Improved Local Search for Job Shop Scheduling with Uncertain Durations

Inés González-Rodríguez
Dept. of Mathematics, Statistics and Computing,
University of Cantabria,
39005 Santander (Spain)
e-mail: ines.gonzalez@unican.es

Camino R. Vela and **Jorge Puente**
and **Alejandro Hernández-Arauzo**
Dept. of Computer Science and A.I. Centre,
University of Oviedo, 33271 Gijón (Spain)
e-mail: {crvela,puente,alex}@uniovi.es

Abstract

This paper is concerned with local search methods to solve job shop scheduling problems with uncertain durations modelled as fuzzy numbers. Based on a neighbourhood structure from the literature, a reduced set of moves and the consequent structure are defined. Theoretical results show that the proposed neighbourhood contains all the improving solutions from the original neighbourhood and provide a sufficient condition for optimality. Additionally, a makespan lower bound is proposed which can be used to discard neighbours. Experimental results illustrate the good performance of both proposals, which considerably reduce the computational load of the local search, as well as a synergy effect when they are simultaneously used.

Introduction

Scheduling problems form an important body of research since the late fifties, with multiple applications in industry, finance and science (Pinedo 2008). Traditionally, scheduling has been treated as a deterministic problem that assumes precise knowledge of all data. However, modelling real-world problems often involves processing uncertainty stemming from various sources; among others, activity processing times. In the literature we find different proposals for dealing with uncertainty in scheduling (Herroelen and Leus 2005). Perhaps the best-known approach is stochastic scheduling, where uncertain durations are taken to be stochastic variables. It is also possible to model ill-known durations using fuzzy numbers or, more generally, fuzzy intervals in the setting of possibility theory. It is argued that the latter proposes a natural framework, simpler and less data-demanding than probability theory, for handling incomplete knowledge about scheduling data. Although less common, the fuzzy approach has been around for more than two decades and has received the attention of several researchers (c.f. (Dubois, Fargier, and Fortemps 2003),(Słowiński and Hapke 2000)).

The complexity of scheduling problems such as shop problems means that practical approaches to solving them usually involve heuristic strategies. Extending these strategies to problems with uncertain durations represented as fuzzy numbers usually requires a significant reformulation of both the problem and solving methods. For the job shop, we find a neural approach (Tavakkoli-Moghaddam, Safei, and Kah 2008), genetic algorithms (Sakawa and Kubota 2000),(Petrovic et al. 2008),(González Rodríguez et al. 2008a), simulated annealing (Fortemps 1997) and genetic algorithms hybridised with local search (González Rodríguez et al. 2008b).

In the following, we consider a job shop problem with task durations modelled as triangular fuzzy numbers. Based on a definition of criticality and neighbourhood structure from (González Rodríguez et al. 2008b), we propose an improved local search procedure. A new neighbourhood structure is defined and shown to improve the existing one. Also, a lower bound for the makespan is defined and used to improve the efficiency of the local search. The potential of both proposals, used independently or jointly, is illustrated by the experimental results .

The Fuzzy Job Shop Scheduling Problem

The classical *job shop scheduling problem*, *JSP* in short, consists in scheduling a set of jobs $\{J_1, \ldots, J_n\}$ on a set $\{M_1, \ldots, M_m\}$ 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 m tasks $\{\theta_{i1}, \ldots, \theta_{im}\}$ to be sequentially scheduled. There are also *capacity constraints*, whereby each task θ_{ij} requires the uninterrupted and exclusive use of one of the machines for its whole processing time. A feasible schedule is an allocation of starting times for each task such that all constraints hold. The objective is to find a schedule which is *optimal* according to some criterion, most commonly that the *makespan* is minimal.

Uncertain Durations

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 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. 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 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 $Q_\alpha = \{r \in \mathbb{R} : \mu_Q(r) \geq \alpha\}$, $\alpha \in (0, 1]$, are intervals (bounded or not). The *support* of Q is $Q_0 = \{r \in \mathbb{R} : \mu_Q(r) > 0\}$. A *fuzzy number* is a fuzzy quantity whose α-cuts are closed intervals, with compact support and unique modal value.

In the job shop, we essentially need two operations on fuzzy quantities, 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 (Fortemps 1997) and approximate the results of these operations by a TFN, evaluating only the operation on the three defining points of each TFN. The approximated sum coincides with the sum of TFNs as defined by the Extension Principle, so for any pair of TFNs M and N:

$$M + N = (m^1 + n^1, m^2 + n^2, m^3 + n^3) \quad (2)$$

Regarding the maximum, for any two TFNs M, N, if F denotes their maximum and $G = (\max\{m^1, n^1\}, \max\{m^2, n^2\}, \max\{m^3, n^3\})$ its approximated value, it holds that:

$$\forall \alpha \in [0, 1], \quad \underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha. \quad (3)$$

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 can be trivially extended to the case of more than two 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 (Liu and Liu 2002), given for a TFN A by

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

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. 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 $E[M] \leq E[N]$. Clearly, for any two TFNs A and B, if $\forall i, a^i \leq b^i$, then $A \leq_E B$.

