{1}{2}$, corresponding to a decision maker who compromises between optimism and pessimism) gives the most robust solutions, whereas using a ranking with an optimistic interpretation seems to be the worst choice in scheduling problems.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

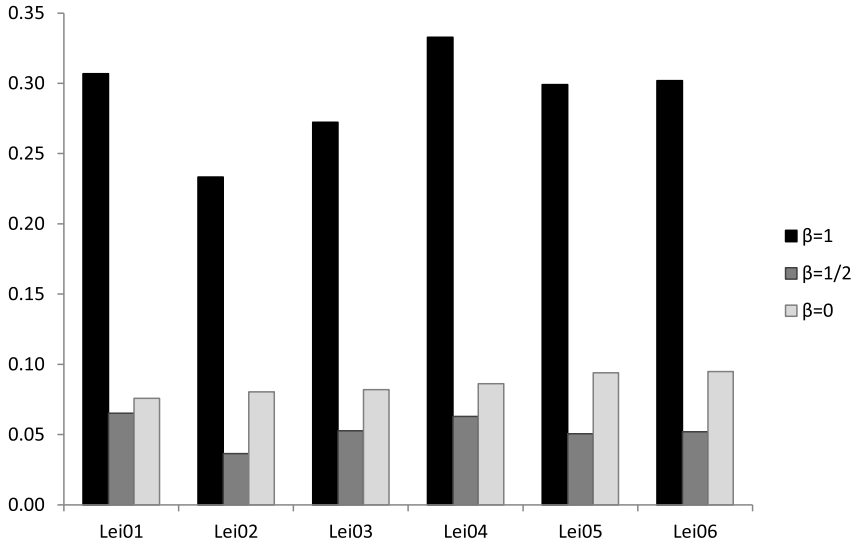


Fig. 2. Mean $\bar{\epsilon}_\beta$ values for the different ranking methods.

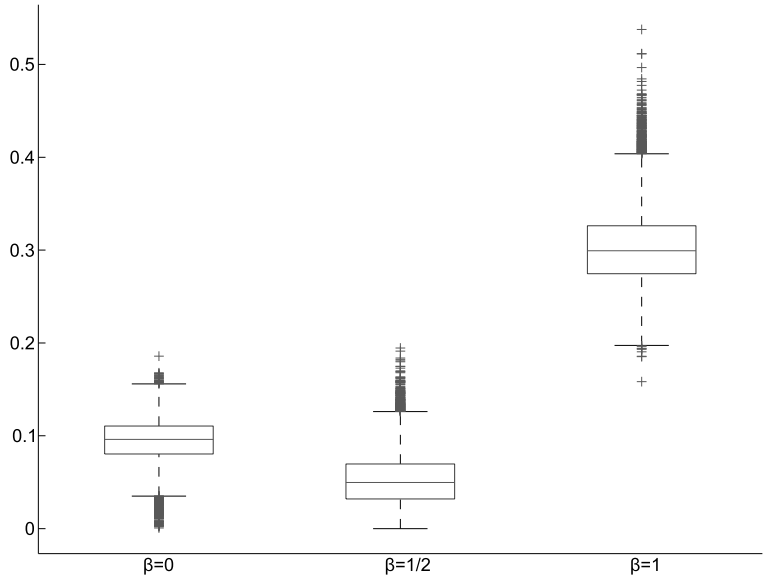


Fig. 3. $\bar{\epsilon}_\beta$ values for the different ranking methods on instance 06.

A final set of experiments is conducted to assess the benefit in terms of robustness of using fuzzy sets. Indeed, although modelling uncertainty seems to be a natural approach to exploit all the available information, it may happen that from a practical point of view this makes no great difference with respect to solving the deterministic problem obtained by taking only the most likely duration of each task, that is the modal values. If this was the case, then it would not be worth in practice to increase the complexity of the problem by considering fuzzy numbers. To check if this is the case, we have run some additional experiments in the same line as we did above, now comparing the predictive fuzzy schedule obtained with $E_{\frac{1}{2}}$ and the predictive deterministic schedule obtained solving the defuzzified problem, again with 30 runs of CELS. The predictive schedules obtained from the defuzzified problem have a slightly better makespan value, compared to the expected makespan of the fuzzy schedules. However, the deterministic solutions are much worse than the fuzzy ones in terms of robustness, as illustrated in Fig. 4. Indeed, the $\bar{\epsilon}$ values obtained after solving the deterministic instances are in average 73.62% worse than the $\bar{\epsilon}_{1/2}$ values obtained with the fuzzy schedules as shown above. We conclude that, even though considering the defuzzified problem in principle seems to yield better solutions, in fact the slightly better quality of these solutions may not compensate their lower robustness,

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

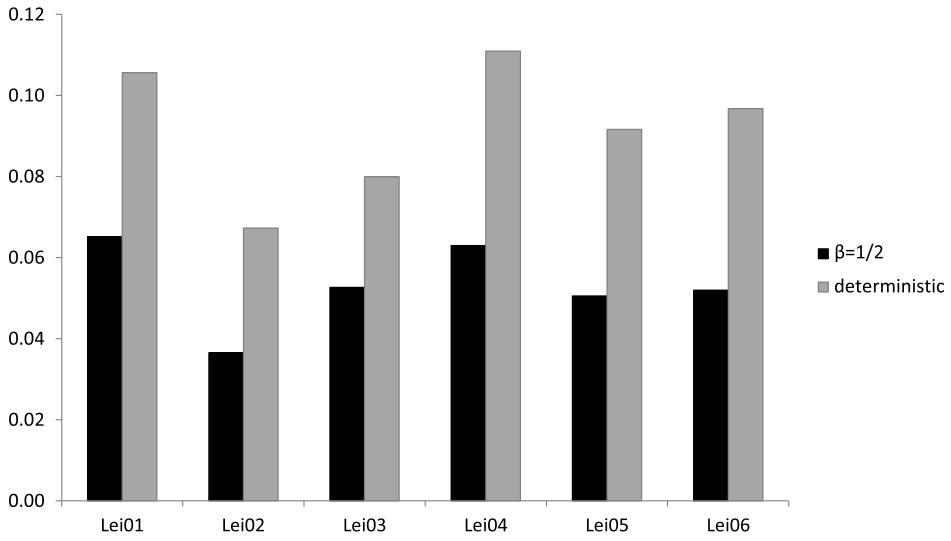


Fig. 4. $\bar{\epsilon}$ values for deterministic and fuzzy predictive schedules.

given that the outcome of a real execution is much more unpredictable than when fuzzy information about durations is taken into account.

In summary, the above results mean that the ranking method chosen to compare TFNs (and hence, to minimise makespan) has a clear influence in the robustness of the resulting predictive schedule, with the compromising approach being the one that yields better robustness among all the considered approaches. Notice however that this does not necessarily mean that the solution with optimal makespan according to the compromising approach is a solution with optimal robustness. Additionally, we have seen that the predictive schedules benefit from using fuzzy sets in terms of increased robustness compared to solving a defuzzified problem.

8. Conclusions

We have tackled the flexible job shop scheduling problem with fuzzy durations and have proposed a new cooperative coevolutionary algorithm hybridised with local search, named CELS, to solve it. The experimental results have assessed the quality of the initial heuristic seeding and the synergy between coevolution and local search. They have also shown that CELS outperforms the state-of-the-art methods, establishing new best known solutions and, in two cases, even finding the optimal solution. In addition, we have assessed the behaviour of several ranking methods for TFNs in these problems by means of a robustness measure. This measure accounts for the average behaviour of predictive schedules in real situations. We have seen that the use of different ranking methods actually affects the robustness of the algorithm's outcome. Due to the nature of scheduling problems, this outcome is more robust when ranking is based on the expected value of the TFNs, modelling a decision maker who compromises between pessimism and optimism, followed by the approach modelling a pessimistic decision maker, with the optimistic method being definitely worse. These results strongly support the use of the expected value as a ranking method for fuzzy scheduling problems. Additionally, the robustness analysis further supports the use of fuzzy sets throughout the optimisation process, compared to the option of simplifying the problem by turning it into a deterministic one.

Acknowledgements

This research has been supported by the Spanish Government (TIN2013-46511-C2-2-P) under Grants FEDER TIN2010-20976-C02-02 and MTM2010-16051 and by the Principality of Asturias Government under FICYT Grants FC-13-COF13-035 and BP13106.

References

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

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

[2] F. Herrera, J.L. Verdegay, Fuzzy sets and operations research: perspectives, Fuzzy Sets Syst. 90 (1997) 207–218.
[3] D. Dubois, H. Fargier, P. Fortemps, Fuzzy scheduling: modelling flexible constraints vs. coping with incomplete knowledge, Eur. J. Oper. Res. 147 (2003) 231–252.
[4] P. Fortemps, Editorial. Fuzzy sets in scheduling and planning, Eur. J. Oper. Res. 147 (2003) 229–230.
[5] B.K. Wong, V.S. Lai, A survey of the application of fuzzy set theory in production and operations management: 1998–2009, Int. J. Prod. Econ. 129 (2011) 157–168.
[6] J. Wang, A fuzzy robust scheduling approach for product development projects, Eur. J. Oper. Res. 152 (2004) 180–194.
[7] A. Kasperski, M. Kule, Choosing robust solutions in discrete optimization problems with fuzzy costs, Fuzzy Sets Syst. 160 (2009) 667–682.
[8] J.J. Palacios, I. González-Rodríguez, C.R. Vela, J. Puente, Robust swarm optimisation for fuzzy open shop scheduling, Nat. Comput. 13 (2) (2014) 145–156.
[9] 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.
[10] M. Brunelli, J. Mezei, How different are ranking methods for fuzzy numbers? A numerical study, Int. J. Approx. Reason. 54 (2013) 627–639.
[11] E.-G. Talbi, Metaheuristics. From Design to Implementation, Wiley, 2009.
[12] D. Lei, A genetic algorithm for flexible job shop scheduling with fuzzy processing time, Int. J. Prod. Res. 48 (10) (2010) 2995–3013.
[13] L. Wang, G. Zhou, Y. Xu, L. Min, A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem, Int. J. Prod. Res. 51 (12) (2013) 3593–3608.
[14] 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, Int. J. Prod. Res. 51 (12) (2013) 3779–3793.
[15] D. Lei, X. Guo, Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling, Int. J. Prod. Res. 50 (6) (2012) 1639–1649.
[16] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Appl. Soft Comput. 12 (2012) 2237–2245.
[17] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[18] E. Popovici, A. Bucci, R.P. Wiegand, E.D. de Jong, Coevolutionary principles, in: Handbook of Natural Computing, Springer, 2012, pp. 987–1033.
[19] L. Antonio, C. Coello Coello, Use of cooperative coevolution for solving large scale multiobjective optimization problems, in: 2013 IEEE Congress on Evolutionary Computation (CEC), 2013, pp. 2758–2765.
[20] J. Derrac, S. García, F. Herrera, IFS-CoCo: instance and feature selection based on cooperative coevolution with nearest neighbor rule, Pattern Recognit. 43 (2010) 2015–2082.
[21] C.-K. Goh, K.C. Tan, A competitive–cooperative coevolutionary paradigm for dynamic multiobjective optimization, IEEE Trans. Evol. Comput. 13 (1) (2009) 103–127.
[22] Y.K. Kim, K. Park, J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, Comput. Oper. Res. 30 (8) (2003) 1151–1171.
[23] Z. Hong, W. Jian, A cooperative coevolutionary algorithm with application to job shop scheduling problem, in: IEEE International Conference on Service Operations and Logistics, and Informatics 2006, SOLI'06, IEEE, 2006, pp. 746–751.
[24] J. Gu, M. Gu, C. Cao, X. Gu, A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem, Comput. Oper. Res. 37 (5) (2010) 927–937.
[25] S. Nguyen, M. Zhang, M. Johnston, K.C. Tan, Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming, IEEE Trans. Evol. Comput. 18 (2) (2014) 193–208, http://dx.doi.org/10.1109/TEVC.2013.2248159.
[26] Z. Xiang, B. Zhenqiang, W. Guijun, P. Quanke, Optimization of fuzzy job-shop scheduling with multi-process routes and its co-evolutionary algorithm, in: 2011 International Conference on Intelligent Computation Technology and Automation (ICICTA), vol. 1, IEEE, 2011, pp. 866–870.
[27] M. Kuroda, Z. Wang, Fuzzy job shop scheduling, Int. J. Prod. Econ. 44 (1996) 45–51.
[28] P. Fortemps, Jobshop scheduling with imprecise durations: a fuzzy approach, IEEE Trans. Fuzzy Syst. 7 (1997) 557–569.
[29] M. Sakawa, R. Kubota, Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms, Eur. J. Oper. Res. 120 (2000) 393–407.
[30] I. González Rodríguez, J. Puente, C.R. Vela, R. Varela, Semantics of schedules for the fuzzy job shop problem, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 38 (3) (2008) 655–666.
[31] D. Lei, Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems, Int. J. Adv. Manuf. Technol. 37 (2008) 157–165.
[32] Q. Niu, B. Jiao, X. Gu, Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time, Appl. Comput. Math. 205 (2008) 148–158.
[33] W. Van Leekwijck, E.E. Kerre, Defuzzification: criteria and classification, Fuzzy Sets Syst. 108 (1999) 159–178.
[34] W. Wang, Z. Wang, Total orderings defined on the set of all fuzzy numbers, Fuzzy Sets Syst. 243 (2014) 131–141.
[35] L.M. Campos Ibañez, A. González Muñoz, A subjective approach for ranking fuzzy numbers, Fuzzy Sets Syst. 29 (1989) 145–153.
[36] T.L. Liou, M.J. Wang, Ranking fuzzy numbers with integral value, Fuzzy Sets Syst. 50 (1992) 247–255.
[37] R.R. Yager, A procedure for ordering fuzzy subsets of the unit interval, Inf. Sci. 24 (1981) 143–161.
[38] S. Heilpern, The expected value of a fuzzy number, Fuzzy Sets Syst. 47 (1992) 81–86.
[39] P. Fortemps, M. Roubens, Ranking and defuzzification methods based on area compensation, Fuzzy Sets Syst. 82 (1996) 319–330.
[40] S. Chanas, M. Nowakowski, Single value simulation of fuzzy variable, Fuzzy Sets Syst. 25 (1988) 43–57.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

[41] B. Liu, Y.K. Liu, Expected value of fuzzy variable and fuzzy expected value models, IEEE Trans. Fuzzy Syst. 10 (2002) 445–450.
[42] B. Asady, A. Zendehman, Ranking fuzzy numbers by distance minimization, Appl. Math. Model. 31 (2007) 2589–2598.
[43] D. Dubois, H. Prade, The mean value of a fuzzy number, Fuzzy Sets Syst. 24 (1987) 279–300.
[44] D. Dubois, Possibility theory an statistical reasoning, Comput. Stat. Data Anal. 51 (2006) 47–69.
[45] J.M. Adamo, Fuzzy decision trees, Fuzzy Sets Syst. 4 (3) (1980) 207–219.
[46] X. Wang, E.E. Kerre, Reasonable properties for the ordering of fuzzy quantities (I), Fuzzy Sets Syst. 118 (2001) 375–385.
[47] X. Wang, E.E. Kerre, Reasonable properties for the ordering of fuzzy quantities (II), Fuzzy Sets Syst. 118 (2001) 387–405.
[48] I. Couso, S. Destercke, Ranking of fuzzy numbers seen through the imprecise probabilistic lense, in: B. De Baets, J. Fodor, S. Montes (Eds.), Proceedings of EUROFUSE 2013, University of Oviedo, 2013, pp. 73–82.
[49] D. Dubois, The role of fuzzy sets in decision sciences: old techniques and new directions, Fuzzy Sets Syst. 184 (2011) 3–28.
[50] K. Nakamura, Preference relation on a set of fuzzy utilities as a basis for decision making, Fuzzy Sets Syst. 20 (2) (1986) 147–162.
[51] C. Carlsson, R. Fullér, On possibilistic mean value and variance of fuzzy variables, Fuzzy Sets Syst. 122 (2) (2001) 315–326.
[52] S.-H. Chen, Ranking fuzzy numbers with maximizing set and minimizing set, Fuzzy Sets Syst. 17 (2) (1985) 113–129.
[53] S. Bodjanova, Median value and median interval of a fuzzy number, Inf. Sci. 172 (1) (2005) 73–89.
[54] D. Dubois, H. Prade, Unfair coins and necessity measures: towards a possibilistic interpretations of histograms, Fuzzy Sets Syst. 10 (1983) 15–20.
[55] W. Herroelen, R. Leus, Project scheduling under uncertainty: survey and research potentials, Eur. J. Oper. Res. 165 (2005) 289–306.
[56] R. Kalaï, C. Lamboray, D. Vanderpooten, Lexicographic α-robustness: an alternative to min–max criteria, Eur. J. Oper. Res. 220 (2012) 722–728.
[57] J. Bidot, T. Vidal, P. Laboire, A theoretic and practical framework for scheduling in stochastic environment, J. Sched. 12 (2009) 315–344.
[58] D. Dubois, H. Prade, S. Sandri, On possibility/probability transformations, in: Fuzzy Logic, in: Theory and Decision Library, vol. 12, Kluwer Academic, 1993, pp. 103–112.
[59] H. Aissi, C. Bazgan, D. Vanderpooten, Min–max and min–max regret versions of combinatorial optimization problems: a survey, Eur. J. Oper. Res. 197 (2009) 427–438.
[60] G. Danoy, B. Dorronsoro, P. Bouvry, Multi-objective cooperative coevolutionary algorithms for robust scheduling, in: Proc. of EVOLVE-A Bridge Between Probability, Set Oriented Numerics and Evolutionary Computation, 2011.
[61] J. Ren, M. Harman, M. Di Penta, Cooperative co-evolutionary optimization of software project staff assignments and job scheduling, in: Search Based Software Engineering, in: Lecture Notes in Computer Science, vol. 6956, Springer, 2011, pp. 127–141.
[62] S. Su, H. Yu, Z. Wu, W. Tian, A distributed coevolutionary algorithm for multiobjective hybrid flowshop scheduling problems, Int. J. Adv. Manuf. Technol. 70 (1–4) (2014) 477–494.
[63] C. Bierwirth, A generalized permutation approach to jobshop scheduling with genetic algorithms, OR Spektrum 17 (1995) 87–92.
[64] I. Ono, M. Yamamura, S. Kobayashi, A genetic algorithm for job-shop scheduling problems using job-based order crossover, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, IEEE, 1996, pp. 547–552.
[65] 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.
[66] M. Mastrolilli, L. Gambardella, Effective neighborhood functions for the flexible job shop problem, J. Sched. 3 (1) (2000) 3–20.
[67] 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.
[68] 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.

Please cite this article in press as: J.J. Palacios et al., Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.12.003

7.2 Genetic tabu search for the fuzzy flexible job shop problem

In this section, we include the following publication.

- **Title:** Genetic tabu search for the fuzzy flexible job shop problem.
- **Journal:** Computers & Operations Research.
- **Year:** 2015.
- Impact Factor (JCR 2013): 1.718
- Impact Factor (5-year): 2.335
- Journal Ranking:
  - Engineering, Industrial: 10/43 **Q1 (T1)**
  - Operations Research & Management Science: 19/79 **Q1 (T1)**
  - Computer Science, Interdisciplinary Applications: 33/102 **Q2 (T1)**

This publications contains pieces of work described in Section 4.2.3.

Palacios, J.J., González, M.A., Vela, C.R., González-Rodríguez, I., Puente, J.: Genetic tabu search for the fuzzy flexible job shop problem. Computers & Operations Research 54, 74-89 (2015). Doi: 10.1016/j.cor.2014.08.023.

Computers & Operations Research 54 (2015) 74–89

Contents lists available at ScienceDirect

Computers & Operations Research

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

Genetic tabu search for the fuzzy flexible job shop problem

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

[a] Department of Computing, University of Oviedo, Spain
[b] Department of Mathematics, Statistics and Computing, University of Cantabria, Spain

ARTICLE INFO

Available online 6 September 2014

Keywords:
Genetic algorithms
Neighbourhood structure
Local search
Heuristics
Flexible job shop scheduling
Fuzzy processing times

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.

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

* Corresponding author.
E-mail addresses: palaciosjuan@uniovi.es (J.J. Palacios), mig@uniovi.es (M.A. González), crvela@uniovi.es (C.R. Vela), gonzalezri@unican.es (I. González-Rodríguez), puente@uniovi.es (J. Puente).

http://dx.doi.org/10.1016/j.cor.2014.08.023
0305-0548/

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 needs to only provide an interval of possible values and the most typical value, which is usually easier than accurately defining a probability distribution. Quantitative 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 (cf. [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 a 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 no 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 us 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 us 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 [16,17] 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]. Ref. [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 $O=\bigcup_{1\le i\le n}O_i$ 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 *α-cuts* $A_\alpha=\{u\in\mathbb{R}:\mu_A(u)\ge\alpha\}$, $\alpha\in(0,1]$, are intervals. A fuzzy interval is a *fuzzy number* if its α-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:

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

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

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

It induces a total ordering $\le_E$ in the set of fuzzy intervals [20], where for any two fuzzy intervals M,N $M\le_E N$ if and only if $E[M]\le E[N]$. 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 $\forall i, a^i\le b^i$, then $A\le_E B$.

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 and 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

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

where $[\underline{f}_\alpha,\overline{f}_\alpha]$ is the α-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 [20,24,43–48,23,21] 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 $\max_R(A,B)\le_E\max_I(A,B)$, $\forall A,B$ (and therefore $\max_R(A,B)\le_R\max_I(A,B)$ too). Notice as well that it is not guaranteed that $\max_R$ maintains the support or 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 (α, π) where α 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 π 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, respectively, and $PM_{o_{ij}}$ and $SM_{o_{ij}}$ the operations right before and after o_{ij} in the machine sequence in a solution (α,π), respectively. 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 (α,π) 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 α and the operation processing ordering π 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 (α, π), 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 (α, π) 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 α and a feasible operation processing order π can be represented by an acyclic directed graph $G(\alpha, \pi) = (V, A \cup R(\alpha, \pi))$ where each node 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 π for all operations assigned by α 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, Fig. 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 α is explicit in the label of each node. In accordance with the 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 side of Fig. 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 are 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 ith 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.

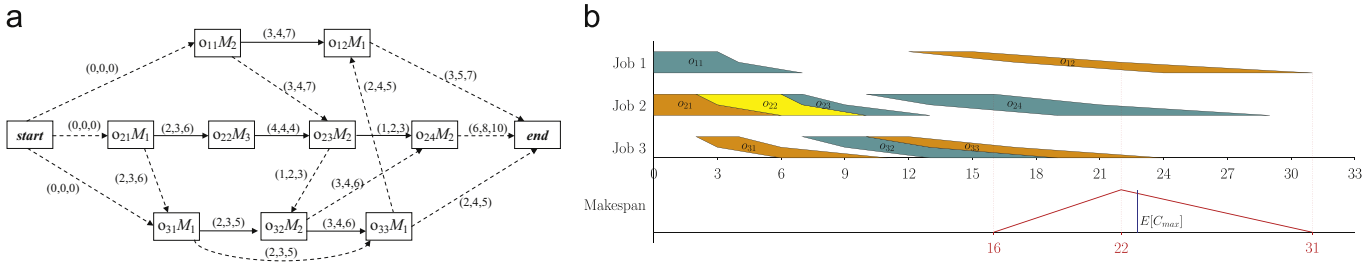


Fig. 1. A feasible schedule to a problem with three jobs and three machines. The makespan is (16, 22, 31). (a) Solution graph. (b) Gantt chart (job-oriented). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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 the Lamarckian evolution). The flow chart of the resulting hybrid algorithm can be seen in Fig. 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 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 three 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.

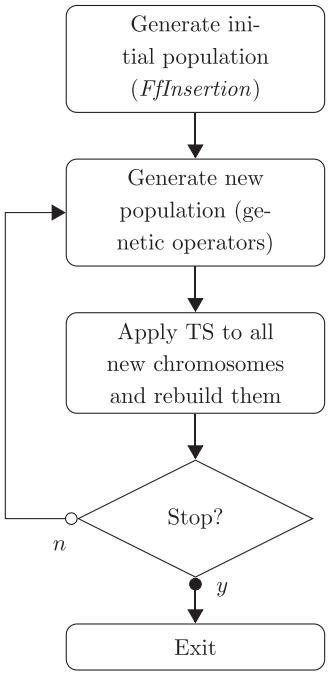


Fig. 2. Flow chart of the hybrid algorithm.

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 Ω 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 Ω 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 Ω and its successor in the job sequence is added to Ω, provided that it exists. The process finishes when Ω becomes empty.

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, $t_k^S + p_{o_{ij}k} \le t_k^E$, and $t_k^S \ge C_{o_{i(j-1)}}$ (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 $EST_{o_{ij}k}$, is the smallest t_k^S that can be found. Thus, the earliest completion time for o_{ij} is $C^\star = \min\{EST_{o_{ij}k} + p_{o_{ij}k}, M_k \in M(o_{ij})\}$. We schedule o_{ij} in any machine M_k such that $EST_{o_{ij}k} + p_{o_{ij}k} = C^\star$.

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

Algorithm 1. The *FfInsertion*.

Input A *FfJSP* instance
Output An operation processing order and a machine assignment which determines a schedule
$\Omega \leftarrow \{o_{i1}, 1 \le i \le n\}$;
while $\Omega \neq \varnothing$ **do**
  $o_{ij} \leftarrow$ an operation selected at random from Ω;
  for each $M_k \in M(o_{ij})$ **do**
    Compute $EST_{o_{ij}k}$;
  $C^\star \leftarrow \min\{EST_{o_{ij}k} + p_{o_{ij}k}, M_k \in M(o_{ij})\}$;
  $K \leftarrow \{M_k \in M(o_{ij}), EST_{o_{ij}k} + p_{o_{ij}k} = C^\star\}$;
  Choose a machine $M_{k^\star}$ from K at random;
  Schedule the operation o_{ij} in machine $M_{k^\star}$;
  {fix the value of $S_{o_{ij}} = EST_{o_{ij}k^\star}$};
  $\Omega \leftarrow \Omega - \{o_{ij}\}$
  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 $\{S_{o_{ij}} : 1 \le i \le n, 1 \le j \le N_i\}$ and the sequence and assignment vectors

4.3. Tabu search

Tabu search (TS) is an advanced local search technique, proposed in [56,57], 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, π, the neighbourhood of π is the set of operation processing orders obtained from π 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 α be a machine assignment, π a feasible operation processing order, x an operation and $k' \in M(x)$ a machine such that $k' \neq \alpha(x)$. Let $\alpha_{\langle x,k'\rangle}$ denote the assignment obtained from α after reassigning operation x to machine k'. The neighbourhood structure N^A obtained from α is defined as $N^A(\alpha,\pi) = \{(\alpha_{\langle x,k'\rangle}, \pi) : x \text{ is critical}, k' \in M(x), k' \neq \alpha(x)\}$.

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

structure obtained from π, N^P, is given by $N^P(\alpha,\pi) = \{(\alpha, \pi_{(v)}) : v \in R(\alpha,\pi)$ 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 α be a machine assignment, π a feasible processing order, $\beta = \alpha_{\langle x,k'\rangle}$ and $\sigma = \pi_{(v)}$ where x and v are not critical in $G(\alpha,\pi)$. Then*

$$\forall i, \quad C^i_{max}(\alpha,\pi) \le C^i_{max}(\alpha,\sigma).$$
$$\forall i, \quad C^i_{max}(\alpha,\pi) \le C^i_{max}(\beta,\pi). \tag{5}$$

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 α be a machine assignment and let π 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 π, 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 the absence of these characteristics, N^{AP} is the same as N. Hence, we start with the following Lemma.

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

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

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

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

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 α^*.

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

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

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 α. 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 depends 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 (α,π) is optimal.

Secondly, we shall prove that if (α,π) 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 $\alpha(x) \ne \alpha^*(x)$.

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 $G^i(\alpha^*,\pi^*) = (V, A \cup \overline{R^i(\alpha^*,\pi^*)})$. As above, unlike the non-flexible case, the length of P^i in $G^i(\alpha,\pi)$ may be different from its length in $G^i(\alpha^*,\pi^*)$ 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 $G^i(\alpha^*,\pi^*)$ 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

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

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

Then, if (α,π) 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 $\alpha(x) \ne \alpha^*(x)$. 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. $\alpha^*(x) \ne \alpha^*(y)$, but given that $\alpha(x) = \alpha(y)$, at least one of them, suppose it is x, verifies that $\alpha^*(x) \ne \alpha(x)$. □

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

Proof. Let (α^*,π^*) be any optimal solution and let $\{\lambda_k\}_{k\ge 0}$ be the sequence of solutions defined recursively as follows:

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

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

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

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 (α^*, π^*) and such that at least one of them is processed in other machine in the current assignment α; $\overline{R(\alpha^*, \pi^*)}$ 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 λ_k to λ_{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 λ_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 λ_{k+1}. If $\lambda_{k+1}$is obtained from λ_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 λ_{k+1} is obtained from λ_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 λ_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 us 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 α, 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 (α,π) 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. $N^P_r = \{v \in N^P : v \text{ is in an extreme of a critical block }\}$.

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 $r_x = \max\{r_{PJ_x} + p_{PJ_x k_1}, r_{PM_x} + p_{PM_x k}\}$, 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 $q_x = \max\{q_{SJ_x} + p_{SJ_x k_2}, q_{SM_x} + p_{SM_x k}\}$, 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 (α,π) is a solution and $v=(x,y)$ is a critical arc in $G(\alpha,\pi)$, reversing arc v produces a feasible solution (α,σ) 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 π, $r_a = r'_a$ and for every operation b posterior to y in π, $q_b = q'_b$. For x and y, the heads and tails after the move are calculated as follows:

$$\begin{aligned}
r'_y &= \max\{r_{PJ_y} + p_{PJ_y k_3}, r_{PM_x} + p_{PM_x k}\},\\
r'_x &= \max\{r_{PJ_x} + p_{PJ_x k_1}, r'_y + p_{yk}\},\\
q'_x &= \max\{q_{SJ_x} + p_{SJ_x k_2}, q_{SM_y} + p_{SM_y k}\},\\
q'_y &= \max\{q_{SJ_y} + p_{SJ_y k_4}, q'_x + p_{xk}\},
\end{aligned} \tag{9}$$

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 (α,σ).

Regarding moves in N^A, let x be a critical operation in (α,π) and $k=\alpha(x)$, assigning x to another machine $M_{k'} \in M(x)$ maintaining the processing order π produces a new feasible solution (β,π), where $\beta = \alpha_{\langle x,k'\rangle}$, 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

$$r'_x = \max\{r_{PJ_x} + p_{PJ_x k_1}, r_{PM_x} + p_{PM_x k'}\},$$
$$q'_x = \max\{q_{SJ_x} + p_{SJ_x k_2}, q_{SM_x} + p_{SM_x k'}\}. \quad (10)$$

Therefore, the makespan of (β, π) 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 (β, π).

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 [35–38]. 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 24 GB 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

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

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

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

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:

$$LB_F = E\left[\max_i \left\{\sum_{j=1}^{N_i} pm_{o_{ij}}\right\}\right] \quad (13)$$

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

Regarding the benchmarks for the *FfJSP*, we consider here those proposed in [34,37,38] 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 [63,64,29]. 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 six 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 Eq. (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 Brandimarte [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

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*)

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 sensitivities 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 or 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 Coevolutionary Genetic Algorithm (CGA) proposed by Lei [37], the Swarm-based Neighbourhood Search Algorithm (SNSA) proposed by Lei and Guo [38], the Hybrid Artificial Bee Colony Algorithm (hABC) proposed by Wang et al. [35] and the Estimation Distribution Algorithm (EDA) proposed by Wang et al. [36]. CGA is implemented in Microsoft Visual C++ 6.0 and run on a 512MB RAM 1.7 GHz PC, it uses a population of 150 chromosomes and a number of 1000 generations and the time taken ranges from 8 to 11 s for a single run. SNSA is coded in Microsoft Visual C++ 6.0 and run on a 2GB RAM 2.2 GHz PC, the swarm size is 100, the number of iterations is limited to 500 and the time taken varies from 9 to 14 s a single run. hABC is implemented in C++ and run on a 3.2 GB RAM 2.83 GHz 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 s per single run. Finally, EDA is coded in C++ and run on Thinkpad T420 2 GB RAM 2.3 GHz; 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 s 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 and 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, 8 and 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.

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	

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 [64,65], the set of instances from [63], the *BRdata* from Brandimarte [29], the *BCdata* from [66], the *DPdata* from [30] and the *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 the absence of other information, we have calculated lower bounds for these instances using Eq. (13) (without expected values as it corresponds to crisp instances). The next five 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 Thomalla's benchmark [63] are compared with the results reported for the PSO by Girish and Jawahar [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 Eq. (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.

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 González et al. [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

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	

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	

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

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 three 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 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 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. Fig. 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. [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 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, Fig. 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 [11,32] 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* ∪ *BCdata* ∪ *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 $p^3\in[2p-p^1,1.2p]$. 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 Eq. (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, regarding the 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 07*a* to 18*a* 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

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

[3] 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.

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	

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**[a] (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	

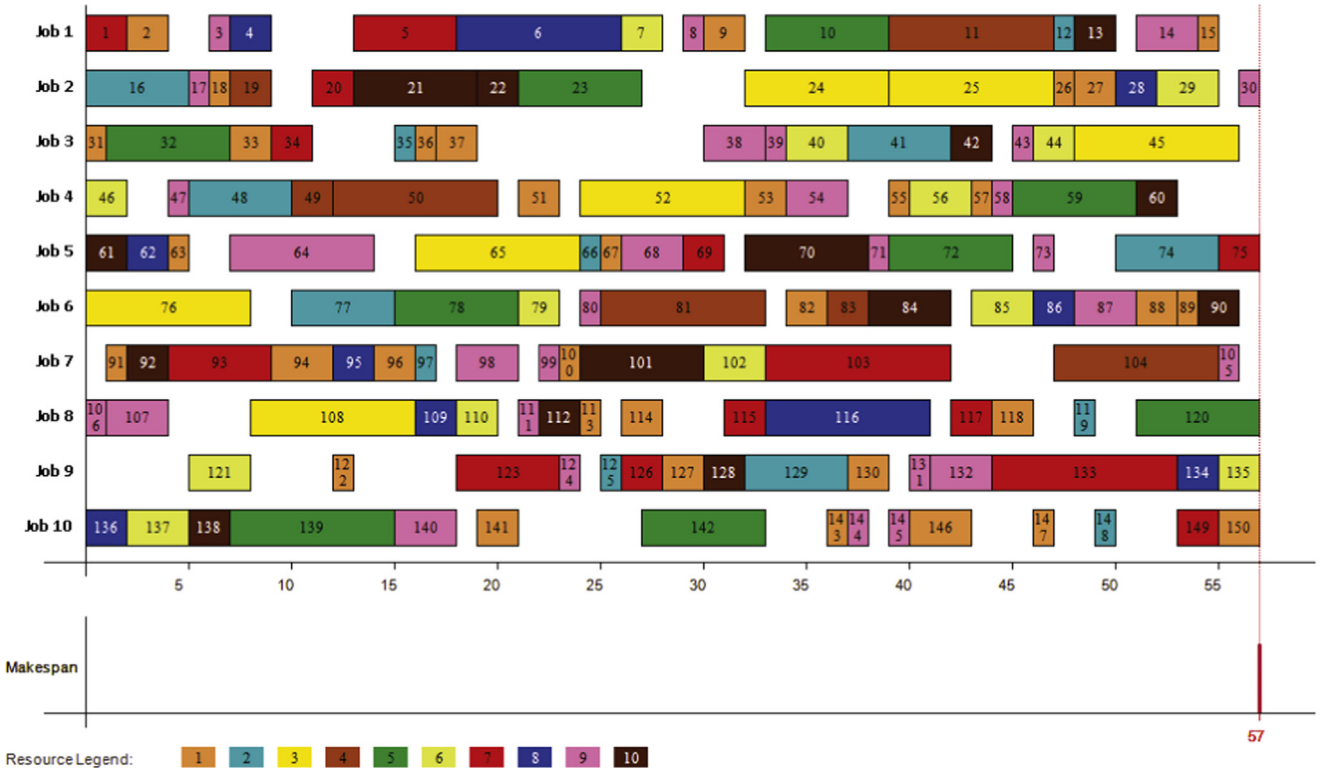


Fig. 3. Gantt chart, job-oriented, of new best solution of instance *MK06* from *BRdata*.

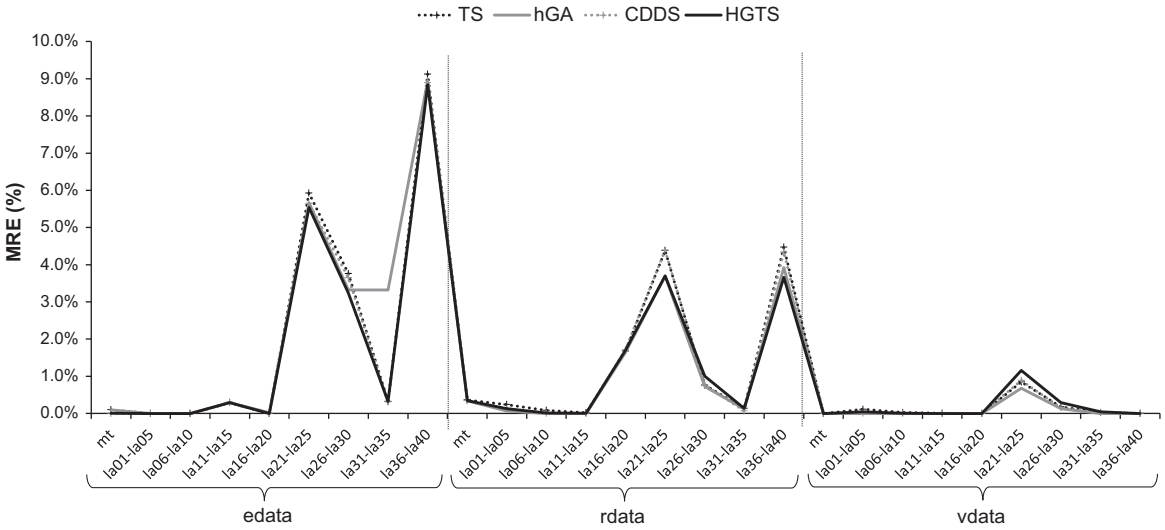


Fig. 4. Main relative errors on the *HUdata*.

structure N_r^{AP}, which contains less neighbours and therefore has been used across all 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. 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 us to better evaluate the effect of the heuristic seeding in the HGTS. Fig. 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

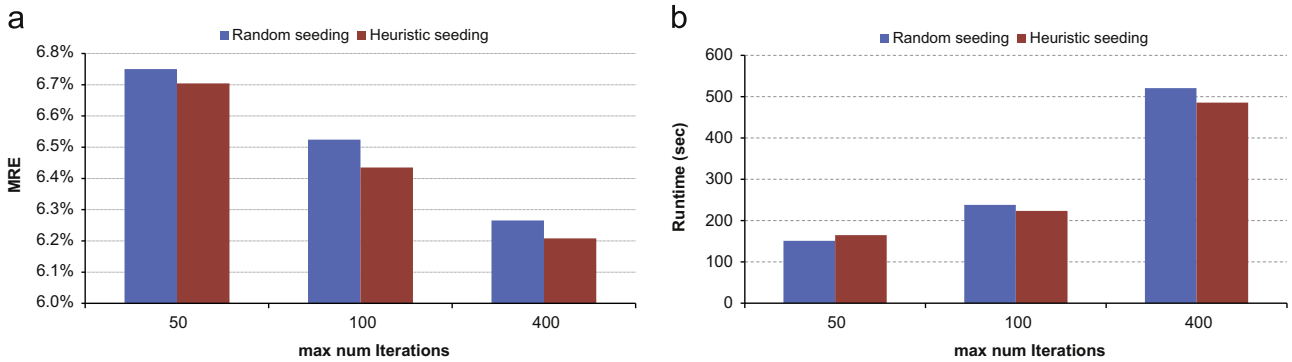


Fig. 5. Comparison between random and heuristic seeding in the new *FfJSP* benchmark. (a) MRE average values. (b) Average computational time.

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*		

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 us to speed-up the evaluation of new chromosomes. We have 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.

Acknowledgments

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] Pinedo ML. Scheduling: theory, algorithms, and systems. 3rd ed. New York: Springer; 2008.
[2] Garey M, Johnson D, Sethi R. The complexity of flowshop and jobshop scheduling. Math Oper Res 1976;1(2):117–29.
[3] Herroelen W, Leus R. Project scheduling under uncertainty: survey and research potentials. Eur J Oper Res 2005;165:289–306.
[4] Klerides E, Hadjiconstantinou E. A decomposition-based stochastic programming approach for the project scheduling problem under time/cost trade-off settings and uncertain durations. Comput Oper Res 2010;37:2131–40.
[5] Bruni M, Beraldi F, Guerriero F, Pinto E. A heuristic approach for resource constrained project scheduling with uncertain activity durations. Math Methods Oper Res 2011;38:1305–18.
[6] Dubois D, Prade H, Smets P. Representing partial ignorance. IEEE Trans Syst Man Cybern: Part A 1996;26(3):361–77.