The Disjunctive Graph Model Representation

A job shop problem instance may be represented by a directed graph $G = (V, A \cup D)$. Each node in the set V represents a task of the problem, with the exception of the dummy nodes *start* or 0 and *end* or $nm + 1$, representing tasks with null processing times. Task θ_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, is represented by node $x = m(i-1) + j$. Arcs in A are called *conjunctive arcs* and represent precedence constraints (including arcs from node 0 to the first task of each job and arcs form the last task of each job to node $nm+1$). Arcs in D are called *disjunctive arcs* and represent capacity constraints; $D = \cup_{i=1,\ldots,m} D_i$, where D_i corresponds to machine M_i and includes two arcs (x, y) and (y, x) for each pair (x, y) of tasks requiring that machine. Each arc is weighted with the processing time of the task at the source node (a TFN in our case). A feasible processing order of tasks σ corresponds to an acyclic subgraph $G(\sigma) = (V, A \cup R(\sigma))$ of G, where $R(\sigma) = \cup_{i=1\ldots m} R_i(\sigma)$, $R_i(\sigma)$ being a hamiltonian selection of D_i. Using forward propagation in $G(\sigma)$, we can obtain the starting and completion times for all tasks and, therefore, the makespan $C_{max}(\sigma)$.

Since task 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 tasks and the makespan are fuzzy intervals, interpreted as possibility distributions on the values that the times may take. However, the task processing ordering σ that determines the schedule is crisp; there is no uncertainty regarding the order in which tasks are to be processed.

To illustrate these ideas, consider a problem of 3 jobs and 2 machines with the following matrices for fuzzy processing times and machine allocation:

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

A feasible task processing order for this problem is given by σ =(1 3 5 4 2 6); its corresponding solution graph can be seen in Figure 1.

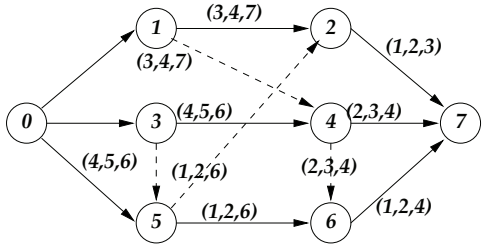

Figure 1: Solution graph $G(\sigma)$ for the processing order σ =(1 3 5 4 2 6). $C_{max}(\sigma) = (7, 10, 16)$

If S_x and C_x denote, respectively, the starting and completion times of a task x, it is easy to check that, for instance for task 4, $S_4 = \max\{C_3, C_1\} = (4, 5, 7)$ and $C_4 = S_4 + p_4 = (6, 8, 11)$. Figure 2 corresponds to the Gantt chart (adapted to TFNs following (Fortemps 1997)) of the resulting schedule, where each block is labelled with the corresponding task number. It shows the final makespan as well as the starting times of tasks scheduled in each machine: for

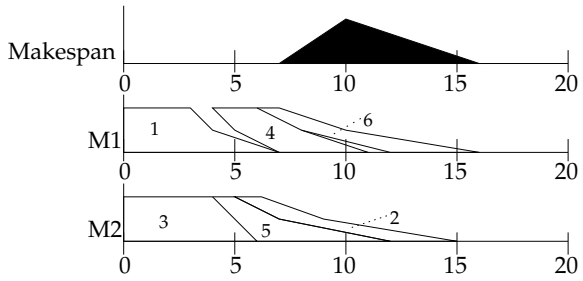

Figure 2: Gantt chart of the schedule represented by (1 3 5 4 2 6).

M_1, $S_1 = (0, 0, 0)$, $S_4 = (4, 5, 7)$, $S_6 = (6, 8, 12)$, and for M_2, $S_3 = (0, 0, 0)$, $S_5 = (4, 5, 6)$, $S_2 = (5, 7, 12)$.

Expected Makespan Model

We have stated the goal of the job shop problem as finding a schedule which is optimal in the sense that the makespan is minimal. However, neither the maximum nor its approximation define a total ordering in the set of TFNs. In a similar approach to stochastic scheduling, it is possible to use the concept of expected value for a fuzzy quantity and the total ordering it provides, so the objective is to minimise the expected makespan $E[C_{max}(\sigma)]$, a crisp objective function.