[7] Ludwig A, Möhring RH, Stork F. A computational study on bounding the makespan distribution in stochastic project networks. Ann Oper Res 2001;102:49–64.
[8] Dubois D, Fargier H, Fortemps P. Fuzzy scheduling: modelling flexible constraints vs. coping with incomplete knowledge. Eur J Oper Res 2003;147:231–52.
[9] Słowiński R, Hapke M, editors. Scheduling under fuzziness, Studies in fuzziness and soft computing, vol. 37. Poznań, Poland: Physica-Verlag; 2000.
[10] Zhang Q, Manier H, Manier M-A. A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. Comput Oper Res 2012;39:1713–23.
[11] Gao J, Sun L, Gen M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 2008;35:2892–907.
[12] Vela CR, Varela R, González MA. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. J Heuristics 2010;16:139–65.
[13] Puente J, Vela CR, González-Rodríguez I. Fast local search for fuzzy job shop scheduling. In: Proceedings of the ECAI 2010. Lisbon, Portugal: IOS Press; 2010. p. 739–44.
[14] González M, Vela CR, Varela R. 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. p. 91–9.
[15] Pezzella F, Morganti G, Ciaschetti G. A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 2008;35:3202–12.
[16] Zhang CY, Li P, Rao Y, Guan Z. A very fast TS/SA algorithm for the job shop scheduling problem. Comput Oper Res 2008;35:282–94.
[17] Beck JC, Feng T, Watson J-P. Combining constraint programming and local search for job-shop scheduling. Inf J Comput 2011;23:1–14.
[18] Nowicki E, Smutnicki C. An advanced tabu search algorithm for the job shop problem. J Sched 2005;8:145–59.
[19] Meeran S, Morshed M. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. J Intell Manuf 2012;23:1063–78.
[20] Fortemps P. Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Trans Fuzzy Syst 1997;7:557–69.
[21] Sakawa M, Kubota R. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. Eur J Oper Res 2000;120:393–407.
[22] Lei D. Solving fuzzy job shop scheduling problems using random key genetic algorithm. Int J Adv Manuf Technol 2010;49:253–62.
[23] Niu Q, Jiao B, Gu X. Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. Appl Math Comput 2008;205:148–58.
[24] González Rodríguez I, Puente J, Vela CR, Varela R. Semantics of schedules for the fuzzy job shop problem. IEEE Trans Syst Man Cybern: Part A 2008;38(3):655–66.
[25] Zheng Y, Li Y, Lei D. Swarm-based neighbourhood search for fuzzy job shop scheduling. Int J Innov Comput Appl 2011;3(3):144–51.
[26] Li J-q, Pan Y-x. A hybrid discrete particle swarm optimization algorithm for solving fuzzy job shop scheduling problem. Int J Adv Manuf Technol App 2012;66(1-4):583–96.
[27] Zheng Y-L, Li Y-X. Artificial bee colony algorithm for fuzzy job shop scheduling. Int J Comput Appl Technol 2012;44(2):124–9.
[28] Brucker P, Schlie R. Job-shop scheduling with multi-purpose machines. Computing 1990;45(4):369–75.
[29] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 1993;41:157–83.
[30] Dauzère-Pérès S, Paulli J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 1997;70(3):281–306.
[31] Mastrolilli M, Gambardella L. Effective neighborhood functions for the flexible job shop problem. J Sched 2000;3(1):3–20.
[32] Hmida A, Haouari M, Huguet M, Lopez P. Discrepancy search for the flexible job shop scheduling problem. Comput Oper Res 2010;37:2192–201.
[33] Yuan Y, Xu H. An integrated search heuristic for large-scale flexible jobshop scheduling problems. Comput Oper Res 2013;40:2864–77.
[34] Lei D. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. Int J Prod Res 2010;48(10):2995–3013.
[35] Wang L, Zhou G, Xu Y, Min L. A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. Int J Prod Res 2013;51(12):3593–608.
[36] Wang S, Wang L, Xu Y, Min L. An effective estimation of distribution algorithm for the flexible job-shop scheduling problem with fuzzy processing time. Int J Prod Res 2013;51(12):3779–93.
[37] Lei D. Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. Appl Soft Comput 2012;12:2237–45.
[38] Lei D, Guo X. Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling. Int J Prod Res 2012;50(6):1639–49.
[39] Dubois D, Prade H. Possibility theory: an approach to computerized processing of uncertainty. New York, USA: Plenum Press; 1986.
[40] Heilpern S. The expected value of a fuzzy number. Fuzzy Sets Syst 1992;47:81–6.
[41] Sakawa M, Mori T. An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate. Comput Ind Eng 1999;36:325–41.
[42] Bortolan G, Degani R. A review of some methods for ranking fuzzy subsets. In: Dubois D, Prade H, Yager R, editors. Readings in fuzzy sets for intelligence systems. Amsterdam, NL: Morgan Kaufmann; 1993. p. 149–58.
[43] Celano G, Costa A, Fichera S. An evolutionary algorithm for pure fuzzy flowshop scheduling problems. Int J Uncertain Fuzziness Knowled Based Syst 2003;11:655–69.
[44] Chen S-M, Chang T-H. Finding multiple possible critical paths using fuzzy PERT. IEEE Trans Syst Man Cybern: Part B 2001;31(6):930–7.
[45] González Rodríguez I, Vela CR, Hernández-Arauzo A, Puente J. 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. p. 154–61.
[46] Kuroda M, Wang Z. Fuzzy job shop scheduling. Int J Prod Econ 1996;44:45–51.
[47] Lei D. Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. Int J Adv Manuf Technol 2008;37:157–65.
[48] Petrovic S, Song X. A new approach to two-machine flow shop problem with uncertain processing times. Optim Eng 2006;7:329–42.
[49] Wang B, Li Q, Yang X, Wang X. Robust and satisfactory job shop scheduling under fuzzy processing times and flexible due dates. In: Proceedings of the 2010 IEEE international conference on automation and logistics; 2010. p. 575–80.
[50] Lei D. Fuzzy job shop scheduling problem with availability constraints. Comput Ind Eng 2010;58:610–7.
[51] González Rodríguez I, Vela CR, Puente J, Varela R. 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. p. 124–31.
[52] Varela R, Vela CR, Puente J, Gómez A. A knowledge-based evolutionary strategy for scheduling problems with bottlenecks. Eur J Oper Res 2003;145:57–71.
[53] Bierwirth C. A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spect 1995;17:87–92.
[54] Essafi I, Mati Y, Dauzère-Pérès S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Comput Oper Res 2008;35:2599–616.
[55] González M, Vela CR, Varela R. A competent memetic algorithm for complex scheduling. Nat Comput 2012;11:151–60.
[56] Glover F. Tabu search. Part I. ORSA J Comput 1989;1(3):190–206.
[57] Glover F. Tabu search. Part II. ORSA J Comput 1990;2(1):4–32.
[58] Dell'Amico M, Trubian M. Applying tabu search to the job-shop scheduling problem. Ann Oper Res 1993;41:231–52.
[59] Van Laarhoven P, Aarts E, Lenstra K. Job shop scheduling by simulated annealing. Oper Res 1992;40:113–25.
[60] Bozejko W, Uchronski M, Wodecki M. Parallel hybrid metaheuristics for the flexible job shop problem. Comput Ind Eng 2010;59(2):323–33.
[61] Jin Y. Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol Comput 2011;1(2):61–70.
[62] Taillard E. Benchmarks for basic scheduling problems. Eur J Oper Res 1993;64:278–85.
[63] Thomalla CS. Job shop scheduling with alternative process plans. Int J Prod Econ 2001;74:125–34.
[64] Kacem I, Hammadi S, Borne P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybern. Part C: Appl Rev 2002;32(1):1–13.
[65] Kacem I, Hammadi S, Borne P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 2002;60:245–76.
[66] Barnes J, Chambers J. 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] Hurink E, Jurisch B, Thole M. Tabu search for the job shop scheduling problem with multi-purpose machine. Oper Res Spekt 1994;15:205–15.
[68] Xing L, Chen Y, Wang P, Zhao Q, Xion J. A knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 2010;10(3):888–96.
[69] Li J, Pan Q, Suganthan P, Chua T. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. Int J Adv Manuf Technol 2011;52(5-8):683–97.
[70] Wang L, Zhou G, Xu Y, Min L. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. Int J Adv Manuf Technol 2012;60:303–15.
[71] Wang L, Wang S, Zhou G, Min L. A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. Comput Ind Eng 2012;62:917–26.
[72] Girish BS, Jawahar N. A particle swarm optimization algorithm for flexible job shop scheduling problem. In: Proceedings of the fifth annual IEEE conference on automation science and engineering; 2009. p. 298–303.
[73] García S, Fernández A, Luengo J, Herrera F. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf Sci 2010;180:2044–64.

7.3 A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem

In this section, we include the following publication.

- **Title:** A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem.
- **Journal:** AI Communications.
- **Year:** 2015.
- Impact Factor (JCR 2013): 0.466
- Impact Factor (5-year): 0.582
- Journal Ranking:
  - Computer Science, Artificial Intelligence: 105/121 **Q4 (T3)**

This publications contains pieces of work described in Sections 4.2.1 and 4.3.1.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem. AI Communications 28(2), 239-257 (2015). Doi: 10.3233/AIC-140637.

AI Communications 28 (2015) 239–257
DOI 10.3233/AIC-140637
IOS Press

A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem

Juan José Palacios [a], Inés González-Rodríguez [b,*], Camino R. Vela [a] and Jorge Puente [a]
[a] *Department of Computer Science, University of Oviedo, Gijón, Spain*
E-mails: palaciosjuan@uniovi.es, crvela@uniovi.es, puente@uniovi.es
[b] *Department of Mathematics, Statistics and Computing, University of Cantabria, Santander, Spain*
E-mail: ines.gonzalez@unican.es

Abstract. In the sequel, we consider a multiobjective open shop scheduling problem with uncertain durations modelled as fuzzy numbers. Given crisp due dates, the objective is to minimise both the makespan and the maximum tardiness. We formulate the multiobjective problem as a fuzzy goal programming model based on lexicographical minimisation of expected values. The resulting problem is solved using a particle swarm optimisation approach searching in the space of possibly active schedules. To asses the performance of this algorithm, we present results of an extensive experimental study on several problem instances, including: a parametric analysis, the experimental evaluation of different priority structures compared to single-objective approaches in terms of objective values as well target achievement, an experimental analysis of the relationship between lexicographical and Pareto solutions and an empirical study based on a-posteriori semantics showing the advantages of taking into account the uncertainty along the scheduling process.

Keywords: Open shop scheduling, fuzzy processing times, particle swarm optimisation, lexicographic goal programming

1. Introduction

The open shop scheduling problem (OSP) is a problem with an increasing presence in the literature and clear applications in industry, for instance, in testing facilities, where units go through a series of diagnostic tests that need not be performed in a specified order and where different testing equipment is usually required for each test [33]. Traditionally, the open shop as well as other scheduling problems have been treated as deterministic, assuming precise knowledge of all data involved, in contrast with the uncertainty and vagueness pervading real-world problems. To reduce the gap between theory and applications, an increasing part of the research is devoted to modelling this lack of certainty with great diversity of approaches [21]. In particular, fuzzy sets have been used in different manners to represent incomplete or vague states of information [10]: using fuzzy priority rules with linguistic qualifiers, modelling soft as well as fuzzy temporal constraints, and as a means of improving solution robustness, a much-desired property in real-life applications [3,24,32,43].

The open shop is NP-complete for a number of machines $m \geqslant 3$, so approaches to solving it usually make use of metaheuristic techniques. In particular, in [1] a heuristic approach is proposed to minimise the expected makespan for an open shop problem with stochastic processing times and random breakdowns; in [18] a genetic algorithm is combined with a local search method to minimise the expected makespan of an open shop with fuzzy durations; a particle swarm optimisation algorithm is used to minimise the expected fuzzy makespan in [31] and a possibilistic mixed-integer linear programming method is proposed in [30] for an OSP with setup times, fuzzy processing times and fuzzy due dates to minimize total weighted tardiness and total weighted completion times. Far from being trivial, extending heuristic strategies to uncertain settings usually requires a significant reformulation of both the problem and solving methods.

*Corresponding author: Inés González-Rodríguez, Departamento de Matemáticas, Estadística y Computación, Facultad de Ciencias, Universidad de Cantabria, Av. Los Castros s/n, 39005, Santander, Spain. E-mail: ines.gonzalez@unican.es.

Additionally, many real-life applications require taking into account several conflicting points of view corresponding to multiple objectives. Pareto optimality is undoubtedly the most extended approach to multicriteria optimisation but, quoting [12], "it is not the end of the story". Among the alternative approaches, lexicographic and goal programming methods are some of the most popular ones [6]. The philosophy behind goal programming can be traced back to the theories of rational decision developed in the 1950s, especially the concept of satisficing solutions: in a complex environment, the decision maker's aim may be to reach a certain satisfactory level for every relevant objective, rather than optimising its value [34]. Also, lexicographic problems arise naturally when conflicting objectives exist that have to be considered in a hierarchical manner. Recent examples of real-world problems where these techniques are applied can be found in [8,9] and [27]. Additionally, there exist interesting relationships between lexicographic and Pareto-optimal solutions. Indeed, "lexicographic minimisation is well-suited to seek a compromise between conflicting interests, as well as reconciling this requirement with the crucial notion of Pareto-optimality" [4].

In the sequel, we describe an open shop problem with fuzzy durations and crisp due dates and where the objective is to minimise both the project's makespan and the maximum tardiness w.r.t. the given job due dates. We adopt a generic multiobjective model so the objective function is defined in order to lexicographically minimise the expected values of several fuzzy goals (here, makespan and tardiness). In addition to the priority structure for the lexicographical minimisation, target levels for each objective are introduced, in order to balance possibly incompatible goals. The resulting problem is solved by means of a multiobjective particle swarm optimisation (MOPSO) algorithm searching in the space of possibly active schedules. We evaluate the performance of the MOPSO algorithm on a set of problem instances based on the expected values of each objective.

The rest of the paper is organised as follows. In Section 2 we describe how to work with fuzzy numbers. Section 3 is devoted to the fuzzy open shop problem, including the definition of the main concepts related to this problem as well as the multiobjective model proposed herein and a semantics for fuzzy schedules. In Section 4 a particle swarm optimisation algorithm is proposed to solve the resulting problem. Section 5 includes a parametric analysis of this algorithm, an experimental evaluation of the obtained results as well an additional analysis based on the semantics of the fuzzy solutions. Finally, in Section 6 we offer some conclusions and proposals for future work.

2. Uncertain processing times

In real-life applications, it is often the case that the exact duration of a task is not known in advance. However, based on previous experience, an expert may be able to estimate an interval for the possible processing time or its most typical values. In the literature, it is common to use fuzzy intervals to represent such ill-known processing times, as an alternative to probability distributions, which require a deeper knowledge of the problem and usually yield a complex calculus.

As a natural extension of human originated confidence intervals we find fuzzy intervals, where some values appear to be more plausible than others. Fuzzy intervals and fuzzy numbers have been extensively studied in the literature (cf. [11]). A *fuzzy interval* N is a fuzzy set on the reals (with membership function $\mu_N : \mathbb{R} \to [0,1]$) such that its α-*cuts* $N_\alpha = \{u \in \mathbb{R}\colon\ \mu_N(u) \geqslant \alpha\}$, $\alpha \in (0,1]$, are intervals. A fuzzy interval is a *fuzzy number* if its α-cuts (denoted $[\underline{n}_\alpha, \overline{n}_\alpha]$) are closed, its *support* $N_0 = \{u \in \mathbb{R}\colon\ \mu_N(u) > 0\}$ is compact (closed and bounded) and there is a unique modal value $u*$, $\mu_N(u*) = 1$. 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 single plausible 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:

$$\mu_A(x) = \begin{cases} \dfrac{x - a^1}{a^2 - a^1}: & a^1 \leqslant x \leqslant a^2, \\ \dfrac{x - a^3}{a^2 - a^3}: & a^2 < x \leqslant a^3, \\ 0: & x < a^1 \text{ or } a^3 < x. \end{cases} \tag{1}$$

In order to work with fuzzy numbers, it is necessary to extend the usual arithmetic operations on real numbers. In particular, for the open shop we need to compute the addition, substraction and maximum of fuzzy numbers. In general, this is done using the Extension Principle, but computing the resulting equation is cumbersome, if not intractable. It can be somewhat simplified if the function $f : \mathbb{R}^2 \to \mathbb{R}$ to be extended is continuous and isotonic; in this case, the First Decomposition theorem provides us with a simpler formula based on evaluating f on each α-cut.

For the addition and substraction of TFNs, this boils down to operating on the three defining points, that is, for any pair of TFNs M and N:

$$\begin{aligned} M + N &= \big(m^1 + n^1, m^2 + n^2, m^3 + n^3\big), \\ M - N &= \big(m^1 - n^3, m^2 - n^2, m^3 - n^1\big). \end{aligned} \tag{2}$$

Unfortunately, for the maximum of TFNs there is no such simplified expression and its computation in general requires evaluating maxima of all α-cuts, $\alpha \in (0, 1]$. Also, the set of TFNs is not closed under the maximum operation. For the sake of simplicity and tractability of numerical calculations, we follow Fortemps [13] and approximate all results of isotonic algebraic operations on TFNs by a TFN, so instead of evaluating the intervals corresponding to all α-cuts, we evaluate only those intervals corresponding to the support and $\alpha = 1$, which is equivalent to working only with the three defining points of each TFN. This is an approach also taken, for instance, in [7,26] and [29]. For any two TFNs M and N, their maximum will then be approximated as follows:

$$\begin{aligned} &\max(M, N) \\ &\quad \sim \big(\max\big(m^1, n^1\big), \max\big(m^2, n^2\big), \\ &\qquad \max\big(m^3, n^3\big)\big). \end{aligned} \tag{3}$$

Despite not being equal, if F denotes the actual maximum and G its approximated value, it holds that $\forall \alpha \in [0, 1]$, $\underline{f}_\alpha \leqslant \underline{g}_\alpha$, $\overline{f}_\alpha \leqslant \overline{g}_\alpha$. In particular, F and G have identical support and modal value: $F_0 = G_0$ and $F_1 = G_1$. The approximated maximum can be trivially extended to $n > 2$ TFNs.

The membership function μ_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 [28], given for a TFN A by $E[A] = \frac{1}{4}(a^1 + 2a^2 + a^3)$. 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 [10]. It induces a total ordering $\leqslant_E$ in the set of fuzzy intervals [13], where for any two fuzzy intervals M, N, $M \leqslant_E N$ if and only if $E[M] \leqslant E[N]$.

3. The fuzzy open shop scheduling problem

The *open shop scheduling problem*, or *OSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job J_i consists of m tasks or operations o_{ij} ($j = 1, \ldots, m$), where o_{ij} requires the exclusive use of a machine M_j for its whole processing time p_{ij} without preemption, i.e. all tasks must be processed without interruption. In total, there are mn tasks, $\{o_{ij}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$. Additionally, for each job J_i there may be a due date d_i, $i = 1, \ldots, n$ before which it is desirable that the job be terminated. A solution to this problem is a *schedule* – an allocation of starting times for all tasks – which is *feasible*, in the sense that all constraints hold, and is also optimal according to some criteria. Here, we shall consider two objectives: minimising the makespan $C_{\max}$, that is, the time lag from the start of the first task until the end of the last one, as well as minimising the maximum tardiness $T_{\max}$, that is, the maximum delay of any job with respect to its due date. This problem may be denoted $O|d_i|\mathit{multicrit}(C_{\max}, T_{\max})$ using the three-field notation from [19], extended to multiobjective problems as proposed in [20].

3.1. Fuzzy schedules from crisp task orderings

A schedule for a open shop problem of size $n \times m$ (n jobs and m machines) may be determined by a decision variable $\mathbf{z} = (z_1, \ldots, z_{nm})$ representing a task processing order, where $1 \leqslant z_l \leqslant nm$ for $l = 1, \ldots, nm$. This is a permutation of the set of tasks where each task o_{ij} is represented by the number $(i - 1)m + j$. The task processing order represented by the decision variable uniquely determines a feasible schedule; it should be understood as expressing partial orderings for every set of tasks requiring the same machine and for every set of tasks belonging to the same job.

Let us assume that the processing time p_{ij} of each task o_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$ is a fuzzy variable (a particular case of which are TFNs), so the problem may be represented by a matrix of fuzzy processing times $\mathbf{p}$ of size $n \times m$. For a given task processing order $\mathbf{z}$ and a task o_{ij}, its starting time $S_{ij}(\mathbf{z}, \mathbf{p})$ is the maximum between the completion times of the task preceding o_{ij} in its job, let it be denoted o_{ik}, and the task preceding o_{ij} in its machine, let it be denoted o_{lj}:

$$S_{ij}(\mathbf{z}, \mathbf{p}) = \max\big(C_{ik}(\mathbf{z}, \mathbf{p}), C_{lj}(\mathbf{z}, \mathbf{p})\big), \tag{4}$$

where $C_{ik}(\mathbf{z}, \mathbf{p})$ or $C_{lj}(\mathbf{z}, \mathbf{p})$ are taken to be zero if o_{ij} is the first task to be processed either in its job or

its machine. Then, its completion time $C_{ij}(\mathbf{z}, \mathbf{p})$ is obtained by adding its duration p_{ij} to $S_{ij}(\mathbf{z}, \mathbf{p})$:

$$C_{ij}(\mathbf{z}, \mathbf{p}) = S_{ij}(\mathbf{z}, \mathbf{p}) + p_{ij}. \tag{5}$$

The completion time of a job J_i will then be the maximum completion time of all its tasks, that is, $C_i(\mathbf{z}, \mathbf{p}) = \max_{1 \leqslant j \leqslant m}\{C_{ij}(\mathbf{z}, \mathbf{p})\}$. Then, the tardiness with respect to the due date d_i for job J_i will be $T_i(\mathbf{z}, \mathbf{p}) = \max(C_i(\mathbf{z}, \mathbf{p}) - d_i, 0)$ (notice that both d_i and 0 can be seen as particular cases of fuzzy numbers).

For this schedule, the *fuzzy makespan* $C_{\max}(\mathbf{z}, \mathbf{p})$ and the *fuzzy maximum tardiness* (fuzzy tardiness for short) $T_{\max}(\mathbf{z}, \mathbf{p})$ are defined as follows:

$$C_{\max}(\mathbf{z}, \mathbf{p}) = \max_{1 \leqslant i \leqslant n} \big(C_i(\mathbf{z}, \mathbf{p})\big), \tag{6}$$

$$T_{\max}(\mathbf{z}, \mathbf{p}) = \max_{1 \leqslant i \leqslant n} \big(T_i(\mathbf{z}, \mathbf{p})\big). \tag{7}$$

In the case where no confusion is possible, we may drop the decision variable $\mathbf{z}$ and the processing times matrix $\mathbf{p}$ and simply write $C_{\max}$ and $T_{\max}$.

Given a fuzzy schedule, our objective is to *optimise* its makespan and maximum tardiness. However, it is not trivial when dealing with fuzzy values to decide on the precise meaning of "optimality", since neither the maximum nor its approximation define a total ordering in the set of TFNs. Using ideas similar to stochastic scheduling, we use the total ordering provided by the expected value and consider that the objective is to minimise the expected makespan $E[C_{\max}]$ and the expected tardiness $E[T_{\max}]$, so the resulting problem may be denoted $O|\mathit{fuzzp}_i, d_i|\mathit{multicrit}(E[C_{\max}], E[T_{\max}])$.

Let us illustrate the previous definitions with an example. Consider a problem of 3 jobs and 2 machines with the following matrices for fuzzy processing times and due dates:

$$\mathbf{p} = \begin{pmatrix} (3,4,7) & (3,4,7) \\ (2,3,3) & (4,5,6) \\ (3,4,6) & (1,2,4) \end{pmatrix}, \qquad \mathbf{d} = \begin{pmatrix} 10 \\ 6 \\ 16 \end{pmatrix}.$$

Here $p_{21} = (2,3,3)$ is the processing time of task o_{21}, the task of job J_2 to be processed in machine M_1 and $d_2 = 6$ is the due date for job J_2. Figure 1 shows the Gantt chart adapted to TFNs of the schedule given by the decision variable $\mathbf{z} = (1,4,6,3,5,2)$; it is inspired in the charts appearing in [13] and it represents the partial schedules on each job obtained from this decision variable. Tasks must be processed in the following order: o_{11}, o_{22}, o_{32}, o_{21}, o_{31}, o_{12}. Given this ordering, the starting time for task o_{21} will be the maximum of the completion times of o_{22} and o_{11}, which are respectively the preceding tasks in the job and in the machine: $S_{21} = \max(C_{22}, C_{11}) = \max((4,5,6),(3,4,7)) = (4,5,7)$. Consequently, its completion time will be $C_{21} = S_{21} + p_{21} = (4,5,7) + (2,3,3) = (6,8,10)$. The fuzzy time gap when task o_{21} is processed corresponds to the shaded polygon labelled o_{21} in the left-hand side of Fig. 1. This polygon is determined on the left by the starting time $(4,5,7)$ and on the right by the completion time $(6,8,10)$; notice that in the case that starting and completion times were real numbers, the polygon would turn into a rectangle, which is the standard way of representing task execution times in deterministic Gantt charts. The makespan can be easily calculated as $C_{\max} = (9,12,17)$, so $E[C_{\max}] = 12.5$. This fuzzy makespan is also depicted above the job partial schedules in Fig. 1; it can be seen that the first and second components of $C_{\max}$ are determined by the last task of J_3 whilst the third component of $C_{\max}$ is determined by

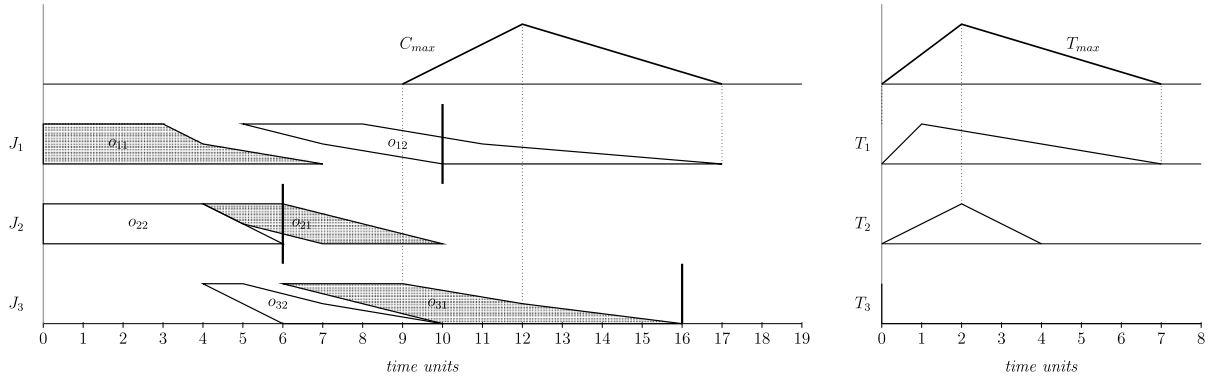


Fig. 1. Gantt chart of the schedule represented by the decision variable $(1,4,6,3,5,2)$ for the example in Section 3.1.

the last task of job J_1. Regarding due dates, depicted as vertical lines on each partial schedule, job J_3 always terminates before d_3, whereas the execution times of the last tasks for J_1 and J_2 overlap with their respective due dates d_1 and d_2. Indeed, $T_3 = 0$, whereas for job J_2, $T_2 = (0, 2, 4)$, and analogously $T_1 = (0, 1, 7)$. Hence, the maximum tardiness will be $T_{\max} = (0, 2, 7)$ and $E[T_{\max}] = 2.75$. These job and maximum tardiness values (all of them TFNs) are depicted on the right-hand side of Fig. 1, illustrating which job determines each component of the maximum tardiness.

3.2. Multiobjective models

With multiple goals it is often the case that some are achievable only at the expense of others. A well-established approach to dealing with multiple and possibly conflicting objectives is lexicographic goal programming [39]. It assumes there exists a hierarchy of importance for these goals so as to satisfy as many as possible in the specified order.

In general, for k objectives $f_1, \ldots, f_k$ such priority structure should be established by the decision maker (DM) and may be represented by a one-to-one mapping ρ from $\{f_1, \ldots, f_k\}$ onto $\{1, \ldots, k\}$, such that $\rho(f_i)$ is the priority level of f_i, $i = 1, \ldots, k$, where 1 represents the highest priority. For instance, if $f_1 = C_{\max}$ and $f_2 = T_{\max}$ and the DM considers that the most prioritary objective is minimising the expected tardiness, then $\rho(f_2) = 1$. Without loss of generality, in the remaining of this section we can assume that the objective functions $f_i, i = 1, \ldots, k$ are ordered according to their priority, that is, $\rho(f_i) = i$. Then, based on similar ideas presented for the job shop in [17] we may formulate the following *expected multiobjective model* for the fuzzy open shop problem (FOSP):

$$\begin{cases} \text{lexmin}\big(E\big[f_1(\mathbf{z}, \mathbf{p})\big], \ldots, E\big[f_k(\mathbf{z}, \mathbf{p})\big]\big) \\ \text{subject to:} \\ \quad 1 \leqslant z_l \leqslant nm, \quad l = 1, \ldots, nm, \\ \quad z_l \neq z_k, \quad k \neq l, l, k = 1, \ldots, nm, \\ \quad z_l \in \mathbb{Z}^+, \quad l = 1, \ldots, nm, \end{cases} \tag{8}$$

where lexmin denotes lexicographically minimising the objective vector.

Pure lexicographical models may get stuck in the first goals and never consider the remaining ones. To balance the multiple conflicting objectives, we may use a goal programming model and consider target levels established by the DM for the different objectives, so $E[f_i(\mathbf{z}, \mathbf{p})]$ should not exceed a given target value $b_i \geqslant 0, i = 1, \ldots, k$. This translates into the following goal constraints:

$$E\big[f_i(\mathbf{z}, \mathbf{p})\big] + \Delta_i^- - \Delta_i^+ = b_i, \quad i = 1, \ldots, k, \tag{9}$$

where Δ_i^+, the positive deviation from the target, should be minimised. We thus obtain the following *expected goal multiobjective model* for the FJSP:

$$\begin{cases} \text{lexmin}\big(\Delta_1^+, \ldots, \Delta_k^+\big) \\ \text{subject to:} \\ \quad E\big[f_i(\mathbf{z}, \mathbf{p})\big] + \Delta_i^- - \Delta_i^+ = b_i, \\ \quad\quad i = 1, \ldots, k, \\ \quad \Delta_i^-, \Delta_i^+ \geqslant 0, \quad i = 1, \ldots, k, \\ \quad 1 \leqslant z_l \leqslant nm, \quad l = 1, \ldots, nm, \\ \quad z_l \neq z_k, \quad k \neq l, l, k = 1, \ldots, nm, \\ \quad z_l \in \mathbb{Z}^+, \quad l = 1, \ldots, nm. \end{cases} \tag{10}$$

Notice that (8) is a particular case of (10). Indeed, the latter is general enough to comprise all possible fuzzy goals, priority structures and target levels established by the DM.

The resulting problem may be denoted $O|\mathit{fuzzp}_i, d_i|\mathit{LexGP}(E[f_1], \ldots, E[f_k])$ according to the three-field notation from [19] extended to multicriteria scheduling in the spirit of [20] and [41].

3.3. Semantics for fuzzy schedules

In [16], a semantics for fuzzy schedules was proposed and then used to measure the performance of such schedules in real environments. According to this semantics, solutions to the fuzzy job shop (and by extension to the FOSP) are interpreted as *a-priori solutions*, found when the duration of tasks is not exactly known. In this setting, it is impossible to predict what the exact time-schedule (exact task starting times) will be, because it depends on the realisation of the task's durations, which is not known yet. Each schedule corresponds to a crisp ordering of tasks; it is not until tasks are executed according to this ordering that we know their real duration and, hence, obtain a real schedule, the *a-posteriori solution* with crisp job completion times. Hence, the main interest of a solution to the FOSP would lie in the ordering of tasks that it provides a priori, when information about the problem is incomplete. Ideally, this ordering should yield good schedules in the moment of its practical use, when tasks do have real durations.

Given this interpretation, for each fuzzy problem, we propose to run a Monte-Carlo simulation to evaluate the behaviour of the fuzzy solution across different real environments. In particular, a family of K crisp open shop problems is generated from the fuzzy problem, so they can be interpreted as its realisations. Such possible realisations are simulated by generating exact durations for each task at random according to different scenarios.

Here we shall consider four different scenarios to cover a wide range of possible situations corresponding to the cases where a-priori estimations of task durations are respectively accurate, pessimistic, optimistic or very inaccurate. This translates into generating crisp durations (possible realisations) as follows:

- Scenario I: The crisp duration of task o_{ij} corresponds to a probability distribution which is coherent with the possibility distribution defined by the fuzzy duration p_{ij}.
- Scenario II: From an optimistic point of view, given the fuzzy duration (p^1, p^2, p^3), the crisp duration is generated randomly in the interval $[p^1, p^1 + 0.25(p^3 - p^1)]$.
- Scenario III: From a pessimistic point of view, given the fuzzy duration (p^1, p^2, p^3), the crisp duration is generated randomly in the interval $[p^3 - 0.25(p^3 - p^1), p^3]$.
- Scenario IV: In a particularly adverse scenario characterized by a wrong prediction, given the fuzzy duration (p^1, p^2, p^3), the crisp duration is generated randomly in the interval $[p^1, p^1 + 0.25(p^3 - p^1)] \cup [p^3 - 0.25(p^3 - p^1), p^3]$.

Given a solution to the FOSP, we consider the ordering of tasks it provides, represented by the decision variable $\mathbf{z}$. For a crisp version of the FOSP, let $\mathbf{p}^c$ be the matrix of crisp durations where p^c_{ij} is the a-posteriori duration of task o_{ij}, generated according to one of the four scenarios above. Then, the ordering $\mathbf{z}$ can be used to obtain a crisp time-schedule as explained in Section 3.1, using real durations instead of fuzzy ones. If instead of a single crisp instance we consider the whole family of K crisp problems, each with a duration matrix $\mathbf{p}^c_k$, we obtain K values of makespan $C_{\max}(\mathbf{z}, \mathbf{p}^c_k)$ and K values of tardiness $T_{\max}(\mathbf{z}, \mathbf{p}^c_k)$, $k = 1, \ldots, K$. In the context of the lexicographic goal programming model, the performance of a solution can then be assessed using the percentage of the K instances where the solution satisfies the objective target values, giving more weight to the objective with highest priority, given the hierarchical nature of the proposed model.

Notice that the idea of evaluating different alternatives according to various scenarios as a way of dealing with imprecise or poorly defined data is not new. In fact, there is a clear connection between the semantics for fuzzy schedules and the related performance measure proposed herein and the increasingly popular concept of robustness. Indeed, a "robust" solution is, intuitively, a solution that performs "well" or "not too bad" in all scenarios [23], and an approach based on finding robust solutions should prevent from taking decisions with disastrous consequences in the case that a particularly adverse scenario should prevail at the end.

4. Particle swarm optimisation for the FOSP

Particle swarm optimisation (PSO) is a population-based stochastic optimisation technique inspired by bird flocking or fish schooling [25]. In PSO, each position in the search space corresponds to a solution of the problem and particles in the swarm cooperate to explore the space and find the best position (hence best solution). Particle movement is mainly affected by the three following factors:

- Inertia: Velocity of the particle in the latest iteration.
- *pbest*: The best position found by the particle.
- *gbest*: The best position found by the swarm so far ("the best *pbest*").

Potential solutions are represented by particles flying through the problem space, changing their position and velocity to follow the current optimum particles *pbest* and *gbest*. A generic PSO algorithm can be found in Algorithm 1: first, the initial swarm is generated and evaluated and then the swarm evolves until a termination criterion is satisfied. In each iteration, a new swarm is built from the previous one by changing the position and velocity of each particle to move towards its *pbest* and *gbest* locations.

In [31] a PSO algorithm was proposed to minimise the expected makespan of fuzzy open shop. This algorithm was in turn inspired by the method proposed in [37] for the deterministic OSP, which improved the best results published so far. Here we extend this algorithm to the multiobjective setting described in the previous section.

Algorithm 1. A generic PSO algorithm

Input A FOSP instance
Output A schedule for the input instance
  Generate and evaluate the initial swarm;
  Compute *gbest* and *pbest* for each particle;
  while no Termination Criterion is satisfied **do**
    for each particle k **do**
      for each dimension d **do**
        Update velocity v_d^k;
        Update position x_d^k;
      Evaluate particle k;
      Update *pbest* and *gbest* values;
  return The schedule from the best particle evaluated so far

4.1. Position representation and evaluation

Particle positions are represented using a priority-based representation. A decision variable **z** is encoded as a *priority array* $\mathbf{x}^k = (x_l^k)_{l=1,\ldots,nm}$, where x_l^k denotes the priority of task l, so a task with smaller x_l^k has a higher priority to be scheduled.

Given a decision variable **z**, which is a permutation of tasks representing an OSP solution, we can transfer this permutation to a priority array as follows. First, from **z** we obtain a position array, denoted $\mathbf{pos^z}$, such that $pos_l^{\mathbf{z}}$ is the position of task l in **z** ($pos_l^{\mathbf{z}} = i$ if and only if $z_i = l$). For instance, given the following decision variable for a problem with $n = 2$ jobs and $m = 3$ machines:

$$\mathbf{z} = (4, 1, 5, 2, 3, 6)$$

the position array for the above decision variable is:

$$\mathbf{pos^z} = (2, 4, 5, 1, 3, 6).$$

Then, the priority array **x** is obtained by setting x_l to a random value in the interval $(pos_l^{\mathbf{z}} - 0.5, pos_l^{\mathbf{z}} + 0.5)$, so a task with smaller x_l has higher priority to be scheduled. For the above permutation, a possible particle position would be:

$$\mathbf{x} = (2.3, 3.7, 5.4, 0.8, 2.8, 5.9).$$

Conversely, from every particle position **x** we can obtain a position array $\mathbf{pos^x}$ where $pos_i^{\mathbf{x}}$ is the position of x_i if the elements of **x** were reordered in non-decreasing order. Given this representation, the PSO does not record in *gbest* and *pbest* the best positions found so far, but rather the corresponding priority arrays.

Algorithm 2. *pFG&T*

Input A FOSP instance and a particle position $\mathbf{x}^k$
Output A schedule for the input instance considering the priorities given by $\mathbf{x}^k$
  $\Omega \leftarrow \{1, \ldots, nm\}$;
  while $\Omega \neq \emptyset$ **do**
    Compute $\{S_l\colon\ l \in \Omega\}$ and $\{C_l\colon\ l \in \Omega\}$ considering only tasks previously scheduled;
    $C^* \leftarrow \min_{l\in\Omega}\{E[C_l]\}$;
    $S^* \leftarrow \min_{l\in\Omega}\{E[S_l]\}$;
    Identify the conflict set $O \leftarrow \{l\colon\ E[S_l] < S^* + \delta \times (C^* - S^*), l \in \Omega\}$;
    Choose the task l^* from O with smallest x_l^k;
    Schedule the operation l^*; */*fix S_{l^*} */*
    $\Omega \leftarrow \Omega - \{l^*\}$;
  return The schedule s given by $\{S_l\colon\ l \in \{1, \ldots, nm\}\}$

A particle may be decoded in several ways. For the crisp job shop and by extension for the open shop, it is common to use the G&T algorithm [14], which is an active schedule builder. A schedule is *active* if one task must be delayed for any other one to start earlier. Active schedules are good in average and, most importantly, the space of active schedules contains at least an optimal one, that is, the set of active schedules is *dominant*. For these reasons it is worth to restrict the search to this space. In [15] a narrowing mechanism was incorporated to the G&T algorithm in order to limit machine idle times using a delay parameter $\delta \in [0, 1]$, thus searching in the space of so-called parameterised active schedules. In the deterministic case, for $\delta < 1$ the search space is reduced so it may no longer contain optimal schedules and, at the extreme $\delta = 0$ the search is constrained to *non-delay* schedules, where a resource is never idle if a requiring task is available. This variant of G&T has been applied in [37] to the deterministic OSP, under the name "parameterized active schedule generation algorithm".

Algorithm 2, denoted *pFG&T*, is an extension of parameterised G&T to the case of fuzzy processing times proposed in [31]. It should be noted that, due to the uncertainty in task durations, even for $\delta = 1$, we cannot guarantee that the produced schedule will indeed be active when it is actually performed (and tasks have exact durations). We may only say that the obtained fuzzy

schedule is *possibly active*. Throughout the algorithm, Ω denotes the set of tasks that have not been scheduled, $\mathbf{x}^k$ the priority array and S_l and C_l the starting and completion time of task o_{ij} such that $l = (i-1)m + j$. Notice that the *pFG&T* algorithm may change the task processing order given by the particle position.

4.2. Particle movement and velocity

Particle velocity is traditionally updated depending on the distance to *gbest* and *pbest*. Instead, this PSO only considers whether the position value x_l^k is larger or smaller than $pbest_l^k$ ($gbest_l$). For any particle, its velocity is represented by an array of the same length as the position array where all the values are in the set $\{-1, 0, 1\}$. The initial values are set randomly and updating is controlled by the inertia weight w and probabilities $p_1, p_2 \in [0, 1]$ such that $p_1 + p_2 \leqslant 1$. For each particle k and dimension d, if $v_d^k \neq 0$, v_d^k will be set to 0 with probability $1 - w$, meaning that if x_d^k was either increasing or decreasing, x_d^k stops at this iteration. After this, if $v_d^k = 0$ (the particle has stopped either in this or in previous iterations), with probability p_1, v_d^k and x_d^k will be updated depending on $pbest_d^k$ and with probability p_2 they will be updated depending on $gbest_d$, always introducing an element of randomness. Further detail on particle updating is given in Algorithm 3.

Position mutation. After a particle moves to a new position, we randomly choose a task and then mutate its priority value x_d^k independently of v_d^k with probability p_m. For a problem of size $n \times m$, if $x_d^k < (nm/2)$, x_d^k will take a random value in $[mn - n, mn]$, and $v_d^k = 1$; otherwise (if $x_d^k > (nm/2)$), x_d^k will take a random value in $[0, n]$ and $v_d^k = -1$.

Diversification strategy. If all particles have the same *pbest* solutions, they may get trapped into local optima. To prevent such situation, a diversification strategy is proposed in [37] that keeps the *pbest* solutions different. In this strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far, where N is the size of the swarm. Once any particle generates a new solution, the *pbest* solutions will be updated as follows: if the new solution equals the objective values of any *pbest* solution, the latter will be replaced with the new solution; else if the new solution is better than the worst *pbest* solution and it is different from all *pbest* solutions, then the worst *pbest* solution is replaced by the new one.

Algorithm 3. Particle movement

Input A particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$, best particle and swarm positions $pbest^k$ and *gbest*, inertia w and probabilities p_1, p_2
Output The updated particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$
for each dimension d **do**
  generate random value *rand* $\sim U(0, 1)$;
  if $v_d^k \neq 0$ and *rand* $\geqslant w$ **then**
    $v_d^k \leftarrow 0$;
  if $v_d^k = 0$ **then**
    generate random value *rand* $\sim U(0, 1)$;
    if *rand* $\leqslant p_1$ **then**
      if $pbest_d^k \geqslant x_d^k$ **then** $v_d^k \leftarrow 1$;
      else $v_d^k \leftarrow -1$;
      generate random value $rand_2 \sim U(0, 1)$;
      $x_d^k \leftarrow pbest_d^k + rand_2 - 0.5$;
    else if $p_1 <$ *rand* $\leqslant p_1 + p_2$ **then**
      if $gbest_d \geqslant x_d^k$ **then** $v_d^k \leftarrow 1$;
      else $v_d^k \leftarrow -1$;
      generate random value $rand_2 \sim U(0, 1)$;
      $x_d^k \leftarrow gbest_d + rand_2 - 0.5$;
  else
    $x_d^k \leftarrow x_d^k + v_d^k$;
return The updated particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$

5. Experimental evaluation

We now proceed to empirically evaluate the proposed method in several steps, using a total of 120 problem instances. First, a parametric analysis will be conducted to decide on a good parameter-configuration for the PSO as well as for the schedule generation algorithm *pG&T*. Then, we will present results of expected makespan and tardiness minimisation obtained by the PSO considering two different objective priority structures. As no multiobjective approach to fuzzy open shop can be found in the literature, we shall compare the multiobjective PSO to the single-objective PSO that considers only the primary objective and we shall also compare it with a memetic algorithm from the literature. We will then proceed to analyse the relationship between the lexicographic solutions and solutions of a dominance-based approach. Finally, we shall present an analysis of the solutions obtained with the PSO based on the a-posteriori semantics of fuzzy schedules introduced in Section 3.3.

5.1. Experiment setting

For the experimental evaluation, we use some of the instances proposed in [18]. These were obtained based on the well-known benchmark from [5], which consists of 6 families, denoted $J3, J4, \ldots, J8$, of sizes $3 \times 3, 4 \times 4, \ldots, 8 \times 8$, containing 8 or 9 instances each. Each family is divided into three sets of problems *per0*, *per10* and *per20* according to the difference between minimum and maximum workloads of jobs and machines (the number in the name refers to this difference in percentage). We shall only consider the largest instances, pertaining to the blocks of size 7×7 and 8×8, comparing our results to those of the memetic algorithm (MA) proposed in [18]. There are 10 fuzzy versions of each original problem instance, generated by transforming the original crisp processing times into symmetric TFNs such that their modal value corresponds to the original duration. To add a crisp due date d_i for each job J_i we follow [2] and compute $d_i = TF \sum_{j=1}^{m} p_{ij}^2$, where *TF* is the tightness factor for the due date, in our case, $TF = 1.1$.

Given the method for generating due dates, in *per0* instances, where all jobs have the same workload (and consequently the same due-date), the difference between tardiness and makespan is only a constant (the due date value). Both objectives are thus strongly correlated, making these instances unsuitable for our multi-objective study. Hence for the experimental analysis we shall restrict to the instances *per10* and *per20* of size 7×7 and 8×8 (in total, 120), these being the hardest problems to solve considering both objectives. These problem instances and more detailed results of the experiments presented in this section can be found at http://www.di.uniovi.es/iscop.