Criticality

In the crisp case, a *critical path* is defined as the longest path in a solution graph from node *start* to node *end* and a *critical arc* or *critical activity* is an arc or activity in a critical path. It is not trivial to extend these concepts and related algorithms to the problem with uncertain durations (cf. (Dubois, Fargier, and Fortemps 2003)). For the fuzzy job shop considered herein it may even be the case that the makespan (a TFN) does not coincide with the completion time of one job (unlike the crisp case).

In (González Rodríguez et al. 2008b), a definition of criticality is proposed based on the fact that all arithmetic operations used in the scheduling process are performed on the three defining points or components of the TFNs. Let $G(\sigma) = (V, A \cup R(\sigma))$ be a solution graph, where the cost of any arc $(x, y) \in A \cup R(\sigma)$ is a TFN representing the processing time p_x of task x. From $G(\sigma)$, we obtain the *parallel solution graphs* $G^i(\sigma)$, $i = 1, 2, 3$, with identical structure to $G(\sigma)$ but where the cost of any arc (x, y) is p^i_x, the i-th component of p_x. Since durations in each parallel graph $G^i(\sigma)$ are deterministic, a critical path in $G^i(\sigma)$ is the longest path from node *start* to node *end*. Notice that it is not necessarily unique; for instance, Figure 3, shows the three parallel graphs generated from the graph in Figure 1; we see that if $p^3_6 = 3$ there would be two critical paths in $G^3(\sigma)$: (3 5 2) and (3 5 6).

Using the parallel graph representation, criticality for the fuzzy job shop is defined as follows:

Definition 1. *A path P in $G(\sigma)$ is a* critical path *if and only if P is critical in some $G^i(\sigma)$. Nodes and arcs in a critical path are 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 of a critical path requiring the same machine.*

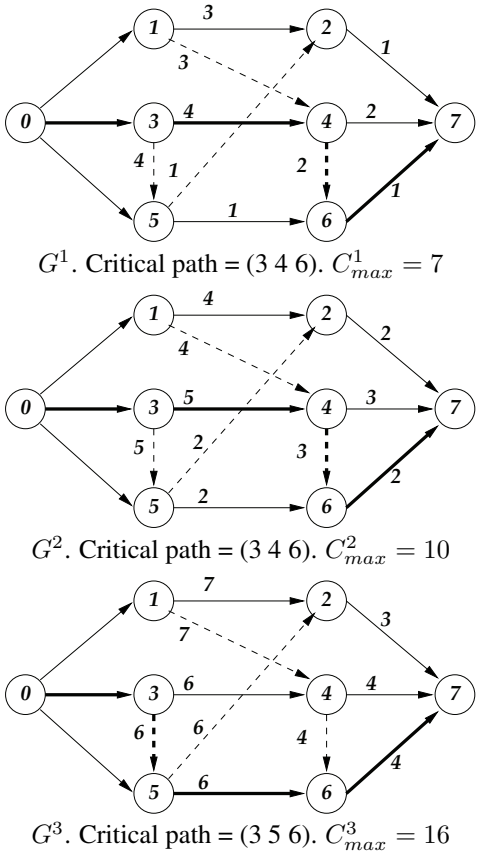

Figure 3: Parallel graphs corresponding to the graph in Figure 1, with critical paths in bold.

Clearly, the sets of critical paths, arcs, tasks and blocks in $G(\sigma)$ are respectively the union of critical paths, arcs, tasks and blocks in the parallel solution graphs. The makespan of the schedule is not necessarily the cost of a critical path, but each component $C^i_{max}(\sigma)$ is the cost of a critical path in the corresponding solution parallel graph $G^i(\sigma)$.

Improved Local Search

Part of the interest of critical paths stems from the fact that they may be used to define neighbourhood structures for local search. Roughly speaking, a typical local search schema starts from a given processing order, calculates its neighbourhood and then neighbours are evaluated in the search of an improving solution. In *steepest descent hill-climbing*, all neighbours are evaluated, the best one is selected and it replaces the original solution if it is an improving neighbour. In *simple hill-climbing*, evaluation stops as soon as the first improving neighbour is found. Local search starts again from that improving neighbour, so the procedure finishes when no neighbour satisfies the acceptation criterion.

Previous Approaches

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 (Van Laarhoven, Aarts, and Lenstra 1992), extended to the fuzzy case in (González Rodríguez et al. 2008b) using the given definition of criticality as follows:

Definition 2. *Given a task processing order π and an arc $v = (x, y) \in R(\pi)$, let $\pi_{(v)}$ denote the processing order obtained from π after an exchange in the processing order of tasks in arc v. Then, the* neighbourhood structure *obtained from π is given by $H(\pi) = \{\pi_{(v)} : v \in R(\pi)$ is critical$\}$.*

This neighbourhood H has a highly desirable property:

Theorem 1. *Let π be a feasible task processing order; then all elements in $H(\pi)$ are feasible.*

Feasibility limits the local search to the subspace of feasible task orders and avoids feasibility checks for the neighbours, hence reducing computational load and avoiding the loss of feasible solutions usually encountered for feasibility checking procedures.