Given the two fuzzy goals $f_1 = C_{\max}$ and $f_2 = T_{\max}$, we consider four objective functions: two single-objective functions given by the expected values $E[f_1]$ and $E[f_2]$ and two multiobjective functions that result from incorporating two different priority structures in expression (10). The first multiobjective function, denoted l_{12}, corresponds to the priority structure defined by $\rho(f_i) = i$, that is, the most prioritary goal is the makespan f_1. The second objective function l_{21} corresponds to $\rho(f_1) = 2$, $\rho(f_2) = 1$, i.e. the most prioritary goal is to minimise tardiness. These hierarchies correspond to probably the most common objectives in the open shop literature, namely minimise makespan or maximise due-date satisfaction.

5.2. Parametric analysis

As an initial configuration for the PSO, we take the best parameter values obtained in [37]: swarm size $N = 60$, guiding probabilities $p_1 = 0.7$, $p_2 = 0.1$, mutation probability 1 and inertia weight w linearly decreasing from 0.9 to 0.3. Regarding the filtering mechanism of the search space given in the schedule generator, in [31] it is suggested to take $\delta = 0.25$ for the instances considered herein. The termination criterion is the number of iterations, which depends on the problem size: 2800 for 7×7 instances and 3000 for 8×8 instances. Finally, to fix the target values we emulate the DM and use the experience gained using $E[f_1]$ and $E[f_2]$ as single objectives, setting b_1 (resp. b_2) equal to the worst value of $E[f_1]$ ($E[f_2]$) across 30 runs of the PSO.

To measure the quality of each configuration, since in most of the problems the target value of the objective with highest priority is reached, we compute the distance from the average solution (across the 10 runs) to the target value of the secondary objective and try to minimise it. To compare results on different instances, distance values are normalised and finally, for those instances where the algorithm does not reach the target with highest priority, we set the normalised distance to 1 in order to penalise that configuration. Tables and figures in the following show the average of these values across all instances.

5.2.1. PSO algorithm parameters

Starting with this initial configuration, we proceed to perform a parametric analysis for the algorithm. In this analysis, we try different values for the PSO algorithm's parameters (in bold we highlight the values of the starting configuration):

- Inertia: Linearly decreasing from ω_s to ω_e, with $\omega_s \in \{0.5, 0.7, \mathbf{0.9}\}$ and $\omega_e \in \{0.1, \mathbf{0.3}, 0.5\}$.
- Mutation probability: Values in $\{0, 0.25, 0.50, 0.75, \mathbf{1}\}$.
- Guiding probabilities p_1 and p_2: all possible pairs of the values in $\{\mathbf{0.1}, 0.3, 0.5, \mathbf{0.7}, 0.9\}$, provided that they add up to a maximum of 1.

The main drawback of a full factorial design is its high computational cost, especially when the number of parameters (factors) or their domain values are large, that is, a very large number of experiments must be realised [38]. A common approach to parameter setting in the literature is to resort to a sequential design: this has the advantage of keeping the complexity of

the parametric analysis within reasonable bounds at the cost of failing to fully consider possible interactions between different parameters. This may raise concerns about the influence of the sequential parameter ordering on the final results. In our experimental analysis we have actually tried the 6 possible orderings for the three parameters under consideration, obtaining in the end three different winning configurations (there exist different orderings yielding the same setup):

- Inertia ω from 0.7 to 0.3, guiding probabilities $p_1 = 0.9$, $p_2 = 0.1$ and mutation probability $p_m = 1.0$;
- Inertia ω from 0.9 to 0.3, guiding probabilities $p_1 = 0.9$, $p_2 = 0.1$ and mutation probability $p_m = 0.5$;
- Inertia ω from 0.7 to 0.5, guiding probabilities $p_1 = 0.7$, $p_2 = 0.1$ and mutation probability $p_m = 1.0$.

While the initial configuration from [37] obtains an average distance value of 0.55, the above configurations obtain, respectively, average distance values equal to 0.335, 0.341 and 0.324. The latter is the one offering best results in terms of quality, although differences between configurations are quite small. This configuration is also the one with smallest variation depending on the objective function used (l_{12} or l_{21}). For these reasons, the ordering followed to obtain the third configuration will be taken to be the best one.

We now describe in detail the process followed to fix parameter values with this ordering. Starting with the initial algorithm's setup from [37], at each step, we run the PSO with l_{12} and l_{21} 10 times on a fuzzy instance of each 8×8 problem, testing different values for a each parameter as follows: first inertia, then mutation probability and finally guiding probabilities.

Figure 2 shows the average across the 8×8 problems of the normalised distance values obtained with l_{12} and l_{21} using different inertia values linearly decreasing in the intervals specified on the X-axis. An additional bar has been added to show the "overall" performance of the algorithm, corresponding to the average value obtained using both objective functions. We can see that the best configuration for this average value is to have inertia going from 0.7 to 0.5; this is also a good configuration for both objective functions l_{12} and l_{21} individually.

Once the inertia is fixed, we try different mutation probability values p_m. Figure 3 illustrates the importance of this parameter for the behaviour of the algorithm, with larger probability values yielding the best

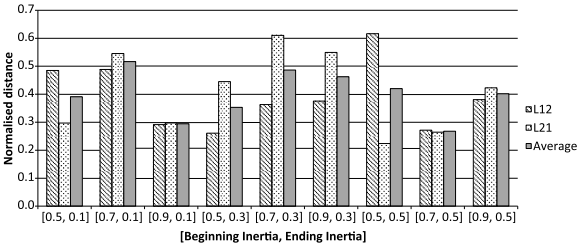


Fig. 2. Algorithm's performance with varying inertia weight.

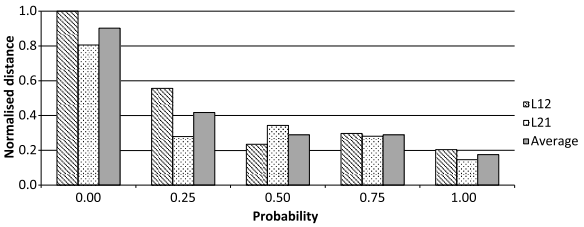


Fig. 3. Algorithm's performance with varying mutation probability.

results. As it happens in the single-objective version of the PSO, the best option is to mutate the particles with probability $p_m = 1$. High mutation probabilities may strike the reader as unexpected but are not unusual in the literature for discrete combinatorial optimisation. Indeed, in traditional PSO all dimensions are typically independent of each other, so particle updates are performed independently in each dimension with larger velocity values allowing the particle to explore more distant areas. However, in PSO for discrete optimisation – such as the one in this paper, where dimensions codify task relative orders – independence no longer holds and velocity is limited to absolute values representing differences instead of distances between particles, thus reducing the exploration potential. Mutation – which in our case, translates to changing the relative order of one of the nm tasks – is introduced to deal with this shortcoming, reducing the possibility of getting easily trapped into local minima [22].

After fixing the mutation probability, we test the guiding probabilities p_1 and p_2; the results are shown in Table 1. At first sight, it is tempting to conclude that the best option in terms of normalised distance is to take $p_1 = 0.9$, $p_2 = 0.1$; however results vary greatly with the objective function used (0.06 for l_{21} but 0.13 for l_{12}). If we look at the results obtained for $p_1 = 0.7$, $p_2 = 0.1$, there is little change in the average distance value but in this case the quality of the solution is clearly more independent of the objective function used. Consequently, we take $p_1 = 0.7$, $p_2 = 0.1$ as the optimal configuration.

Table 1
Algorithm's performance with varying guiding probabilities

Values	l_{12}	l_{21}	*Average*
$p_1 = 0.1, p_2 = 0.1$	0.49	0.34	0.42
$p_1 = 0.1, p_2 = 0.3$	0.60	0.78	0.69
$p_1 = 0.1, p_2 = 0.5$	0.80	0.78	0.79
$p_1 = 0.1, p_2 = 0.7$	0.94	0.80	0.87
$p_1 = 0.1, p_2 = 0.9$	0.97	0.83	0.90
$p_1 = 0.3, p_2 = 0.1$	0.23	0.32	0.28
$p_1 = 0.3, p_2 = 0.3$	0.57	0.37	0.47
$p_1 = 0.3, p_2 = 0.5$	0.46	0.58	0.52
$p_1 = 0.3, p_2 = 0.7$	0.70	0.64	0.67
$p_1 = 0.5, p_2 = 0.1$	0.23	0.14	0.19
$p_1 = 0.5, p_2 = 0.3$	0.47	0.29	0.38
$p_1 = 0.5, p_2 = 0.5$	0.49	0.32	0.41
$p_1 = 0.7, p_2 = 0.1$	0.11	0.09	**0.10**
$p_1 = 0.7, p_2 = 0.3$	0.15	0.25	0.20
$p_1 = 0.9, p_2 = 0.1$	0.13	0.06	**0.09**

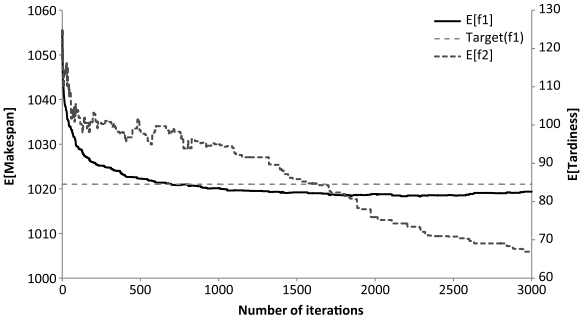


Fig. 4. Evolution of $E[f_1]$ and $E[f_2]$ using l_{12} for *j8-per20-0* instance.

Regarding the stopping criterion, we keep the number of iterations proposed initially. To illustrate that this ensures proper convergence, Fig. 4 shows the average evolution of l_{12} along 3000 iterations for one of the fuzzy instances generated from *J8-per20-0*. We can see how, initially, the algorithm minimises the expected makespan while the behaviour of the expected tardiness is erratic. However, once the algorithm has reached the expected makespan target (around iteration 600), it starts to mimimise tardiness as well. The remaining instances follow the same convergence pattern.

We now test the effect of increasing the swarm size. Notice that this may affect the runtime, being and important penalisation factor. A common approach in the literature is to measure the computational effort of a metaheuristic in terms of the number of objective function evaluations, which is independent of the computer system. For this reason, we adjust the number of iterations so the PSO evaluates the same number of particles for all possible values of swarm size. We try three different swarm sizes: 60, 80 and 100, with 3000, 2250 and 1800 iterations respectively so the number of evaluations is kept the same.

Table 2
Algorithm's performance with varying swarm size

Swarm size	l_{12}	l_{21}
60	0.17	0.30
80	0.45	0.39
100	0.28	0.44

Table 3
Algorithm's performance with varying the delay parameter

Delay value δ	l_{12}	l_{21}	*Average*
0	0.59	0.83	0.71
0.25	0.01	0.07	0.04
0.5	0.09	0.10	0.10
0.75	0.90	0.85	0.87
1	1.00	1.00	1.00

The performance of the obtained solutions, measured in terms of normalised distance to the target value of the secondary objective, can be seen in Table 2; attending to these values, the option providing the best objective values at the same computational cost is to keep 60 particles in the swarm. This would be in line with the results obtained for continuous PSO in [42], where medium swarm sizes obtain the best results.

5.2.2. Parameterisation of the search space

There is another parameter in the algorithm, the delay parameter δ used by the schedule builder (Algorithm 2). Notice that, unlike the other parameters, the differences in algorithm performance due to variations of the delay are a consequence of changes in the search space explored not in the search process followed by the PSO algorithm. We have tried four possibilities: $\delta = 0$, which corresponds to exploring the set of non-delay schedules; $\delta = 1$, to explore the set of active schedules; and $\delta = 0.25, 0.5$, and 0.75, to explore three proper subsets of the space of active schedules. Table 3 shows the results (normalised distances to the target values of the secondary objectives) obtained, showing that both multi-objective approaches perform better for $\delta = 0.25$.

5.3. Results of the multiobjective optimisation

Given the optimal configuration obtained with the above parametric analysis, we now proceed to eval-

Table 4
Results obtained by the MOPSO

Prob.	Obj.	Size 7 × 7				Size 8 × 8			
		$E[C_{\max}]$		$E[T_{\max}]$		$E[C_{\max}]$		$E[T_{\max}]$	
		Avg.	Std. dev.	Avg.	Std. dev.	Avg.	Std. dev.	Avg.	Std. dev.
per10-0	$E[f_1]$	1030	2.26	19.2	5.79	1050	2.64	23.0	5.70
	l_{12}	1031	1.66	14.3	3.41	1052	2.42	18.5	4.42
	$E[f_2]$	1056	9.14	11.1	0.92	1072	10.62	13.5	1.32
	l_{21}	1040	5.80	11.8	0.97	1061	7.93	14.8	1.19
per10-1	$E[f_1]$	1017	1.22	20.1	1.46	1030	3.76	30.3	7.88
	l_{12}	1018	1.71	19.5	1.66	1033	3.75	23.5	5.65
	$E[f_2]$	1048	9.29	12.0	1.32	1064	11.44	14.1	1.52
	l_{21}	1036	5.34	13.8	0.67	1049	10.89	15.6	1.32
per10-2	$E[f_1]$	1031	2.74	25.1	7.36	1033	5.58	27.4	7.67
	l_{12}	1033	3.51	21.1	5.76	1036	5.03	20.8	5.45
	$E[f_2]$	1066	12.25	11.5	1.14	1063	13.04	14.2	1.53
	l_{21}	1058	8.93	12.6	1.47	1052	10.13	15.8	1.47
per20-0	$E[f_1]$	1001	0.12	72.4	10.75	1016	2.64	99.4	20.65
	l_{12}	1001	0.31	46.8	9.68	1019	2.09	72.1	20.19
	$E[f_2]$	1030	8.95	15.8	0.74	1071	11.58	13.4	1.32
	l_{21}	1021	7.92	16.9	0.97	1059	11.38	14.7	1.21
per20-1	$E[f_1]$	1028	2.43	90.1	22.84	1001	0.86	93.1	20.29
	l_{12}	1031	1.74	56.0	18.98	1003	0.92	57.0	21.40
	$E[f_2]$	1081	7.54	16.4	0.93	1023	9.87	16.9	1.58
	l_{21}	1073	9.44	17.5	1.27	1013	7.42	18.6	1.61
per20-2	$E[f_1]$	1021	2.41	65.6	26.56	1014	2.49	78.6	23.70
	l_{12}	1023	2.31	49.9	23.11	1018	2.35	47.3	17.11
	$E[f_2]$	1072	15.01	14.7	1.28	1060	15.25	15.0	1.59
	l_{21}	1059	12.72	16.4	1.34	1048	12.51	16.3	1.56

uate the performance of the multiobjective PSO algorithm in terms of objective-value minimisation. As mentioned above, we run the PSO with the single-objective functions $f_1 = C_{\max}$ and $f_2 = T_{\max}$ (minimising their expected values) as well as two different multiobjective functions, l_{12} and l_{21}, corresponding to the two possible priority structures for f_1 and f_2.

Table 4 contains a summary of the results: for each objective function we measure $E[f_1]$ and $E[f_2]$ in the obtained schedule and compute the average values and standard deviations across the 30 executions of the PSO and the 10 fuzzy instances generated from the same original problem. According to this table, the multiobjective versions with l_{12} and l_{21} behave similarly to the corresponding single-objective ones, $E[f_1]$ and $E[f_2]$ regarding their primary goal. Additionally, they improve considerably on the other goal. Indeed, in all instances, l_{12} reaches the expected makespan target while the expected tardiness values obtained with l_{12} are 26% better in average than using $E[f_1]$, being better in all problem instances. If we measure the reduction of the gap between the expected tardiness and its corresponding target, the multi-objective approach l_{12} is 49% better than using $E[f_1]$ in average. Clearly, minimising the makespan does not always imply minimising tardiness. If we now compare l_{21} with $E[f_2]$, results are similar: in all instances l_{21} reaches the expected tardiness target whereas the gap between the expected makespan and its target value is reduced 43% in average when the multi-objective approach is used.

According to Table 4, the improvement in expected tardiness with l_{12} is greater for *per20* problems than for *per10* ones. This is not surprising: with higher workload variability, due dates are less uniform in *per20* in-

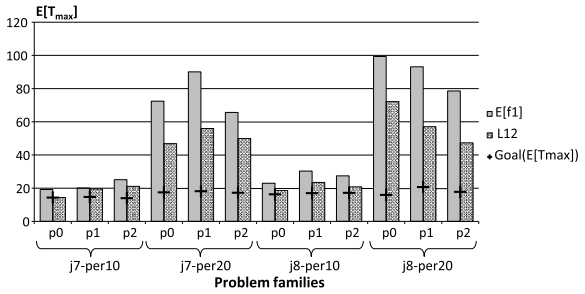


Fig. 5. Comparison between $E[f_1]$ and l_{12} regarding the secondary objective function ($E[T_{\max}]$).

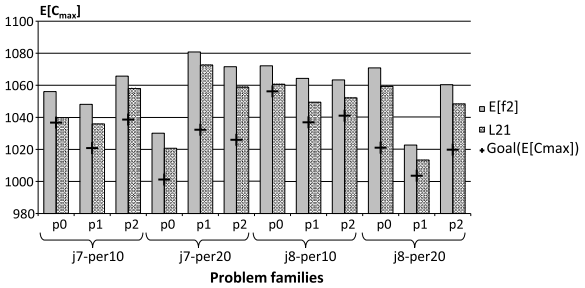


Fig. 6. Comparison between $E[f_2]$ and l_{21} regarding the secondary objective function ($E[C_{\max}]$).

stances. Indeed when only expected makespan is optimised, tardiness values are considerably larger for *per20* than for *per10* instances, the latter being closer to their target values. In consequence, for *per20* instances there is greater room for improvement. Figures 5 and 6 illustrate the behaviour of the algorithms l_{12} and l_{21} with respect to the single criteria algorithms. As they always reach the goal having the highest priority, the figures plot only the values for the function with least priority.

Notice that comparisons between different multiobjective functions do not make sense, as they model different priority requirements. In a hierarchical approach such as this, the decision maker is responsible for establishing an adequate hierarchy among goals according to his/her knowledge of the problem. The relevance of the target values and the resulting influence in a good performance of the algorithm must be taken into account. In the case of a very hard target for the objective with highest priority, the algorithm behaves like the corresponding single-objective one. In experiments not reported here we have tried different values for tardiness target and the results showed that for a difficult target, l_{21} behaves similarly to $E[f_2]$ in terms of makespan and tardiness. On the other hand, when the tardiness target is relaxed, l_{21} reached the target in early iterations of the algorithm and then optimised the makespan, thus behaving like $E[f_1]$.

Table 5
Comparison of results for $E[C_{\max}]$

Problem	Size 7×7		Size 8×8	
	MA-$E[f_1]$	PSO-l_{12}	MA-$E[f_1]$	PSO-l_{12}
per10-0	1066.04	1031.37	1083.08	1052.32
per10-1	1052.36	1017.89	1065.86	1032.83
per10-2	1067.27	1032.89	1070.77	1036.19
per20-0	1004.22	1000.97	1036.72	1019.14
per20-1	1043.73	1030.87	1013.79	1002.53
per20-2	1042.05	1022.96	1034.66	1017.77

Finally, we compare the PSO using l_{12} with the single-objective MA algorithm from [17]. Table 5 contains expected makespan results for both methods, with average solutions obtained by both methods across the 10 fuzzy instances of each original crisp problem, MA optimising only $E[C_{\max}]$ and PSO with l_{12}. In [31] the PSO optimising $E[C_{\max}]$ compared favourably with the MA algorithm and we see that this is still the case when we use the multiobjective function l_{12}.

5.4. Relationship with Pareto-optimality

Dominance-based algorithms constitute the most common approach to multiobjective optimisation problems. Their aim is to find sets of non-dominated solutions, also known as Pareto-optimal or efficient solutions. There are known relationships between lexicographically optimal solutions and efficient solutions; in particular, a lexicographically optimal solution is always non-dominated and optimal solutions to a lexicographical goal-programming problem are non-dominated solutions to the problem of minimising deviations w.r.t. to target values [12].

The aim of this section is to empirically explore the relationship between the solutions obtained by our algorithm and those obtained by a multiobjective dominance-based approach, which assumes that no hierarchy can be established among objectives. To this end, we modify our PSO to handle sets of non-dominated solutions, in the same way that the original PSO from [37] is changed to become a multiobjective dominance-based PSO (DB-PSO for short) in [35] and [36].

The resulting DB-PSO is run 10 times on the *per20* family of problems, these being the instances with the smallest correlation between the objective functions. Figure 7 illustrates the behaviour in the objective space

of the solutions obtained by DB-PSO for a *per20-0* instance, with a vertical and a horizontal line marking the target values for each objective. If S_i denotes the set of non-dominated solutions obtained on the ith run, $i = 1, \ldots, 10$, the circles in the figure represent the solutions in the union $S = S^1 \cup S^2 \cup \cdots \cup S^{10}$. We can appreciate the familiar shape or distribution usually obtained by dominance-based algorithms. Additionally, the dotted line is formed by joining the solutions in the set PO_1 of non-dominated elements from S.

Also in Fig. 7 we can see solutions obtained with the lexicographic approach: there is a set $S_{12} = \{s^1_{12}, \ldots, s^{10}_{12}\}$ of ten solutions (represented as triangles) which correspond to 10 runs of the PSO algorithm with the objective function l_{12} and another set $S_{21} = \{s^1_{21}, \ldots, s^{10}_{21}\}$ of ten solutions (represented as squares) obtained with 10 runs using l_{21}. The solid line is obtained by connecting all the elements in PO_2, the set of non-dominated solutions in the union $S \cup S_{12} \cup S_{21}$ of all solutions. We can appreciate how the lexicographic solutions complement those obtained by the DB-PSO, by focusing on the "extreme ends" of the non-dominated front.

In order to obtain quantitative results, we take for every *per20* problem 10 lexicographic solutions s^i_{kl}, $i = 1, \ldots, 10$, $k \neq l$, $k, l \in \{1, 2\}$ and 10 sets of non-dominated solutions S^j, $j = 1, \ldots, 10$, and we count all pairs (s^i_{kl}, S^j) out of the 100 possible ones where s^i_{kl} is a member of the set of non-dominated solutions from $\{s^i_{kl}\} \cup S^j$ (in other words, s^i_{kl} is not dominated by any solution in S^j). It turns out that, in average across all *per20* problems, s^i_{12} is part of the set of non-dominated solutions in 88% of the cases; this proportion grows to 95% when l_{21} is used instead of l_{12}. It is also remarkable that in 77% of the cases, a solution s^i_{21} is non-dominated by the solutions in S^j and at the same time does not dominate the solutions in S^j; we believe this stems from the fact that the target values fixed for tardiness are hard to achieve.

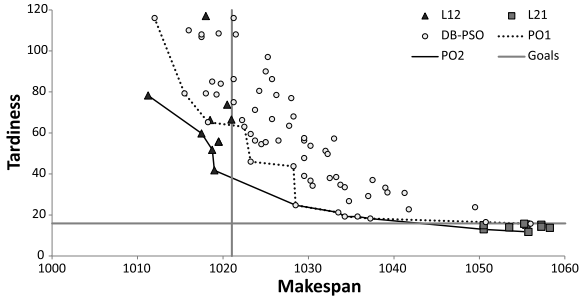


Fig. 7. Solutions obtained with l_{12}, l_{21} and DB-PSO across 10 runs on *J8-per20-0* instance.

5.5. Study using a-posteriori semantics

To evaluate our proposal in terms of the semantics for fuzzy schedules described in Section 3.3, we use the six largest problems – those from *j8-per10-0* to *j8-per20-2* – and we take a single fuzzy instance of each one. This provides a sufficiently diverse set of problem instances while keeping it to a reasonable size given the computational cost of the following experimental study. In particular, here we take the first-generated fuzzy instance of each original problem. We run l_{12} on the fuzzy problem instances 30 times, thus obtaining 30 permutations $\mathbf{z}^F_1, \ldots, \mathbf{z}^F_{30}$ representing 30 solutions for each instance. Now, one of these permutations needs to be selected as the solution representative of the performance of the PSO on the fuzzy problem. This selection should be made so as to avoid as much as possible the random effect of stochastic algorithms. For this reason, we consider the set of 30 vectors of objective values obtained with the permutations and take the permutation lying closest to the centre of gravity of this set. Then, in order to establish a comparison, we repeat the process running the PSO algorithm on the original crisp problems, thus obtaining 30 new permutations $\mathbf{z}^C_1, \ldots, \mathbf{z}^C_{30}$ for each problem from which one is selected following the same criterion as above. This provides us with prototypical solutions z^F and z^C of the fuzzy and crisp instance of each problem, which correspond to taking into account or overlooking the uncertainty in task durations during the search of a solution. We can now compare these two prototypical solutions based on the a-priori semantics, generating four sets containing 1000 deterministic instances each, which correspond to the possible realisations of the problem according to the four scenarios described in Section 3.3.

To compare the fuzzy solutions z^F with the crisp ones z^C, we shall use the percentage of instances (a-posteriori realisations of the problem) where the target values are satisfied, paying special attention to the objective with highest priority. Table 6 contains the average of these percentage values across all instances. It only reports results under scenarios I and IV because results under the remaining scenarios are trivial:

Table 6
Percentage of instances that reach the target value

	Makespan		Tardiness	
	z^F	z^C	z^F	z^C
Scenario I	48%	37%	41%	46%
Scenario IV	12%	7%	21%	17%

100% satisfaction in scenario II – where all durations are small – and 0% in scenario III – where all durations are large and the target is never reached.

For the non-trivial scenarios, the fuzzy solutions perform clearly better in terms of makespan, which is the highest-priority goal. For scenario I, the percentage of instances that reach the makespan target when scheduled with the fuzzy permutations is a 30% higher than when the crisp permutations are used. More importantly, when the initial fuzzy durations are not accurately estimated (scenario IV), fuzzy solutions reach target values over 70% times more than the crisp ones. Regarding the secondary objective, we can observe that under scenario I the percentage of instances that the percentage of instances where the tardiness target is reached is slightly higher when they are scheduled using the crisp permutation. However, the tardiness target is reached more often when permutations are obtained using fuzzy information under scenario IV, when differences between "a priori" predictions and real durations are bigger.

For a better insight into the behavior of the task processing orders given by the PSO when the fuzzy information is maintained along the search, versus those that only use the most likely information, we present some figures that illustrate different characteristics on a particular instance, *j8-per10-0*.

Figure 8 shows box-plots corresponding to makespan and tardiness values obtained by the samples generated in each scenario evaluating the two processing orders z^F and z^C (identified as "fuzzy" and "crisp" in the figure) across the 1000 crisp instances. This graphic shows that the smallest values are obtained under scenario II, being 0 in tardiness. On the opposite, scenario III yields the largest values; this is also the scenario where the differences between the crisp and the fuzzy permutations are larger, specially for tardiness values. The overall performance across all scenarios is slightly better for the fuzzy permutation, being clearly better under scenario III.

Figures 9–11 illustrate other properties for the same instance. The left-hand side of each figure corresponds to makespan values while the right-hand side corresponds to tardiness values. Each subfigure contains two triangles (TFNs) corresponding to the makespan (resp. tardiness) obtained after scheduling the original fuzzy problem following the task orderings $\mathbf{z}^F$ and $\mathbf{z}^C$. Together with the TFNs, we depict the normalised histograms of the corresponding crisp makespan values $C_{\max}(\mathbf{z}^C, \mathbf{p}_l^c)$ and $C_{\max}(\mathbf{z}^F, \mathbf{p}_l^c)$, $l = 1, \ldots, 1000$ and crisp tardiness values $T_{\max}(\mathbf{z}^C, \mathbf{p}_l^c)$ and $T_{\max}(\mathbf{z}^F, \mathbf{p}_l^c)$ under scenarios I, III and IV. Scenario II is again not included because of its trivial behaviour explained above. Even if these figures correspond to a single instance, they illustrate a standard behaviour across all instances. For example, the TFNs obtained with the fuzzy ordering $\mathbf{z}^F$ always less width than those obtained with the crisp ordering $\mathbf{z}^C$. This naturally translates into histograms with smaller spread. We also observe a strong correlation between the shape of the TFNs and the histograms under scenario I, reinforcing the interpretation of the fuzzy objective values as possibility distributions on the actual values that the objectives may take when tasks are actually executed and take exact durations (assuming that the original fuzzy durations are accurately estimated). Finally, under scenarios III and IV the behaviour of $\mathbf{z}^F$ is clearly better than that of $\mathbf{z}^C$, with significant differences for the histograms.

Finally, we would like to show not just an instance or an average behaviour, but the situation of the whole sample. In Fig. 12 we depict, for each permutation z^F and z^C and each a-posteriori scenario, the objective values for the 1000 crisp instances (a-posteriori realisations) as a cloud of bidimensional points together

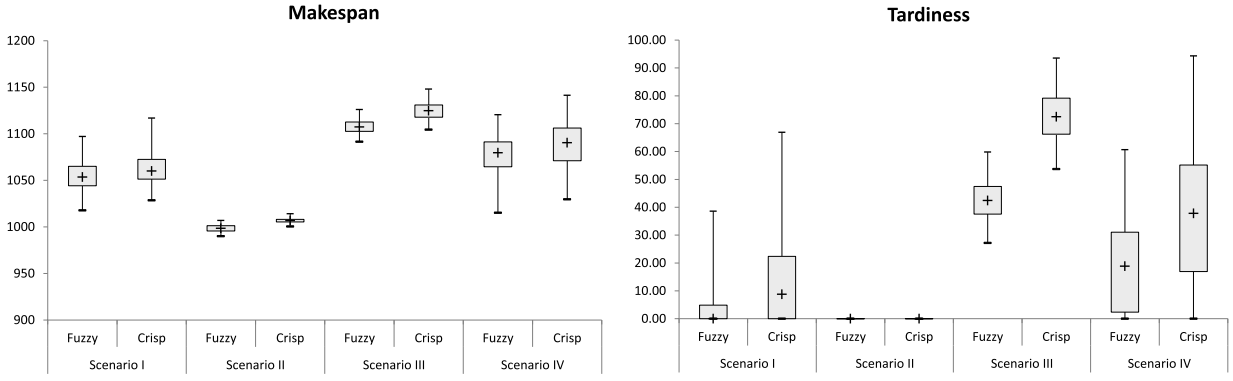


Fig. 8. Box plots for crisp makespan and tardiness values for instance *j8-per10-0* under each scenario.

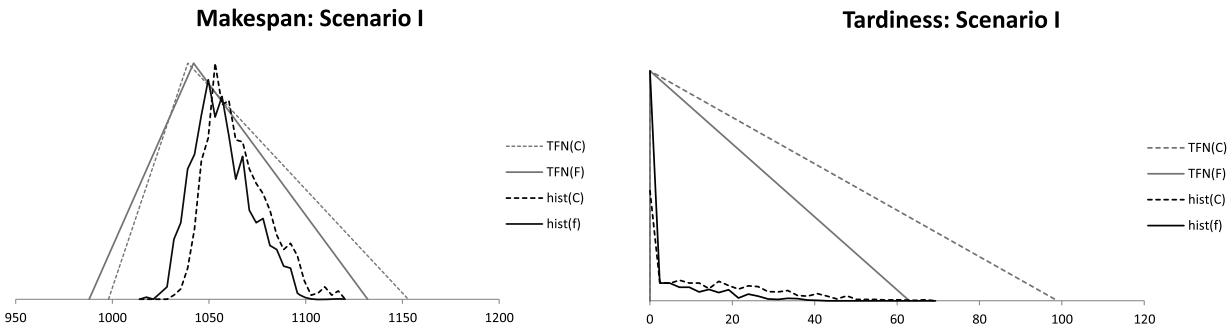


Fig. 9. $C_{\max}(\mathbf{z}^F, \mathbf{p})$, $C_{\max}(\mathbf{z}^C, \mathbf{p})$, $T_{\max}(\mathbf{z}^F, \mathbf{p})$, $T_{\max}(\mathbf{z}^C, \mathbf{p})$, and histograms of crisp $C_{\max}$ and $T_{\max}$ values obtained with $\mathbf{z}^F$ and $\mathbf{z}^C$ for instance *j8-per10-0* under scenario I.

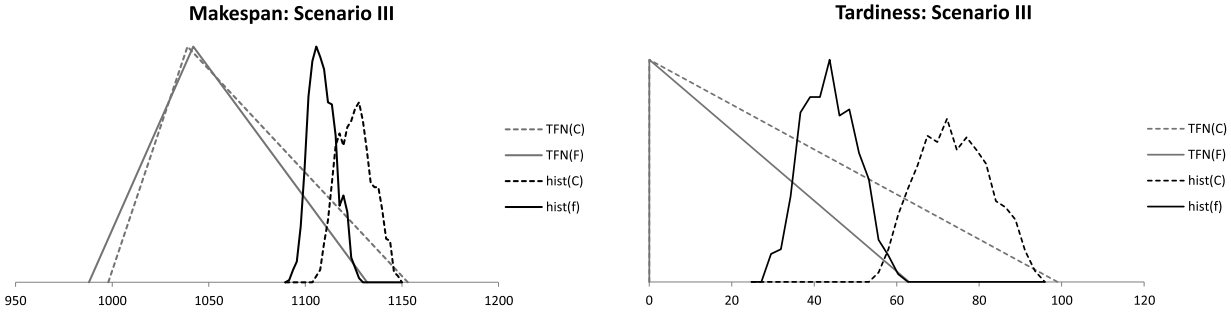


Fig. 10. $C_{\max}(\mathbf{z}^F, \mathbf{p})$, $C_{\max}(\mathbf{z}^C, \mathbf{p})$, $T_{\max}(\mathbf{z}^F, \mathbf{p})$, $T_{\max}(\mathbf{z}^C, \mathbf{p})$, and histograms of crisp $C_{\max}$ and $T_{\max}$ values obtained with $\mathbf{z}^F$ and $\mathbf{z}^C$ for instance *j8-per10-0* under scenario III.

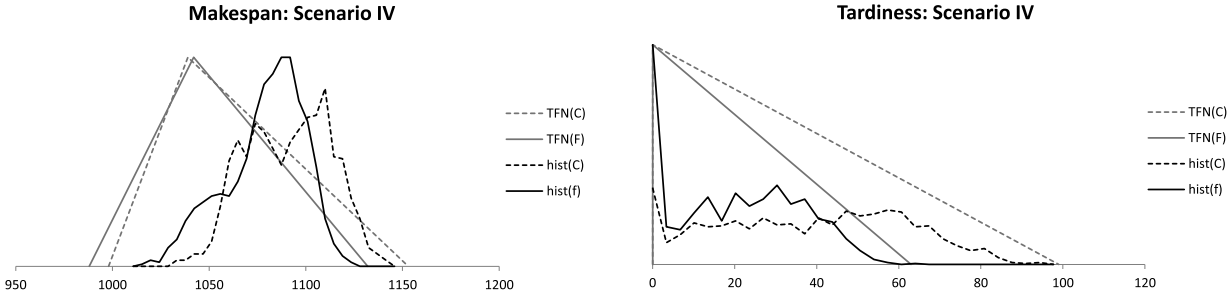


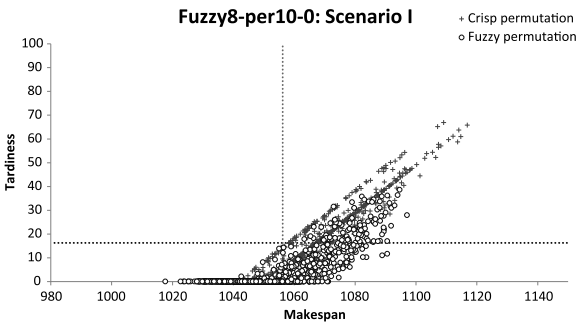
Fig. 11. $C_{\max}(\mathbf{z}^F, \mathbf{p})$, $C_{\max}(\mathbf{z}^C, \mathbf{p})$, $T_{\max}(\mathbf{z}^F, \mathbf{p})$, $T_{\max}(\mathbf{z}^C, \mathbf{p})$, and histograms of crisp $C_{\max}$ and $T_{\max}$ values obtained with $\mathbf{z}^F$ and $\mathbf{z}^C$ for *j8-per10-0* under scenario IV.

with a cross marking the centre of gravity of that cloud of points. Regarding both objective values, fuzzy permutations are overall clearly better than the crisp ones: for makespan crisp solutions are "on the right" of fuzzy ones and also for tardiness fuzzy solutions are "under" the crisp ones. Differences are greatest under scenario III, that is, when several delays occur for all tasks.

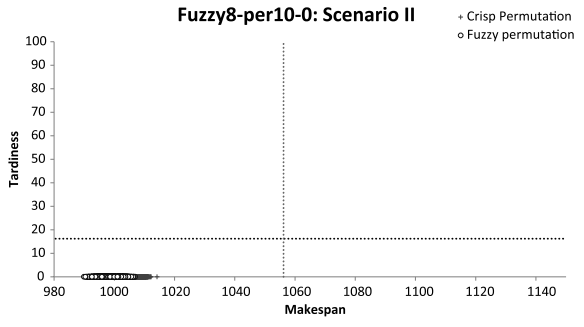
A summary of the above would be that the solutions obtained when the algorithm takes into account the uncertainty in task durations are more robust than those obtained considering only the most plausible duration for each task. When the real durations are close to the modal value (scenario I) both methods have a similar behaviour but when there are incidences and task durations increase (scenario III), then fuzzy solutions become much more reliable.

6. Conclusions and future work

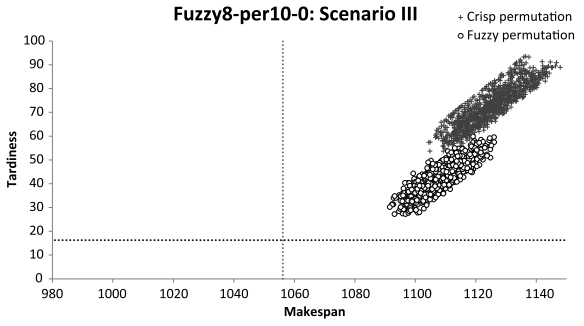
We have considered an open shop problem with uncertain durations modelled using TFNs and where the goal is to find a schedule optimising the fuzzy makespan and fuzzy maximum tardiness. We have pro-



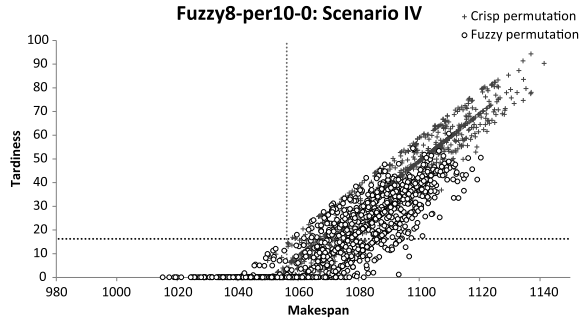
(a)



(b)



(c)



(d)

Fig. 12. Clouds of points with objective values for the scenario-samples generated for instance *J8-per-10-0*. (a) Scenario I. (b) Scenario II. (c) Scenario III. (d) Scenario IV.

posed to formulate the multiobjective problem as a lexicographical fuzzy goal programming model according to a generic priority structure and target levels established by the decision maker, using the expected value of the fuzzy quantities. As solving method, a PSO with codification based on priority arrays has been described. After experimentally tuning the algorithm's parameter values, we have presented experimental results on fuzzy versions of well-known crisp problem instances that illustrate the potential of both the proposed multiobjective formulation and the PSO. We have also seen how the lexicographical-approach solutions complement the solutions obtained with a Pareto-like approach, appropriate for the case when no hierarchy can be established for the objectives. Finally, we have studied with an analysis of possible a-posteriori behaviour of solutions the advantages of modelling the uncertainty and incorporating it to the solving process instead of using a deterministic algorithm on crisp problems considering only the modal values.

In the future, the multiobjective approach will be further analysed on a more varied set of problem instances, considering the influence of the target values as well as measuring the robustness of the obtained solutions. It would also be interesting to extend our PSO algorithm to automatically tune the parameters as suggested, for instance, in [40]. Finally, we would like to study different schedule generation schemes for the fuzzy OSP: their theoretical properties and their influence in the search metaheuristic's outcome.

Acknowledgements

We would like to thank the anonymous referees for their insightful and constructive comments. This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02 and MTM2010-16051.

References

[1] D. Alcaide, A. Rodriguez-Gonzalez and J. Sicilia, A heuristic approach to minimize expected makespan in open shops subject to stochastic processing times and failures, *International Journal of Flexible Manufacturing Systems* **17** (2006), 201–226.

[2] M. Andresen, H. Bräsel, M. Mörig, J. Tusch, F. Werner and P. Willenius, Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop, *Mathematical and Computer Modelling* **48** (2008), 1279–1293.

[3] S. Badaloni, M. Falda and M. Giacomin, Integrating quantitative and qualitative fuzzy temporal constraints, *AI Communications* **17**(4) (2004), 187–200.

[4] S. Bouveret and M. Lemaître, Computing leximin-optimal solutions in constraint networks, *Artificial Intelligence* **173** (2009), 343–364.

[5] P. Brucker, J. Hunrink, B. Jurisch and B. Wöstmann, A branch & bound algorithm for the open-shop problem, *Discrete Applied Mathematics* **76** (1997), 43–59.

[6] R. Caballero, T. Gómez and F. Ruiz, Goal programming: Realistic targets for the near future, *Journal of Multi-Criteria Decision Analysis* **16** (2009), 79–110.

[7] S.-M. Chen and 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.

[8] J.T. Coshall and R. Charlesworth, A management orientated approach to combination forecasting of tourism demand, *Tourism Management* **32** (2011), 759–769.

[9] L. Diaz-Balteiro and C. Romero, Making forestry decisions with multiple criteria: A review and an assessment, *Forest Ecology and Management* **255** (2008), 3222–3241.

[10] D. Dubois, H. Fargier and P. Fortemps, Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge, *European Journal of Operational Research* **147** (2003), 231–252.

[11] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*, Plenum Press, New York, USA, 1986.

[12] M. Ehrgott, *Multicriteria Optimization*, 2nd. edn, Springer, 2005.

[13] P. Fortemps, Jobshop scheduling with imprecise durations: A fuzzy approach, *IEEE Transactions of Fuzzy Systems* **7** (1997), 557–569.

[14] B. Giffler and G.L. Thompson, Algorithms for solving production scheduling problems, *Operations Research* **8** (1960), 487–503.

[15] J.F. Gonçalves, J.J. de Magalhães Mendes and M.G.C. Resende, A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research* **167** (2005), 77–95.

[16] I. González Rodríguez, J. Puente, C.R. Vela and 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.

[17] I. González-Rodríguez, C.R. Vela and J. Puente, A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty, *Journal of Intelligent Manufacturing* **21** (2010), 65–73.

[18] I. González-Rodríguez, J.J. Palacios, C.R. Vela and J. Puente, Heuristic local search for fuzzy open shop scheduling, in: *Proceedings IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2010*, IEEE, 2010, pp. 1858–1865.

[19] R. Graham, E. Lawler, J. Lenstra and A. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics* **4** (1979), 287–326.

[20] W. Herroelen, E. Demeulemeester and B. De Reyck, A note on the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by P. Brucker et al., *European Journal of Operational Research* **128** (2001), 679–688.

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

[22] X. Hu, Y. Shi and R. Eberhart, Recent advances in particle swarm, in: *Congress on Evolutionary Computation, CEC2004*, IEEE, 2004, pp. 90–97.

[23] R. Kalaï, C. Lamboray and D. Vanderpooten, Lexicographic α-robustness: An alternative to min-max criteria, *European Journal of Operational Research* **220** (2012), 722–728.

[24] A. Kasperski and M. Kule, Choosing robust solutions in discrete optimization problems with fuzzy costs, *Fuzzy Sets and Systems* **160** (2009), 667–682.

[25] J. Kennedy and R. Eberhart, Particle swarm optimization, in: *IEEE International Conference on Neural Networks*, IEEE Press, New Jersey, 1995, pp. 1942–1948.

[26] J.-q. Li and Y.-x. Pan, A hybrid discrete particle swarm optimization algorithm for solving fuzzy job shop scheduling problem, *International Journal of Advanced Manufacturing Technology* **66** (2013), 583–596.

[27] F. Liberatore, M.T. Ortuño, G. Tirado, B. Vitoriano and M.P. Scaparra, A hierarchical compromise model for the joint optimization of recovery operations and distribution of emergency goods in humanitarian logistics, *Computers & Operations Research* **42** (2014), 3–13.

[28] B. Liu and Y.K. Liu, Expected value of fuzzy variable and fuzzy expected value models, *IEEE Transactions on Fuzzy Systems* **10** (2002), 445–450.

[29] Q. Niu, B. Jiao and 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.

[30] S. Noori-Darvish, I. Mahdavi and N. Mahdavi-Amiri, A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due-dates, *Applied Soft Computing* **12** (2012), 1399–1416.

[31] J.J. Palacios, I. González-Rodríguez, C.R. Vela and J. Puente, Particle swarm optimisation for open shop problems with fuzzy durations, in: *Proceedings of IWINAC 2011*, Lecture Notes in Computer Science, Vol. 6686, Springer, 2011, pp. 362–371, Part I.

[32] S. Petrovic, S. Fayad and D. Petrovic, Sensitivity analysis of a fuzzy multiobjective scheduling problem, *International Journal of Production Research* **46**(12) (2008), 3327–3344.

[33] M.L. Pinedo, *Scheduling. Theory, Algorithms, and Systems*, 3rd edn, Springer, 2008.

[34] C. Romero, Extended lexicographic goal programming: A unifying approach, *Omega* **29** (2001), 63–71.

[35] D. Sha and H.-H. Lin, A multi-objective PSO for job-shop scheduling problems, *Expert Systems with Applications* **37**(2) (2010), 1065–1070.

[36] D. Sha, H.-H. Lin and C. Hsu, A modified particle swarm optimization for multi-objective open shop scheduling, in: *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Vol. 3, 2010.

[37] D.Y. Sha and H. Cheng-Yu, A new particle swarm optimization for the open shop scheduling problem, *Computers & Operations Research* **35** (2008), 3243–3261.

[38] E. Talbi, *Metaheuristics. From Design to Implementation*, Wiley, 2009.

[39] M. Tamiz, D. Jones and C. Romero, Goal programming for decision making: An overview of the current state-of-the-art, *European Journal of Operations Research* **111** (1998), 569–581.

[40] G.S. Tewolde, D.M. Hanna and R.E. Haskell, Enhancing performance of PSO with automated parameter tuning technique, in: *IEEE Swarm Intelligence Symposium, SIS'09*, 2009, pp. 67–73.

[41] V. T'kindt and J.-C. Billaut, *Multicriteria Scheduling. Theory, Models and Algorithms*, 2nd edn, Springer, 2006.

[42] I.C. Trelea, The particle swarm optimization algorithm: Convergence analysis and parameter selection, *Information Processing Letters* **85** (2003), 317–325.

[43] J. Wang, A fuzzy robust scheduling approach for product development projects, *European Journal of Operational Research* **152** (2004), 180–194.

7.4 Robust swarm optimisation for fuzzy open shop scheduling

In this section, we include the following publication.

- **Title:** Robust swarm optimisation for fuzzy open shop scheduling.
- **Journal:** Natural Computing.
- **Year:** 2014.
- Impact Factor (JCR 2013): 0.539
- Impact Factor (5-year): Not available
- Journal Ranking:
  - Computer Science, Theory and Methods: 74/102 **Q3 (T3)**
  - Computer Science, Artificial Intelligence: 97/121 **Q4 (T3)**

This publications contains pieces of work described in Sections 4.2.1 and 4.3.2.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Robust swarm optimisation for fuzzy open shop scheduling. Natural Computing 13(2), 145-156 (2014). Doi: 10.1007/s11047-014-9413-1

Nat Comput (2014) 13:145–156
DOI 10.1007/s11047-014-9413-1

Robust swarm optimisation for fuzzy open shop scheduling

Juan José Palacios · Inés González-Rodríguez · Camino R. Vela · Jorge Puente

Published online: 31 January 2014

Abstract In this paper we consider a variant of the open shop problem where task durations are allowed to be uncertain and where uncertainty is modelled using fuzzy numbers. Solutions to this problem are fuzzy schedules, which we argue should be seen as predictive schedules, thus establishing links with the concept of robustness and a measure thereof. We propose a particle swarm optimization (PSO) approach to minimise the schedule's expected makespan, using priorities to represent particle position, as well as a decoding algorithm to generate schedules in a subset of possibly active ones. Our proposal is evaluated on a varied set of several benchmark problems. The experimental study includes a parametric analysis, results of the PSO compared with the state-of-the-art, and an empirical study of the robustness of taking into account uncertainty along the scheduling process.

Electronic supplementary material The online version of this article (doi:10.1007/s11047-014-9413-1) contains supplementary material, which is available to authorized users.

J. J. Palacios · C. R. Vela · J. Puente
Department of Computer Science, University of Oviedo, Oviedo, Spain
e-mail: palaciosjuan@uniovi.es
URL: http://di002.edv.uniovi.es/iscop

C. R. Vela
e-mail: crvela@uniovi.es
URL: http://di002.edv.uniovi.es/iscop

J. Puente
e-mail: puente@uniovi.es
URL: http://di002.edv.uniovi.es/iscop

I. González-Rodríguez (✉)
Department of Mathematics, Statistics and Computing, University of Cantabria, Santander, Spain
e-mail: gonzalezri@unican.es

Keywords Open shop scheduling · Fuzzy durations · Particle swarm optimisation · Robustness

1 Introduction

The open shop scheduling problem is a problem with an increasing presence in the scheduling literature and with clear applications in industry—consider for instance testing facilities, where units go through a series of diagnostic tests that need not be performed in a specified order and where different testing equipment is usually required for each test [27]. For a number of machines $m \geq 3$, this problem is *NP*-complete; in consequence, it is usually tackled via metaheuristic techniques. For makespan minimisation, in [15] two heuristic methods to obtain a list of operation priorities are described and later used in a list-scheduling algorithm; [23] proposes a tabu search algorithm; ant colony optimisation is hybridised with beam search in [3]; [30] proposes a solution based on particle swarm optimisation; ant bee colony optimisation is used in [18] and a hybrid genetic algorithm is proposed in [1]. These last three algorithms conform the state-of-the art for open shop with makespan minimisation.

To enhance the range of applications of scheduling, part of the research is devoted to incorporating the uncertainty and vagueness pervading real-world situations. Part of this uncertainty translates into variability of input data which can be somehow modelled and anticipated, leading to proactive or robust scheduling [16]. The approaches are diverse and, among these, fuzzy sets have been used in a wide variety of ways [5, 6]. Far from being trivial, incorporating uncertainty and extending heuristic strategies to the resulting setting usually requires a significant reformulation of both the problem and solving methods. Some

heuristic methods have so far been proposed for fuzzy flow and job shop problems, where uncertain durations are modelled via fuzzy sets; among others, in the last years we find genetic algorithms in [12, 22], a fuzzy-neural approach in [32], a memetic algorithm combining evolution and local search in [28], swarm-based neighbourhood search in [33] or differential evolution in [17]. However, to the best of our knowledge, the open shop problem has received little attention in the fuzzy framework: in [21] fuzzy sets are used to represent flexible job start and due dates, a possibilistic mixed-integer linear programming method is proposed in [25] for a multiobjective open shop with setup times, fuzzy processing times and fuzzy due dates and in [26] a genetic algorithm is proposed to solve the open shop with fuzzy durations, and in [11] this genetic algorithm is combined with a local search method.

In the following, we consider the fuzzy open shop problem with expected makespan minimisation, denoted $O|fuzzp_i|E[C_{max}]$ and propose a particle swarm technique to solve it. The rest of the paper is organized as follows. In Sect. 2 we formulate the problem and associated concepts and introduce a measure of robustness. Then, in Sect. 3, we describe the main components of the particle swarm optimisation (PSO) algorithm proposed to solve the problem. Section 4 includes a parametric analysis of the PSO, experimental results to evaluate the competitiveness of our proposal and an additional analysis of the usefulness of scheduling with fuzzy durations in order to improve solution robustness. Finally, in Sect. 5 we summarise the main conclusions and propose ideas for future work.

2 Open shop scheduling with uncertain durations

The *open shop scheduling problem*, or *OSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job consists of m tasks or operations, each requiring the exclusive use of a different machine for its whole processing time without preemption, i.e. all operations must be processed without interruption. In total, there are $n \times m$ tasks (nm for short), denoted $\{o_{ij}, 1 \leq i \leq n, 1 \leq j \leq m\}$. A solution to this problem is a *schedule*—an allocation of starting times for all tasks—which is *feasible*, in the sense that all constraints hold, and is also optimal according to some criterion. Here, the objective will be minimising the makespan C_{max}, that is, the time lag from the start of the first task until the end of the last one, a problem often denoted $O||C_{max}$ in the literature [14].

2.1 Uncertain durations

In real-life applications, it is often the case that it is not known in advance the exact time it will take to process one operation and only some uncertain knowledge is available, for instance, an interval of possible durations, or a most likely duration with a certain error margin. Such knowledge can be modelled using a *triangular fuzzy number* or TFN, given by an interval $[n^1, n^3]$ of possible values and a modal value n^2 in it [7]. For a TFN N, denoted $N = (n^1, n^2, n^3)$, the membership function takes the following triangular shape:

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

In the open shop, we essentially need two operations on processing times (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, we follow [8] and approximate the results of these operations, evaluating the operation only on the three defining points of each TFN. It turns out that for any pair of TFNs M and N, the approximated sum $M + N \approx (m^1 + n^1, m^2 + n^2, m^3 + n^3)$ coincides with the actual sum of TFNs; this is not necessarily so for the maximum $\max\{M, N\} \approx (\max\{m^1, n^1\}, \max\{m^2, n^2\}, \max\{m^3, n^3\})$, although they have identical support and modal value.

The membership function of a fuzzy number can be interpreted as a possibility distribution on the real numbers. This allows to define its expected value [24], given for a TFN N by $E[N] = \frac{1}{4}(n^1 + 2n^2 + n^3)$. It coincides with the neutral scalar substitute of a fuzzy interval and the centre of gravity of its mean value [6]. It induces a total ordering $\leq_E$ in the set of fuzzy intervals [8], where for any two fuzzy intervals M, N $M \leq_E N$ if and only if $E[M] \leq E[N]$.

2.2 Fuzzy open shop scheduling

If processing times of operations are uncertain and such uncertainty is modelled using TFNs, the resulting schedule is fuzzy in the sense that starting and completion times for each operation and hence the makespan are TFNs, where each TFN can be seen as a possibility distribution on the actual values that the corresponding time may take. However, there is no uncertainty regarding the order in which operations must be processed.

Indeed, a schedule for an open shop problem of size $n \times m$ (n jobs and m machines) may be determined by a priority vector $\pi = (\pi_1, \ldots, \pi_{nm})$ representing a task processing order, where $\forall k, l = 1, \ldots, nm$ $1 \leq \pi_l \leq nm$ and, if $k \neq l$, then $\pi_k \neq \pi_l$, that is, π is a permutation of the set of tasks where each task o_{ij} may be represented by the number

$(i-1)m+j$. The task processing order represented by the priority vector uniquely determines a feasible schedule; it should be understood as expressing partial orderings for every set of tasks requiring the same machine and for every set of tasks requiring the same job.

Let us assume that the processing time p_{ij} of each task $o_{ij}, i = 1,\ldots,n,\ j = 1,\ldots,m$ is a TFN, so the problem may be represented by a matrix of fuzzy processing times $\boldsymbol{p}$ of size $n \times m$. For a given priority vector π and a task o_{ij}, its starting time $S_{ij}(\pi,\boldsymbol{p})$ is the maximum between the completion times of the task preceding o_{ij} in its job according to π, let it be denoted o_{ik}, and the task preceding o_{ij} in its machine according to π, let it be denoted o_{lj}:

$$S_{ij}(\pi,\boldsymbol{p}) = \max(\boldsymbol{C_{ik}}(\boldsymbol{\pi},\boldsymbol{p}), \boldsymbol{C_{lj}}(\boldsymbol{\pi},\boldsymbol{p})) \tag{2}$$

where $C_{ik}(\pi,\boldsymbol{p})$ or $C_{lj}(\pi,\boldsymbol{p})$ are taken to be zero if o_{ij} is the first task to be processed either in its job or its machine. Then, its completion time $C_{ij}(\pi,\boldsymbol{p})$ is obtained by adding its duration p_{ij} to $S_{ij}(\pi,\boldsymbol{p})$:

$$C_{ij}(\pi,\boldsymbol{p}) = \boldsymbol{S_{ij}}(\boldsymbol{\pi},\boldsymbol{p}) + \boldsymbol{p_{ij}} \tag{3}$$

The completion time of a job J_i will then be the maximum completion time of all its tasks, that is,

$$C_i(\pi,\boldsymbol{p}) = \max_{\boldsymbol{1 \le j \le m}} \{\boldsymbol{C_{ij}}(\boldsymbol{\pi},\boldsymbol{p})\} \tag{4}$$

so the *fuzzy makespan* $C_{max}(\pi,\boldsymbol{p})$ will be given by the following:

$$\boldsymbol{C_{max}}(\boldsymbol{\pi},\boldsymbol{p}) = \max_{\boldsymbol{1 \le i \le n}} (\boldsymbol{C_i}(\boldsymbol{\pi},\boldsymbol{p})) \tag{5}$$

In the case where no confusion is possible, we may drop the priority vector π and the processing times matrix $\boldsymbol{p}$ and simply write C_{max}.

An important issue with fuzzy times is to decide on the meaning of "optimal makespan". It is not trivial to optimise a fuzzy makespan, since neither the maximum nor its approximation define a total ordering in the set of TFNs. Using ideas similar to stochastic scheduling, we follow the approach taken for the fuzzy job shop in [13]. Given the total ordering provided by the expected value, we consider that the objective is to minimise the expected makespan $E[C_{max}]$. The resulting problem may be denoted $O|fuzz\ p_i|E[C_{max}]$ using the three-field notation [14].

Let us illustrate the previous definitions with an example. Consider a problem of three jobs and two machines with the following matrix for fuzzy processing times:

$$\boldsymbol{p} = \begin{pmatrix} (\boldsymbol{3},\boldsymbol{4},\boldsymbol{7}) & (\boldsymbol{3},\boldsymbol{4},\boldsymbol{7}) \\ (\boldsymbol{2},\boldsymbol{3},\boldsymbol{3}) & (\boldsymbol{4},\boldsymbol{5},\boldsymbol{6}) \\ (\boldsymbol{3},\boldsymbol{4},\boldsymbol{6}) & (\boldsymbol{1},\boldsymbol{2},\boldsymbol{4}) \end{pmatrix}$$

Here $p_{21} = (2, 3, 3)$ is the processing time of task o_{21}, the task of job J_2 to be processed in machine M_1. Figure 1 shows the Gantt chart adapted to TFNs of the schedule given by the priority vector $\pi = (1, 4, 6, 3, 5, 2)$. It represents the partial schedules on each machine obtained from this decision variable. Tasks must be processed in the following order: o_{11}, o_{22}, o_{32}, o_{21}, o_{31}, o_{12}. Given this ordering, the starting time for task o_{21} will be the maximum of the completion times of o_{22} and o_{11}, which are respectively the preceding tasks in the job and in the machine:

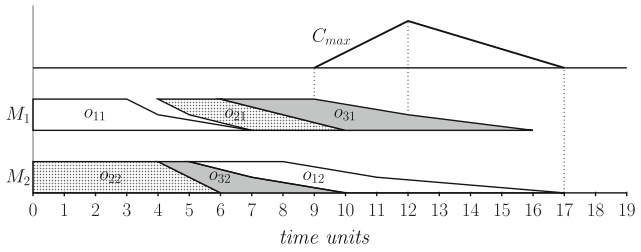


Fig. 1 Gantt chart of the schedule represented by the priority vector (1, 4, 6, 3, 5, 2)

$$S_{21} = \max(C_{22}, C_{11}) = \max((4,5,6),(3,4,7)) = (4,5,7).$$

Consequently, its completion time will be

$$C_{21} = S_{21} + p_{21} = (4,5,7) + (2,3,3) = (6,8,10).$$

Also, it is easy to see that the makespan is $C_{max} = (9, 12, 17)$, so $E[C_{max}] = 12.5$.

2.3 Robust schedules

A fuzzy schedule does not provide exact starting times for each task. Instead, it gives a fuzzy interval of possible values for each starting time, provided that tasks are executed in the order determined by the schedule. In fact, it is impossible to predict what the exact time-schedule will be, because it depends on the realisation of the tasks durations, which is not known yet. This idea is the basis for a semantics for fuzzy schedules from [12] by which solutions to the fuzzy open shop should be understood as a-priori solutions, also called baseline or predictive schedules in the literature [16]. These solutions are found when the duration of tasks is not exactly known and a set of possible scenarios must be taken into account. When tasks are executed according to the ordering provided by the fuzzy schedule we shall know their real duration and, hence, obtain a real (executed) schedule, the a-posteriori solution with deterministic times.

Clearly, fuzzy solution should yield reasonably good executed schedules in the moment of its practical use. Also, the estimates for starting and completion times and, in particular, for the makespan, should be reasonably accurate for each possible scenario of task durations. This leads us to the concept of solution robustness. As [19] puts it, "Intuitively, a solution can be considered as robust if it

behaves "well" or "not too bad" in all the scenarios." This is the idea underlying a definition of ϵ-robustness given in [2] for stochastic scheduling which can be adapted to the fuzzy open shop as follows.

A predictive schedule is considered to be *robust* if the quality of the eventually executed schedule is close to the quality of the predictive schedule. In particular, a predictive schedule with objective value f^{pred} is ϵ-*robust* for a given ϵ if the objective value f_{exec} of the eventually executed schedule is such that:

$$(1-\epsilon) \leq \frac{f^{exec}}{f^{pred}} \leq (1+\epsilon) \tag{6}$$

or, equivalently,

$$\frac{|f^{exec} - f^{pred}|}{f^{pred}} \leq \epsilon \tag{7}$$

That is, the relative error of the estimation made by the predictive schedule is bounded by ϵ. Obviously, the smaller ϵ is, the better.

3 Particle swarm optimization for the FOSP

Given the complexity of the open shop, different metaheuristic techniques have been proposed to solve the general *m*-machine problem. In particular, a method based on particle swarm optimisation has been proposed in [30] which is considered the state-of-the-art for crisp open shop.

Require: A FOSP instance
Ensure: A schedule for the input instance
1. generate and evaluate the initial swarm.
2. compute gbest and pbest for each particle.
while no termination criterion is satisfied **do**
  for each particle k **do**
    for each dimension d **do**
      3. update velocity v_d^k.
      4. update position x_d^k.
    5. evaluate particle k.
    6. update pbest and gbest values.
return The schedule from the best particle evaluated so far;

Alg. 1: A generic PSO algorithm

Particle swarm optimisation (PSO) is a population-based stochastic optimisation technique inspired by bird flocking or fish schooling [20]. In PSO, each position in the search space corresponds to a solution of the problem and particles in the swarm cooperate to find the best position (hence best solution) in the space. Particle movement is mainly affected by the three following factors:

- Inertia: Velocity of the particle in the latest iteration.
- *pbest*: The best position found by the particle.
- *gbest*: The best position found by the swarm so far (the best *pbest*).

The potential solutions or particles fly through the problem space changing their position and velocity by following the current optimum particles *pbest* and *gbest*.

Algorithm 1 describes the structure of a generic PSO algorithm. First, the initial swarm is generated and evaluated. Then the swarm evolves until a termination criterion is satisfied and in each iteration, a new swarm is built from the previous one by changing the position and velocity of each particle following its *pbest* and *gbest* locations.

Following this general structure, we now extend the successful algorithm from [30] to the fuzzy framework.

3.1 Position representation and evaluation

We use a priority-based representation for particle positions. Thus a schedule is encoded as a priority matrix $X^k = (x_{ij}^k)_{i=1\ldots n, j=1\ldots m}$, where x_{ij}^k denotes the priority of operation o_{ij}, the task of job i processed on machine j. An operation with smaller x_{ij}^k has a higher priority to be scheduled.

If we represent a FOSP solution as a task processing order π, which is a permutation of tasks, we can transfer this permutation to a priority matrix and viceversa. For instance, given the following solution for a problem of size 3×3:

$$\pi = (\, o_{11} \quad o_{13} \quad o_{23} \quad o_{12} \quad o_{31} \quad o_{33} \quad o_{21} \quad o_{32} \quad o_{22} \,)$$

a particle in the space can be obtained by randomly setting x_{ij} in the interval $(p-0.5, p+0.5)$ where p is the location of o_{ij} in π. Therefore, the operation with smaller x_{ij} has higher priority to be scheduled. The above permutation list can be transferred to:

$$X^k = \begin{pmatrix} 1.2 & 4.0 & 1.7 \\ 6.6 & 9.4 & 2.7 \\ 5.3 & 7.9 & 6.4 \end{pmatrix}$$

Decoding of a particle may be done in different ways. For the crisp job shop and by extension for the open shop, it is common to use the G&T algorithm [9], which is an active schedule builder. A schedule is *active* if one task must be delayed for any other one to start earlier. Active schedules are good in average and, most importantly, the space of active schedules contains at least an optimal one, that is, the set of active schedules is *dominant*. For these reasons it is worth to restrict the search to this space. In [10] a narrowing mechanism was incorporated to the G&T algorithm in order to limit machine idle times by means of a delay parameter $\delta \in [0,1]$, thus searching over the space of so-called parameterised active schedules. In the deterministic case, for $\delta < 1$ the search space is reduced so it

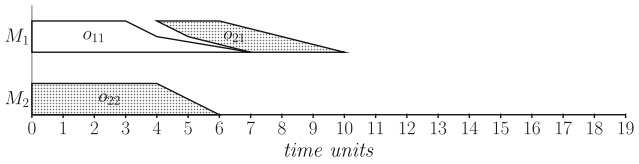


Fig. 2 Gantt chart of the partial schedule for the example in Sect. 2.2 where only o_{11}, o_{22} and o_{21} have been scheduled

may no longer contain optimal schedules and, at the extreme $\delta = 0$ the search is constrained to *non-delay* schedules, where a resource is never idle if a requiring operation is available. This variant of G&T has been applied in [30] to the deterministic OSP, under the name "parameterized active schedule generation algorithm".

In Algorithm 2 we propose an extension of parameterised G&T to the case of fuzzy processing times, denoted *pfG&T*. It should be noted that, due to the uncertainty in task durations, even for $\delta = 1$, we cannot guarantee that the produced schedule will indeed be active when it is actually performed (and tasks have exact durations). We may only say that the obtained fuzzy schedule is *possibly active*. Throughout the algorithm, Ω denotes the set of the operations that have not been yet scheduled, X^k the priority matrix, S_{ij} the starting time of the operation o_{ij} and C_{ij} the completion time of the operation o_{ij}.

Let us illustrate the decoding algorithm with an example. Consider the problem proposed in Sect. 2.2 to illustrate the concept of fuzzy schedule and the following priority matrix for it:

$$X^k = \begin{pmatrix} 1.2 & 5.3 \\ 2.7 & 1.7 \\ 4.0 & 6.4 \end{pmatrix}$$

Figure 2 shows the Gantt chart of the partial schedule in which the operations o_{11}, o_{22} and o_{21} have been already scheduled. In this situation, $\Omega = \{o_{12}, o_{31}, o_{32}\}$. Table 1 depicts the values of the starting and completion times of the operations in Ω in this iteration of the algorithm as well as its expected values. The s^* and c^* values are shown in bold. Considering $\delta = 1$, the conflict set is $O = \{o_{12}, o_{32}\}$. Operation o_{31} is not contained in the O set, although it has the highest priority in Ω, because the possibility that operation o_{32} can be completed before the earliest beginning of o_{31} is 1, so by selecting o_{31} before o_{31} we would generate a non possibly active schedule. Additionally, reducing the δ value to 0.1, the set of operations that are candidates to be scheduled is further restricted to $O = \{o_{32}\}$ even though o_{32} is the lowest-priority operation of Ω.

Table 1 Partial schedule values

Oper.	S_{ij}	$E[S_{ij}]$	C_{ij}	$E[C_{ij}]$
o_{12}	(4,5,7)	5.25	(7,9,14)	9.75
o_{31}	(6,8,10)	8.00	(9,12,16)	12.25
o_{32}	(4,5,6)	**5.00**	(5,7,10)	**7.25**

Notice that the *pfG&T* algorithm only uses the priority vector to break ties among tasks in the conflict set, so the task processing order in the resulting schedule may differ from that in the particle. Given that the essence of a particle is the task ordering it represents, *gbest* and *pbest* do not record the actual best positions found so far, but rather the best operation sequences of the schedules generated by the decoding operator.

Require: A FOSP instance and a particle position X^k
Ensure: A schedule for the input instance considering the priorities given by X^k
$\Omega \leftarrow \{o_{ij} : 1 \le i \le n, 1 \le j \le m\}$;
while $\Omega \neq \emptyset$ **do**
  $c^* \leftarrow \min_{o_{ij} \in \Omega}\{E[C_{ij}]\}$;
  $s^* \leftarrow \min_{o_{ij} \in \Omega}\{E[S_{ij}]\}$;
  $O \leftarrow \{o_{ij} : E[S_{ij}] < s^* + \delta \times (c^* - s^*), o_{ij} \in \Omega\}$;
  Choose the operation o^*_{ij} from O with smallest x^k_{ij};
  Schedule the operation o^*_{ij};
  $\Omega \leftarrow \Omega - \{o^*_{ij}\}$;
return The schedule given by $\{S_{ij} : 1 \le i \le n, 1 \le j \le m\}$

Alg. 2: The $pfG\&T$ algorithm

3.2 Particle movement and velocity

Particle movement depends not only on its position, but also on its velocity. For any particle, its velocity is represented by an array of the same length as the position array where all the values are in the set $\{-1, 0, 1\}$. Initially, the values in the array are set at random. Afterwards, particle position and velocity are updated depending on *gbest* and *pbest*. Traditionally, this updating depends on distance values. Instead, this PSO considers whether the position value x^k_{ij} is larger or smaller than $pbest^k_{ij}$ ($gbest_{ij}$). Updating is controlled at the beginning of each iteration by the inertia weight w as well as two other constants $0 \le C_1, C_2$ such that $C_1 + C_2 \le 1$, representing the probability that the updating is guided either by *pbest* or by *gbest*. Further detail on the updating process can be found in Algorithm 3.

Require: A particle position X^k and velocity V^k, best particle and swarm positions $pbest^k$ and $gbest$, inertia w and updating probabilities C_1, C_2
Ensure: The updated particle position X^k and velocity V^k
for each dimension d **do**
  generate random value $rand \sim U(0,1)$.
  if $v_d^k \neq 0$ and $rand \geq w$ **then**
    $v_d^k \leftarrow 0$.
  if $v_d^k = 0$ **then**
    generate random value $rand \sim U(0,1)$.
    if $rand \leq C_1$ **then**
      if $pbest_d^k \geq x_d^k$ **then**
        $v_d^k \leftarrow 1$.
      else
        $v_d^k \leftarrow -1$.
      generate random value $rand_2 \sim U(0,1)$.
      $x_d^k \leftarrow pbest_d^k + rand_2 - 0.5$.
    if $C_1 < rand \leq C_1 + C_2$ **then**
      if $gbest_d \geq x_d^k$ **then**
        $v_d^k \leftarrow 1$.
      else
        $v_d^k \leftarrow -1$.
      generate random value $rand_2 \sim U(0,1)$.
      $x_d^k \leftarrow gbest_d + rand_2 - 0.5$.
  else
    $x_d^k \leftarrow x_d^k + v_d^k$.
return The updated particle position X^k and velocity V^k;

Alg. 3: Particle movement

3.2.1 *Position mutation*

In order to introduce diversity, after a particle moves to a new position, we mutate it with probability p_M by choosing an operation and then randomly changing its priority value x_d^k independently of v_d^k. As in [30], for a problem of size $n \times m$, if $x_d^k < (nm/2)$, x_d^k will take a random value in $[mn - n, mn]$, and $v_d^k = 1$; else, if $x_d^k > (nm/2)$, x_d^k will take a random value in [0,n] and $v_d^k = -1$.

3.2.2 *Diversification strategy*

If all particles have the same *pbest* solutions, they will be trapped into local optima. To prevent such situation, a diversification strategy is adopted that keeps the *pbest* solutions different. In this strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far, where N is the size of the swarm. Once any particle generates a new solution, the *pbest* solutions will be updated in certain cases as follows:

- if the makespan of the particle solution equals that of any *pbest* solution, replace that *pbest* solution with the new particle solution;
- if the makespan of the particle solution improves the worst *pbest* solution and is different from all *pbest* solutions, set the worst *pbest* solution to be the particle solution.

4 Experimental evaluation

We now proceed to empirically evaluate the proposed method in several steps, using a total of 520 problem instances. First, a parametric analysis will be conducted to decide on a good parameter-configuration for the PSO search process as well as for the schedule generation algorithm *pG&T*. Then, we present results of expected makespan minimisation obtained by the PSO and we compare them with the best results obtained so far in the literature by a memetic algorithm. Finally, we shall present some results to illustrate the benefits in terms of robustness of using fuzzy numbers along the scheduling process, instead of the more straightforward approach of scheduling a crisp problem that results from defuzzification.

4.1 Experiment setting

For the experimental study we use the test bed given in [11], where the authors follow [8] and generate a set of fuzzy problem instances from well-known open shop benchmark problems [4]. Given a crisp problem instance, each crisp processing time t is transformed into a symmetric fuzzy processing time $p(t)$ such that its modal value is $p^2 = t$ and p^1, p^3 are random values, symmetric w.r.t. p^2 and generated so the TFN's maximum range of fuzziness is 30 % of p^2. The original problem instances consist of six families, denoted $J3, J4, \ldots, J8$, of sizes 3×3 to 8×8, containing 8 or 9 instances each. Ten fuzzy versions of each crisp problem instance were generated, so in total there are 520 problem instances. The obtained benchmark instances for the fuzzy open shop are available at http://www.di.uniovi.es/iscop.

If a lower bound for the expected makespan were known, the algorithm's performance may be evaluated by measuring the distance between the obtained expected makespan and the lower bound. Thanks to the symmetry in the TFNs, the optimal solution (if known) to the crisp problem provides a lower bound, denoted LB_c, for the expected makespan of the fuzzified version [8]. An alternative lower bound LB_f can be obtained directly from the fuzzy instance as follows:

$$LB_f = E\left[\max\left\{\max_j\left\{\sum_{i=1}^{n} p_{ij}\right\}, \max_i\left\{\sum_{j=1}^{m} p_{ij}\right\}\right\}\right] \quad (8)$$

This lower bound adapts the lower bound proposed in [31] for crisp problems to the fuzzy setting. The maximum of both quantities yield a tighter lower bound for the expected makespan of the optimal solution:

$$LB = \begin{cases} \max(LB_f, LB_c) & \text{if } LB_c \text{ is known} \\ LB_f & \text{otherwise} \end{cases} \quad (9)$$

We can now compute the makespan relative error with respect to LB as follows:

$$RE = \frac{E[C_{max}] - LB}{LB} \quad (10)$$

This relative error will be the basis for evaluating the obtained results in the remaining of this section.

All the experiments reported in this section, correspond to a C++ implementation running on a PC with Xeon E5520 processor and 24 GB RAM running Linux (SL 6.0).

4.2 Parametric analysis

For the PSO we take as initial parameter configuration the values proposed after a parameter analysis for the crisp OSP in [30]: swarm size $N = 60$, $C_1 = 0.7$, $C_2 = 0.1$, mutation probability $P_M = 1$ and inertia weight w linearly decreasing from 0.9 to 0.3. Regarding the filtering mechanism of the search space given in the schedule generator *pfG&T*, an initial value of $\delta = 0.25$ is adopted. This has been done after some preliminary experiments consisting in generating random solutions in the search space with varying values of δ and adopting the value with better solutions in average.

Starting with this initial configuration, we proceed to perform a parametric analysis for both the PSO algorithm and the decoding scheme. First, we try different values for the PSO algorithm's parameters (in bold we highlight the values of the starting configuration) as follows:

- Stopping criterion: A maximum number of iterations *MaxIter* less or equal than **5,000** iterations.
- Guiding constants C_1 and C_2: all possible pairs of the values in {**0.1**, 0.3, 0.5, **0.7**, 0.9}, provided that they add up to a maximum of 1.
- Inertia: Linearly decreasing from ω_s to ω_e, with $\omega_s \in \{0.5, 0.7, \mathbf{0.9}\}$ and $\omega_e \in \{0.1, \mathbf{0.3}, 0.5\}$.
- Mutation probability: Values in {0, 0.25, 0.50, 0.75, **1**}.
- Swarm Size N: Values in {**60**, 80, 100}.

We follow an incremental process in which we test one of the parameters until it is optimised, then we fix it and proceed in the same way with the next one, until all the parameters are fixed.

Table 2 Final value for the stopping criterion *MaxIter* depending on problem size

Size	*MaxIter*
3 × 3	100
4 × 4	100
5 × 5	750
6 × 6	1500
7 × 7	2100
8 × 8	2700

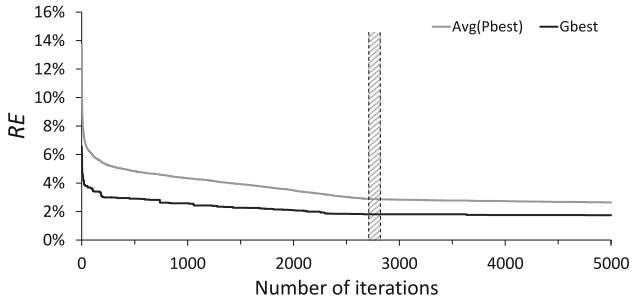


Fig. 3 Evolution along 5,000 iterations of $E[C_{max}]$ for *gbest* (in *black*) and the average $E[C_{max}]$ for *pbest* (in *grey*) for *j8-per10-1* instance; convergence is obtained at iteration 2700

4.2.1 Stopping criterion MaxIter

First, for each problem size we estimate the number of iterations *MaxIter* needed by the algorithm to converge. The procedure to obtain this number of iterations is as follows: For an arbitrary fuzzy instance of each original crisp problem, the algorithm is run 10 times for 5,000 iterations. At each iteration, we record the average makespan of all *pbest* elements in the swarm and calculate its relative error w.r.t. the lower bound LB. Then, we pick the first iteration for which this error will decrease less than 1 % in the next 100 iterations. This provides us with an stopping iteration for each problem. Putting together all problems of the same size, we select the number of iterations in the third quartile as *MaxIter* for the group of problem instances of the same size.The resulting values for *MaxIter* depending on problem size can be seen in Table 2. Additionally, Figure 3 shows the evolution of the expected makespan for *gbest* and the average expected makespan for *pbest* particles along 5,000 iterations of the PSO for problem instance $J8 - per10 - 1$; it highlights the first interval of 100 iterations where the error improvement is less than 1 %.

4.2.2 Guiding constants C_1 and C_2

To decide on the values of the remaining parameters, at each step, the PSO is run 10 times on a random fuzzy

Table 3 Algorithm's performance with varying guiding constants C_1 and C_2

Values	*RE* (%)
$C_1 = 0.1$, $C_2 = 0.1$	2.96
$C_1 = 0.1$, $C_2 = 0.3$	2.83
$C_1 = 0.1$, $C_2 = 0.5$	2.81
$C_1 = 0.1$, $C_2 = 0.7$	2.79
$C_1 = 0.1$, $C_2 = 0.9$	2.92
$C_1 = 0.3$, $C_2 = 0.1$	2.63
$C_1 = 0.3$, $C_2 = 0.3$	2.74
$C_1 = 0.3$, $C_2 = 0.5$	2.77
$C_1 = 0.3$, $C_2 = 0.7$	2.80
$C_1 = 0.5$, $C_2 = 0.1$	2.64
$C_1 = 0.5$, $C_2 = 0.3$	2.64
$C_1 = 0.5$, $C_2 = 0.5$	2.74
$\boldsymbol{C_1 = 0.7, C_2 = 0.1}$	**2.58**
$C_1 = 0.7$, $C_2 = 0.3$	2.65
$\boldsymbol{C_1 = 0.9, C_2 = 0.1}$	**2.56**

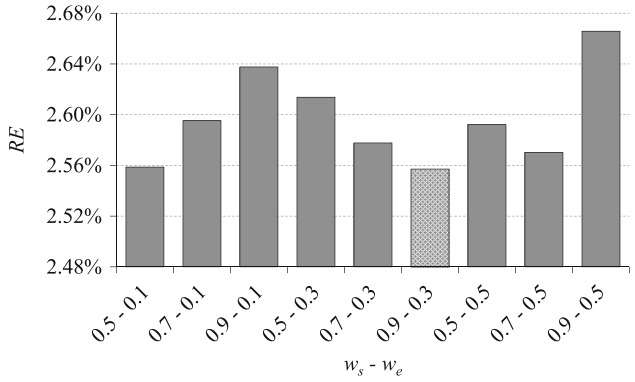


Fig. 4 Algorithm's performance with varying inertia weight from w_s to w_e

instance from each 8 × 8 original problem and the quality of the configuration used is measured using *RE*, the relative error w.r.t. the problem's lower bound *LB*, averaged across the 10 runs. First, we test the guiding probabilities C_1 and C_2. Table 3 shows the average across the 8 × 8 problems of the relative error *RE*. Two configurations (in bold in the table) perform clearly better than the rest: (C_1, C_2) = (0.7, 0.1) and (C_1, C_2) = (0.9, 0.1). Since the latter yields slightly better results, we take it to be the definite one.

4.3 Inertia weight bounds w_s and w_e

Regarding the inertia parameter, Fig. 4 shows the average *RE* obtained using different inertia values linearly decreasing in the corresponding intervals [w_s, w_e] on the *X*-axis. As above there are two configurations ($w_s = 0.9$ to $w_e = 0.3$ and $w_s = 0.5$ to $w_e = 0.1$) behaving similarly, with slightly better results for the inertia going from 0.9 to 0.3. However, since there are two configurations which are similar in terms of quality both for the guiding constants and the inertia, we have additionally tested all the combinations of those values. Table 4 shows the obtained results, which support taking $C_1 = 0.9$, $C_2 = 0.1$ and w linearly decreasing in [0.9, 0.3].

Table 4 Algorithm's performance depending on different combinations of guiding constants and inertia weights

Guiding	Inertia weights w	
Constants	0.5 → 0.1	0.9 → 0.3
$C_1 = 0.7$, $C_2 = 0.1$	2.590 %	2.580 %
$C_1 = 0.9$, $C_2 = 0.1$	2.559 %	**2.557** %

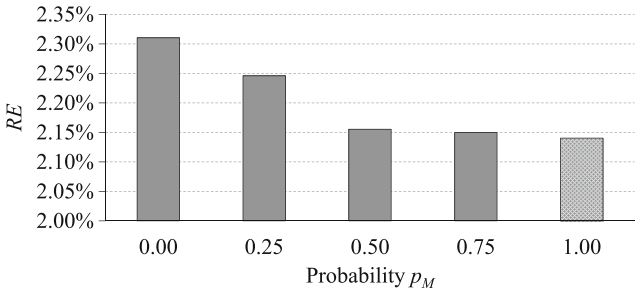


Fig. 5 Algorithm's performance with varying mutation probability

4.3.1 Mutation probability p_M

Having fixed the inertia, we try different mutation probability values p_M. Figure 5 illustrates the importance of this parameter for the behaviour of the algorithm, with larger probability values yielding the best results. As it happens in the crisp version of the PSO, the best option is to mutate the particles with probability $p_M = 1$.

4.3.2 Swarm size N

Finally, we test different swarm sizes: 60, 80 and 100. Here we need to pay attention not only to makespan values, but also to runtime, this being an important penalisation factor. Figure 6 shows the distance (bars) and runtime (line) of each configuration. Clearly, although the largest swarm size provides the best results in terms of relative error w.r.t. the lower bound, the improvement does not compensate the increased computation-time cost. In fact, if we increase the swarm size from 60 to 100, *RE* decreases 4.3 % compared

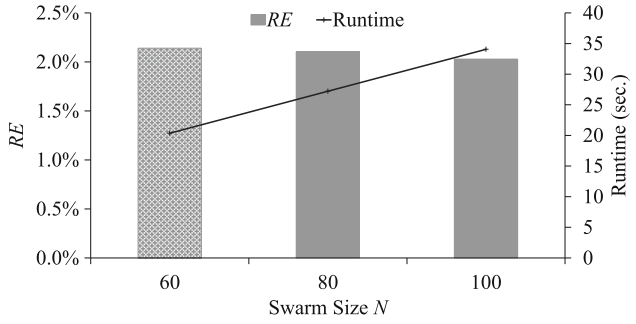


Fig. 6 Algorithm's performance with varying swarm size N

Table 5 Algorithm's performance with varying the delay parameter δ

Delay value δ	*RE* (%)	Std. dev
0.00	3.44	2.32
0.25	2.56	1.96
0.50	2.73	2.02
0.75	3.32	2.26
1.00	5.95	3.38

Table 6 Parameter values adopted after the parametric analysis

Inertia w	Mutation p_m	Guiding const. C_1,C_2	Delay δ
from 0.9 to 0.3	1	0.9, 0.1	0.25

to a 67.3 % increase in runtime. Therefore, we keep $N = 60$.

4.3.3 Delay parameter δ

There is another parameter in the algorithm, the delay parameter δ used by the schedule builder (Algorithm 2). Unlike the other parameters, the differences in algorithm performance caused by variations of δ are a consequence of changes in the subset of the search space which is explored, not of changes in the search process followed by the PSO algorithm. We have tried five possibilities: $\delta = 0$, which corresponds to exploring the set of non-delay schedules; $\delta = 1$, to exploring the set of active schedules; and $\delta = 0.25, 0.5, 0.75$, to exploring three proper subsets of the space of active schedules. Table 5 reports the obtained results, suggesting that the best performance of the PSO is obtained with $\delta = 0.25$.

A summary of the parameter values (except *MaxIter*) adopted after the parametric analysis can be seen in Table 6.

Table 7 Comparison between PSO and MA

Problem family	PSO			MA		
	RE		*T*(*s*)	*RE*		*T*(*s*)
	AoB	*AoA*		*AoB*	*AoA*	
J3	0.112	0.112	0.05	0.112	0.112	0.13
J4	0.645	0.757	0.10	0.645	0.799	0.23
J5	0.667	0.687	1.29	0.874	2.234	1.29
J6	0.861	1.019	4.24	0.929	2.698	7.57
J7	1.591	1.971	9.76	2.425	4.710	14.46
J8	2.051	2.693	19.53	3.565	5.807	26.15

4.4 Algorithm's evaluation

To our knowledge, the best results obtained so far for the FOSP have been published in [11]. In that paper, the GA from [26] is combined with local search using a new neighbourhood structure, providing a memetic algorithm (MA) which not only improves on solution quality but is also more "reliable" in the sense that there is less variability in solution quality across different executions. Thus, in the following we shall evaluate our PSO in comparison with this MA.

For the experimental evaluation, we shall use the optimal configuration obtained above with the exception of the smallest problems (3×3, 4×4). Here, since the search space is small, we take $\delta = 1$ as delay value for the *pfG&T* algorithm. This allows to explore the whole space of active schedules, thus keeping the chance of finding an optimal solution. For medium size problems (5×5, 6×6) and large problems (7×7, 8×8), we use the best delay value found during the parametric analysis, that is, $\delta = 0.25$.