An earlier proposal to extend Van Laarhoven's neighbourhood structure to the fuzzy case can be found in (Fortemps 1997), using an alternative definition of critical path. It turns out that the set of critical arcs, according to the definition based on parallel graphs, is a strict subset of the critical arcs according to the earlier definition. Hence, the neighbourhood H is strictly included in that defined in (Fortemps 1997), denoted H'. It can be shown that those neighbours from $H' - H$ can never improve the makespan. This is a consequence of the following property:

Proposition 1. *Let π be a feasible processing order and let $\sigma = \pi_{(v)}$ where v is not critical in $G(\pi)$. Then*

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

In (González Rodríguez et al. 2008b), a local search procedure using H is hybridised with a genetic algorithm, obtaining a memetic algorithm, for which competitive experimental results are reported. Despite being satisfactory, the results also suggest that the algorithm has reached its full potential and, importantly, most of the computational time it requires corresponds to the local search. In order to obtain better metaheuristics for the fuzzy job shop, it is necessary to improve the neighbourhood structure and reduce the computational cost of the local search. Additionally, the parallel graph framework causes neighbourhood structures in the fuzzy case to usually contain considerably more individuals than in the classical setting, with the consequent increase in computational cost. This further justifies the need of a greater effort to improve efficiency.

In the following, we propose to improve the local search in three ways. A first idea is to change the scheduling method in the local search algorithm, evaluating neighbours in a more efficient manner. A second idea is to reduce the number of neighbours with more complex structures than "simply" inverting any critical arc, an approach successful both for the classical job shop and for the job shop with setup times (Nowicki and Smutnicki 1996),(González, Vela, and Varela 2008). A third possibility is to avoid evaluating certain neighbours by calculating lower bounds of their makespan, as done, for instance, in (Taillard 1994).

Makespan Calculation for Neighbours

The well-known concepts of head and tail of a task are easily extended to the fuzzy framework. For a solution graph $G(\pi)$ and a task x, let $P\nu_x$ and $S\nu_x$ denote the predecessor and successor nodes of x on the machine sequence (in $R(\pi)$) and let PJ_x and SJ_x denote the predecessor and successor nodes of x on the job sequence (in A). The *head* of task x is the starting time of x, a TFN given by $r_x = \max\{r_{PJ_x} + p_{PJ_x}, r_{P\nu_x} + p_{P\nu_x}\}$, and the *tail* of task x is the time lag between the moment when x is finished until the completion time of all tasks, a TFN given by $q_x = \max\{q_{SJ_x} + p_{SJ_x}, q_{S\nu_x} + p_{S\nu_x}\}$.

Clearly, the makespan coincides with the head of the last task and the tail of the first task: $C_{max} = r_{nm+1} = q_0$; other basic properties that hold for each parallel graph $G^i(\pi)$ are the following: r^i_x is the length of the longest path from node 0 to node x; $q^i_x + p^i_x$ is the length of the longest path from node x to node $nm + 1$; and $r^i_x + p^i_x + q^i_x$ is the length of the longest path from node 0 to node $nm + 1$ through node x: it is a lower bound for $C^i_{max}(\pi)$, being equal if node x belongs to a critical path in $G^i(\pi)$.

Given a task processing order π and a critical arc (x, y) in $G(\pi)$, the reversal of that arc produces a new feasible processing order $\sigma = \pi_{(x,y)}$ with solution graph $G(\sigma)$. This situation is illustrated in Figure 4. The makespan after the

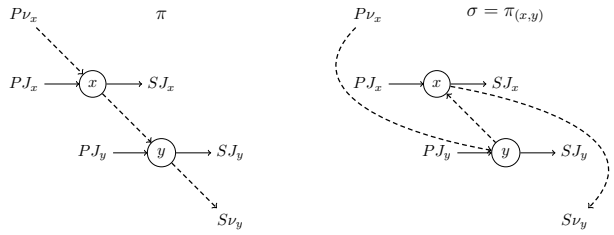

Figure 4: Situation before (π) and after (σ) the reversal of a critical arc (x, y).

move may be calculated as for any solution, using forward propagation in the graph $G(\sigma)$ from 0. This has been the method used in (González Rodríguez et al. 2008b). Alternatively, for the classical job shop, "the evaluation of the makespan of neighbouring solutions may be done very quickly (in time $O(N)$)" (Taillard 1994). This is still so in the fuzzy case.