To evaluate the performance, we run the proposed PSO 30 times for each problem instance, recording the best and average relative error of the expected makespan with respect to its lower bound across these 30 runs. Table 7 contains a summary of the results, with average values across 30 executions on each the 80–90 instances of the same size (detailed results for each problem, which require 520 rows, can be found at http://www.di.uniovi.es/iscop). There are three columns per method, PSO and MA, showing the Average of the Best values (*AoB*), the Average of the Average values (*AoA*), and the average runtime in seconds [*T*(*s*)] across 30 runs. We can see that the performance of both PSO and MA is similar on the small (3×3, 4×4) problem instances. However, differences in solution quality between the PSO and the MA increase with problem size, with the PSO obtaining better results. The average increase in *RE* value from the PSO to the MA across all problems is 28 % for AoB and 108 % for AoA. It is

noticeable that the a maximum increase in the value of AoB in Table 7 is 74 % for problems of size 8 × 8.

More detailed results are presented in Table 8, where each row corresponds to a set of ten fuzzy versions of each of the original crisp problems of size 8 × 8. It shows relative makespan errors w.r.t. the lower bound for both methods: the best (B) error and the average (A) error across 30 runs of each method. As expected, the PSO compares favourably with MA in all instances. Notice as well that the relative errors for the best (B) and average (A) solution do not differ greatly, suggesting that the PSO is quite stable. Figure 7 depicts the average relative makespan error for each set of fuzzy problems, clearly illustrating the difference between the proposed PSO and the MA.

4.5 Why not simply defuzzify?

It is tempting to think that a simpler approach to fuzzy open shop problems is to use defuzzification: substitute the uncertain durations for a crisp value (e.g. their expected value) and then solve the resulting deterministic open shop problem. This would provide a deterministic predictive schedule, including a task processing order. Tasks can then be processed according to this order, even if their starting times are likely to change given the variability in the durations. The advantages of doing so are clear: a simpler operational setting and the availability of different solving methods from the literature. However, before embracing defuzzification, we should also consider its effect (if any) on the robustness of the obtained solutions.

In this subsection, we propose to evaluate, in terms of ϵ-robustness, the predictive schedules obtained with both approaches: solving the fuzzy problem or, alternatively, defuzzifying durations and solving the resulting crisp problem. To do so, we simulate N possible realisations or scenarios of the problem: crisp durations for tasks are generated following a probability distribution which is coherent with the possibility distribution defined by each TFN. For each scenario $i = 1, \ldots, N$, let $C^i_{max} {C_{max}}^i$ denote the makespan obtained by executing tasks according to the ordering provided by a predictive schedule. Then, the mean ϵ-robustness of the predictive schedule, denoted $\overline{\epsilon}$, is calculated as:

$$\overline{\epsilon} = \frac{1}{N} \sum_{i=1}^{N} \frac{|C^i_{max} - C^{pred}_{max}|}{C^{pred}_{max}} \tag{11}$$

where C^{pred}_{max} is the makespan estimated by the predictive schedule. In our case, two predictive schedules are considered: the schedule obtained from solving the fuzzy problem, so C^{pred}_{max} is the expected makespan $E[C_{max}]$, and the schedule obtained from solving the defuzzified problem where TFNs are substitued by their expected value, in which case C^{pred}_{max} is a crisp makespan value.

For this robustness analysis, we concentrate on the largest problem instances, those of size 8 × 8 and consider, for each problem instance, $N = 1000$ deterministic instances corresponding to possible realisations. Figure 8 depicts, for each problem, the mean ϵ-robustness value of each predictive schedule, the fuzzy one (denoted $\overline{\epsilon}_F$) and the defuzzified one (denoted $\overline{\epsilon}_C$), across the N simulated scenarios. Clearly, the predictive schedule obtained from the fuzzy problem is much more robust (with smaller prediction error ϵ) than the schedule obtained from the defuzzified problem. In fact, the robustness error of the

Table 8 Average *RE* (in %) for sets of problems of size 8 × 8

Problem	PSO		MA	
	B	*A*	*B*	*A*
J8-per0-1	4.410	5.421	7.533	10.493
J8-per0-2	5.402	5.909	6.923	9.715
J8-per10-0	2.951	3.425	4.209	6.688
J8-per10-1	1.614	2.375	3.419	5.959
J8-per10-2	1.352	2.650	3.994	6.455
J8-per20-0	0.393	0.872	1.170	2.960
J8-per20-1	0.000	0.111	0.155	1.349
J8-per20-2	0.288	0.783	1.114	2.837

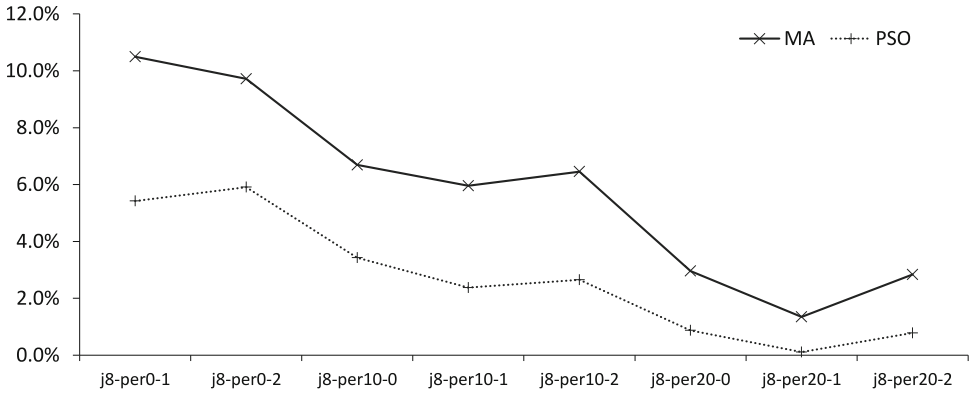


Fig. 7 Average makespan error *RE* (%) for 8 × 8 problems

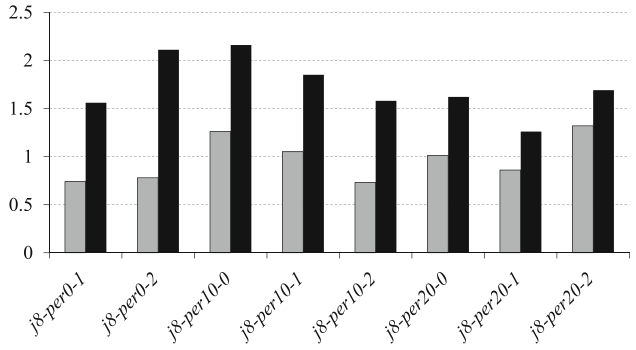


Fig. 8 Mean ϵ-robustness value of the predictive schedules obtained from the fuzzy problem ($\bar{\epsilon}_F$, in *grey*) and the defuzzified one ($\bar{\epsilon}_C$, in *black*)

defuzzified solution is always significantly higher than that of the fuzzy solution, with error increases ranging from 28.03 to 170.51 % and an average error increase of 85.04 %. We may conclude that it is more robust to take into account all the available information about task durations and solve the fuzzy problem than solve the defuzzified problem.

5 Conclusions and future work

We have considered an open shop problem with uncertain durations modelled as triangular fuzzy numbers where the objective is to minimise the expected makespan, a problem denoted $O|fuzz\ p_{ij}|E[C_{max}]$. We have proposed a PSO method to solve this problem. An extensive experimental analysis has shown that the PSO obtains good results both in terms of relative makespan error and also in comparison to a memetic algorithm from the literature. Additionally, we have argued that it is more robust to find solutions to the fuzzy problem, taking into account the uncertainty in the durations along the scheduling process, instead of the straightforward approach of defuzzifying the durations and scheduling the resulting deterministic problem.

These promising results suggest directions for future work. First, the PSO should be tested on more difficult problems, fuzzy versions of other benchmark problems from the literature. Also, the PSO provides a solid basis for the development of more powerful hybrid methods, in combination with local search techniques, an already successful approach in fuzzy job shop problems [29]. It would also be interesting to adapt this successful PSO method to the fuzzy job shop problem.

Acknowledgments This work Has been funded by the Spanish Ministry of Science and Education under research grants MEC-FEDER TIN2010-20976-C02-02 and MTM2010-16051 and by the Principality of Asturias (Spain) under grant Severo Ochoa BP13106.

References

1. Ahmadizar F, Farahani MH (2012) A novel hybrid genetic algorithm for the open shop scheduling problem. Int J Adv Manuf Technol 62:775–787
2. Bidot J, Vidal T, Laboire P (2009) A theoretic and practical framework for scheduling in stochastic environment. J Sched 12:315–344
3. Blum C (2005) Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
4. Brucker P, Hunrink J, Jurisch B, Wöstmann B (1997) A branch & bound algorithm for the open-shop problem. Discrete App Math 76:43–59
5. Dubois D, Fargier H, Fortemps P (2008) Scheduling under flexible constraints and uncertain data: the fuzzy approach. In: Production scheduling, chap. 11, Wiley, Weinheim, p 301–332
6. Dubois D, Fargier H, Fortemps P (2003) Fuzzy scheduling: modelling flexible constraints vs. coping with incomplete knowledge. Eur J Oper Res 147:231–252
7. Dubois D, Prade H (1986) Possibility theory: an approach to computerized processing of uncertainty. Plenum Press, New York
8. Fortemps P (1997) Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Trans Fuzzy Syst 7:557–569
9. Giffler B, Thompson GL (1960) Algorithms for solving production scheduling problems. Oper Res 8:487–503
10. Gonçalves J, Mendes J, de M RM (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167:77–95
11. González-Rodríguez I, Palacios JJ, Vela CR, Puente J (2010) Heuristic local search for fuzzy open shop scheduling. In: Proceedings IEEE International conference on fuzzy systems, FUZZ-IEEE2010. pp 1858–1865. IEEE
12. González-Rodríguez I, Puente J, Vela CR, Varela R (2008) Semantics of schedules for the fuzzy job shop problem. IEEE Trans Syst Man Cybern Part A 38(3):655–666
13. González-Rodríguez I, Vela CR, Puente J (2007) A memetic approach to fuzzy job shop based on expectation model. In: Proceedings of IEEE international conference on fuzzy systems, FUZZ-IEEE2007. pp 692–697. IEEE, London
14. Graham R, Lawler E, Lenstra J, Rinnooy Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discret Math 4:287–326
15. Guéret C, Prins C (1998) Classical and new heuristics for the open-shop problem: a computational evaluation. Eur J Oper Res 107:306–314
16. Herroelen W, Leus R (2005) Project scheduling under uncertainty: Survey and research potentials. Eur J Oper Res 165:289–306
17. Hu Y, Yin M, Li X (2011) A novel objective function for job-shop scheduling problem with fuzzy processing time and fuzzy due date using differential evolution algorithm. Int J Adv Manuf Technol 56:1125–1138
18. Huang YM, Lin JC (2011) A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems. Expert Syst Appl 38(5):5438–5447
19. Kalaï R, Lamboray C, Vanderpooten D (2012) Lexicographic α-robustness: an alternative to min-max criteria. Eur J Oper Res 220:722–728
20. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: IEEE international conference on neural networks. pp 1942–1948. IEEE Press, New Jersey
21. Konno T, Ishii H (2000) An open shop scheduling problem with fuzzy allowable time and fuzzy resource constraint. Fuzzy Sets Syst 109:141–147

22. Lei D (2010) Solving fuzzy job shop scheduling problems using random key genetic algorithm. International Int J Adv Manuf Technol 49:253–262
23. Liaw CF (1999) A tabu search algorithm for the open shop scheduling problem. Comput Oper Res 26:109–126
24. Liu B, Liu YK (2002) Expected value of fuzzy variable and fuzzy expected value models. IEEE Trans Fuzzy Syst 10:445–450
25. Noori-Darvish S, Mahdavi I, Mahdavi-Amiri N (2012) A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due-dates. Appl Soft Comput 12:1399–1416
26. Palacios JJ, Puente J, Vela CR, González-Rodríguez I (2009) A genetic algorithm for the open shop problem with uncertain durations. In: Proceedings of IWINAC 2009, Part I. Lecture notes in computer science, vol 5601, pp 255–264. Springer
27. Pinedo ML (2008) Scheduling: theory, algorithms, and systems, 3rd edn. Springer, Secaucus
28. Puente J, Vela CR, González-Rodríguez I (2010) Fast local search for fuzzy job shop scheduling. In: Proceedings of 19th European conference on artificial intelligence, ECAI 2010. pp 739–744. IOS Press
29. Sha DY, Cheng-Yu H (2006) A modified parameterized active schedule generation algorithm for the job shop scheduling problem. In: Proceedings of the 36th international conference on computers and industrial engineering, ICCIE2006. pp 702–712
30. Sha DY, Cheng-Yu H (2008) A new particle swarm optimization for the open shop scheduling problem. Comput Oper Res 35:3243–3261
31. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
32. Tavakkoli-Moghaddam R, Safei N, Kah M (2008) Accessing feasible space in a generalized job shop scheduling problem with the fuzzy processing times: a fuzzy-neural approach. J Oper Res Soc 59:431–442
33. Zheng Y, Li Y, Lei D (2011) Swarm-based neighbourhood search for fuzzy job shop scheduling. Int J Innovative Comput Appl 3(3):144–151

7.5 Swarm lexicographic goal programming for fuzzy open shop scheduling

In this section, we include the following publication.

- **Title:** Swarm lexicographic goal programming for fuzzy open shop scheduling.
- **Journal:** Journal of Intelligent Manufacturing.
- **Year:** In press. Published online in 2013.
- Impact Factor (JCR 2013): 1.142
- Impact Factor (5-year): 1.658
- Journal Ranking:
  - Computer Science, Artificial Intelligence: 64/121 **Q3 (T2)**
  - Engineering, Manufacturing: 22/39 **Q3 (T2)**

This publications contains pieces of work described in Section 4.2.1.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Swarm lexicographic goal programming for fuzzy open shop scheduling. Journal of Intelligent Manufacturing. In press (2013). Doi: 10.1007/s10845-013-0850-y

J Intell Manuf
DOI 10.1007/s10845-013-0850-y

Swarm lexicographic goal programming for fuzzy open shop scheduling

Juan José Palacios · Inés González-Rodríguez · Camino R. Vela · Jorge Puente

Received: 11 April 2013 / Accepted: 15 November 2013

Abstract In this work we consider a multiobjective open shop scheduling problem with uncertain processing times and flexible due dates, both modelled using fuzzy sets. We adopt a goal programming model based on lexicographic multiobjective optimisation of both makespan and due-date satisfaction and propose a particle swarm algorithm to solve the resulting problem. We present experimental results which show that this multiobjective approach achieves as good results as single-objective algorithms for the objective with the highest priority, while greatly improving on the second objective.

Keywords Open shop scheduling · Fuzzy processing times · Flexible due dates · Particle swarm optimisation · Lexicographic goal programming

Introduction

The open shop scheduling problem (OSP) is a problem with an increasing presence in the literature and clear applications in industry—consider for instance testing facilities where units go through a series of diagnostic tests that need not be performed in a specified order and where different testing equipment is usually required for each test (see Pinedo 2008). For a number of machines $m \geq 3$ this problem is NP-complete; in consequence, it is usually tackled via metaheuristics techniques. For instance, for makespan minimisation, Guéret and Prins (1998) describe two heuristic methods to obtain a list of operation priorities later used in a list-scheduling algorithm; Liaw (1999) proposes a tabu search algorithm; Blum (2005) hybridises ant colony optimisation with beam search and Sha and Cheng-Yu (2008) propose a solution based on particle swarm optimisation. To minimise total tardiness, Naderi et al. (2011) propose two metaheuristics based on genetic algorithms and variable neighbourhood search and for multiobjective open shop we find an ant colony algorithm combined with simulated annealing in Panahi et al. (2008) and particle swarm optimisation in Sha et al. (2010).

Traditionally, scheduling has been treated as a deterministic problem that assumes precise knowledge of all data involved, in contrast with the uncertainty and vagueness pervading real-world problems. To enhance the range of applications of scheduling, an increasing part of the research is devoted to modelling this lack of certainty with great diversity of approaches (Herroelen and Leus 2005). In particular, fuzzy sets have been used in different manners, ranging from representing incomplete or vague states of information to using fuzzy priority rules with linguistic qualifiers or preference modelling and as an interesting tool for improving solution robustness and stability (Guiffrida and Nagi 1998; Dubois et al. 2003; Petrovic et al. 2008).

Far from being trivial, extending heuristic strategies to uncertain settings usually requires a significant reformulation of both the problem and solving methods. This is patent in the available literature on job shop problems with uncertain processing times and/or flexible constraints. For instance, Dubois et al. (1995) extend a constrained-based approach,

J. J. Palacios · C. R. Vela · J. Puente
Department of Computing, University of Oviedo,
Campus de Viesques, 33204 Gijón, Spain
e-mail: palaciosjuan@uniovi.es

C. R. Vela
e-mail: crvela@uniovi.es

J. Puente
e-mail: puente@uniovi.es

I. González-Rodríguez (✉)
Department of Mathematics, Statistics and Computing, University of Cantabria, Los Castros s/n, 39005 Santander, Spain
e-mail: gonzalezri@unican.es; ines.gonzalez@unican.es

Published online: 30 November 2013

Fortemps (1997) uses simulated annealing and Sakawa and Kubota (2000) propose a genetic algorithm in what can be seen as pioneering works in the application of metaheuristic strategies, followed by many authors, e.g. González Rodríguez et al. (2008), Puente et al. (2010), Niu et al. (2008) or Zheng et al. (2011). However, while there are many contributions to solve fuzzy job shop problems, the literature on fuzzy open shop is still scarce. Indeed, the open shop with uncertainty constitutes a relatively new and complex research line. Among the few existing proposals, in (Alcaide et al. 2006) a heuristic approach is proposed to minimise the expected makespan for an open shop problem with stochastic processing times and random breakdowns; González-Rodríguez et al. (2010) minimise the expected makespan of an open shop with fuzzy durations using a genetic algorithm hybridised with local search, while Palacios et al. (2011) use a particle swarm optimisation algorithm for the same problem. Finally, a possibilistic mixed-integer linear programming method is proposed in Noori-Darvish et al. (2012) for an OSP with setup times, fuzzy processing times and fuzzy due dates to minimise total weighted tardiness and total weighted completion times.

Another issue that must be taken into account to reduce the gap between academic and real-world problems is the fact that many real-life applications require taking into account several conflicting points of view corresponding to multiple objectives. This is one of the reasons why the applications of multiobjective decision making techniques in engineering have grown in the recent decades (Pasandideh and Niaki 2013). Although Pareto optimality is undoubtedly the most extended approach to multicriteria optimisation, as Ehrgott (2005) puts it, "it is not the end of the story", with other approaches to multiobjective optimisation in the literature (Ehrgott and Gandibleux 2000). Among these techniques, lexicographic and goal programming methods are some of the most popular ones (Farahani et al. 2010). The philosophy behind goal programming (Romero 2001) can be traced back to the theories of rational decision developed in the 1950s, especially the concept of satisficing solutions: in a complex environment, the decision maker's aim may be to reach a certain satisfactory level for every relevant objective, rather than optimising its value. Also, lexicographic problems arise naturally when conflicting objectives exist in a decision problem but for reasons outside the control of the decision maker the objectives have to be considered in hierarchical manner. Recent examples of real-world problems where these techniques are applied can be found, for instance, in Ehrgott (2005), Diaz-Balteiro and Romero (2008), Puente et al. (2013), Coshall and Charlesworth (2011), and Liberatore et al. (2013). Additionally, there exist interesting relationships between lexicographic and Pareto-optimal solutions. Indeed, "lexicographic minimisation is well-suited to seek a compromise between conflicting interests, as well as reconciling this requirement with the crucial notion of Pareto-optimality" (Bouveret and Lemaître 2009).

To our knowledge, a lexicographical goal programming approach to solve multiobjective instances of fuzzy open shop has never been taken in the still scarce literature on this problem. This paper attempts to contribute to filling this gap. To this end, in the sequel we propose a multiobjective particle swarm optimisation (MOPSO) algorithm to solve instances of open shop where uncertain processing times are modelled with triangular fuzzy numbers and flexible due dates are modelled with fuzzy sets. In "Uncertain processing times and flexible constraints" section we provide some background on fuzzy sets, which will be used in "The fuzzy open shop scheduling problem" section to formulate the Fuzzy Open Shop Problem (FOSP). We adopt a lexicographic goal programming approach to define an objective function which combines minimisation of the expected fuzzy makespan and maximisation of overall due-date satisfaction. The resulting problem is solved by means of a particle swarm optimization method searching in the space of possibly active schedules, as proposed in "Particle swarm optimization for the FOSP" section. "Experimental results" section reports results from the experimental study which illustrate the potential of the proposed method. Finally, in " Conclusions and future work" section we summarise the main conclusions and propose some ideas for future work.

Uncertain processing times and flexible constraints

In real-life applications, it is often the case that the exact duration of a task is not known in advance. However, based on previous experience, an expert may be able to estimate, for instance, an interval for the possible processing time or its most typical value. In literature, it is common to use fuzzy intervals to represent such processing times, as an alternative to probability distributions, which require a deeper knowledge of the problem and usually yield a complex calculus.

Fuzzy interval arithmetic to model processing times

Fuzzy intervals are a natural extension of human originated confidence intervals when some values appear to be more plausible than others. The simplest model is a *triangular fuzzy number* or *TFN*, using an interval $[a^1, a^3]$ of possible values and a single plausible 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:

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

Triangular fuzzy numbers and more generally fuzzy intervals have been extensively studied in the literature (cf. Dubois and Prade 1986). A *fuzzy interval* Q is a fuzzy quantity (a fuzzy set on the reals) whose α-*cuts* $Q_\alpha = \{u \in \mathbb{R} : \mu_Q(u) \geq \alpha\}$, $\alpha \in (0.1]$, are convex, i.e. they are intervals (bounded or not). The *core* of Q consists of those elements with full membership $\mu_Q(u) = 1$, also called *modal values* and its *support* is $Q_0 = \{u \in \mathbb{R} : \mu_Q(u) > 0\}$. A *fuzzy number* is a fuzzy quantity whose α-cuts are closed intervals, with compact (i.e. closed and bounded) support and unique modal value. Thus, real numbers can be seen as a particular case of fuzzy ones.

In order to work with fuzzy numbers, it is necessary to extend the usual arithmetic operations on real numbers. In general, if f is a function $f : \mathbb{R}^2 \to \mathbb{R}$ and Q_1, Q_2 are two fuzzy quantities, the fuzzy quantity $f(Q_1, Q_2)$ is calculated according to the *Extension Principle*. However, computing the resulting equation is in general cumbersome, if not intractable. It can be somewhat simplified for two fuzzy numbers M and N, so the α-cuts M_α and N_α are closed bounded intervals of the form $[\underline{m}_\alpha, \overline{m}_\alpha]$ and $[\underline{n}_\alpha, \overline{n}_\alpha]$, if f is a continuous isotonic mapping from $\mathbb{R}^2$ into $\mathbb{R}$, that is, if for any $u \geq u'$ and $v \geq v'$ it holds $f(u, v) \geq f(u', v')$. In this case, the First Decomposition Theorem provides us with an alternative formula for $f(M, N)$:

$$f(M, N) = \cup_{\alpha \in (0,1]}[f(\underline{m}_\alpha, \underline{n}_\alpha),\ f(\overline{m}_\alpha, \overline{n}_\alpha)] \tag{2}$$

In the open shop, we essentially need the following operations on fuzzy durations: addition and maximum. In the case of TFNs, the addition is fairly easy to compute, since it is reduced to operating on the three defining points, that is, for any pair of TFNs M and N:

$$M + N = (m^1 + n^1, m^2 + n^2, m^3 + n^3). \tag{3}$$

Unfortunately, for the maximum of TFNs there is no such simplified expression. Being an isotonic function, we can use Eq. (2) above, but in general this still requires an infinite number of computations, since we have to evaluate maxima for each value $\alpha \in (0, 1]$. For the sake of simplicity and tractability of numerical calculations, we follow (Fortemps 1997) and approximate all results of isotonic algebraic operations on TFNs by a TFN. Instead of evaluating the intervals corresponding to all α-cuts, we evaluate only those intervals corresponding to the support and $\alpha = 1$, which is equivalent to working only with the three defining points of each TFN. This is an approach also taken, for instance, in Niu et al. (2008) and Chen and Chang (2001). Therefore, for any two TFNs M and N, their maximum will be approximated as follows:

$$\begin{aligned} \max(M, N) \sim M \sqcup N = (&\max(m^1, n^1), \\ &\max(m^2, n^2), \max(m^3, n^3)). \end{aligned} \tag{4}$$

Despite not being equal, for any two TFNs M, N, if $F = \max(N, M)$ denotes their maximum and $G = N \sqcup M$ its approximated value, it holds that $\forall \alpha \in [0, 1]$, $\underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha$. In particular, F and G have identical support and modal value: $F_0 = G_0$ and $F_1 = G_1$. The approximated maximum can be trivially extended to $n > 2$ TFNs.

For a fuzzy number N, its membership function μ_N can be interpreted as a possibility distribution on the real numbers. This allows to define the *expected value* of a fuzzy number (Liu and Liu 2002), given for a TFN A by

$$E[A] = \frac{1}{4}(a^1 + 2a^2 + a^3). \tag{5}$$

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 (Dubois et al. 2003). It induces a total ordering $\leq_E$ in the set of fuzzy intervals (Fortemps 1997), where for any two fuzzy intervals M, N $M \leq_E N$ if and only if $E[M] \leq E[N]$.

Modelling flexible due dates

In practice, if due-date constraints exist, they are often flexible. For instance, customers may have a preferred delivery date d^1, but some delay will be allowed until a later date d^2, after which the order will be cancelled. The satisfaction of a due-date constraint becomes a matter of degree, our degree of satisfaction that a job is finished on a certain date. A common approach to modelling such satisfaction levels is to use a fuzzy set D with linear decreasing membership function:

$$\mu_D(x) = \begin{cases} 1 & : x \leq d^1 \\ \frac{x - d^2}{d^1 - d^2} & : d^1 < x \leq d^2 \\ 0 & : d^2 < x \end{cases} \tag{6}$$

This expresses a flexible threshold "less than", representing the satisfaction level $sat(t) = \mu_D(t)$ for the ending date t of the job (Dubois et al. 2003). When the job's completion time is no longer a real number t but a TFN C, the degree to which C satisfies the due-date constraint D may be measured using the following *agreement index* (Sakawa and Kubota 2000; Celano et al. 2003):

$$AI(C, D) = \frac{area(D \cap C)}{area(C)} \tag{7}$$

where $area(D \cap C)$ and $area(C)$ denote the areas under the membership functions of $(D \cap C)$ and C respectively. The intuition behind this definition is to measure the degree to which C is contained in D (the degree of subsethood).

The fuzzy open shop scheduling problem

The *open shop scheduling problem*, or *OSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job J_i consists of m tasks or operations o_{ij} $(j = 1, \ldots, m)$, where o_{ij} requires the exclusive use of a machine M_j for its whole processing time p_{ij} without preemption, i.e. all tasks must be processed without interruption. In total, there are mn tasks. Additionally, for each job J_i there may be a due date d_i, $i = 1, \ldots, n$ before which it is desirable that the job be finished. A solution to this problem is a schedule (a starting time for all tasks) which, besides being *feasible*, in the sense that precedence and capacity constraints hold, is optimal according to some criteria, for instance, that due-date satisfaction is maximal or that the project's makespan is minimal.

Fuzzy schedules from crisp task orderings

A schedule s for an open shop problem of size $n \times m$ (n jobs and m machines) may be determined by a decision variable $\mathbf{z} = (z_1, \ldots, z_{nm})$ representing a task processing order, where $1 \leq z_l \leq nm$ for $l = 1, \ldots, nm$. This is a permutation of the set of tasks where each task o_{ij} is represented by the number $(i-1)m + j$. The task processing order represented by the decision variable uniquely determines a feasible schedule; it should be understood as expressing partial orderings for every set of tasks requiring the same machine and for every set of tasks belonging to the same job.

Let us assume that the processing time p_{ij} of each task o_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$ is a fuzzy variable (a particular case of which are TFNs), so the problem may be represented by a matrix of fuzzy processing times $\mathbf{p}$ of size $n \times m$. For a given task processing order $\mathbf{z}$ and a task o_{ij}, its starting time $S_{ij}(\mathbf{z}, \mathbf{p})$ is the maximum (Eq. 4) between the completion times of the task preceding o_{ij} in its job, let it be denoted o_{ik}, and the task preceding o_{ij} in its machine, let it be denoted o_{lj}:

$$S_{ij}(\mathbf{z}, \mathbf{p}) = C_{ik}(\mathbf{z}, \mathbf{p}) \sqcup C_{lj}(\mathbf{z}, \mathbf{p}) \tag{8}$$

where $C_{ik}(\mathbf{z}, \mathbf{p})$ or $C_{lj}(\mathbf{z}, \mathbf{p})$ are taken to be zero if o_{ij} is the first task to be processed either in its job or its machine. Then, its completion time $C_{ij}(\mathbf{z}, \mathbf{p})$ is obtained by adding its duration p_{ij} to $S_{ij}(\mathbf{z}, \mathbf{p})$:

$$C_{ij}(\mathbf{z}, \mathbf{p}) = S_{ij}(\mathbf{z}, \mathbf{p}) + p_{ij} \tag{9}$$

The completion time of a job J_i will then be the maximum completion time of all its tasks, that is, $C_i(\mathbf{z}, \mathbf{p}) = \sqcup_{1 \leq j \leq m} \{C_{ij}(\mathbf{z}, \mathbf{p})\}$.

For this schedule, the *fuzzy makespan* $C_{max}(\mathbf{z}, \mathbf{p})$ is defined as the maximum of job completion times:

$$C_{max}(\mathbf{z}, \mathbf{p}) = \sqcup_{1 \leq i \leq n} (C_i(\mathbf{z}, \mathbf{p})) \tag{10}$$

Notice that when uncertain durations are given as fuzzy intervals the schedule s will be fuzzy in the sense that the starting and completion times of all tasks as well as the makespan are fuzzy intervals. These may be interpreted as possibility distributions on the values that each time may take. Fuzzy intervals are thus used to represent our incomplete knowledge of problem parameters related to durations and, in consequence, our incomplete knowledge of starting and completion times for all tasks. However, the task processing order represented by $\mathbf{z}$ that determines such schedule is crisp: there is no uncertainty regarding the order in which tasks are to be processed.

Given a fuzzy schedule, it is necessary to give a precise definition of what "optimal makespan" means, since neither the maximum nor its approximation define a total ordering in the set of TFNs. Using ideas similar to stochastic scheduling, we use the total ordering provided by the expected value and consider that the objective of minimising the makespan translates, in practice, into minimising its expected value $E[C_{max}]$ (Eq. 5).

While also being fuzzy sets, due dates d_i for jobs J_i, $i = 1, \ldots, n$, do not model uncertainty. Instead, they model flexible constraints, introducing grades in the traditionally Boolean notion of feasibility (cf. Dubois 2011) and the references therein for the semantics of fuzzy sets and their role in decision making). In this setting, the *agreement index*, $AI(C_i(\mathbf{z}, \mathbf{p}), d_i)$ (Eq. 7), denoted $AI_i(\mathbf{z}, \mathbf{p})$ for short, measures to what degree the flexible due date d_i is satisfied by the fuzzy time $C_i(\mathbf{z}, \mathbf{p})$. The degree of overall due-date satisfaction for schedule s may then be obtained by aggregating the satisfaction degrees $AI_i(\mathbf{z}, \mathbf{p}), i = 1, \ldots, n$. In particular, we shall consider two aggregation functions, the minimum and the average, previously used in the literature concerning shop scheduling with soft constraints, for instance, in Sakawa and Kubota (2000), González Rodríguez et al. (2008), Lei (2008). The minimum is inspired by the seminal paper on fuzzy decision making (Bellman and Zadeh 1970), while the average provides an alternative for which the compensation property holds. Hence, the degree $AI_{ag}(\mathbf{z}, \mathbf{p})$ to which a schedule s determined by an ordering $\mathbf{z}$ satisfies due dates will be determined by one of the two following formula:

$$AI_{av}(\mathbf{z}, \mathbf{p}) = \frac{1}{n} \sum_{i=1}^{n} AI_i(\mathbf{z}, \mathbf{p}), \tag{11}$$

$$AI_{min}(\mathbf{z}, \mathbf{p}) = \min_{i=1,\ldots,n} AI_i(\mathbf{z}, \mathbf{p}) \tag{12}$$

Clearly both $AI_{av}(\mathbf{z}, \mathbf{p})$ and $AI_{min}(\mathbf{z}, \mathbf{p})$ should be maximised. Notice however that they model different requirements and encourage different behaviours. In the cases when

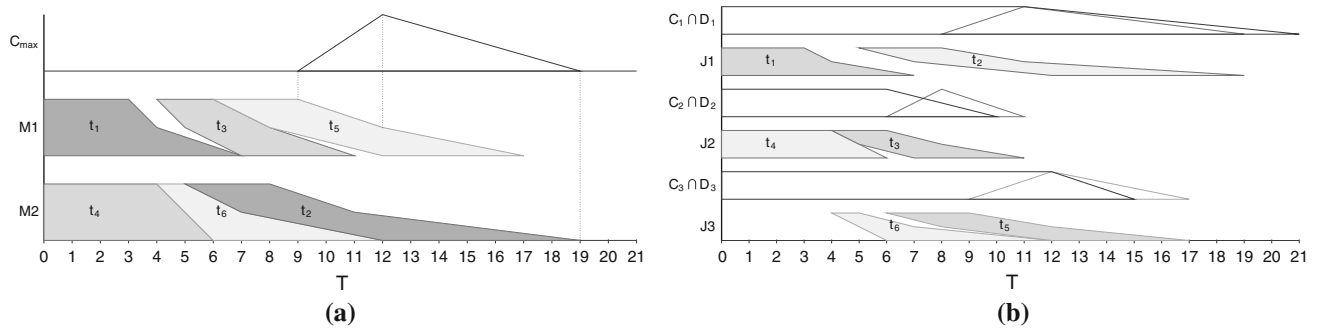


Fig. 1 Gantt charts of the schedule represented by the decision variable (1, 4, 6, 3, 5, 2). **a** Machine oriented, **b** Job oriented

there is no possible confusion regarding the order $\mathbf{z}$ or the processing times $\mathbf{p}$, we may simplify the notation and write AI_{ag} or C_{max}.

Let us illustrate the previous definitions with an example. Consider a problem of 3 jobs and 2 machines with the following matrices for fuzzy processing times and due dates:

$$\mathbf{p} = \begin{pmatrix} (3,4,7) & (3,4,7) \\ (2,3,4) & (4,5,6) \\ (3,4,5) & (1,2,6) \end{pmatrix} \mathbf{d} = \begin{pmatrix} (11,21) \\ (6,10) \\ (12,15) \end{pmatrix}$$

Here $p_{21} = (2, 3, 4)$ is the processing time of task o_{21}, the task of job J_2 to be processed in machine M_1 and $d_2 = (6, 10)$ is the flexible due date for job J_2. Figure 1a, b show the Gantt charts (both machine and job oriented) adapted to TFNs of the schedule given by the decision variable $\mathbf{z} = (1, 4, 6, 3, 5, 2)$. They represent the partial schedules on each machine and each job obtained from this decision variable. Tasks must be processed in the following order: $o_{11}, o_{22}, o_{32}, o_{21}, o_{31}, o_{12}$. Given this ordering, the starting time for task o_{21} will be the maximum of the completion times of o_{22} and o_{11}, which are respectively the preceding tasks in the job and in the machine: $S_{21} = C_{22} \sqcup C_{11} = (4, 5, 6) \sqcup (3, 4, 7) = (4, 5, 7)$. Consequently, its completion time will be $C_{21} = S_{21} + p_{21} = (4, 5, 7) + (2, 3, 4) = (6, 8, 11)$. Also, it is easy to see that $C_{max} = (9, 12, 19)$ (see Fig. 1a), so $E[C_{max}] = 13$. Regarding due dates, in Fig. 1b we can see that the completion time of job J_1 always satisfies its due date, so $AI_1 = 1$, whereas for job J_2 $area(C_2) = 5/2$ and $area(d_2 \cap C_2) = 4/3$, so $AI_2 = 0.53$, and analogously $AI_3 = 0.75$. Hence, the aggregated degrees of due date satisfaction will be $AI_{min} = 0.53$ and $AI_{av} = 0.76$.

Multiobjective model

For the fuzzy open shop problem we are interested both in maximising the aggregated due-date satisfaction AI_{ag} and minimising the expected makespan $E[C_{max}]$. A well-established approach dealing with multiple and possibly conflicting objectives is lexicographic goal programming (Ehrgott 2005; Tamiz et al. 1998), assuming that the decision makers establish a priority structure as well as target levels for the different objectives.

Before we formulate the resulting problem, notice that $AI_{ag}(\mathbf{z}, \mathbf{p}) \in [0, 1]$ for both aggregation operators. Hence, maximising $AI_{ag}(\mathbf{z}, \mathbf{p})$ is equivalent to minimising $1 - AI_{ag}(\mathbf{z}, \mathbf{p})$, which could be interpreted as the degree to which due dates are violated. In consequence, we can restate the objective of our problem as minimising both $E[C_{max}(\mathbf{z}, \mathbf{p})]$ and $1 - AI_{ag}(\mathbf{z}, \mathbf{p})$.

Let C_{max} and $1 - AI_{ag}$ be ordered according to their priority, and let f_1 denote the objective with highest priority and f_2 denote the secondary objective. Also, let us assume that the decision makers establish target values $b_1, b_2 \geq 0$ for f_1 and f_2. Clearly, these values should not be exceeded, which translates into the following goal constraints:

$$f_i(\mathbf{z}, \mathbf{p}) + \Delta_i^- - \Delta_i^+ = b_i, \quad i = 1, 2 \tag{13}$$

where $\Delta_1^+, \Delta_2^+ \geq 0$, the positive deviations from the targets, should be minimised. This results in the following *lexicographic goal programming model* for the fuzzy open shop problem (FOSP):

$$\begin{cases} \text{lexmin} & (\Delta_1^+, \Delta_2^+) \\ \text{subject to:} & \\ & f_i(\mathbf{z}, \mathbf{p}) + \Delta_i^- - \Delta_i^+ = b_i, \quad i = 1, 2, \\ & b_i \geq 0, \quad i = 1, 2, \\ & \Delta_i^-, \Delta_i^+ \geq 0, \\ & 1 \leq z_l \leq nm, \quad l = 1, \ldots, nm, \\ & z_l \neq z_k, \quad k \neq l \\ & z_l \in \mathbb{Z}^+, \quad l = 1, \ldots, nm, \end{cases} \tag{14}$$

where lexmin denotes lexicographically minimising the objective vector (Δ_1^+, Δ_2^+).

The resulting problem can be denoted $O|fuzz\ p_i, fuzz\ d_i|LexGP(E[C_{max}], 1-AI_{av})$ according to the three-field notation from (Graham et al. 1979), extended to multicriteria scheduling in the spirit of T'kindt and Billaut (2006).

Particle swarm optimization for the FOSP

Particle swarm optimisation (PSO) is a population-based stochastic method inspired by bird flocking or fish schooling, first proposed in Kennedy and Eberhart (1995) which has been successfully applied to solve complex combinatorial optimization problems; recent examples of this success can be found in Belmecheri et al. (2013), Jia and Seo (2013), and Kim and Son (2012). In particular, it has been applied to scheduling problems, among others, in Tassopoulos and Beligiannis (2012), Vijay Chakaravarthy et al. (2013), and Marinakis and Marinaki (2013) as well as the already mentioned references devoted to the open shop problem (Sha and Cheng-Yu 2008; Sha et al. 2010).

In PSO, each position in a multidimensional search space corresponds to a solution of the problem and particles in the swarm cooperate to explore the space and find the best position (hence best solution). Particle movement is mainly affected by the three following factors:

- Inertia: Velocity of the particle in the latest iteration,
- *pbest*: The best position found by the particle,
- *gbest*: The best position found by the swarm so far ("the best *pbest*"),

Potential solutions are represented by multidimensional particles flying through the problem space, changing their position and velocity by following the current optimum particles *pbest* and *gbest*. A generic PSO algorithm is given in Algorithm 1: first, the initial swarm is generated and evaluated and then the swarm evolves until a termination criterion is satisfied. In each iteration, a new swarm is built from the previous one by changing the position and velocity of each particle to move towards its *pbest* and *gbest* locations.

Input A FOSP instance
Output A schedule for the input instance
  Generate and evaluate the initial swarm;
  Compute *gbest* and *pbest* for each particle;
  while no Termination Criterion is satisfied **do**
    for each particle k **do**
      Update velocity $\mathbf{v}^k$;
      Update position $\mathbf{x}^k$;
      Evaluate particle k;
      Update *pbest* and *gbest* values;
  return The schedule from the best particle evaluated so far;

Algorithm 1: A generic PSO algorithm

In the following, we present a multiobjective PSO algorithm for the FOSP with lexicographic goal programming defined in the previous section. A preliminary version of this algorithm was presented in Palacios et al. (2011) to minimise the expected makespan of fuzzy open shop.

Position representation and evaluation

For each particle k in the swarm, its position $\mathbf{x}^k$ is represented with a priority-based representation. Thus, the decision variable $\mathbf{z}^k$ is encoded as a *priority array* $\mathbf{x}^k = (x_l^k)_{l=1\ldots nm}$ where x_l^k denotes the priority of task l, so a task with smaller x_l^k has a higher priority to be scheduled.

Given a FOSP solution represented by a decision variable $\mathbf{z}$, which is a permutation of tasks, we can transfer this permutation to a priority array as follows. First, from $\mathbf{z}$ we obtain a position array, denoted $pos^{\mathbf{z}}$, such that $pos_l^{\mathbf{z}}$ is the position of task l in $\mathbf{z}$ ($pos_l^{\mathbf{z}} = i$ if and only if $z_i = l$). For instance, for a problem with $n = 2$ jobs and $m = 3$ machines we can have a decision variable $\mathbf{z}$ and the corresponding position array $pos^{\mathbf{z}}$ as follows:

$$\mathbf{z} = (4, 1, 5, 2, 3, 6) \qquad \mathbf{pos^z} = (2, 4, 5, 1, 3, 6)$$

Then, the priority array $\mathbf{x}$ is obtained by randomly setting x_l in the interval $\left(pos_l^{\mathbf{z}} - 0.5,\ pos_l^{\mathbf{z}} + 0.5\right)$, so a task with smaller x_l has higher priority to be scheduled. For the above decision variable, a possible particle position would be:

$$\mathbf{x} = (2.3, 3.7, 5.4, 0.8, 2.8, 5.9)$$

Conversely, from every particle position $\mathbf{x}$ we can obtain a position array $\mathbf{pos^x}$ (and the corresponding decision variable) where pos_i^x is the position of x_i if the elements of $\mathbf{x}$ were reordered in non-decreasing order.

A particle may be decoded in several ways. For deterministic job shop and, by extension, for open shop scheduling, it is common to use the G&T algorithm (Giffler and Thompson 1960), which is an active schedule builder. A schedule is *active* if one task must be delayed for any other one to start earlier. Active schedules are good in average and, most importantly, the space of active schedules contains at least an optimal one, that is, the set of active schedules is *dominant*. For these reasons it is worth to restrict the search to this space. In Gonçalves et al. (2005) a narrowing mechanism was incorporated to the G&T algorithm in order to limit machine idle times using a delay parameter $\delta \in [0, 1]$, thus searching in the space of so-called parametrised active schedules. In the deterministic case, for $\delta < 1$ the search space is reduced so it may no longer contain optimal schedules and at the extreme $\delta = 0$ the search is constrained to non-delay schedules where a resource is never idle if a requiring operation is available. This variant of G&T has been applied in Sha and Cheng-Yu (2008) to the deterministic OSP, under the name "parameterized active schedule generation algorithm". Algorithm 2, denoted $pFG\&T$, is an extension of parametrised G&T to the case of fuzzy processing times proposed in Palacios et al. (2011). Throughout the algorithm, Ω denotes the set of tasks that have not been scheduled yet, $\mathbf{x}^k$ denotes the priority array and S_l and C_l denote the starting and completion time of task o_{ij} such that $l = (i-1)m + j$. It should be noted

Input A FOSP instance and a particle position $\mathbf{x}^k$
Output A schedule for the input instance considering the priorities given by $\mathbf{x}^k$
$\Omega \leftarrow \{1, \ldots, nm\}$;
while $\Omega \neq \emptyset$ **do**
Compute $\{E[S_l] : l \in \Omega\}$ and $\{E[C_l] : l \in \Omega\}$ considering only tasks previously scheduled;
$C^* \leftarrow \min_{l \in \Omega}\{E[C_l]\}$;
$S^* \leftarrow \min_{l \in \Omega}\{E[S_l]\}$;
Identify the conflict set $O \leftarrow \{l : E[S_l] < S^* + \delta \times (C^* - S^*), l \in \Omega\}$;
Choose the task l^* from O with smallest x_l^k;
Schedule the operation l^*; {fix the value of S_{l^*}}
$\Omega \leftarrow \Omega - \{l^*\}$;
return The schedule s given by $\{S_l : l \in \{1, \ldots, nm\}\}$

Algorithm 2: The $pFG\&T$

that, due to the uncertainty in task durations, even for $\delta = 1$ we cannot guarantee that the produced schedule will indeed be active when it is actually performed (and tasks have exact durations). We may only say that the obtained fuzzy schedule is *possibly active*.