Let r and q denote the heads and tails in $G(\pi)$ (before the move) and let r' and q' denote the heads and tails in $G(\sigma)$ (after the move). For every task a previous to x in π, $r_a = r'_a$ and for every task b posterior to y in π, $q_b = q'_b$. The heads and tails for x and y after the move (see Figure 4) are given

by the following:

$$\begin{aligned} r'_y &= \max\{r_{PJ_y} + p_{PJ_y}, r_{P\nu_x} + p_{P\nu_x}\}, \\ r'_x &= \max\{r_{PJ_x} + p_{PJ_x}, r'_y + p_y\} \\ q'_x &= \max\{q_{SJ_x} + p_{SJ_x}, q_{S\nu_y} + p_{S\nu_y}\} \\ q'_y &= \max\{q_{SJ_y} + p_{SJ_y}, q'_x + p_x\} \end{aligned} \tag{6}$$

To calculate the makespan $C_{max}(\sigma)$, we need only recalculate the heads of tasks from x onwards in the graph $G(\sigma)$. We propose to incorporate this way of evaluating neighbours in H to the local search algorithm, an idea which, albeit simple, may prove a considerable reduction in computational load.

Makespan Lower Bound

At each iteration of the local search, only those neighbours with improving makespan are of interest. Another way of reducing computational cost is to foresee, by means of a makespan lower bound, that certain neighbours are for sure not improving. A well-known and inexpensive lower bound for the makespan in the crisp case was proposed by Taillard in (Taillard 1994). In the following, we generalise this lower bound to the fuzzy case.

For a processing order σ and tasks x and y, let $P_\sigma(x \vee y)$ denote the set of all paths in the solution graph $G(\sigma)$ containing x or y, $P_\sigma(x \wedge y)$ denote the set of all paths in $G(\sigma)$ containing both x and y and let $P_\sigma(\neg x)$ denote de set of all paths in $G(\sigma)$ not containing x. Also, for a given set of paths P, let $D[P]$ denote the TFN such that $D^i[P]$ is the length of the longest path from P in the parallel graph G^i, $i = 1, 2, 3$.

Proposition 2. *Let $\sigma = \pi_{(v)}$, where $v = (x, y)$ is an arc in $G(\pi)$. Then, the makespan for the new solution is given by:*

$$C_{max}(\sigma) = \max\{D[P_\sigma(x \vee y)], D[P_\pi(\neg x)]\} \tag{7}$$

Proof. For every $i = 1, 2, 3$ it holds that:

$$C^i_{max}(\sigma) = \max\{D^i[P_\sigma(x \vee y)], D^i[P_\sigma(\neg x \wedge \neg y)]\} \tag{8}$$

Since the only arcs that change between $G(\pi)$ and $G(\sigma)$ are $(P\nu_x(\pi), x)$, (x, y), $(y, S\nu_y(\pi))$, those paths not containing x nor y are the same in both graphs $G(\pi)$ and $G(\sigma)$, so:

$$C^i_{max}(\sigma) = \max\{D^i[P_\sigma(x \vee y)], D^i[P_\pi(\neg x \wedge \neg y)]\} \tag{9}$$

Now, for every path in $G(\pi)$ containing y but not containing x, either it starts in y or it contains the arc (PJ_y, y). In both cases, the subpath to y is identical in both $G(\pi)$ and $G(\sigma)$. If the path does not contain $S\nu_y$, it is still a path in $G(\sigma)$ and if it does contain the arc $(y, S\nu_y)$, then substituting (x, y) by $(y, x), (x, S\nu_y)$ we obtain a longer path in $G(\sigma)$. Therefore, $D^i[P_\pi(\neg x \wedge y)] \leq D^i[P_\sigma(x \vee y)]$ and we may rewrite (9) as follows:

$$\begin{aligned} C^i_{max}(\sigma) = \max\{&D^i[P_\sigma(x \vee y)], \\ &D^i[P_\pi(\neg x \wedge \neg y)], D^i[P_\pi(\neg x \wedge y)]\} \\ = \max\{&D^i[P_\sigma(x \vee y)], D^i[P_\pi(\neg x)]\} \end{aligned}$$

□

The previous proposition shows that $C_{max}(\sigma)$ can be calculated as the maximum of two elements; the former provides an easy-to-compute lower bound for $C_{max}(\sigma)$:

Corollary 1. *The TFN defined as:*

$$LB = \max\{r'_x + p_x + q'_x, r'_y + p_y + q'_y\} \tag{10}$$

provides a lower bound *for the makespan $C_{max}(\sigma)$ obtained after the reversal of arc (x, y) (for $i = 1, 2, 3$ $LB^i \leq C^i_{max}(\sigma)$ and hence $LB \leq_E C_{max}(\sigma)$).*

This lower bound, extending that in (Taillard 1994) for the crisp job shop, can be calculated in time $O(1)$.

Reduced Neighbourhood

A third possibility to increase the efficiency of local search is to reduce the neighbourhood size without discarding improving solutions. For the crisp case, it is proposed in (Nowicki and Smutnicki 1996) to consider only the reversal of arcs which are in the extreme of a critical block.

In the fuzzy setting if (x, y) is critical in some component and is in the extreme of a critical block, its reversal may yield an improving solution but it may also be the case that the three arcs $(P\nu_x, x)$, (x, y), $(y, S\nu_y)$ are all critical arcs (in different components) and do not belong to the same critical block or they may be inside a critical block for one component and in the extreme of a block for another component. In these cases, the reversal of arc (x, y) may produce an improvement.

Let π be feasible processing order; for each $G^i(\pi)$ we select a single critical path $u_i = (u_{i,1}, \ldots, u_{i,\omega_i})$ containing ω_i tasks, which can be decomposed into critical blocks $u_i = B_{i,1}, \ldots, B_{i,r_i}$. Let us denote the tasks in each block as follows: $B_{i,j} = (u_{a_{i,j}}, u_{a_{i,j}+1}, \ldots, u_{b_{i,j}})$, so $a_{i,j}$ and $b_{i,j}$ denote respectively the indices of the first and the last task in the block $B_{i,j}$, $j = 1, \ldots, r_i$. With this notation, $u_{i,1} = u_{a_{i,1}}$, $u_{i,\omega_i} = b_{i,r_i}$ and $a_{i,1} \leq b_{i,1} < b_{i,1} + 1 = a_{i,2} \leq b_{i,2} \ldots a_{i,r_i} \leq b_{i,r_i}$. For each block $B_{i,j}$, the arcs candidate for a move will be those in the extreme of the block: $(u_{a_{i,j}}, u_{a_{i,j}+1})$ and $(u_{b_{i,j}-1}, u_{b_{i,j}})$. More formally:

Definition 3. *The* set of arcs candidates for a move on component i *is $\mathcal{V}(\pi, i) = \cup_{j=1}^{r_i} \mathcal{V}_j(\pi, i)$ where:*

$$\mathcal{V}_1(\pi, i) = \begin{cases} \{(u_{b_{i,1}-1}(i), u_{b_{i,1}})\} & \text{if } a_{i,1} \neq b_{i,1}, r_i > 1, \\ \emptyset & \text{otherwise.} \end{cases}$$

$$\mathcal{V}_j(\pi, i) = \begin{cases} \{(u_{a_{i,j}}, u_{a_{i,j}+1}), (u_{b_{i,j}-1}, u_{b_{i,j}})\} \\ \quad \text{if } a_{i,j} \neq b_{i,j} \\ \emptyset \quad \text{otherwise.}, \qquad j = 2, \ldots, r_i - 1 \end{cases} \tag{11}$$

$$\mathcal{V}_{r_i}(\pi, i) = \begin{cases} \{(u_{a_{i,r_i}}, u_{a_{i,r_i}+1})\} & \text{if } a_{i,r_i} \neq b_{i,r_i}, r_i > 1, \\ \emptyset & \text{otherwise.} \end{cases}$$

The set of arcs from π candidates for a move *is obtained as the union over the three components: $\mathcal{V}(\pi) = \cup_{i=1,2,3} \mathcal{V}(\pi, i)$.*

It follows directly from the definition that $\mathcal{V}(\pi)$ is a subset of the critical arcs in $G(\pi)$. The reduced neighbourhood will be defined based on moves of arcs in $\mathcal{V}(\pi)$:

Definition 4. *The* reduced neighbourhood structure *obtained from π is given by*

$$H_R(\pi) = \{\pi_{(v)} : v \in \mathcal{V}(\pi)\}. \tag{12}$$

Clearly, $H_R(\pi) \subset H(\pi)$ and hence contains only feasible schedules. The following result shows that those discarded neighbours from $H(\pi) - H_R(\pi)$ never improve the makespan of the original solution π.

Theorem 2. *Let π be a feasible processing order.*

$$\forall \sigma \in H(\pi) - H_R(\pi), E[C_{max}(\pi)] \leq E[C_{max}(\sigma)] \quad (13)$$

Proof. For $i = 1, 2, 3$ let $u_i = (u_{i,1}, \ldots, u_{i,\omega_i})$ be the single critical path in $G^i(\pi)$ selected to generate $H_R(\pi)$. Let $\sigma \in H(\pi) - H_R(\pi)$, i.e., $\sigma = \pi_{(v)}$ where $v = (x, y)$ such that: (1) $(x, y) \in R(\pi)$; (2) for at least one component i, x, y belong to a critical path u' in $G^i(\pi)$ (u' need not be equal to u_i); (3) $(x, y) \notin \mathcal{V}(\pi)$.