Particle movement

Velocity update

Particle velocity is traditionally updated depending on the distance to $gbest$ and $pbest$. Instead, this PSO only considers whether the position value x_l^k is greater or smaller than $pbest_l^k$ ($gbest_l$). For any particle, its velocity is represented by an array of the same length as the position array where all the values are in the set $\{-1, 0, 1\}$. The initial values for the velocity array are set randomly. Velocity and particle updating is controlled by the inertia weight w according to Algorithm 3. In the updating process of each particle k and dimension d an element of randomness is introduced, making it dependent on $pbest_d^k$ with probability p_1 and on $gbest_d$ with probability p_2, where $p_1, p_2 \in [0, 1]$ are constants such that $p_1 + p_2 \leq 1$.

Mutation

When adapting PSO to discrete optimisation, there is a risk of getting stuck in local minima when velocity is limited to absolute values (Hu et al. 2003). In order to introduce diversity, after a particle k moves to a new position, we randomly choose a dimension d and then mutate its priority value x_d^k independently of v_d^k. For a problem of size $n \times m$, if $x_d^k < (nm/2)$, x_d^k will take a random value in $[mn - n, mn]$, and $v_d^k = 1$; otherwise (if $x_d^k \geq (nm/2)$), x_d^k will take a random value in $[0, n]$ and $v_d^k = -1$.

Input A particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$, best particle and swarm positions $pbest^k$ and $gbest$, inertia w and updating probabilities p_1, p_2
Output The updated particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$
for each dimension d **do**
generate random value $rand \sim U(0, 1)$;
if $v_d^k \neq 0$ and $rand \geq w$ **then**
$v_d^k \leftarrow 0$;
if $v_d^k = 0$ **then**
generate random value $rand \sim U(0, 1)$;
if $rand \leq p_1$ **then**
if $pbest_d^k \geq x_d^k$ **then** $v_d^k \leftarrow 1$;
else $v_d^k \leftarrow -1$;
generate random value $rand_2 \sim U(0, 1)$;
$x_d^k \leftarrow pbest_d^k + rand_2 - 0.5$;
if $p_1 < rand \leq p_1 + p_2$ **then**
if $gbest_d \geq x_d^k$ **then** $v_d^k \leftarrow 1$;
else $v_d^k \leftarrow -1$;
generate random value $rand_2 \sim U(0, 1)$;
$x_d^k \leftarrow gbest_d + rand_2 - 0.5$;
else
$x_d^k \leftarrow x_d^k + v_d^k$;
return The updated particle position $\mathbf{x}^k$ and velocity $\mathbf{v}^k$;

Algorithm 3: Particle movement

Diversification strategy

In the case that all particles had the same $pbest$ solution, they could be trapped into local optima. To prevent such situation, a diversification strategy is proposed in Sha and Cheng-Yu (2008) in order to keep the different $pbest$ solutions. According to this strategy, the $pbest$ solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far, where N is the size of the swarm. Once any particle generates a new solution, the $pbest$ solutions will be updated as follows: if the new solution equals the makespan of any $pbest$ solution, the latter will be replaced with the new solution; else if the new solution has better makespan than the worst $pbest$ solution and has a different makespan from all $pbest$ solutions, then the worst $pbest$ solution is replaced by the new one; else, the set of N $pbest$ solutions remains unchanged.

Experimental results

For the experimental study, we use the fuzzy open shop instances proposed in González-Rodríguez et al. (2010). These were obtained by fuzzyfying the well-known benchmark from (Brucker et al. 1997), consisting of 6 families, denoted *J3, J4, …, J8*, of sizes 3×3 to 8×8, with 8 or 9 instances each. Each family is divided into three sets of problems *per0*, *per10* and *per20* according to the difference between minimum and maximum workloads of jobs and machines (the number in the name refers to this

difference in percentage). We shall only consider the largest instances, pertaining to the blocks of size 7×7 and 8×8 and compare our results on expected makespan to those of the memetic algorithm (MA) proposed in González-Rodríguez et al. (2010), which combines a genetic algorithm with a local search schema. According to the results reported in González-Rodríguez et al. (2010), this MA outperforms the genetic algorithm alone when run under equivalent running conditions; additionally, on crisp instances of OSP it improves two GAs from Liaw (2000) and Prins (2000) and is competitive with two PSO algorithms from Sha and Cheng-Yu (2008), one of them hybridised with beam search.

For each original deterministic problem instance there are 10 fuzzy versions, generated by transforming the original crisp processing times into symmetric TFNs such that their modal value corresponds to the original duration. To add a due date d_i for each job J_i we follow Andresen et al. (2008): first, we define a generic due date $d_i = TF \times \sum_{j=1}^{m} p_{ij}^2$, where TF is a tightness factor; then, we use two different tightness factors to have the earliest and latest due dates: d_i^1, with $TF = 1.10$, and d_i^2, with $TF = 1.15$.

Given the method for generating due dates, in *per0* instances, where all jobs have the same workload (and consequently the same due date), the makespan and due date satisfaction are strongly correlated objectives, making these instances unsuitable for our multiobjective study. Therefore, the experimental analysis will be conducted on the instances *per10* and *per20* of size 7×7 and 8×8, making it a total of 120 instances, these being the hardest ones to solve when both objectives are considered.

For each problem instance, we have run the PSO algorithm using different objectives: we have considered the three single-objective functions $E[C_{max}]$, AI_{av} and AI_{min} and the four multiobjective functions that result from combining the two choices of aggregation function for due date satisfaction ($AI_{ag} = AI_{min}$ or $AI_{ag} = AI_{av}$) and the two possible priority structures for objectives ($f_1 = C_{max}$, $f_2 = AI_{ag}$ or $f_1 = AI_{ag}$, $f_2 = C_{max}$).

For the multiobjective cases, it is necessary that the target values for both objectives be fixed. As already mentioned, in practice these target values should be given by the DM based on his/her expertise in the problem. Unfortunately, such expert knowledge is not available for the set of synthetic instances used herein. Instead, we emulate the DM and try to gain insight into the problem instances with some preliminary runs of the PSO using $E[C_{max}]$, AI_{av} and AI_{min} as single objectives, using the parameter values proposed in Sha and Cheng-Yu (2008). Then, we set b_1 (resp. b_2 for $1 - AI_{ag}$) equal to the worst value of $E[C_{max}]$ ($1 - AI_{ag}$) across 30 runs of the PSO.

Table 1 Parameter settings

Parameters	Factor level			
	1	2	3	4
Swarm size (N)	60	80	100	120
Inertia weight (w) linearly decreasing [from,to]	[0.9, 0.3]	[0.7, 0.1]	[0.9, 0.7]	[0.7, 0.5]
Guiding probabilities ($gp = (p_1, p_2)$)	(0.7, 0.1)	(0.5, 0.3)	(0.3, 0.5)	(0.1, 0.7)
Delay parameter (δ)	0	0.25	0.75	1

Parameter setting

To ensure that the algorithm yields reliable solutions within a reasonable amount of time, the Taguchi method is used for parameter tuning. Table 1 shows the parameters of our algorithm together with the four possible values (factor levels in the Taguchi terminology) considered for each of them. A caveat in changing the swarm size N is its considerable effect on the algorithm's runtime if a constant number of iterations is considered. Now, it is common in literature to measure the computational effort of a metaheuristic in terms of the number of objective-function evaluations, which is independent of the computer system. This suggests adjusting the number of iterations in such a way that the PSO evaluates roughly the same number of particles for all possible swarm sizes: for $N = 60$, 80, 100 and 120, the number of iterations $Niter$ is set respectively to 3,000, 2,250, 1,800 and 1,500. As for the second parameter, the inertia weight w, it should be linearly decreasing from a starting value, thus stimulating the exploration of the PSO. We consider two possible starting values, 0.9 and 0.7, and two possible slopes, $0.6/Niter$ and $0.2/Niter$, which should allow to analyse the behaviour of the PSO with either more exploration or more exploitation in the last iterations. In consequence, w will be linearly decreasing in four possible intervals, as shown in Table 1. Regarding the guiding probabilities, p_1 and p_2, since their sum must be less or equal to 1, we consider them as a single factor: given the values 0.7, 0.5, 0.3 and 0.1, p_1 and p_2 simultaneously traverse these values in increasing and decreasing order respectively, that is, first $p_1 = 0.7$ and $p_2 = 0.1$, then $p_1 = 0.5$ and $p_2 = 0.3$ and so forth. Thus, we always ensure that the constraint $p_1 + p_2 \leq 1$ holds, while covering a varied sample of values for both probabilities. Finally, for the delay parameter we consider the two extremes values, $\delta = 0$—which in the deterministic case restricts the search to the space of *non-delay* schedules—and $\delta = 1$, together with two intermediate values $\delta = 0.25$ and $\delta = 0.75$.

With a total of four parameters and four factor levels each, the orthogonal array L'_{16} is pertinent for the Taguchi analysis. For every combination of parameter values given by the orthogonal array we run the PSO with the four multiobjective functions: $L(C_{max}, AI_{av})$, $L(C_{max}, AI_{min})$,

Table 2 Orthogonal tabulation and average performance values

Exp.	Parameter levels				Average performance			
	N	w	gp	δ	$L(C_{max}, AI_{min})$	$L(AI_{min}, C_{max})$	$L(C_{max}, AI_{av})$	$L(AI_{av}, C_{max})$
1	1	1	1	1	0.919	0.727	0.524	0.727
2	1	2	2	2	0.051	0.043	0.208	0.178
3	1	3	3	3	1.266	1.344	1.302	1.409
4	1	4	4	4	1.701	1.759	1.661	1.871
5	2	1	2	3	1.084	1.104	1.100	1.015
6	2	2	1	4	1.282	1.423	1.361	1.546
7	2	3	4	1	0.970	0.792	1.035	0.764
8	2	4	3	2	0.324	0.258	0.556	0.186
9	3	1	3	4	1.650	1.693	1.702	1.776
10	3	2	4	3	1.222	1.217	1.193	1.294
11	3	3	1	2	0.293	0.226	0.175	0.004
12	3	4	2	1	1.023	0.775	0.881	0.740
13	4	1	4	2	0.822	0.429	0.441	0.197
14	4	2	3	1	1.055	0.802	1.063	0.953
15	4	3	2	4	2.000	2.000	2.000	1.983
16	4	4	1	3	0.912	0.930	0.920	0.605

$L(AI_{av}, C_{max})$, and $L(AI_{min}, C_{max})$ on a fuzzy instance of each 8×8 problem.

To measure the quality of each configuration we need a value that can consistently combine such heterogeneous values as C_{max}, AI_{av} and AI_{min} while taking into account the lexicographical goal programming nature of the model. First, we consider the distance of each value to its corresponding target, averaged across ten runs of the algorithm and normalised so as to unify scales (notice that such distance is taken to be zero if the target is reached). Let d_1 and d_2 denote, respectively, the normalised distance values for the primary and secondary objective. These values will allow us to characterise the algorithm's performance for the Taguchi analysis as follows: if the first target is reached, i.e. $d_1 = 0$, then the performance is given by d_2 (the distance to the second objective); in the worse case that the primary objective does not reach its target ($d_1 > 0$), then the performance is given by $1 + d_1$. Since $0 \leq d_2 \leq 1$, this guarantees that the algorithm is always considered to perform worse when the target for the primary objective is not reached, as well as discriminating among solutions taking into account how far they are from reaching each target. We have opted for using this performance measure directly, instead of the classical signal-to-noise ratio, in the line of the use of the Taguchi method in Jia and Seo (2013) and Wang et al. (2013) for scheduling problems.

Table 2 shows, for every combination of factor levels in the orthogonal array, the average performance value for each of the four multiobjective functions considered. It is based on these values that we can compute the response value of each parameter and analyse their significance rank. As a summary, Fig. 2 depicts the response values of each parameter for each of the four objective functions, illustrating the effect of the parameter levels on the algorithm's performance. Clearly, the most significant parameter for all objective functions is δ, with a difference between the highest and lowest level over 1.25 of a maximum possible difference of 2.00 (see Fig. 2d). The second most significant parameter is the pair of guiding probabilities (Fig. 2c), although their effect is significantly smaller. Finally, the smallest effect on the performance for all functions is obtained with the swarm size and the inertia weight (see Fig. 2a, b). Additionally, for the two most significant parameters it can be clearly seen that the best level remains the same for all four objective functions. This is not the case for swarm size and inertia weight, where the best levels differ for $L(C_{max}, AI_{av})$ and $L(AI_{av}, C_{max})$; however, the difference is relatively small, 0.079 for swarm size and 0.142 for inertia weight. In consequence, we will take the factor level that performs best for all but one objective functions, this being a good value in all cases.

As a result of this analysis, the parameter setting in what follows will be $\delta = 0.25$, $gp = (p_1, p_2) = (0.7, 0.1)$, w linearly decreasing from 0.7 to 0.1, and swarm size $N = 80$ for all objective functions.

Highest priority for makespan minimisation

Let us first consider the case where C_{max} is the objective with highest priority and let $L(C_{max}, AI_{av})$ and $L(C_{max}, AI_{min})$ denote the resulting multiobjective functions for $AI_{ag} = AI_{av}$ and $AI_{ag} = AI_{min}$ respectively.

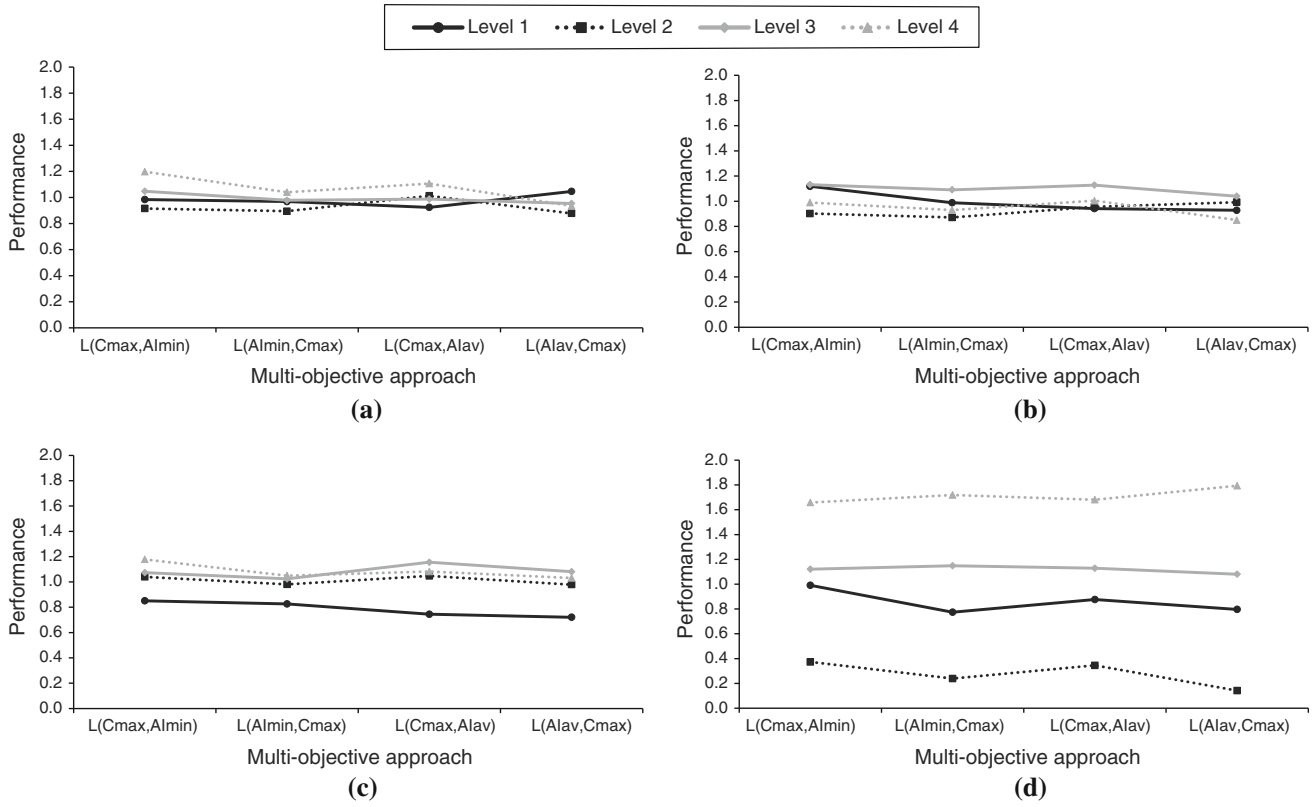


Fig. 2 Average performance of the four multiobjecive-PSO for each parameter level. **a** Swarm size (N). **b** Inertia weight (w). **c** Guiding probabilities (gp). **d** Delay parameter (δ)

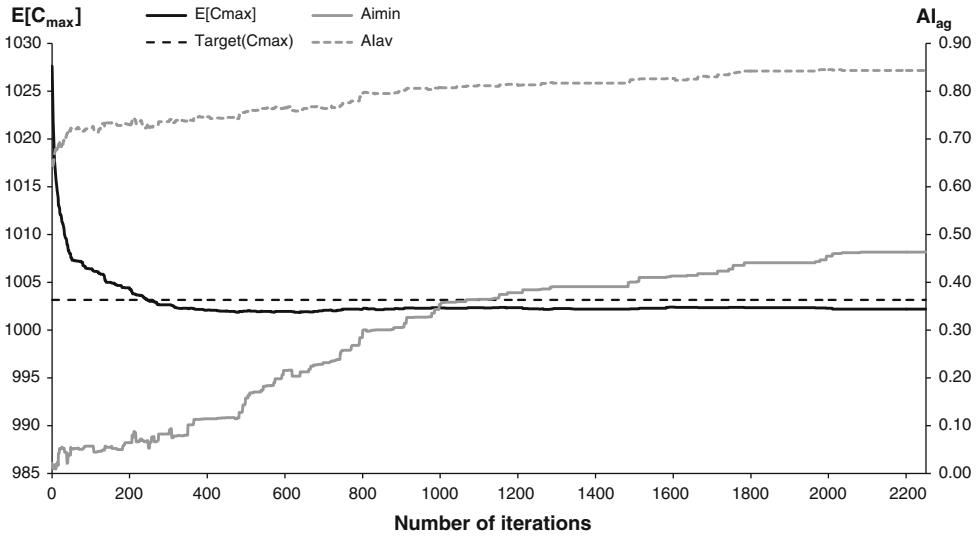


Fig. 3 Evolution of $E[C_{max}]$ and $E[AI_{min}]$ on the $L(C_{max}, AI_{min})$ version for the *J8-per20-1* instance

In order to illustrate the algorithm's convergence, we first focus on a single problem instance. Figure 3 shows the convergence pattern of $L(C_{max}, AI_{min})$ for a fuzzy instance generated from *J8-per20-1*, one of the largest and hardest instances. We can see how the algorithm shows a proper convergence: initially the algorithm minimises the expected makespan $E[C_{max}]$ while the behaviour of AI_{min} is erratic. However, once the algorithm has reached the expected makespan target (around the 250th iteration), it starts maximising AI_{min} instead. We can also observe the evolution of the AI_{av} value and its correlated behaviour w.r.t. AI_{min}. Analogous convergence curves show that the number of iterations can be reduced for 7×7 to 2,100 iterations.

Tables 3 and 4 contain a summary of the results obtained when makespan minimisation has the highest priority. For each objective function used by the PSO they report the

Table 3 Comparison of results for $E[C_{max}]$ highest priority on instances of size 7×7

	Objective	$E[C_{max}]$			AI_{min}			AI_{av}		
		Targ.	Avg	SD	Target	Avg	SD	Target	Avg	SD
J7-per10-0	$L[C_{max}, AI_{min}]$	1035	**1032**	1.655	0.9569	0.9155	0.0992	0.9923	0.9818	0.0225
	$L[C_{max}, AI_{av}]$	1035	**1032**	1.878	0.9569	0.9114	0.0925	0.9923	0.9815	0.0205
	$E[C_{max}]$	–	**1030**	1.946	–	0.8182	0.1157	–	0.9669	0.0226
	AI_{min}	–	1058	9.363	–	**0.9873**	0.0108	–	**0.9968**	0.0035
	AI_{av}	–	1057	10.473	–	**0.9841**	0.0162	–	**0.9971**	0.0029
J7-per10-1	$L[C_{max}, AI_{min}]$	1019	**1017**	1.478	0.9303	0.7562	0.0321	0.9857	0.9352	0.0117
	$L[C_{max}, AI_{av}]$	1019	**1017**	1.664	0.9303	0.7529	0.0340	0.9857	0.9351	0.0137
	$E[C_{max}]$	–	**1017**	1.157	–	0.7502	0.0253	–	0.9346	0.0102
	AI_{min}	–	1049	8.212	–	**0.9723**	0.0212	–	**0.9916**	0.0083
	AI_{av}	–	1047	9.655	–	**0.9653**	0.0298	–	**0.9934**	0.0049
J7-per10-2	$L[C_{max}, AI_{min}]$	1038	**1033**	3.410	0.9358	0.7685	0.1091	0.9817	0.9252	0.0301
	$L[C_{max}, AI_{av}]$	1038	**1034**	2.935	0.9358	0.7495	0.1322	0.9817	0.9347	0.0275
	$E[C_{max}]$	–	**1031**	2.915	–	0.6873	0.1511	–	0.9091	0.0353
	AI_{min}	–	1072	13.352	–	**0.9713**	0.0183	–	**0.9898**	0.0073
	AI_{av}	–	1071	14.196	–	**0.9670**	0.0257	–	**0.9938**	0.0045
J7-per20-0	$L[C_{max}, AI_{min}]$	1001	**1001**	0.294	0.8278	0.3915	0.1116	0.9459	0.7322	0.0683
	$L[C_{max}, AI_{av}]$	1001	**1001**	0.343	0.8278	0.3140	0.1501	0.9459	0.7838	0.0433
	$E[C_{max}]$	–	**1001**	0.145	–	0.1412	0.0888	–	0.6426	0.0695
	AI_{min}	–	1030	7.829	–	**0.8700**	0.0167	–	0.9419	0.0170
	AI_{av}	–	1027	7.934	–	**0.8367**	0.0412	–	**0.9635**	0.0086
J7-per20-1	$L[C_{max}, AI_{min}]$	1032	**1031**	1.561	0.8337	0.3143	0.1404	0.9531	0.7730	0.0560
	$L[C_{max}, AI_{av}]$	1032	**1031**	1.603	0.8337	0.2204	0.1767	0.9531	0.7960	0.0468
	$E[C_{max}]$	–	**1028**	2.329	–	0.0884	0.1144	–	0.7312	0.0435
	AI_{min}	–	1082	9.013	–	**0.8781**	0.0210	–	**0.9550**	0.0147
	AI_{av}	–	1082	10.664	–	**0.8534**	0.0400	–	**0.9698**	0.0069
J7-per20-2	$L[C_{max}, AI_{min}]$	1027	**1024**	2.246	0.8658	0.4303	0.2055	0.9617	0.8225	0.0562
	$L[C_{max}, AI_{av}]$	1027	**1024**	2.974	0.8658	0.3934	0.2045	0.9617	0.8333	0.0517
	$E[C_{max}]$	–	**1021**	2.742	–	0.2688	0.2400	–	0.7972	0.0606
	AI_{min}	–	1074	15.165	–	**0.9158**	0.0211	–	**0.9665**	0.0121
	AI_{av}	–	1076	13.906	–	**0.8979**	0.0412	–	**0.9771**	0.0082

values of $E[C_{max}]$, AI_{av} and AI_{min} in the solution, averaged across the 30 executions of the PSO and the 10 fuzzy instances generated from the same original problem, together with the standard deviations. The average values are shown in bold when they reach the target for the corresponding objective.

A first look at Tables 3 and 4 confirms the strong correlation between the values of AI_{min} and A_{av}, both measuring the overall due-date satisfaction. In most cases, the single-objective version using any of these aggregated values reaches the target value established for the other aggregated measure. That is, when any one of these aggregated measures is optimised, the alternative one is also optimised.

Let us now compare results obtained by the proposed multiobjective approach using $L(C_{max}, AI_{min})$ and $L(C_{max}, AI_{av})$ with the results obtained when optimising a single criterion. For the objective with the highest priority—minimisation of expected makespan—we see that both multiobjective approaches behave similarly to the single-objective function. In particular, they always reach the expected makespan target. Additionally, the multiobjective approach obtains a clear improvement in due-date satisfaction. Indeed, for all instances, AI_{min} values obtained with $L(C_{max}, AI_{min})$ are in average 159 % better than those obtained using $E[C_{max}]$ as single-objective function. There are however remarkable differences in the improvement rate depending on the instance type. For example, due-date satisfaction improves only 8 % for *J7-per10* instances and 16 % for *J8-per10*, but this improvement scales up to 164 and 450 % in *per20* instances of sizes 7×7 and 8×8 respectively.

Table 4 Comparison of results for $E[C_{max}]$ highest priority on instances of size 8×8

	Objective	$E[C_{max}]$			AI_{min}			AI_{av}		
		Targ.	Avg	SD	Target	Avg	SD	Target	Avg	SD
J8-per10-0	$L[C_{max}, AI_{min}]$	1055	**1052**	2.587	0.9026	0.8292	0.0889	0.9756	0.9515	0.0261
	$L[C_{max}, AI_{av}]$	1055	**1052**	2.676	0.9026	0.8017	0.0999	0.9756	0.9545	0.0225
	$E[C_{max}]$	–	**1050**	3.105	–	0.7504	0.1156	–	0.9399	0.0271
	AI_{min}	–	1073	10.333	–	**0.9598**	0.0273	–	**0.9874**	0.0100
	AI_{av}	–	1072	9.972	–	**0.9453**	0.0384	–	**0.9898**	0.0063
J8-per10-1	$L[C_{max}, AI_{min}]$	1036	**1032**	3.391	0.8653	0.7242	0.0987	0.9664	0.9062	0.0360
	$L[C_{max}, AI_{av}]$	1036	**1033**	2.904	0.8653	0.6721	0.1333	0.9664	0.9141	0.0355
	$E[C_{max}]$	–	**1030**	3.744	–	0.6087	0.1316	–	0.8858	0.0375
	AI_{min}	–	1064	11.386	–	**0.9386**	0.0342	–	**0.9801**	0.0151
	AI_{av}	–	1063	12.490	–	**0.9342**	0.0485	–	**0.9881**	0.0088
J8-per10-2	$L[C_{max}, AI_{min}]$	1041	**1036**	5.139	0.8656	0.7958	0.1082	0.9658	0.9357	0.0364
	$L[C_{max}, AI_{av}]$	1041	**1037**	4.521	0.8656	0.7533	0.1275	0.9658	0.9436	0.0330
	$E[C_{max}]$	–	**1033**	5.225	–	0.6735	0.1283	–	0.9108	0.0371
	AI_{min}	–	1065	13.246	–	**0.9330**	0.0339	–	**0.9778**	0.0149
	AI_{av}	–	1062	13.724	–	**0.9279**	0.0472	–	**0.9862**	0.0088
J8-per20-0	$L[C_{max}, AI_{min}]$	1022	**1020**	2.339	0.8644	0.1645	0.1259	0.9668	0.7168	0.0712
	$L[C_{max}, AI_{av}]$	1022	**1020**	2.001	0.8644	0.0771	0.1324	0.9668	0.7728	0.0550
	$E[C_{max}]$	–	**1015**	2.255	–	0.0219	0.0515	–	0.6685	0.0604
	AI_{min}	–	1074	11.741	–	**0.9394**	0.0322	–	**0.9814**	0.0123
	AI_{av}	–	1072	11.650	–	**0.9250**	0.0481	–	**0.9870**	0.0083
J8-per20-1	$L[C_{max}, AI_{min}]$	1003	**1002**	0.800	0.7574	0.2573	0.1703	0.9225	0.7709	0.0716
	$L[C_{max}, AI_{av}]$	1003	**1002**	0.793	0.7574	0.1541	0.1846	0.9225	0.7914	0.0630
	$E[C_{max}]$	–	**1001**	0.862	–	0.0411	0.0790	–	0.6946	0.0589
	AI_{min}	–	1023	9.613	–	**0.8358**	0.0417	–	**0.9288**	0.0255
	AI_{av}	–	1025	11.223	–	**0.7799**	0.0920	–	**0.9513**	0.0147
J8-per20-2	$L[C_{max}, AI_{min}]$	1018	**1017**	2.292	0.8246	0.3463	0.2001	0.9502	0.7961	0.0700
	$L[C_{max}, AI_{av}]$	1018	**1017**	2.191	0.8246	0.2714	0.2040	0.9502	0.8239	0.0567
	$E[C_{max}]$	–	**1014**	2.538	–	0.1269	0.1498	–	0.7593	0.0561
	AI_{min}	–	1065	15.408	–	**0.9026**	0.0347	–	**0.9628**	0.0185
	AI_{av}	–	1062	15.974	–	**0.8879**	0.0611	–	**0.9758**	0.0127

This variability is due to the fact that, as mentioned above, the dependency between $E[C_{max}]$ and AI_{min} is greater for *per10* instances, given the way in which the original benchmark was created. In consequence, for *per10* problems, when the makespan is optimised, due-date satisfaction is also being optimised to a certain extent; however this is not always the case for an arbitrary open shop problem.

Regarding AI_{av} values, they improve 7.2 % when using $L(C_{max}, AI_{av})$. Again there is a remarkable variability in the improvement depending on the family of problems: 2.1 % for *per10* instances and 12.2 % for *per20* instances. It is also tempting to conclude that the gain obtained with $L(C_{max}, AI_{min})$ is much higher than that obtained with $L(C_{max}, AI_{av})$. However, this is only a scale effect. If instead of considering absolute gains, we measure the reduction of the gap between the AI_{min} and the AI_{av} values and their corresponding targets, the multiobjective approach $L(C_{max}, AI_{min})$ is in average over 36 % better than $E[C_{max}]$ and $L(C_{max}, AI_{av})$ is also over 36 % better than $E[C_{max}]$ w.r.t. the corresponding secondary target. In any case, it is worth noticing that for *per10* instances $L(C_{max}, AI_{min})$ performs better than $L(C_{max}, AI_{av})$ whereas for *per*20 instances the best performance corresponds to $L(C_{max}, AI_{av})$ in terms of gap reduction w.r.t. its secondary target. A possible explanation is that AI_{min} is a more demanding aggregation operator. If it is relatively "easy" to satisfy the due dates for all jobs (at least to a certain extent), then $0 < AI_{min} \leq AI_{av}$ and AI_{min} will probably provide a better guide for maximising due date satisfaction. However, as long as it is likely that one of the due dates is not satisfied at all in schedules

Table 5 Comparison between PSO and MA in terms of $E[C_{max}]$

Problem	Target	MA	PSO	
		$E[C_{max}]$	$L(C_{max}, AI_{min})$	$L(C_{max}, AI_{av})$
J7-per10-0	1,035	1,066	1,032	1,032
J7-per10-1	1,019	1,052	1,017	1,017
J7-per10-2	1,038	1,067	1,033	1,034
J7-per20-0	1,001	1,004	1,001	1,001
J7-per20-1	1,032	1,044	1,031	1,031
J7-per20-2	1,027	1,042	1,024	1,024
J8-per10-0	1,055	1,083	1,052	1,052
J8-per10-1	1,036	1,066	1,032	1,033
J8-per10-2	1,041	1,071	1,036	1,037
J8-per20-0	1,022	1,037	1,020	1,020
J8-per20-1	1,003	1,014	1,002	1,002
J8-per20-2	1,018	1,035	1,017	1,017

with good makespan values (as is the case for *per20* problems), then $AI_{min} = 0$ with high probability, thus providing a poor guide for the optimisation process.

Finally, the correlation between both aggregation operators is further confirmed if we look at the behaviour of AI_{av} in case of $L(C_{max}, AI_{min})$ and AI_{min} in case of $L(C_{max}, AI_{av})$: both multiobjective approaches significantly reduce the alternative due-date objective, with a gap-improvement of approximately 26 % in both cases.

Let us now compare the multiobjective PSO using $L(C_{max}, AI_{av})$ or $L(C_{max}, AI_{min})$ with the single-objective memetic algorithm (MA) from González Rodríguez et al. (2010) in terms of expected makespan minimisation. Table 5 contains the expected makespan results for each method—MA optimising only $E[C_{max}]$, PSO with $L(C_{max}, AI_{av})$ and PSO with $L(C_{max}, AI_{min})$—with values averaged across the 10 instances of each size and 30 runs of the algorithm. Clearly, the PSO with both multiobjective functions $L(C_{max}, AI_{av})$ and $L(C_{max}, AI_{min})$ compares favourably with the single-objective MA in terms of makespan values. Indeed, the multiobjective PSO reduces $E[C_{max}]$ values about 2.25 % (slightly over 3 % for *per10* instances and slightly below 1.5 % for *per20* instances), with no significant differences between different problem sizes or different aggregated measures for due-date satisfaction. This reduction may not seem very important in absolute values. However, on a closer look we can see that the MA never reaches the expected makespan target value, whereas the multiobjective PSO reaches this target in all instances. We can conclude that our multiobjective PSO outperforms the previous single-objective algorithm when it comes to optimising the objective with the highest priority (makespan), while also optimising the secondary objective.

Highest priority for due-date satisfaction

We now consider the alternative priority structure where due-date satisfaction becomes the primary objective; let $L(AI_{ag}, C_{max})$ denote the resulting lexicographic goal programming multiobjective function. If we now compare each $L(AI_{ag}, C_{max})$ with the corresponding aggregated due-date satisfaction value AI_{ag} (AI_{min} or AI_{av}), the results are analogous to the case where makespan was the first objective. In all instances $L(AI_{ag}, C_{max})$ reaches the corresponding target for due-date satisfaction value whereas the gap between the expected makespan and its target value is reduced 36 % in average when $AI_{ag} = AI_{min}$ and 40 % in the case that $AI_{ag} = AI_{av}$. Figure 4 shows the $E[C_{max}]$ values (averaged across the 10 fuzzy instances of every original problem and the 30 executions of the PSO algorithm) obtained with AI_{min}, AI_{av} and the corresponding multiobjective functions on each family of problems. It also depicts the $E[C_{max}]$ target values for each family. We can clearly appreciate how the expected makespan behaves better in the multiobjective approach. We can also observe that AI_{av} used as single objective function obtains in general slightly better $E[C_{max}]$ values than the alternative AI_{min}. Also, its multiobjective counterpart $L(AI_{av}, C_{max})$ performs slightly better (in terms of makespan minimisation) than $L(AI_{min}, C_{max})$. The explanation, again, lies in the fact that AI_{min} is a more pessimistic aggregator of individual job due-date satisfaction. The figure also illustrates that, as above, the solutions are in general closer to the target values for *per10* instances than for *per20* ones.

Conclusions and future work

We have proposed a multiobjective approach for solving the open shop scheduling problem with uncertain durations and flexible due dates modelled using fuzzy sets. We have adopted a lexicographic goal programming framework to deal with the multiple objectives of minimising the project's makespan and maximising due-date satisfaction. The resulting problem has been solved by adapting a particle swarm optimisation algorithm to the hierarchical multiobjective framework. The experimental results, on fuzzy instances of well-known benchmark problems, illustrate the potential of our proposal. In general, the multiobjective approaches perform as well as their single-objective counterparts when it comes to optimising the objective with the highest priority, reaching the target levels in all cases. Additionally, the multiobjective approaches greatly improve on the secondary objective. Also, the multiobjective PSO algorithm compares favourably to a memetic algorithm from the literature in terms of makespan minimisation, when this is the objective with the highest priority.

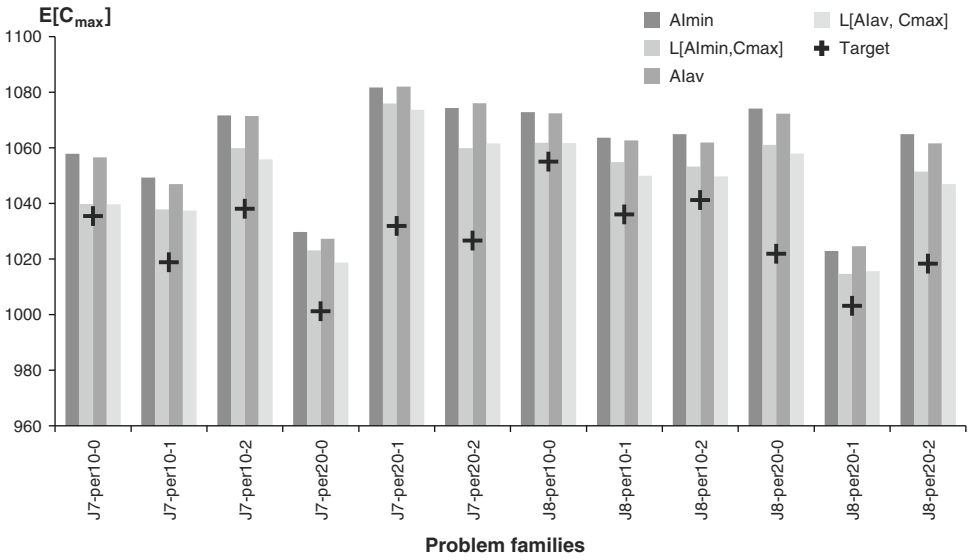


Fig. 4 Average $E[C_{max}]$ values obtained with AI_{min}, AI_{av} and the corresponding multiobjective $L(AI_{min}, C_{max})$ and $L(AI_{av}, C_{max})$

In the future, we would like to contemplate an alternative approach to multiobjective optimisation, appropriate for the case when no priority structure among multiple objectives can or needs to be established. We would like to explore the known relationships between lexicographic and Pareto optimality, as well as extending the PSO algorithm to directly work with sets of non-dominated solutions. We would also like to adapt the PSO algorithm to other scheduling problems with uncertainty, such as job shop or resource-constrained project scheduling.

Acknowledgments We would like to thank the anonymous referees for their insightful and constructive comments. This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02 and MTM2010-16051 and by the Principality of Asturias (Spain) under grant Severo Ochoa BP13106.

References

Alcaide, D., Rodriguez-Gonzalez, A., & Sicilia, J. (2006). A heuristic approach to minimize expected makespan in open shops subject to stochastic processing times and failures. *International Journal of Flexible Manufacturing Systems*, *17*, 201–226.

Andresen, M., Bräsel, H., Mörig, M., Tusch, J., Werner, F., & Willenius, P. (2008). Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Mathematical and Computer Modelling*, *48*, 1279–1293.

Bellman, R. E., & Zadeh, L. A. (1970). Decision-making in a fuzzy environment. *Management Science*, *17*(4), 141–164.

Belmecheri, F., Prins, C., & Yalaoui, F. L. A. (2013). Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *Journal of Intelligent Manufacturing*, *24*(4), 775–789.

Blum, C. (2005). Beam-ACO—hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, *32*(6), 1565–1591.

Bouveret, S., & Lemaître, M. (2009). Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, *173*, 343–364.

Brucker, P., Hunrink, J., Jurisch, B., & Wöstmann, B. (1997). A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, *76*, 43–59.

Celano, G., Costa, A., & Fichera, S. (2003). An evolutionary algorithm for pure fuzzy flowshop scheduling problems. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *11*, 655–669.

Chen, S. M., & Chang, T. H. (2001). Finding multiple possible critical paths using fuzzy PERT. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, *31*(6), 930–937.

Coshall, J. T., & Charlesworth, R. (2011). A management orientated approach to combination forecasting of tourism demand. *Tourism Management*, *32*, 759–769.

Diaz-Balteiro, L., & Romero, C. (2008). Making forestry decisions with multiple criteria: A review and an assessment. *Forest Ecology and Management*, *255*, 3222–3241.

Dubois, D. (2011). The role of fuzzy sets in decision sciences: Old techniques and new directions. *Fuzzy Sets and Systems*, *184*, 3–28.

Dubois, D., & Prade, H. (1986). *Possibility theory: An approach to computerized processing of uncertainty*. New York, NY, USA: Plenum Press.

Dubois, D., Fargier, H., & Prade, H. (1995). Fuzzy constraints in job-shop scheduling. *Journal of Intelligent Manufacturing*, *6*, 215–234.

Dubois, D., Fargier, H., & Fortemps, P. (2003). Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, *147*, 231–252.

Ehrgott, M. (2005). *Multicriteria optimization* (2nd ed.). Berlin: Springer.

Ehrgott, M., & Gandibleux, X. (2000). A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, *22*, 425–460.

Farahani, R. Z., SteadieSeifi, M., & Asgari, N. (2010). Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, *34*, 1689–1709.

Fortemps, P. (1997). Jobshop scheduling with imprecise durations: A fuzzy approach. *IEEE Transactions of Fuzzy Systems*, *7*, 557–569.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, *8*, 487–503.

Gonçalves, J., Magalhaes Mendes, J. J., & Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, *167*, 77–95.

González Rodríguez, I., Puente, J., Vela, C. R., & Varela, R. (2008). Semantics of schedules for the fuzzy job shop problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *38*(3), 655–666.

González-Rodríguez, I., Palacios, J.J., Vela, C.R., & Puente, J. (2010). Heuristic local search for fuzzy open shop scheduling. In: *Proceedings IEEE international conference on fuzzy systems, FUZZ-IEEE2010* (pp. 1858–1865). IEEE.

González Rodríguez, I., Vela, C. R., & Puente, J. (2010). A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. *Journal of Intelligent Manufacturing*, *21*, 65–73.

Graham, R., Lawler, E., & Lenstra, J. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *4*, 287–326.

Guéret, C., & Prins, C. (1998). Classical and new heuristics for the open-shop problem: A computational evaluation. *European Journal of Operational Research*, *107*, 306–314.

Guiffrida, A. L., & Nagi, R. (1998). Fuzzy set theory applications in production management research: A literature survey. *Journal of Intelligent Manufacturing*, *9*, 39–56.

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

Hu, X., Eberhart, R.C., & Shi, Y. (2003). Swarm intelligence for permutation optimization: A case study of n-queens problem. In: *Swarm intelligence symposium, 2003. SIS'03. Proceedings of the 2003 IEEE* (pp. 243–246). IEEE.

Jia, Q., & Seo, Y. (2013). An improved particle swarm optimization for the resource-constrained project scheduling problem. *International Journal of Advanced Manufacturing Technology* , *67*(9–12), 2627–2638.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *IEEE international conference on neural networks* (pp. 1942–1948). New Jersey: IEEE Press.

Kim, B. I., & Son, S. J. (2012). A probability matrix based particle swarm optimization for the capacitated vehicle routing problem. *Journal of Intelligent Manufacturing*, *23*, 1119–1126.

Lei, D. (2008). Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, *37*, 157–165.

Liaw, C. F. (1999). A tabu search algorithm for the open shop scheduling problem. *Computers and Operations Research*, *26*, 109–126.

Liaw, C. F. (2000). A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, *124*, 28–42.

Liberatore, F., Ortuño, M.T., Tirado, G., Vitoriano, B., & Scaparra, M. P. (2014). A hierarchical compromise model for the joint optimization of recovery operations and distribution of emergency goods in humanitarian logistics. *Computers & Operations Research*, *42*, 3–13.

Liu, B., & Liu, Y. K. (2002). Expected value of fuzzy variable and fuzzy expected value models. *IEEE Transactions on Fuzzy Systems*, *10*, 445–450.

Marinakis, Y., & Marinaki, M. (2013). Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Computing*, *17*(7), 1159–1173.

Naderi, B., Fatemi Ghomi, S. M. T., Aminnayeri, M., & Zandieh, M. (2011). A study on open shop scheduling to minimise total tardiness. *International Journal of Production Research*, *49*(15), 4657–4678.

Niu, Q., Jiao, B., & Gu, X. (2008). Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. *Applied Mathematics and Computation*, *205*, 148–158.

Noori-Darvish, S., Mahdavi, I., & Mahdavi-Amiri, N. (2012). A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due-dates. *Applied Soft Computing*, *12*, 1399–1416.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., & Puente, J. (2011). Particle swarm optimisation for open shop problems with fuzzy durations. In: *Proceedings of IWINAC 2011, Part I, Springer, Lecture Notes in Computer Science* (Vol. 6686, pp. 362–371).

Panahi, H., Rabbani, M., & Tavakkoli-Moghaddam, R. (2008). Solving an open shop scheduling problem by a novel hybrid multi-objective ant colony optimization. In: *Eighth international conference on hybrid intelligent systems* (pp. 320–325). IEEE.

Pasandideh, S. H. R., Niaki, S. T. A., & Hajipour, V. (2013). A multi-objective facility location model with batch arrivals: two parameter-tuned meta-heuristic algorithms. *Journal of Intelligent Manufacturing*, *24*(2), 331–348.

Petrovic, S., Fayad, S., & Petrovic, D. (2008). Sensitivity analysis of a fuzzy multiobjective scheduling problem. *International Journal of Production Research*, *46*(12), 3327–3344.

Pinedo, M. L. (2008). *Scheduling, theory, algorithms, and systems* (3rd ed.). Berlin: Springer.

Prins, C. (2000). Competitive genetic algorithms for the open-shop scheduling problem. *Mathematical Methods of Operations Research*, *52*, 389–411.

Puente, J., Vela, C. R., & González-Rodríguez, I. (2010). Fast local search for fuzzy job shop scheduling. In: *Proceedings of ECAI 2010* (pp. 739–744). IOS Press.

Puente, J., Vela, C. R., González-Rodríguez, I., Rodríguez, L. J., & Palacios, J. J. (2013). GRASPing examination board assignments for university-entrance exams. In: *IEA-AIE 2013, Proceedings of, Springer, Lecture notes in computer science* (Vol 7906, pp. 171–180).

Romero, C. (2001). Extended lexicographic goal programming: a unifying approach. *Omega*, *29*, 63–71.

Sakawa, M., & Kubota, R. (2000). Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy due-date through genetic algorithms. *European Journal of Operational Research*, *120*, 393–407.

Sha, D., Lin, H. H., & Hsu, C. (2010). A modified particle swarm optimization for multi-objective open shop scheduling. In: *Proceeding of the international multiconference of engineers and computer scientists*, Vol 3.

Sha, D. Y., & Cheng-Yu, H. (2008). A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, *35*, 3243–3261.

Tamiz, M., Jones, D., & Romero, C. (1998). Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operations Research*, *111*, 569–581.

Tassopoulos, I. X., & Beligiannis, G. N. (2012). Using particle swarm optimization to solve effectively the school timetabling problem. *Soft Computing*, *16*, 1229–1252.

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

Vijay Chakaravarthy, G., Marimuthu, S., & Naveen Sait, A. (2013). Performance evaluation of proposed differential evolution and particle swarm optimization algorithms for scheduling m-machine flow shops with lot streaming. *Journal of Intelligent Manufacturing*, *24*, 175–191.

Wang, L., Zhou, G., Xu, Y., & Min, L. (2013). A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. *International Journal of Production Research*, *51*(2), 3593–3608.

Zheng, Y., Li, Y., & Lei, D. (2011). Swarm-based neighbourhood search for fuzzy job shop scheduling. *International Journal of Innovative Computing and Applications*, *3*(3), 144–151.

Chapter 8

Most relevant conference papers

In this chapter we include the conference papers that have been presented in conferences that are ranked in the ERA or CORE conference rankings. For the sake of clarity, before each paper we shall include again its data.

8.1 Schedule generation schemes for job shop problems with fuzziness

In this section, we include the following publication.

- **Title:** Schedule generation schemes for job shop problems with fuzziness.
- **Conference:** ECAI 2014. 21st European Conference on Artificial Intelligence.
- **Date:** August 2014.
- Place: Prague (Czech Republic).
- Conference Ranking:
  - ERA 2010: **Rank A**
  - CORE 2014: **Rank A**

This publications contains pieces of work described in Section 4.1.1.

Palacios, J.J., Vela, C.R., González-Rodríguez, I., Puente, J.: Schedule generation schemes for job shop problems with fuzziness. In: Proceedings of ECAI 2014. Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 687-692. IOS Press (2014). Doi: 10.3233/978-1-61499-419-0-687.

ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-687

Schedule Generation Schemes for Job Shop Problems with Fuzziness

Juan José Palacios[1] and **Camino R. Vela**[2] and **Inés González-Rodríguez**[3] and **Jorge Puente**[4]

Abstract. We consider the job shop scheduling problem with fuzzy durations and expected makespan minimisation. We formally define the space of *semi-active* and *active fuzzy schedules* and propose and analyse different schedule-generation schemes (SGSs) in this fuzzy framework. In particular, we study dominance properties of the set of schedules obtained with each SGS. Finally, a computational study illustrates the great difference between the spaces of active and the semi-active fuzzy schedules, an analogous behaviour to that of the deterministic job shop.

1 Introduction

Scheduling is a research field of great importance, involving complex combinatorial constraint-satisfaction optimisation problems and with relevant applications in industry, finance, welfare, education, etc [13]. To enhance the applicability of scheduling, part of the research in this field has been devoted to modelling the uncertainty and vagueness pervading real-world situations, with great diversity of approaches [9]. In particular, fuzzy sets have been used in different manners, ranging from representing incomplete or vague states of information to using fuzzy priority rules with linguistic qualifiers or preference modelling [4]. They are also emerging as an interesting tool for improving solution robustness, a much-desired property in real-life applications [10, 15].

A key issue in scheduling is the definition of subsets of feasible solutions and the study of their properties, in particular, whether they are guaranteed to contain at least one optimal solution. For classical deterministic scheduling, the best known are the sets of semi-active, active and non-delay (or dense) schedules, and it is common practice to restrict the search to some of these subspaces. This is achieved using schedule generation schemes (SGSs) which, given an operation processing order, produce a schedule (an assignment of start times to all operations) based on this ordering. SGSs are extensively used in (meta)heuristic procedures and can also be viewed as branching schemes of exact search methods. It is essential to have proper SGSs, to know which is the set of schedules obtainable with a given SGS and how it relates with the schedule categories and to study the theoretical ability of any SGS to reach the optimum. Surprisingly enough, although we can find some ad-hoc extensions of deterministic SGSs to the fuzzy framework, no effort has been made to give precise definitions for types of schedules when fuzzy times are involved, nor have SGSs been defined and studied systematically in this framework.

[1] University of Oviedo, Spain, email: palaciosjuan@uniovi.es
[2] University of Oviedo, Spain, email: crvela@uniovi.es
[3] University of Cantabria, Spain, email: ines.gonzalez@unican.es
[4] University of Oviedo, Spain, email: puente@uniovi.es

In this paper, we intend to fill the existing gap in the literature. Inspired by the work of [1],[18],[19] for different deterministic scheduling problems, we provide a formal definition of the concepts of semi-active and active schedules as well as several SGSs for the fuzzy job shop problem with expected makespan minimisation (FJSP). We shall study the relationship between different types of schedules and the sets generated by SGSs, and investigate whether such sets necessarily contain one optimal schedule. Finally, we shall provide computational results to compare the different SGSs.

2 The Fuzzy Job Shop Problem

The *job shop scheduling problem*, or *JSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job J_j, $j = 1, \ldots, n$, consists of $m_j \leq m$ tasks or operations $(o(j,1), \ldots, o(j,m_j))$ to be sequentially scheduled (*precedence constraints*). Each task $o(j,l)$ needs the exclusive use of a machine $\mu_{o(j,l)}$ for its whole processing time $d_{o(j,l)} > 0$ (*capacity constraints*). There is no preemption, i.e. all operations must be processed without interruption and no reentrance, i.e., operations within a job are processed by different machines: $\forall j$, $\mu_{o(j,l)} \neq \mu_{o(j,l')}, \forall l \neq l'$. A solution to this problem is a *schedule*–an allocation of starting times for all operations – which is *feasible*, in the sense that all constraints hold, and is also optimal according to some criterion. Here, we consider the objective of minimising the makespan C_{max}, which is the time lag from the start of the first operation until the end of the last one. This is the most commonly considered regular (non-decreasing with task processing times) performance measure.

In order to simplify notation, we assume w.l.o.g. that tasks are indexed from 1 to $N = \sum_{j=1}^{n} m_j$, so we can refer to a task $o(j,l)$ by its index $o = \sum_{i=1}^{j-1} m_i + l$. The machine, duration, starting time and completion time of a task o are denoted respectively μ_o, d_o, S_o and C_o (notice the last two depend on the schedule). The set of tasks is denoted $O = \{0, 1, \ldots, N\}$, where 0 is an initial dummy operation, taken to be the first operation of each job (i.e. $o(j,0) = 0, \forall j = 1, \ldots, n$) and such that $d_0 = S_0 = 0$. Finally, a feasible schedule will be represented by the vector of operation starting times $\boldsymbol{t}$, where $t_o = S_o$ is the starting time of operation $o \in \{1, \ldots, N\}$ (in our case, a triangular fuzzy number, as described below).

2.1 Uncertain Durations as Fuzzy Numbers

In real-life applications, it is often the case that the exact time it takes to process a task is not known in advance and only some uncertain knowledge about the duration is available. The crudest representation

for uncertain processing times would be a human-originated confidence interval. If some values appear to be more plausible than others, a natural extension is a fuzzy interval or fuzzy number (cf. [5]). A *fuzzy interval* A is a fuzzy set on the reals with membership function $\mu_A : \mathbb{R} \to [0,1]$ such that its α-cuts $A_\alpha = \{r \in \mathbb{R} : \mu_A(r) \geq \alpha\}$, $\alpha \in (0,1]$, are intervals (bounded or not). The *support* of A is $A_0 = \{r \in \mathbb{R} : \mu_A(r) > 0\}$ and the *modal values* are those in A_1. A *fuzzy number* B is a fuzzy interval whose α-cuts are closed intervals, denoted $B_\alpha = [\underline{b}_\alpha, \overline{b}_\alpha]$, with compact support and unique modal value.

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. A TFN A is denoted $A = (a^1, a^2, a^3)$ and its membership function takes the following triangular shape:

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

In the job shop, we essentially need two operations on fuzzy numbers, the sum and the maximum. In principle, these are obtained by extending the corresponding operations on real numbers using the *Extension Principle*. However, computing the sum or maximum of two fuzzy numbers is cumbersome if not intractable in general, because it requires evaluating two sums or two maxima for every value $\alpha \in [0,1]$. For the sake of simplicity and tractability of numerical calculations, we follow [6] and approximate the results of these operations by linear interpolation on the three defining points of each TFN (an approach also taken, for instance, in [3] or [11]). The approximated sum coincides with the actual sum, so for any pair of TFNs A and B:

$$A + B = (a^1 + b^1, a^2 + b^2, a^3 + b^3) \quad (2)$$

As for the maximum, for any two TFNs A, B, if $F = \max(A, B)$ denotes their maximum and $G = (\max\{a^1, b^1\}, \max\{a^2, b^2\}, \max\{a^3, b^3\})$ the approximated value, it holds that $\forall \alpha \in [0,1]$, $\underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha$. The approximated maximum G is thus a TFN which artificially increases the value of the actual maximum F, maintaining the support and modal value, that is, $F_0 = G_0$ and $F_1 = G_1$. This approximation can be trivially extended to the case of more than two TFNs.

The membership function μ_A of a fuzzy number A can be interpreted as a possibility distribution on the reals; this allows to define the *expected value* of a fuzzy number [8], given for a TFN A by

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

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 [4]. It induces a total ordering $\leq_E$ in the set of fuzzy intervals [6], where for any two fuzzy intervals A, B $A \leq_E B$ if and only if $E[A] \leq E[B]$. Clearly, for any two TFNs A and B, if $\forall i \in \{1,2,3\}, a^i \leq b^i$, then $A \leq_E B$.

2.2 Problem Statement

In analogy to the original problem, our objective is to find a fuzzy schedule with optimal makespan. However, neither the maximum nor its approximation define a total ordering in the set of TFNs. Using ideas similar to stochastic scheduling, we use the total ordering provided by the expected value, considering that the objective is to minimise the expected makespan $E[C_{max}]$. The resulting problem will be denoted $J|fuzzd_o|E[C_{max}]$ and can be formulated as follows:

$$\min E[C_{max}(S)] = E[\max_{1 \leq j \leq n} C_{o(j,m)}] \quad (4)$$

subject to:

$$\forall i \; C_o^i = S_o^i + d_o^i, \; \forall o \in O \quad (5)$$

$$\forall i \; S_{o(j,l)}^i \geq C_{o(j,l-1)}^i, \; 1 \leq l \leq m_j, 1 \leq j \leq n \quad (6)$$

$$\forall i \; S_o^i \geq C_{o'}^i \; \vee \; \forall i \; S_{o'}^i \geq C_o^i, \forall o \neq o' \in O : \mu_o = \mu_{o'} \quad (7)$$

Where $\forall i$ represents $\forall i \in \{1,2,3\}$.

Clearly, the FJSP is NP-hard, since setting all processing times to crisp figures yields the classical JSP.

Notice that the schedule is fuzzy in the sense that the starting, processing and completion times of each task are fuzzy numbers, seen as possibility distributions on the actual values they may take. However, there is no uncertainty regarding the order in which operations must be processed: once the starting times have been allocated, they establish clear orderings among operations in the same machine.

Algorithm 1 SGS Generic Algorithm

Require: an instance of $J|fuzzp_o|E[C_{max}]$, *P*, and a task order, π
Ensure: a schedule $\boldsymbol{t}$ for *P* according to π
1. $A = \{o(j,1) : 1 \leq j \leq n\}$
while A $\neq \emptyset$ **do**
2. compute the eligible set E $\subseteq$ A
3. select $o(j^*, l^*) = \arg\min\{\pi_{o(j,l)} : o(j,l) \in E\}$
4. $S_{o(j^*,l^*)} = ES_{o(j^*,l^*)}$
5. $A = A - \{o(j^*, l^*)\}(\cup\{o(j^*, l^* + 1)\}$ if $l^* < m_{j^*})$
end while
return $\boldsymbol{t}$, where $t_i = S_i, i = 1, \ldots, N$

3 Schedule Generation Schemes

A general framework for a SGS is provided in Algorithm 1: given a task order π (which can be interpreted as a priority vector), it allows to build different types of schedules, depending on the actual instantiation of some of its actions.

The generic algorithm builds the schedule in N iterations. At each iteration, the SGS computes a set of *eligible* tasks, *E*, which is a subset of the set of *available* tasks, *A*, containing the tasks that are candidates to be scheduled at the current iteration. In steps 3 and 4 the SGS selects the operation $o(j^*, l^*) \in E$ with the highest priority according to π and computes its *Earliest feasible Starting time* (ES) based on an *Appending* (ESA) or *Insertion* (ESI) strategy.

This framework covers a wide range of interesting SGSs, as we shall see in the sequel. However, it does not comprise all possible SGSs, in particular those where a non-available operation may be selected for scheduling or where starting times may be later modified in the schedule-building process.

3.1 Computing Earliest Feasible Starting Times

In the SGS generic algorithm, once a task has been selected, it is scheduled at its earliest feasible starting time ES. Depending on how this value is computed, we distinguish between *appending* SGS and *insertion* SGS.

In an appending scheme, an unscheduled task can be scheduled only after all tasks that have been previously scheduled in its machine

and its job. Formally, let $o(j,l)$ be the task for which the starting time must be computed, let $k = \mu_{o(j,l)}$ be the machine required by $o(j,l)$ and let $\lambda(k) \in O$ denote the latest task scheduled (at the current iteration) on machine k. Then, $ESA_{o(j,l)}$ can be computed in $\mathcal{O}(1)$ as follows:

$$ESA_{o(j,l)} = \max\{C_{\lambda(k)}, C_{o(j,l-1)}\} \tag{8}$$

In an insertion scheme, an unscheduled task $o(j,l)$ may be scheduled before tasks already scheduled on its machine, provided that the starting time of each of these tasks does not change. Hence, the scheme searches for the first insertion position where the selected task can fit without delaying the subsequent tasks already scheduled. Taking into account the definition of starting and completion times in the FJSP, the insertion position must fit "in each component" of the TFN. More precisely, let η_k be the number of tasks scheduled on machine k and let $\sigma_k = (0, \sigma(1,k), \ldots, \sigma(\eta_k,k))$, with $\sigma(\eta_k,k) = \lambda(k)$ denote the partial processing order of tasks already scheduled in machine k. If a position q, $0 \leq q < \eta_k$, is such that for all $i \in \{1,2,3\}$:

$$\max\{C^i_{\sigma(q,k)}, C^i_{o(j,l-1)}\} + d^i_{o(j,l)} \leq S^i_{\sigma(q+1,k)} \tag{9}$$

then q is a *feasible insertion position for operation* $o(j,l)$ between operations $\sigma(q,k)$ (possibly the dummy first task 0) and $\sigma(q+1,k)$. If there exists at least one position q verifying (9), we take $\overline{q} = \min_{q \text{ verifying (9)}} q$ and

$$ESI_{o(j,l)} = \max\{C^i_{\sigma(\overline{q},k)}, C^i_{o(j,l-1)}\} \tag{10}$$

Otherwise $ESI_{o(j,l)} = ESA_{o(j,l)}$

The earliest starting time of an eligible task in an insertion scheme can be computed in $\mathcal{O}(m)$, since there are at most $m-1$ tasks scheduled on machine $k = \mu_{o(j,l)}$

4 Schedule Categories and SGSs

The set Σ of feasible solutions usually constitutes a huge search space. Hence, it is common in deterministic scheduling to restrict the search to smaller subsets of Σ which define categories of schedules. Among these, the best known are the sets of semiactive, active and non-delay schedules [13]. A set of schedules of a given category is said to be *dominant* w.r.t. an objective function if it contains at least one optimal solution. In the following, we will always consider dominance w.r.t. expected makespan. A SGS is *complete* for a category if it can be used to generate all the schedules of this category.

4.1 Semi-active Schedules

For deterministic shop scheduling, the definition of semi-active schedules is based on the concept of *local left shift*, a change that consists in "moving an operation block to the left on the Gantt chart while preserving the operation sequences" [18]. This can be interpreted in the fuzzy case as follows.

Definition 1 *Let $\boldsymbol{t}$ be a feasible schedule, then a* left shift *of a task o in $\boldsymbol{t}$ is a move giving another feasible schedule $\boldsymbol{s}$ where*

$$\begin{gathered}\exists i \in \{1,2,3\} : s^i_o = t^i_o - 1 \wedge \forall j \neq i \; s^j_o = t^j_o \\ s_{o'} = t_{o'} \forall o' \in O - \{o\}\end{gathered} \tag{11}$$

Definition 2 *A semi-active schedule is a feasible schedule in which none of the tasks can be locally left-shifted.*

Notice that for any feasible schedule that is not semi-active, there exists a sequence of local left shifts that produces a semi-active schedule without increasing any of the makespan components, C^i_{max}, and, therefore, without increasing the expected makespan. Hence, the set of semi-active schedules is strictly contained in the set of feasible schedules and is dominant for the FJSP with expected makespan minimization.

We are now in position of defining a SGS that produces semi-active schedules.

Definition 3 SemiActiveSGS *is an appending SGS where the eligible set E equals the set of available operations A, i.e., $E = A$.*

Theorem 1 SemiActiveSGS *generates only semi-active schedules and it is complete in this set.*

Sketch of Proof Schedules generated by SemiActiveSGS are always semi-active because every operation $o \in O$ ESA_o is assigned the least possible value, so it is unfeasible to reduce any of its components, and no local left-shift is available. On the other hand, given a semiactive schedule $\boldsymbol{t}$, we take π to be the topological order from the constraint graph that represents the precedence and capacity constraints between operations in $\boldsymbol{t}$ (this order always exists because, being $\boldsymbol{t}$ feasible, the graph is acyclic). For any operation ordering π, SemiActiveSGS(π) schedules all operations following exactly the same order π, so in particular SemiActiveSGS(π) = $\boldsymbol{t}$. □

Corollary 2 *The set of schedules generated by* SemiActiveSGS *is dominant.*

4.2 Active schedules

Given a feasible schedule $\boldsymbol{t}$ where no local left shifts are possible, a *global left shift* of an operation o is a move that allows "to start an operation earlier without delaying any other operation" [18]. More formally:

Definition 4 *Let $\boldsymbol{t}$ be a feasible schedule, then a* left shift *of an operation o in $\boldsymbol{t}$ is a move giving another feasible schedule $\boldsymbol{s}$ where:*

$$\begin{gathered}\exists i \in \{1,2,3\} : s^i_o < t^i_o \wedge \forall j \neq i \; s^j_o \leq t^j_o \\ s_{o'} = t_{o'} \forall o' \in O - \{o\}\end{gathered} \tag{12}$$

Definition 5 *Let $\boldsymbol{t}$ be a feasible schedule, then a* global left shift *of a task o in $\boldsymbol{t}$ is a left shift of o that is not obtainable by a sequence of local left shifts.*

Definition 6 *An active schedule is a feasible schedule where no global or local left shift lead to a feasible schedule.*

Notice that an active schedule contains no feasible insertion positions, because if an insertion position existed, this would allow for at least one global left shift. Also, given any semi-active but non-active schedule, it is always possible to perform a sequence of global left shift moves in order to build an active schedule without increasing any component of the starting times of tasks. Hence, the set of active schedules is a strict subset of the semi-active ones and remains dominant.

In the following we study different ways of generating active schedules, starting with a straightforward insertion version of the general SGS algorithm.

Definition 7 ActiveSGS *is an insertion SGS where the eligible set E is the whole set of available operations A, i.e., $E = A$.*

Theorem 3 ActiveSGS *generates only active schedules and it is complete in this set.*

Proof Let π be a task processing order, let $\boldsymbol{t} = \text{ActiveSGS}(\pi)$ and let $\sigma_k = (0, \sigma(1,k), \ldots, \sigma(\eta_k, k))$ denote the partial sequencing order in which operations are scheduled on a machine k according to $\boldsymbol{t}$. If $\boldsymbol{t}$ is not active, there must exist a task $o(j,l)$ scheduled in its machine k at a position $p_k \in \{2, \ldots, \eta_k\}$ such that for $o(j,l)$ there exists a feasible insertion position $q < p_k$ in σ_k. Thus, there exists a feasible schedule $\boldsymbol{s}$ such that $s_{o'} = t_{o'}, \forall o' \neq o(j,l)$ and

$$\forall i\ s^i_{o(j,l)} + d^i_{o(j,l)} \leq \min\{t^i_{o(j,l+1)}, t^i_{\sigma(q+1,k)}\},$$
$$\forall i\ s^i_{o(j,l)} = \max\{C^i_{o(j,l-1)}, C^i_{\sigma(q,k)}\} < t^i_{o(j,l)}.$$

But this is absurd because if such feasible insertion position exists at the end of the algorithm, it must also exist when operation $o(j,l)$ is to be scheduled by ActiveSGS and, in this case, $t^i_{o(j,l)} = ESI^i_{o(j,l)}$ can never be greater than $s^i_{o(j,l)}$ for any component i.

Conversely, let $\boldsymbol{t}$ be an active schedule and let π be the task processing order obtained as the topological order of the constraint graph representing $\boldsymbol{t}$. Since it is active, no feasible insertion positions can exist in $\boldsymbol{t}$. Therefore, $\text{ActiveSGS}(\pi)$ will schedule every task $o(j,l)$ with starting time $ESI_{o(j,l)} = ESA_{o(j,l)} = \max\{C_{\lambda(k)}, C_{o(j,l-1)}\}$ where $\lambda(k)$ is the operation preceding $o(j,l)$ in its machine k according to π, i.e. $ESI_{o(j,l)} = t_{o(j,l)}$. It thus follows that $\boldsymbol{t} = \text{ActiveSGS}(\pi) = \text{SemiActiveSGS}(\pi)$ □

Corollary 4 *The set of schedules generated by* ActiveSGS *is dominant.*

4.2.1 *The* fG&T-SGS *algorithms*

The Giffler-Thompson Algorithm or G&T in short ([7]) is probably the most famous active schedule generation scheme for deterministic job shop problem, having been used in a variety of settings. It is an appending algorithm where, given the task o^* with earliest possible completion time C^* at the current step, the set E of eligible operations (also referred to as *conflict set*) is the set of operations processed in the same machine as o^* which may start before C^*.

G&T provides a complete and constructive heuristic method to search for solutions in search spaces of reasonable size and has been used as a branching schema for the deterministic JSP in exact methods, such as branch and bound [2] or best-first search [17]. Also, G&T allows further reductions of the search space by including a parameter that bounds the length of time that a machine is allowed to remain idle on the chance that a "more critical" job will soon become available [19].

We can find some ad-hoc extensions of G&T in the fuzzy scheduling literature, from the earliest one in [16] to the most recent one in [12]. The variety of existing proposals illustrates that extending G&T is far from trivial. The first difficulty appears when computing the earliest completion time C^* at each current step. If it is computed as the minimum completion time of all the unscheduled tasks currently available, it may not correspond to the completion time of any specific task because a set of TFNs is not closed under the minimum. In consequence, it may not make sense to consider only one machine when computing the eligible set.

A possible solution is to build the eligible set E with all tasks o that "can start before C^*", which in fuzzy framework means that $\exists i ESA^i_o < (C^*)^i$, since C^* is previous to ESA_o only if $\forall i, (C^*)^i \leq ESA^i_o$. This is the basis for the first SGS extending G&T:

Definition 8 *The* fG&T-SGS1 *algorithm is an appending SGS where the eligible set E is computed as follows:*

$$\begin{aligned} C^* &= \min\{ESA_o + d_o : o \in A\} \\ E &= \{o \in A : \exists i\ ESA^i_o < (C^*)^i\} \end{aligned} \tag{13}$$

Theorem 5 fG&T-SGS1 *generates only active schedules, but it is not complete in this set and it is not dominant.*

Sketch of Proof We first prove by contradiction that fG&T-SGS1 generates active schedules. Let π be a task processing order and let us suppose that $\boldsymbol{t} = \text{fG\&T-SGS1}(\pi)$ is not active. Let $\sigma_k = (0, \sigma(1,k), \ldots, \sigma(\eta_k, k))$ denote the partial sequencing order in which operations are scheduled on a machine k according to $\boldsymbol{t}$. Reasoning as in Theorem 3, there must exist a task $o(j,l)$ scheduled in its machine k at a position $p_k \in \{2, \ldots, \eta_k\}$, a feasible schedule $\boldsymbol{s}$ with $s_{o'} = t_{o'}, \forall o' \neq o(j,l)$ and a position $q < p_k$ such that

$$\forall i\ s^i_{o(j,l)} + d^i_{o(j,l)} \leq \min\{t^i_{o(j,l+1)}, t^i_{\sigma(q+1,k)}\},$$
$$\forall i\ s^i_{o(j,l)} = \max\{C^i_{o(j,l-1)}, C^i_{\sigma(q,k)}\} < t^i_{o(j,l)}.$$

For the feasible position q to exist in $\boldsymbol{t}$, it must be the case that fG&T-SGS1 has scheduled operation $\sigma(q+1,k)$ before $o(j,l)$.

Also, $o(j,l)$ cannot have been in the set A when $\sigma(q+1,k)$ was to be scheduled. This is proved by contradiction using the fact that q is a feasible insertion position.

A direct consequence is that $o(j,l-1)$ cannot have been scheduled either. In fact, $o(j,l-1)$ cannot even have been in A when $\sigma(q+1,k)$ was to be scheduled. This is again proved by contradiction, using the fact that $\boldsymbol{s}$ and $\boldsymbol{t}$ are identical for every operation other than $o(j,l)$ and that a feasible insertion position exists.

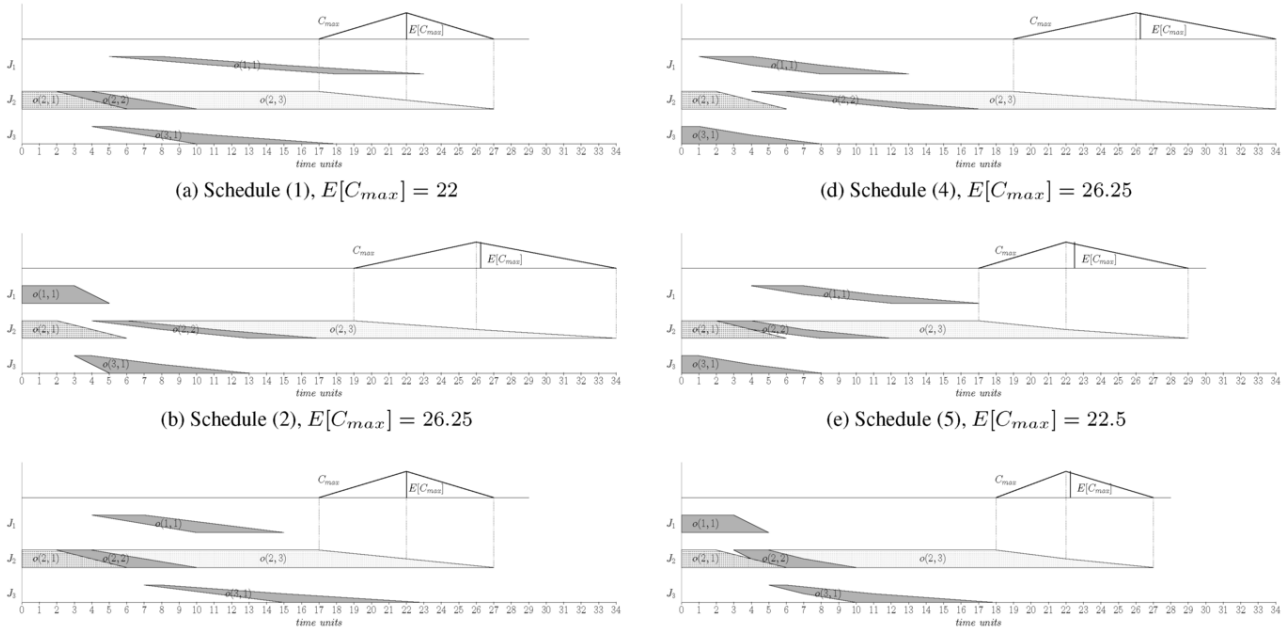
By repeating this argument "backwards" for all operations preceding $o(j,l)$ in its job, we conclude that $o(j,1)$ cannot have been in A when $\sigma(q+1,k)$ was scheduled, which is clearly absurd because A is initialised with the first task of every job.

To show that fG&T-SGS1 does not generate all active schedules nor is it complete, consider a problem with 3 jobs and 3 machines where durations are $d_{o(1,1)} = (3,4,5)$, $d_{o(2,1)} = (2,4,6)$, $d_{o(2,2)} = (2,3,4)$, $d_{o(2,3)} = (13,15,17)$, $d_{o(3,1)} = (1,4,8)$ and with the following machine requirements $\mu_{o(1,1)} = 1$, $\mu_{o(2,1)} = 2$, $\mu_{o(2,2)} = 1$, $\mu_{o(2,3)} = 3$, $\mu_{o(3,1)} = 1$. Figure 1 shows the job-oriented Gantt charts adapted to TFNs (following [6]) of all six feasible active schedules, including the two optimal solutions with $C_{max} = (17,22,27)$ (solutions (1) and (3)). In this case, it is easy to see that fG&T-SGS1 cannot generate any of the optimal (active) solutions. □

The incompleteness of fG&T-SGS1 stems from the fact that a set of TFNs is not closed under the minimum, i.e., C^* may not correspond to the earliest completion time of an operation in A; we can only guarantee that $(C^*)^i$ does correspond to the i-th component of the earliest completion time of an operation in A. Taking this into account, we propose an alternative extension of G&T.

Definition 9 *The* fG&T-SGS2 *algorithm is an appending SGS where the eligible set E is computed as follows.*

$$\begin{aligned} C^* &= \min\{ESA_o + d_o : o \in A\} \\ A^* &= \{o \in A : \exists i\ ESA^i_o + d^i_o = (C^*)^i\} \\ E &= \{o \in A : \forall o' \in A^* \exists i\ ESA^i_o < ESA^i_{o'} + d^i_{o'}\} \end{aligned} \tag{14}$$



(a) Schedule (1), $E[C_{max}] = 22$

(d) Schedule (4), $E[C_{max}] = 26.25$

(b) Schedule (2), $E[C_{max}] = 26.25$

(e) Schedule (5), $E[C_{max}] = 22.5$

(c) Schedule (3), $E[C_{max}] = 22$

(f) Schedule (6), $E[C_{max}] = 22.25$

Figure 1: Gantt chart of the schedules of all active solutions for the example in Theorem 5.

Theorem 6 fG&T-SGS2 *generates only active schedules and it is complete in this set.*

Sketch of Proof The argument that fG&T-SGS2 generates active schedules is analogous to that given for fG&T-SGS1 in Theorem 5.

To see that fG&T-SGS2 is complete, let $\boldsymbol{t}$ be an active schedule, let σ be the task processing order obtained from the topological ordering of the constraint graph represented by $\boldsymbol{t}$ and let σ_k be the partial order determined by σ for a particular machine k. We prove that for an operation processing order π containing all partial orders represented by σ_k and $\boldsymbol{s} = \text{fG\&T-SGS2}(\pi)$, we have $\boldsymbol{s} = \boldsymbol{t}$. It suffices to show that, if σ' is the task processing order obtained from the topological ordering of the constraint graph represented by $\boldsymbol{s}$, $\forall k$ $\sigma_k = \sigma'_k$.

Let us suppose that there exists at least one k such that $\sigma_k \neq \sigma'_k$ and let $a = o(j,l) = \sigma(q,k)$ be the first operation in σ that is scheduled in its machine k in a different order from σ. This means that there exists an operation requiring the same machine as a, $b = \sigma(q',k)$, $q' > q$, that will be scheduled by fG&T-SGS2 before a. Notice that, $b \in E$ and $a \notin E$. Also, without loss of generality, we may assume that $a \in A$. Finally, notice that, being an active schedule, in $\boldsymbol{t}$ there are no feasible insertion positions, that is, $\exists i\ ESA^i_a < ESA^i_b + d^i_b$.

If $b \in A^*$, since $a \in A - E$, there must exist at least one operation $o \in A^* \subseteq E$ such that $\forall i\ ESA^i_o + d^i_o \leq ESA^i_a$. o cannot share job with a or b. If it requires a machine $k' \neq k$, it can be scheduled before b without any change in any of the partial orders in σ. Using this argument a finite number of times, eventually $\forall x \in A^*, \mu_x = k$. This, together with the fact that $\boldsymbol{t}$ is active, leads to having $a \in E$, which is a contradiction. If $b \notin A^*$, $b \in E$ means that $\forall o \in A^*$ $\exists i : ESA^i_b < ESA^i_o + d^i_o$. Reasoning analogously to the case when $b \in A^*$, we conclude that it is impossible to schedule b before a, which is a contradiction. □

Corollary 7 *The set of schedules generated by* fG&T-SGS2 *is dominant.*

5 Empirical Behaviour

Having studied the different features of each proposed SGS, in this section we intend to illustrate their behaviour in practice. To this end, we will analyse the quality of the solutions generated by each SGS from a broad sample of operation processing orders, which will also offer a picture of the different schedule spaces. This study is carried out on the fuzzy instances from [14], a set of 12 fuzzified versions of what are considered to be some of the hardest instances for the JSP. For each instance, we generate $T = 1000$ random feasible task orderings and evaluate each ordering using the four SGSs proposed in this paper.

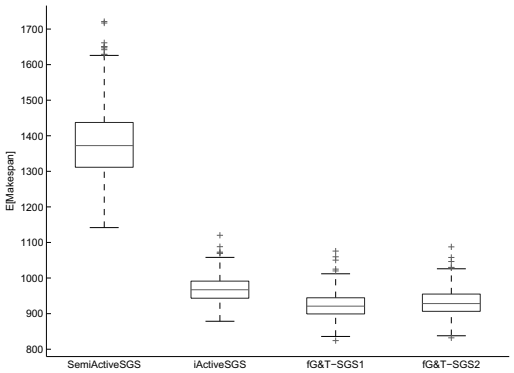


Figure 2: $E[C_{max}]$ for 1000 task orderings for instance ABZ9.

The box-plot in Figure 2 corresponds to the expected makespan obtained with the T task orderings using the different SGSs. It corresponds to instance ABZ9; for the remaining instances, the behaviour is very similar. As expected, the semi-active solutions generated by

SemiActiveSGS are much worse than the active ones obtained with the other SGSs; this is due to the size and features of the related space of solutions. These results confirm the clear difference, also in the fuzzy framework, between the spaces of active schedules and semi-active ones. Differences between active SGSs are on the other hand not so clear, even if ActiveSGS seems to yield slightly worse solutions than the two extensions of G&T.

A better assessment of the SGSs is achieved through a series of non-parametric statistical inference tests, having rejected the hypotheses of normality for all instances with preliminary Kolmogorov-Smirnov tests. For each instance we run a Friedman two-way analysis of variance by ranks. As for the box-plots, results are very similar for all instances, and show that there is a significant difference between the samples corresponding to each SGS. According to the mean ranks provided by the test on ABZ9, the SGS can be ranked according to the average quality of the solutions as follows: the best one would be fG&T-SGS1 (1.4215), followed by fG&T-SGS2 (1.7565), then ActiveSGS (2.822) and finally SemiActiveSGS (4); the results for the remaining instances are very similar. Additionally, a Mann-Whitney U test is run on each pair of samples. According to this test, for instances FT10, FT20 and LA25, there are not significant differences between fG&T-SGS1 and fG&T-SGS2 (with p-values 0.288, 0.206 and 0.129 respectively). For the remainng instances, a p-value< 0.01 indicates that there are significant differences between both extensions of G&T.

An explanation for these results is that fG&T-SGS1 maps the processing orders to a subspace of the active schedules with good solutions in average, even if it is not guaranteed to contain any optimal solution. For large instances with a huge solution space, this reduction may prove worthwhile. However, for small instances (or if the SGS is to be used in a exact algorithm) it may be better to use fG&T-SGS2 or ActiveSGS, which allow to search across the whole space of active schedules. In fact, although both are complete, the mapping defined by fG&T-SGS2 seems significantly better in average quality.

The behaviour shown for the fuzzy setting is consistent with the deterministic JSP, where active schedules are good in average (and much better than semi-active ones) and form a dominant set. Also, in the crisp case the G&T algorithm can be modified in order to further reduce the search space; at the extreme, the search space is constrained to that of non-delay schedules, where a machine cannot be idle if there is an operation that can be executed in it. Experience demonstrates that the mean value of solutions tends to improve with the reduction of the search space, despite the risk of losing the optimal solution.

6 Conclusions

This papers provides the first formal definition and study of types of feasible fuzzy schedules and related schedule generation schemes for the job shop problem with fuzzy processing times. We have shown that dominance and completeness are lost when considering a simple extension of the G&T algorithm, while an insertion SGS algorithm and a more sophisticated extension of the G&T are both complete and dominant. Additional experimental results have confirmed the differences between semi-active and active subspaces and shown that narrowing the search space can improve the average quality of schedules even if dominance is lost. We believe both the theoretical and experimental results can provide a guide for designing SGS and incorporate them both into metaheuristic and exact search methods.

As future work, we plan to extend this study to smaller sets of schedules, such as non-delay. Also, the fuzzy setting allows for alternative definitions of left shifts and, consequently, (semi)active schedules, thus admiting more constraints in the solution space than those existing in the deterministic job shop which may be worth exploring.

ACKNOWLEDGEMENTS

This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02 and MTM2010-16051 and by the Principality of Asturias (Spain) under grants Severo Ochoa BP13106 and FC-13-COF13-035.

REFERENCES

[1] C. Artigues, P. Lopez, and P.D. Ayache, 'Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis', *Annals of Operations Research*, **138**, 21–52, (2005).
[2] P. Brucker, B. Jurisch, and B. Sievers, 'A branch and bound algorithm for the job-shop scheduling problem', *Discrete Applied Mathematics*, **49**, 107–127, (1994).
[3] S-M. Chen and T-H. Chang, 'Finding multiple possible critical paths using fuzzy PERT', *IEEE Transactions on Systems, Man, and Cybernetics–Part B:*, **31**(6), 930–937, (2001).
[4] D. Dubois, H. Fargier, and Ph. Fortemps, 'Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge', *European Journal of Operational Research*, **147**, 231–252, (2003).
[5] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*, Plenum Press, New York (USA), 1986.
[6] P. Fortemps, 'Jobshop scheduling with imprecise durations: a fuzzy approach', *IEEE Transactions of Fuzzy Systems*, **7**, 557–569, (1997).
[7] B. Giffler and G. L. Thompson, 'Algorithms for solving production scheduling problems', *Operations Research*, **8**, 487–503, (1960).
[8] S. Heilpern, 'The expected value of a fuzzy number', *Fuzzy Sets and Systems*, **47**, 81–86, (1992).
[9] W. Herroelen and R. Leus, 'Project scheduling under uncertainty: Survey and research potentials', *European Journal of Operational Research*, **165**, 289–306, (2005).
[10] A. Kasperski and M. Kule, 'Choosing robust solutions in discrete optimization problems with fuzzy costs', *Fuzzy Sets and Systems*, **160**, 667–682, (2009).
[11] Q. Niu, B. Jiao, and X. Gu, 'Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time', *Applied Mathematics and Computation*, **205**, 148–158, (2008).
[12] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente, 'Swarm lexicographic goal programming for fuzzy open shop scheduling', *Journal of Intelligent Manufacturing*, (2013) *In press*.
[13] M. L. Pinedo, *Scheduling. Theory, Algorithms, and Systems.*, Springer, third edn., 2008.
[14] J. Puente, C. R. Vela, and I. González-Rodríguez, 'Fast local search for fuzzy job shop scheduling', in *Proceedings of ECAI 2010*, pp. 739–744. IOS Press, (2010).
[15] S. J. Sadjadi, R. Pourmoayed, and M.B. Aryanezhad, 'A robust critical path in an environment with hybrid uncertainty', *Applied Soft Computing*, **12**(3), 1087–1100, (2012).
[16] M. Sakawa and T. Mori, 'An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate', *Computers & Industrial Engineering*, **36**, 325–341, (1999).
[17] M. Sierra and R. Varela, 'Pruning by dominance in best-first search for the job shop scheduling problem with total flow time.', *Journal of Intelligent Manufacturing*, **21**(1), 111–119, (2010).
[18] A. Sprecher, R. Kolisch, and A. Drexl, 'Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem', *European Journal of Operational Research*, **80**, 94–102, (1995).
[19] R. H. Storer, S. D. Wu, and R. Vaccari, 'New search spaces for sequencing problems with application to job shop scheduling', *Management Science*, **38**(10), 1495–1509, (1992).
[20] J. Wang, 'A fuzzy robust scheduling approach for product development projects', *European Journal of Operational Research*, **152**, 180–194, (2004).

8.2 β-robust solutions for the fuzzy open shop scheduling

In this section, we include the following publication.