For any component j for which (x, y) is not critical, let P be a critical path in $G^j(\pi)$ (with length $C^j_{max}(\pi)$). Clearly, (x, y) does not belong to P. Let us see that there always exists a path in $G^j(\sigma)$ with greater or equal length than P (and, therefore, $C^j_{max}(\sigma) \geq C^j_{max}(\pi)$). We consider three cases:

1. P does not contain any of the arcs $(P\nu_x, x)$, (x, y), $(y, S\nu_y)$ that change in the move from π to σ. In this case, P is also a path in $G^j(\sigma)$.
2. P contains arc $(P\nu_x, x)$. In this case, if $P_{a,b}$ denotes the subpath of P from node a to node b, we have $P = P_{0,P\nu_x}, x, P_{SJ_x,nm+1}$ and $P_{0,P\nu_x}, y, x, P_{SJ_x,nm+1}$ is a path in $G^j(\sigma)$ longer than P.
3. P contains arc $(y, S\nu_y)$. Here the proof is symmetric to the previous case.

For a component i in which (x, y) is critical, if (x, y) is not in the critical path u_i selected to generate $H_R(\pi)$, the same arguments as above (taking $P = u_i$) can be used to prove that $C^i_{max}(\pi) \leq C^i_{max}(\sigma)$. If (x, y) is in u_i, since $(x, y) \notin V(\pi, i)$, we have three possibilities:

1. (x, y) is inside a critical block $B_{i,j}$, $1 < j < r_i$ and therefore $P\nu_x, x, y, S\nu_y$ are in the block $B_{i,j}$.
2. (x, y) is in the first block $B_{i,1}$ and it is not the last arc in the block, therefore, $P\nu_x, x, y, S\nu_y$ are in the block $B_{i,1}$ (where $P\nu_x$ may not exist if x is the first node in the path).
3. (x, y) is in the last block B_{i,r_i} and it is not the first arc in the block, therefore, $P\nu_x, x, y, S\nu_y$ are in the block $B_{i,1}$ (where $S\nu_y$ may not exist if y is the last node in the path).

Let us then suppose, without loss of generality, that $P\nu_x, x, y, S\nu_y$ all exist and are inside a critical block. It follows:

$$C^i_{max}(\pi) = r^i_{S\nu_y} + p^i_{S\nu_y} + q^i_{S\nu_y} \quad (14)$$

$$r^i_{S\nu_y} = r^i_{P\nu_x} + p^i_x + p^i_y + p^i_y \quad (15)$$

In $G(\sigma)$, since the arcs before $P\nu_x$ have not changed, $r'_{P\nu_x} = r_{P\nu_x}$, and in consequence:

$$\begin{aligned} r'^i_{S\nu_y} &\geq r'^i_x + p^i_x \geq r'^i_y + p^i_y + p^i_x \\ &\geq r'^i_{P\nu_x} + p^i_{P\nu_x} + p^i_y + p^i_x \\ &= r^i_{P\nu_x} + p^i_{P\nu_x} + p^i_y + p^i_x \end{aligned} \quad (16)$$

From (15) and (16) we have $r'^i_{S\nu_y} \geq r^i_{S\nu_y}$. Similarly, we obtain the inequality $q'^i_{S\nu_y} \geq q^i_{S\nu_y}$. Therefore:

$$\begin{aligned} C^i_{max}(\sigma) &\geq r'^i_{S\nu_y} + p^i_{S\nu_y} + q'^i_{S\nu_y} \\ &\geq r^i_{S\nu_y} + p^i_{S\nu_y} + q^i_{S\nu_y} = C^i_{max}(\pi) \end{aligned}$$ □

The proposed neighbourhood $H_R(\pi)$ (in fact, the definition of the subset of critical arcs $\mathcal{V}(\pi)$) allows to formulate a sufficient condition for optimality of a processing order.

Proposition 3. *If the set of candidates for a move is empty, $\mathcal{V}(\pi) = \emptyset$, then π is an optimal processing order.*

Proof. By definition, $\mathcal{V}(\pi)$ is empty if and only if $\mathcal{V}(\pi, i)$ is empty for all $i = 1, 2, 3$. $\mathcal{V}(\pi, i)$ can be empty in one of the two following cases:

1. $a_{i,j} = b_{i,j}$, $j = 1, \ldots, r_i$, i.e., all blocks have size 1, so all arcs correspond to precedence constraints in a job. Thus, all tasks from the critical path u_i belong to the same job J_{i*} and the length of this path (the makespan C_{max}) corresponds to the makespan lower bound given by the sum of durations of all tasks in a job: $\sum_{j=1}^{m} p_{i*j}$. Hence the makespan is optimal.
2. $r_i = 1$, i.e., all tasks in u_i belong to the same block B_1 and, therefore, they are all processed in the same machine M_{j*}. As above, the length of the path u_i (the makespan) coincides with the lower bound given by the sum of durations of tasks in a machine: $\sum_{i=1}^{n} p_{ij*}$. □

This guarantees that if the local search procedure stops because the neighbourhood is empty, not only has it found a local optimum, but a global one.