- **Title:** β-robust solutions for the fuzzy open shop scheduling. In: Information Processing and Management of Uncertainty in Knowledge-Based Systems.
- **Conference:** IPMU 2014. 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems.
- **Date:** July 2014.
- Place: Montpellier (France).
- Conference Ranking:
  - ERA 2010: **Rank C**
  - CORE 2014: **Rank C**

This publications contains pieces of work described in Sections 4.3.2 and 4.3.3.

Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente Peinador, J.: β-robust solutions for the fuzzy open shop scheduling. In: Information Processing and Management of Uncertainty in Knowledge-Based Systems. Communications in Computer and Information Science, vol. 442, pp. 447-456. Springer (2014). Doi: 10.1007/978-3-319-08795-5_46.

β-Robust Solutions for the Fuzzy Open Shop Scheduling

Juan José Palacios[1], Inés González-Rodríguez[2], Camino R. Vela[1], and Jorge Puente Peinador[1]

[1] Department of Computer Science, University of Oviedo, Spain
{palaciosjuan,puente,crvela}@uniovi.es http://di002.edv.uniovi.es/iscop
[2] Department of Mathematics, Statistics and Computing, University of Cantabria, Spain
ines.gonzalez@unican.es

Abstract. We consider the open shop scheduling problem with uncertain durations modelled as fuzzy numbers. We define the concepts of necessary and possible β-robustness of schedules and set as our goal to maximise them. Additionally, we propose to assess solution robustness by means of Monte Carlo simulations. Experimental results using a genetic algorithm illustrate the proposals.

1 Introduction

Scheduling problems form an important body of research since the late fifties, with multiple applications in industry, finance and science [1]. In particular, the open shop scheduling problem models situations frequently appearing in testing components of electronic systems, in general repair facilities when repairs can be performed in an arbitrary order, as well as in certain medical diagnosis procedures. However, the open shop is NP-complete for a number of resources $m \geq 3$ and has a significantly large search space. Specific and efficient methods to solve it are necessary but still scarce, despite their increasing presence in the recent literature [2].

Traditionally, it has been assumed that problems are static and certain: all activities and their durations are precisely known in advance and do not change as the solution is being executed. However, for many real-world scheduling problems design variables are subject to perturbations or changes, causing optimal solutions to the original problem to be of little or no use in practice. Therefore, a common practical requirement is to obtain so-called *robust solutions*, which should still work satisfactorily when design variables change slightly, for instance, due to manufacturing tolerances.

A source of changes in scheduling problems is uncertainty in activity durations. There exists great diversity of approaches to dealing with this kind of uncertainty [3]. Perhaps the best-known is stochastic scheduling, although fuzzy sets and possibility theory provide an interesting alternative, with a tradeoff between the expressive power of probability and their associated computational

A. Laurent et al. (Eds.): IPMU 2014, Part I, CCIS 442, pp. 447–456, 2014.

complexity and knowledge demands. Indeed, fuzzy sets have been used in different manners in scheduling, ranging from representing incomplete or vague states of information to using fuzzy priority rules with linguistic qualifiers or preference modelling (cf. [4]).

The approaches to proactive robustness are several and varied. For instance, in stochastic settings, heuristic rules are used to include time buffers or slacks between activities in a baseline schedule [5]. In combinatorial optimisation, min-max or min-max regret criteria are applied to construct solutions having the best possible performance in the worst case [6], an approach already translated to the fuzzy framework [7],[8]. However, this may be deemed as too conservative when the worst case is not crucial and an overall acceptable performance is preferred. This is the basis for the β-robustness approach in stochastic scheduling [9], taking into account the subjective aspect of robustness through a target level specified by the decision maker so the goal is to maximise the likelihood that a solutions's actual performance is not worse than the target. This technique can be related to chance-constrained programming in linear optimisation, which has also been extended to fuzzy and fuzzy stochastic coefficients (cf. [10]).

The open shop problem with uncertainty constitutes a relatively new and complex research line. While there are many contributions to solve fuzzy job shop problems (we can cite, among others, [11],[12], [13] or [14]), the literature on fuzzy open shop is still scarce. Among the few existing proposals, a heuristic approach is proposed in [15] to minimise the expected makespan for an open shop problem with stochastic processing times and random breakdowns; in [16] the expected makespan of an open shop with fuzzy durations is minimised using a genetic algorithm hybridised with local search. Finally, in the framework of multiobjective approach, a possibilistic mixed-integer linear programming method is proposed in [17] for an OSP with setup times, fuzzy processing times and fuzzy due dates to minimise total weighted tardiness and total weighted completion times and in [18] a goal programming model based on lexicographic multiobjective optimisation of both makespan and due-date satisfaction is adopted and solved using a particle swarm algorithm.

In this paper, we intend to advance in the study of the fuzzy open shop problem, and in particular, in the search of robust solutions. In analogy to stochastic scheduling, we shall define the concepts of β_*-robust and β^*-robust schedules in terms of necessity and possibility, so the objective will then be to maximise such robustness. Then, we shall propose to perform an additional analysis of the obtained solutions using a Monte-Carlo simulation method based on the semantics of fuzzy schedules from [13]. Finally, we adapt the genetic algorithm from [19] and provide experimental results to illustrate our proposals.

2 The Fuzzy Open Shop Problem

The *open shop scheduling problem*, or *OSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job consists of m tasks or operations, each requiring

the exclusive use of a different machine for its whole processing time without preemption, i.e. all operations must be processed without interruption. In total, there are mn operations, $\{o_{ij}, 1 \leq i \leq n, 1 \leq j \leq m\}$. A solution to this problem is a *schedule*–an allocation of starting times for all operations– which is *feasible*, in the sense that all constraints hold, and is also optimal according to some criterion, most commonly minimising the makespan C_{max}, that is, the completion time of the last operation (and therefore, of the whole project).

In real-life applications, it is often the case that the exact time it takes to process a task is not known in advance. However, based on previous experience, an expert may have some knowledge (albeit uncertain) about the duration. The crudest representation of such knowledge would be a human-originated confidence interval; if some values appear to be more plausible than others, then a natural extension is a fuzzy interval or fuzzy number. The simplest model is a *triangular fuzzy number* or *TFN*, denoted $A = (a^1, a^2, a^3)$, given by an interval $[a^1, a^3]$ of possible values and a modal value $a^2 \in [a^1, a^3]$, so its membership function takes a triangular shape:

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

Triangular fuzzy numbers and more generally fuzzy intervals have been extensively studied in the literature (cf. [20]) and widely used in scheduling.

In the open shop, we essentially need two operations on fuzzy numbers, the sum and the maximum. For any bivariate continuous isotonic function f and any two fuzzy numbers A and B, if $A_\alpha = [\underline{a}_\alpha, \overline{a}_\alpha]$ denotes the α-cut, the result $f(A, B)$ is a fuzzy number F such that $F_\alpha = [f(\underline{a}_\alpha, \underline{b}_\alpha), f(\overline{a}_\alpha, \overline{b}_\alpha)]$, that is, computing the function is equivalent to computing it on every α-cut. In particular, this is true for both the addition and the maximum. However, evaluating two sums or two maxima for every value $\alpha \in [0, 1]$ is cumbersome if not intractable in general. For the sake of simplicity and tractability of numerical calculations, we follow [11] and approximate the results of these operations by a linear interpolation evaluating only the operation on the three defining points of each TFN (an approach also taken, among others, in [12], [18] or [21]). The approximated sum coincides with the actual sum, so for any pair of TFNs A and Bs:

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

Regarding the maximum, for any two TFNs A, B, if $F = \max(A, B)$ denotes their maximum and $G = (\max\{a^1, b^1\}, \max\{a^2, b^2\}, \max\{a^3, b^3\})$ the approximated value, it holds that $\forall \alpha \in [0, 1]$, $\underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha$. The approximated maximum G is thus a TFN which artificially increases the value of the actual maximum F, although it maintains the support and modal value. This approximation can be trivially extended to the case of more than two TFNs.

Given a task processing order π, the schedule (starting and completion times of all tasks) may be computed as follows. For every task x with processing time

p_x, let $S_x(\pi)$ and $C_x(\pi)$ denote respectively the starting and completion times of x, let $PM_x(\pi)$ and $SM_x(\pi)$ denote the predecessor and successor tasks of x in the machine sequence provided by π, and let $PJ_x(\pi)$ and $SJ_x(\pi)$ denote respectively the predecessor and successor tasks of x in the job sequence provided by π. Then the starting time $S_x(\pi)$ of x is a TFN given by:

$$S_x(\pi) = \max(S_{PJ_x(\pi)} + p_{PJ_x(\pi)}, S_{PM_x(\pi)} + p_{PM_x(\pi)}), \tag{3}$$

Clearly, $C_x(\pi) = S_x(\pi) + p_x(\pi)$. If there is no possible confusion regarding the processing order, we may simplify notation by writing S_x and C_x. The completion time of the last task to be processed according to π thus calculated will be the makespan, denoted $C_{max}(\pi)$ or simply C_{max}. We obtain a *fuzzy schedule* in the sense that the starting and completion times of all tasks and the makespan are fuzzy intervals, interpreted as possibility distributions on the values that the times may take. However, notice that the task processing ordering π that determines the schedule is deterministic; there is no uncertainty regarding the order in which tasks are to be processed.

3 Robust Schedules

The usual objective of deterministic scheduling of minimising the makespan could, in principle, be translated to the fuzzy framework as minimising the expected makespan $E[C_{max}]$. However, minimising the expected makespan may be criticised, since it reduces the information provided by a fuzzy makespan to a single value, thus loosing part of the information. Neither does it address the practical requirement of solution robustness. Therefore we propose instead to find the equivalent to what has been called in the stochastic framework β-robust schedules [9,22], schedules with a certain confidence level that the performance will be within a given threshold.

The membership function μ_D of a fuzzy duration D may be interpreted as a possibility distribution on the real numbers [23,24], representing the set of more or less plausible, mutually exclusive values of a variable y (in our case, the underlying uncertain duration). Since a degree of possibility can be viewed as an upper bound of a degree of probability, μ_D also encodes a whole family of probability distributions.

It is well known that for a given interval $I \subseteq \mathbb{R}$, the *possibility* and *necessity measure* that $D \in I$ are respectively given by $\Pi(D \in I) = \sup_{y \in I} \mu_D(y)$ and $N(D \in I) = \inf_{y \in I} 1 - \mu_D(y) = 1 - \sup_{y \notin I} \mu_D(x) = 1 - \Pi(D \notin I)$, so necessity and possibility are dual measures which provide lower and upper bounds for the probability that y is in I given the information 'y is D': $N(D \in I) \leq Pr(D \in I) \leq \Pi(D \in I)$. In particular, for $A = (a^1, a^2, a^3)$ a TFN, the *necessity* and the *possibility* that A is less than a given real number r are given by:

$$N(A \leq r) = \begin{cases} 0, & r \leq a^2, \\ \frac{r-a^2}{a^3-a^2}, & a^2 \leq r \leq a^3, \\ 1, & a^3 < r \end{cases} \Pi(A \leq r) = \begin{cases} 0, & r \leq a^1, \\ \frac{x-a^1}{a^2-a^1}, & a^1 \leq r \leq a^2, \\ 1, & a^2 < r \end{cases} \tag{4}$$

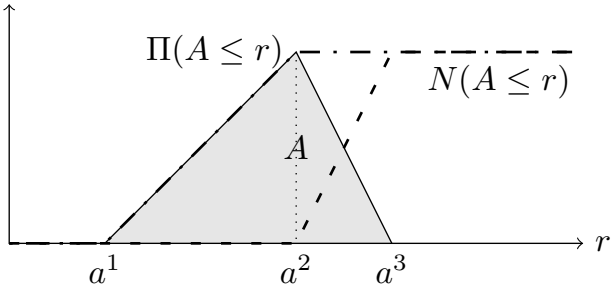


Fig. 1. Necessity $N(A \leq r)$ and possibility $\Pi(A \leq r)$ for varying values of $r \in \mathbb{R}$

Clearly, for any value r, $N(A \leq r) \leq \Pi(A \leq r)$. Figure 1 illustrates both measures.

Assuming we have a target or threshold for the makespan $C^\star$, we may want to maximise the confidence that C_{max} will "for sure" be less than this threshold. In our setting, this means to maximise the necessity degree that C_{max} is less than $C^\star$.

Definition 1. *A schedule with makespan C_{max} is said to be* necessarily β_*-robust *w.r.t. a threshold $C^\star$ if and only if $\beta_* = N(C_{max} \leq C^\star)$. Analogously, the schedule is said to be* possibly β^*-robust *w.r.t. $C^\star$ iff $\beta^* = \Pi(C_{max} \leq C^\star)$. β_* and β^* are respectively the* degrees of necessary and possible robustness *w.r.t. the threshold $C^\star$.*

Clearly, if a schedule is β^*-robust and β_*-robust w.r.t. the same threshold, and $\beta = Pr(C_{max} \leq C^\star)$, we have that $\beta_* \leq \beta \leq \beta^*$.

The degree of necessary robustness represents the degree of confidence that the makespan will certainly be less than the threshold. In the following, we will consider that the objective will be to find a schedule maximising this confidence level, so the resulting problem may be denoted $O|fuzz\,p_i|\beta_*(C^\star)$ following the three-field notation [25]. Obviously, by maximising the degree of necessary robustness we are also maximising the possible robustness of the schedule.

4 Monte-Carlo Simulation Assessment

Assuming we have solved the above optimisation problem and have obtained a β_*-robust schedule w.r.t. $C^\star$, is there a means of assessing the actual robustness of such schedule? In other words, does the concept of β_*-robustness really capture the desired high-level characteristic of robustness? Here, we propose a method for an empirical assessment of solutions to the $O|fuzz\,p_i|\beta_*(C^\star)$ problem, based on using Monte-Carlo simulations and inspired by the semantics for fuzzy schedules from [13].

In [13] fuzzy schedules are interpreted as *a-priori solutions*, found when the duration of tasks is not exactly known. In this setting, it is impossible to predict what the exact time-schedule will be, because it depends on the realisation of the tasks' durations, which is not known yet. Each fuzzy schedule corresponds to

a precise ordering of tasks and it is not until tasks are executed according to this ordering that we know their real duration and, hence, know the exact schedule, the *a-posteriori solution* with exact job completion times and makespan. The practical interest of a solution to the fuzzy open shop would then lie in the ordering of tasks that it provides a priori using the available incomplete information, which should ideally yield good schedules in the moment of its practical use. Its behaviour could therefore be evaluated on a family of K deterministic open shop problems, representing K possible a posteriori realisations of the fuzzy problem. These may be simulated by generating an exact duration $\hat{p}_x$ for each task at random according to a probability distribution which is coherent with the fuzzy duration p_x.

Given a solution to the fuzzy open shop, consider the task processing order π it provides. For a deterministic version of the problem, let $\hat{\boldsymbol{p}}$ be the matrix of precise durations, such that $\hat{p}_{ij}$, the a-posteriori duration of operation o_{ij}, is coherent with the constraint imposed by the fuzzy duration p_{ij}. The ordering π can be used to process the operations, where the duration of each operation o_{ij} is taken to be $\hat{p}_{ij}$. This yields a time-schedule with precise starting and completion times for all tasks and, in particular, a real makespan $C_{max}(\pi, \hat{\boldsymbol{p}})$, which may be under or above the threshold $C^\star$. If instead of a single deterministic instance we consider the whole family of K deterministic problems, each with a duration matrix, we obtain K makespan values; the proportion κ of those values among the K which are actually below the threshold $C^\star$ gives us an empirical measure of the robustness of π. If the β_*-robustness is a good measure of the schedules robustness, then a schedule with high β_* should correspond to a high κ.

5 Genetic Algorithm

To solve the optimisation problem $O|fuzz\ p_i|\beta_*(C^\star)$, we propose to use the genetic algorithm (GA) from [19]. In principle, to do so it would only be necessary to substitute the fitness function therein by the β_*-robustness degree of the schedule represented by each chromosome. However, such a straightforward approach has a serious drawback: the initial population, generated at random, consists of poor schedules, with high makespan values which, most likely, will yield a value $\beta_* = 0$ for any reasonable threshold $C^\star$, thus making it impossible for the GA to evolve.

In order to overcome this drawback, we propose to adapt the GA to use an "adaptive" threshold, with successive approximations $C_0^\star > C_1^\star > \ldots$ until $C^\star$ is reached. Given the first population, a first threshold $C_0^\star$ is obtained as the most pessimistic value of the best makespan in this population, making sure that there will be chromosomes with non-zero fitness values (in fact, the individual with the best makespan will have fitness 1), thus allowing the GA to evolve. The threshold can then be updated along successive generations with new more demanding values $C_g^\star$ linearly decreasing from $C_0^\star$ to $C^\star$. This smooth updating allows the GA to evolve to robust solutions w.r.t. iteratively smaller thresholds. Finally, in order to give the GA the chance of obtaining β_*-robust solutions w.r.t.

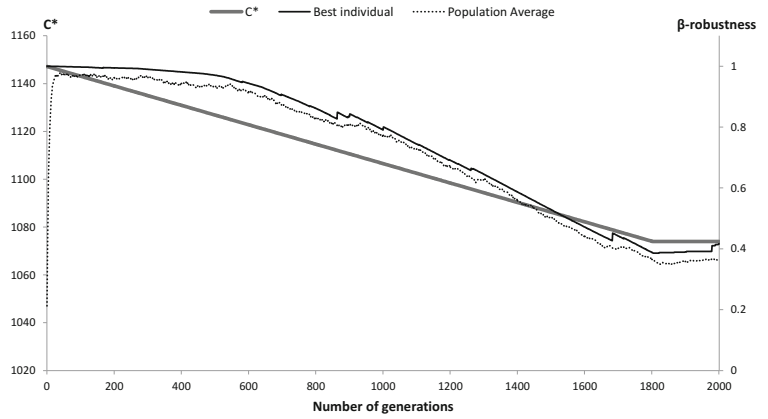


Fig. 2. Evolution of the best and mean solution of GA and the $C_g^\star$ values for the instance j8-per10-1 averaged across 10 runs

$C^\star$, in the last generations of the algorithm the $C^\star$ value is used to compute the β_*-robustness degree as fitness function.

6 Experimental Results

For the experimental study we shall use the test bed given in [16], where the authors follow [11] to generate a set of fuzzy instances from well-known open shop benchmark problems. Given a deterministic instance, each deterministic processing time t is transformed into a symmetric fuzzy processing time $p(t)$ with modal value $p^2 = t$ and where values p^1, p^3 are taken as random integer values such that the resulting TFN is symmetric w.r.t. p^2 and its maximum range of fuzziness is 30% of p^2. The original benchmark consists of 6 families, denoted *J3, J4,. . . , J8*, of sizes from 3×3 to 8×8, containing 8 or 9 instances each. In this work we only consider the largest instances: i.e. the 9 instances of size 7×7 and the 8 instances of size 8×8.

In a real problem, the target value $C^\star$ would be provided by an expert with a reasonable knowledge of the problem. However, as we are using synthetic problems, such expert is not available and in consequence the target values must be set following some criterium. In our case, we have taken the best known solution $A = (a^1, a^2, a^3)$ for each instance [18] and we have defined $C^\star = a^2 + TF \times (a^3 - a^2)$, where TF is a given tightness factor. To obtain the best possible performance, a parametric analysis (not reported here due to the lack of space) was conducted using $TF = 0.75$. The resulting parameter values were: population size=100, crossover probability=0.7, mutation probability=0.05, and number of generations=2000 from which the last 200 use the $C^\star$ value. The GA has been run with these parameters 10 times on each problem instance. Figure 2 shows the convergence pattern for *j8-per10-1*, one of the largest instances, with the remaining instances presenting a similar behaviour. The figure shows the evolution along 2000 generations of the fitness value of the best individual together with the mean fitness of the population and the $C_g^\star$ threshold

used at each generation g to compute the β_*-robustness. As expected, we can appreciate that the algorithm's behaviour is sensitive to the $C_g^\star$ values. Initially, a less-demanding $C_0^\star$ allows the GA to evolve properly so the average quality of the population improves. After the first generations, $C_g^\star$ decreases becoming more demanding and in consequence, despite the fact that the population continues evolving, the robustness deteriorates for some generations (notice that for the same solution, its robustness value is dependent on the threshold $C^\star$). Finally, in the last iterations the goal $C^\star$ remains fixed and robustness values improve again thanks to the algorithm's evolution.

To empirically measure the robustness of the schedules obtained by the GA, we follow the Monte-Carlo simulation assessment introduced in Section 4 and generate samples of $K = 1000$ deterministic problems for each fuzzy instance, with random a-posteriori durations following a probability distribution which is coherent with the TFNs that model the fuzzy durations. We have then obtained the makespan values for each deterministic problem using the ordering provided by the GA on the fuzzy instanc, and we have finally computed the proportion κ out of the K deterministic makespan values which are below the threshold $C^\star$. Table 1 shows, for each fuzzy instance, the threshold $C^\star$, the β_* value of the best, average and worst solution across 10 runs, the CPU time (Runtime) in seconds, and the proportion κ obtained in the simulation for the best solution (κ-robustness). We can appreciate that even for the worst solutions $\beta_* > 0$, so in all solutions the possible β^*-robustness is 1. Moreover, the obtained "real" robustness values (κ) are always 1 or very close to 1, even in those instances

Table 1. Results of the GA and the *a-posteriori* analysis across the largest instances of the Brucker data set

Instance	$C^\star$	β_*-robustness Best	Average	Worst	Runtime	κ-robustness
j7-per0-0	1105.25	0.3682	0.2258	0.1082	9.2s.	0.9830
j7-per0-1	1140.00	0.7439	0.6231	0.4789	9.0s.	1.0000
j7-per0-2	1136.75	0.5493	0.4364	0.3147	9.0s.	0.9980
j7-per10-0	1099.50	0.7500	0.5294	0.2895	8.6s.	1.0000
j7-per10-1	1075.75	0.7319	0.5383	0.1972	8.9s.	1.0000
j7-per10-2	1079.75	0.6408	0.4701	0.2351	9.2s.	1.0000
j7-per20-0	1028.50	0.6477	0.5667	0.4524	9.0s.	1.0000
j7-per20-1	1075.00	0.7541	0.5041	0.1509	9.0s.	1.0000
j7-per20-2	1059.50	0.6288	0.3657	0.1508	9.1s.	1.0000
j8-per0-1	1106.50	0.3750	0.2164	0.0473	13.6s.	0.9190
j8-per0-2	1115.75	0.4696	0.2561	0.1735	13.8s.	0.9630
j8-per10-0	1110.00	0.9054	0.5723	0.3273	13.5s.	1.0000
j8-per10-1	1074.00	0.5714	0.4162	0.2692	13.7s.	0.9830
j8-per10-2	1059.25	0.4179	0.2601	0.0753	13.9s.	0.9850
j8-per20-0	1062.75	0.6433	0.4975	0.3994	13.6s.	1.0000
j8-per20-1	1048.00	0.7164	0.5445	0.4133	13.6s.	1.0000
j8-per20-2	1059.00	0.5444	0.4451	0.3299	13.6s.	0.9960

where β_* is smaller (e.g. j7-per0-0). This could be explained by the conservative character of the necessary robustness. In fact, in all cases where the fuzzy schedule has $\beta_* \geq 0.6$, the makespan values for all deterministic simulations are below the threshold $C^\star$.

7 Conclusions

We have tackled a variant of the open shop scheduling problem where uncertainty in durations is modelled using triangular fuzzy numbers. We have defined necessary and possible β-robustness in terms of scheduling and we have proposed as objective function to maximize the most pessimistic measure which is the necessary β-robustness. Moreover, we have proposed a method to empirically assess the actual robustness of the solutions. We have tested our approach using a genetic algorithm from the literature using an adaptive threshold of makespan values that overcomes the drawback of a likely random search by the GA. Based in the promising results, in the future we intend to improve on the β-robustness by adapting to the fuzzy framework the definition of α-β-robustness, that is, for a given confidence level β (ideally close to 1), try to minimise the threshold α for which this confidence is obtained (as in [22] for stochastic scheduling). We also intend to consider some kind of multiobjective approach that maximises robustness and minimises makespan.

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

References

1. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems, 3rd edn. Springer (2008)
2. Sha, D.Y., Cheng-Yu, H.: A new particle swarm optimization for the open shop scheduling problem. Computers & Operations Research 35, 3243–3261 (2008)
3. Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165, 289–306 (2005)
4. Dubois, D., Fargier, H., Fortemps, P.: Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. European Journal of Operational Research 147, 231–252 (2003)
5. Van de Vonder, S., Demeulemeester, E., Herroelen, W.: Proactive heuristic procedures for robust project scheduling: An experimental analysis. European Journal of Operational Research 189, 723–733 (2008)
6. Aissi, H., Bazgan, C., Vanderpooten, D.: Min-max and min-max regret versions of combinatorial optimization problems: A survey. European Journal of Operational Research 197, 427–438 (2009)
7. Wang, J.: A fuzzy robust scheduling approach for product development projects. European Journal of Operational Research 152, 180–194 (2004)

8. Kasperski, A., Kule, M.: Choosing robust solutions in discrete optimization problems with fuzzy costs. Fuzzy Sets and Systems 160, 667–682 (2009)
9. Daniels, R.L., Carrillo, J.E.: β-robust scheduling for single-machine systems with uncertain processing times. IIE Transactions 29, 977–985 (1997)
10. Aiche, F., Abbas, M., Dubois, D.: Chance-constrained programming with fuzzy stochastic coefficients. Fuzzy Optimization and Decision Making 12, 125–152 (2013)
11. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions of Fuzzy Systems 7, 557–569 (1997)
12. Sakawa, M., Kubota, R.: Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. European Journal of Operational Research 120, 393–407 (2000)
13. González Rodríguez, I., Puente, J., Vela, C.R., Varela, R.: Semantics of schedules for the fuzzy job shop problem. IEEE Transactions on Systems, Man and Cybernetics, Part A 38(3), 655–666 (2008)
14. Puente, J., Vela, C.R., González-Rodríguez, I.: Fast local search for fuzzy job shop scheduling. In: Proc. of ECAI 2010, pp. 739–744. IOS Press (2010)
15. Alcaide, D., Rodriguez-Gonzalez, A., Sicilia, J.: A heuristic approach to minimize expected makespan in open shops subject to stochastic processing times and failures. International Journal of Flexible Manufacturing Systems 17, 201–226 (2006)
16. González-Rodríguez, I., Palacios, J.J., Vela, C.R., Puente, J.: Heuristic local search for fuzzy open shop scheduling. In: Proc. FUZZ-IEEE 2010, pp. 1858–1865. IEEE (2010)
17. Noori-Darvish, S., Mahdavi, I., Mahdavi-Amiri, N.: A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due-dates. Applied Soft Computing 12, 1399–1416 (2012)
18. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Swarm lexicographic goal programming for fuzzy open shop scheduling. Journal of Intelligent Manufacturing (2013)
19. Palacios, J.J., Puente, J., Vela, C.R., González-Rodríguez, I.: A genetic algorithm for the open shop problem with uncertain durations. In: Mira, J., Ferrández, J.M., Álvarez, J.R., de la Paz, F., Toledo, F.J. (eds.) IWINAC 2009, Part I. LNCS, vol. 5601, pp. 255–264. Springer, Heidelberg (2009)
20. Dubois, D., Prade, H.: Possibility Theory: An Approach to Computerized Processing of Uncertainty. Plenum Press, New York (1986)
21. Niu, Q., Jiao, B., Gu, X.: Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. Applied Mathematics and Computation 205, 148–158 (2008)
22. Wu, C.W., Brown, K.N., Beck, J.C.: Scheduling with uncertain durations: Modeling β-robust scheduling with constraints. Computers & Operations Research 36, 2348–2356 (2009)
23. Dubois, D., Prade, H.: Fuzzy Sets and Systems: Theory and Applications. Academic Press, New York (1980)
24. Zadeh, L.A.: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems 1, 3–28 (1978)
25. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 4, 287–326 (1979)

Bibliography

[1] S. Abdullah and M. Abdolrazzagh-Nezhad. Fuzzy job-shop scheduling problems: A review. *Information Sciences*, 278:380–407, 2014.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197:427–438, 2009.

[3] S. Ali, A. Maciejewski, H. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *Parallel and Distributed Systems, IEEE Transactions on*, 15(7):630–641, July 2004.

[4] C. Artigues, P. Lopez, and P. Ayache. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research*, 138:21–52, 2005.

[5] H. Aytung, M. A. Lawley, K. McKay, M. Shantha, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161:86–110, 2005.

[6] O. Bahri, N. Ben Amor, and E.-G. Talbi. Optimization algorithms for multi-objective problems with fuzzy data. In *Computational Intelligence in Multi-Criteria Decision-Making (MCDM), 2014 IEEE Symposium on*, pages 194–201, Dec 2014.

[7] J. Barnes and 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*, 1996.

[8] C. Baudrit and D. Dubois. Practical representations of incomplete probabilistic knowledge. *Computational Statistics & Data Analysis*, 51:86–108, 2006.

[9] J. C. Beck. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, 2007.

[10] J. Bidot, T. Vidal, and P. Laboire. A theoretic and practical framework for scheduling in stochastic environment. *Journal of Scheduling*, 12:315–344, 2009.

[11] C. Blum. Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[12] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[13] C. Blum, A. Roli, and M. Sampels. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer-Verlag, 2008.

[14] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.

[15] J. Branke and D. Mattfeld. Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research*, 43:3103–3129, 2005.

[16] P. Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.

[17] P. Brucker, J. Hunrink, B. Jurisch, and B. Wöstmann. A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76:43–59, 1997.

[18] M. Brunelli and J. Mezei. How different are ranking methods for fuzzy numbers? A numerical study. *International Journal of Approximate Reasoning*, 54:627–639, 2013.

[19] L. M. Campos Ibañez and A. González Muñoz. A subjective approach for ranking fuzzy numbers. *Fuzzy Sets and Systems*, 29:145–153, 1989.

[20] G. Celano, A. Costa, and S. Fichera. An evolutionary algorithm for pure fuzzy flowshop scheduling problems. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11:655–669, 2003.

[21] C.-T. Chen. Extensions of the topsis for group decision-making under fuzzy environment. *Fuzzy Sets and Systems*, 114:1–9, 2000.

[22] M. Chica, O. Cordón, S. Damas, and J. Bautista. Multiobjective memetic algorithms for time and space assembly line balancing. *Engineering Applications of Artificial Intelligence*, 25:254–273, 2012.

[23] O. Cordon, F. Herrera, and T. Sttzle. A review on the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware & Soft Computing*, 9:141–175, 2002.

[24] R. L. Daniels and J. E. Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29:977–985, 1997.

[25] G. Danoy, B. Dorronsoro, and P. Bouvry. Multi-objective cooperative coevolutionary algorithms for robust scheduling. In *Proc. of EVOLVE-A bridge between Probability, Set Oriented Numerics and Evolutionary Computation*, 2011.

[26] S. Dauzère-Pérès and 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):281–306, 1997.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[28] M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, 41:231–252, 1993.

[29] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, Feb 1996.

[30] D. Dubois. Possibility theory an statistical reasoning. *Computational Statistics & Data Analysis*, 51:47–69, 2006.

[31] D. Dubois, H. Fargier, and P. Fortemps. Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147:231–252, 2003.

[32] D. Dubois, L. Foulloy, G. Mauris, and H. Prade. Probability-possibility transformations, triangular fuzzy sets and probabilistic inequalities. *Reliable Computing*, 10:273–297, 2004.

[33] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, 1980.

[34] D. Dubois and H. Prade. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press, New York (USA), 1986.

[35] D. Dubois, H. Prade, and S. Sandri. On possibility/probability transformations. In *Fuzzy Logic*, volume 12 of *Theory and Decision Library*, pages 103–112. Kluwer Academic, 1993.

[36] P. Fortemps. Jobshop scheduling with imprecise durations: a fuzzy approach. *IEEE Transactions of Fuzzy Systems*, 7:557–569, 1997.

[37] M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, second edition edition, 2010.

[38] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[39] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[40] J. Gonçalves, J. Magalhães Mendes, and R. MGC. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.

[41] M. González, C. R. Vela, and 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)*, pages 91–99, 2013.

[42] I. González-Rodríguez, J. J. Palacios, C. R. Vela, and J. Puente. Heuristic local search for fuzzy open shop scheduling. In *Proceedings IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2010*, pages 1858–1865. IEEE, 2010.

[43] I. González-Rodríguez, J. J. Palacios, C. R. Vela, and J. Puente. Heuristic local search for the fuzz open shop schedluing. In *Proceedings of 2010 IEEE International Conference on Fuzzy Systems*, pages 1858–1865. Institute of Electrical and Electronics Engineers (IEEE), 2010.

[44] I. González Rodríguez, J. Puente, and C. R. Vela. A multiobjective approach to fuzzy job shop problem using genetic algorithms. *CAEPIA 2007, Lecture Notes in Artificial Intelligence*, 4788:80–89, 2007.

[45] I. González Rodríguez, J. Puente, C. R. Vela, and R. Varela. Semantics of schedules for the fuzzy job shop problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 38(3):655–666, 2008.

[46] I. González Rodríguez, C. R. Vela, A. Hernández-Arauzo, and 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)*, pages 154–161, Thesaloniki, 2009. AAAI Press.

[47] I. González Rodríguez, C. R. Vela, and J. Puente. A memetic approach to fuzzy job shop based on expectation model. In *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007*, pages 692–697, London, 2007. IEEE.

[48] I. González Rodríguez, C. R. Vela, J. Puente, and 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)*, pages 124–131, Sidney, 2008. AAAI Press.

[49] S. Heilpern. The expected value of a fuzzy number. *Fuzzy Sets and Systems*, 47:81–86, 1992.

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

[51] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The University of Michigan Press, 1975.

[52] Z. Hong and W. Jian. A cooperative coevolutionary algorithm with application to job shop scheduling problem. In *Service Operations and Logistics, and Informatics 2006, SOLI'06, IEEE International Conference on*, pages 746–751. IEEE, 2006.

[53] E. Hurink, B. Jurisch, and M. Thole. Tabu search for the job shop scheduling problem with multi-purpose machine. *Operations Research Spektrum*, 15:205–215, 1994.

[54] IBM. IBM CPLEX Optimizer, 2014.

[55] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[56] R. Kalaï, C. Lamboray, and D. Vanderpooten. Lexicographic α-robustness: An alternative to min-max criteria. *European Journal of Operational Research*, 220:722–728, 2012.

[57] A. Kasperski. Some general properties of a fuzzy single machine scheduling problem. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 15(1):43–56, 2007.

[58] A. Kaufmann and M. Gupta. *Introduction to Fuzzy Arithmetic*. Van Nostrand Reinhold, New York, 1991.

[59] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[60] D. Lei. Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 37:157–165, 2008.

[61] D. Lei. Fuzzy job shop scheduling problem with availability constraints. *Computers & Industrial Engineering*, 58:610–617, 2010.

[62] D. Lei. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *International Journal of Production Research*, 48(10):2995–3013, 2010.

[63] D. Lei. Solving fuzzy job shop scheduling problems using random key genetic algorithm. *International Journal of Advanced Manufacturing Technologies*, 49:253–262, 2010.

[64] D. Lei. Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Applied Soft Computing*, 12:2237–2245, 2012.

[65] D. Lei and X. Guo. Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling. *International Journal of Production Research*, 50(6):1639–1649, 2012.

[66] F.-m. Li, Y.-l. Zhu, C.-w. Yin, and X.-y. Song. Fuzzy programming for multiobjective fuzzy job shop scheduling with alternative machines through genetic algorithms. In *Advances in Natural Computation*, volume 3611 of *Lecture Notes in Computer Science*, pages 992–1004. Springer, 2005.

[67] H. Li and Q. Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 13(2):284–302, April 2009.

[68] J.-q. Li and Y.-x. Pan. A hybrid discrete particle swarm optimization algorithm for solving fuzzy job shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 66:583–596, 2013.

[69] J. Lin. A hybrid biogeography-based optimization for the fuzzy flexible job-shop scheduling problem. *Knowledge-Based Systems*, 78:59–74, 2015.

[70] L. D. Long and A. Ohsato. Fuzzy critical chain method for project scheduling under resource constraints and uncertainty. *International Journal of Project Management*, 26(6):688–698, 2008.

[71] G. Marquet, B. Derbel, A. Liefooghe, and E.-G. Talbi. Shake them all! rethinking selection and replacement in moea/d. In T. Bartz-Beielstein, J. Branke, B. Filipi, and J. Smith, editors, *Parallel Problem Solving from Nature PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2014.

[72] R. Martí, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169:359–372, 2006.

[73] M. Mastrolilli and L. Gambardella. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

[74] Q. Niu, B. Jiao, and X. Gu. Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. *Applied Mathematics and Computation*, 205:148–158, 2008.

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

[76] I. Ono, M. Yamamura, and S. Kobayashi. A genetic algorithm for job-shop scheduling problems using job-based order crossover. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 547–552. IEEE, 1996.

[77] J. J. Palacios, M. A. González, C. R. Vela, I. González-Rodríguez, and J. Puente. Genetic tabu search for the fuzzy flexible job shop problem. *Computers & Operations Research*, 54:74–89, 2015.

[78] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente. Swarm lexicographic goal programming for fuzzy open shop scheduling. *Journal of Intelligent Manufacturing*, 2013.

[79] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente. Robust swarm optimisation for fuzzy open shop scheduling. *Natural Computing*, 13(2):145–156, 2014.

[80] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente. Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop. *Fuzzy Sets and Systems*, In press, 2015.

[81] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente. A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem. *AI Communications*, 28(2):239–257, 2015.

[82] J. J. Palacios, I. González-Rodríguez, C. R. Vela, and J. Puente Peinador. -robust solutions for the fuzzy open shop scheduling. In A. Laurent, O. Strauss, B. Bouchon-Meunier, and R. R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 442 of *Communications in Computer and Information Science*, pages 447–456. Springer, 2014.

[83] J. J. Palacios, J. Puente, I. González-Rodríguez, and C. R. Vela. Hybrid tabu search for fuzzy job shop. In J. M. Ferrández Vicente, J. R. Álvarez Sánchez, F. Paz López, and F. Toledo Moreo, editors, *Natural and Artificial Models in Computation and Biology*, volume 7930 of *Lecture Notes in Computer Science*, pages 376–385. Springer, 2013.

[84] J. J. Palacios, J. Puente, C. R. Vela, and I. González-Rodríguez. A genetic algorithm for the open shop problem with uncertain durations. In *Proceedings of IWINAC 2009, Part I*, volume 5601 of *Lecture Notes in Computer Science*, pages 255–264. Springer, 2009.

[85] J. J. Palacios, C. R. Vela, I. González-Rodríguez, and J. Puente. Schedule generation schemes for job shop problems with fuzziness. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 687–692. IOS Press, 2014.

[86] S. Petrovic and C. Fayad. A fuzzy shifting bottleneck hybridised with genetic algorithm for real-world job shop scheduling. In *Mini-EURO Conference, Managing Uncertainty in Decision Support Models*, pages 1–6, 2004.

[87] M. L. Pinedo. *Scheduling. Theory, Algorithms, and Systems.* Springer, third edition, 2008.

[88] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. an overview. *Swarm Intelligence*, 1:33–57, 2007.

[89] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. de Jong. Coevolutionary principles. In *Handbook of Natural Computing*, pages 987–1033. Springer, 2012.

[90] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[91] J. Puente, C. R. Vela, and I. González-Rodríguez. Fast local search for fuzzy job shop scheduling. In *Proceedings of ECAI 2010*, pages 739–744. IOS Press, 2010.

[92] B. Roy. Robustness in operational research and decision aiding: A multi-faceted issue. *European Journal of Operational Research*, 200:629–638, 2010.

[93] M. Sakawa and R. Kubota. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. *European Journal of Operational Research*, 120:393–407, 2000.

[94] M. Sakawa and T. Mori. An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate. *Computers & Industrial Engineering*, 36:325–341, 1999.

[95] D. Y. Sha and H. Cheng-Yu. A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, 35:3243–3261, 2008.

[96] R. Słowiński and M. Hapke, editors. *Scheduling Under Fuzziness*, volume 37 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 2000.

[97] X. Song, Y. Zhu, C. Yin, and L. Fuming. A hybrid strategy based on ant colony and taboo search algorithms for fuzzy job shop scheduling. In *Proceedings of the 8th World Congress on Intelligent Control and Automation*, pages 7362–7365, 2006.

[98] A. Sprecher, R. Kolisch, and A. Drexl. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80:94–102, 1995.

[99] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[100] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[101] E.-G. Talbi. *Metaheuristics. From Design to Implementation*. Wiley, 2009.

[102] E.-G. Talbi. *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*. Springer-Verlag, 2013.

[103] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling. Theory, Models and Algorithms*. Springer, second edition, 2006.

[104] T.-D. Tran, R. Varela, I. González-Rodríguez, and E.-G. Talbi. Solving fuzzy job-shop scheduling problems with a multiobjective optimizer. In V. N. Huynh, T. Denoeux, D. H. Tran, A. C. Le, and S. B. Pham, editors, *Knowledge and Systems Engineering*, volume 245 of *Advances in Intelligent Systems and Computing*, pages 197–209. Springer, 2014.

[105] Y. Tsujimura, M. Gen, and E. Kubota. Solving job-shop scheduling problem with fuzzy processing time using genetc algorithm. *Journal of Japan Society for Fuzzy Theory and Systems*, 7:1073–1083, 1995.

[106] P. Van Laarhoven, E. Aarts, and K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[107] W. Van Leekwijck and E. E. Kerre. Defuzzification: criteria and classification. *Fuzzy Sets and Systems*, 108:159–178, 1999.

[108] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 16:139–165, 2010.

[109] E. Vicente, A. Mateos, and A. Jiménez. A new similarity function for generalized trapezoidal fuzzy numbers. In *Artificial Intelligence and Soft Computing*, pages 400–411. Springer, 2013.

[110] B. Wang, Q. Li, X. Yang, and 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*, pages 575–580, 2010.

[111] J. Wang. A fuzzy robust scheduling approach for product development projects. *European Journal of Operational Research*, 152:180–194, 2004.

[112] L. Wang, G. Zhou, Y. Xu, and L. Min. A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. *International Journal of Production Research*, 51(12):3593–3608, 2013.

[113] S. Wang, L. Wang, Y. Xu, and 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):3779–3793, 2013.

[114] W. Wang and Z. Wang. Total orderings defined on the set of all fuzzy numbers. *Fuzzy Sets and Systems*, 243:131–141, 2014.

[115] B. K. Wong and V. S. Lai. A survey of the application of fuzzy set theory in production and operations management: 1998–2009. *International Journal of Production Economics*, 129:157–168, 2011.

[116] C. W. Wu, K. N. Brown, and J. C. Beck. Scheduling with uncertain durations: Modeling β-robust scheduling with constraints. *Computers & Operations Research*, 36:2348–2356, 2009.

[117] Z. Xiang, B. Zhenqiang, W. Guijun, and P. Quanke. Optimization of fuzzy job-shop scheduling with multi-process routes and its co-evolutionary algorithm. In *Intelligent Computation Technology and Automation (ICICTA), 2011 International Conference on*, volume 1, pages 866–870. IEEE, March 2011.

[118] Y. J. Xing, Z. Q. Wang, J. Sun, and J. J. Meng. A multi-objective fuzzy genetic algorithm for job-shop scheduling problems. *Journal of Achievements in Materials and Manufacturing Engineering*, 17:297–300, 2006.

[119] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

[120] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, Dec 2007.

[121] Y. Zheng, Y. Li, and D. Lei. Swarm-based neighbourhood search for fuzzy job shop scheduling. *International Journal of Innovative Computing and Applications*, 3(3):144–151, 2011.

[122] Y.-L. Zheng and Y.-X. Li. Artificial bee colony algorithm for fuzzy job shop scheduling. *International Journal of Computer Applications in Technology*, 44 (2):124–129, 2012.