Experimental Results

We now consider 12 benchmark problems for job shop: the well-known FT10 (size 10×10) and FT20 (20×5), and La21, La24, La25 (15×10), La27, La29 (20×10), La38, La40 (15×15), and ABZ7, ABZ8, ABZ9 (20×15), the set of 10 problems identified in (Applegate and Cook 1991) as hard to solve for classical JSP. Ten fuzzy versions of each benchmark are generated following (Fortemps 1997) and (González Rodríguez et al. 2008b), so task durations become symmetric TFNs where the modal value is the original duration, ensuring that the optimal solution to the crisp problem provides a lower bound for the fuzzified version.

The goal of this section is to evaluate empirically the contribution of our proposals to improving local search efficiency. We consider the memetic algorithm presented in (González Rodríguez et al. 2008b), denoted GVPV08 in the following, which compared favourably with previous approaches from the literature in terms of makespan optimisation. GVPV08 combines a genetic algorithm with a local search procedure. In the GA, chromosomes are permutations with repetition, decodified using G&T algorithm, with the expected makespan as fitness function. Chromosomes are paired at random and mated using job order crossover (JOX) to obtain two offsprings; acceptance consists in selecting the best individuals from both parents and their offsprings. LS with simple hill-climbing is applied to every

Table 1: Neighbour scheduling algorithm: CPU time (in seconds) of MA vs. GVPV08

Problem	Size	GVPV08	MA	Red%
FT10	10×10	801.2	588.2	26.59%
FT20	20×5	1693.9	682.1	59.73%
La21	15×10	1769.4	1072.8	39.37%
La24	15×10	1562.4	950.1	39.19%
La25	15×10	1722.8	993.7	42.32%
La27	20×10	4137.8	2242.8	45.80%
La29	20×10	3936.0	2071.7	47.37%
La38	15×15	3037,6	2556.7	15.83%
La40	15×15	3220.4	2652.2	17.64%
ABZ7	20×15	7396.1	5294.7	28.41%
ABZ8	20×15	8098.5	5780.9	28.62%
ABZ9	20×15	7308.0	5652.1	22.66%

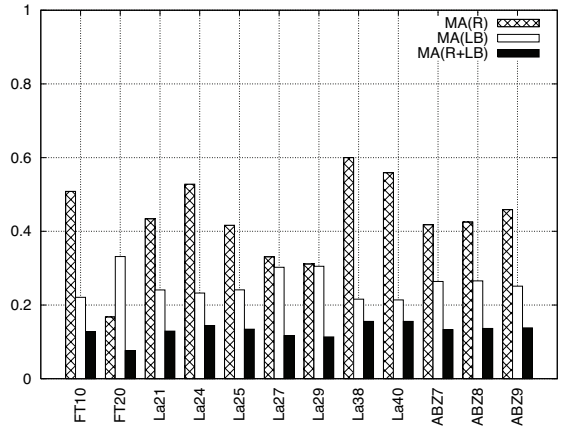

Figure 5: Proportion of evaluated neighbours w.r.t. MA

chromosome immediately after its generation, with a high computational load. We shall use GVPV08 as a baseline algorithm and introduce the different improvements proposed herein in the LS module, to evaluate their contribution to increasing efficiency.

The first experiment aims at evaluating the benefit of scheduling neighbours using heads and tails. We change the way in which neighbours are scheduled in GVPV08 and obtain a new hybrid algorithm denoted MA, which is run with the same parameters as GVPV08. Notice that the schedules will be the same with both methods (it is only the way of calculating them that changes), so the final solution obtained with both proposals is identical in terms of makespan. Table 1 shows the average CPU time in seconds taken by 30 runs of GVPV08 and MA on every family of ten fuzzy instances, showing a clear reduction in time for MA, with the reduction rate varying across different problem families: the minimum (15.83%, 17.64%) is obtained for the square problems of size 15×15 and the maximum (59.73%) is obtained for FT20 of size 20×5. For identical number of jobs, the greater the number of resources, the greater the reduction. In any case, the scheduling based on heads and tails is always more efficient than the original one, with an average reduction in CPU time of 34.5%,

Having ascertained the net contribution of the new scheduling algorithm to local search efficiency, we proceed to analyse the remaining proposals. We consider three variations of the most efficient algorithm MA: incorporating the lower bound to avoid unnecessary evaluations, denoted MA(LB); changing the neighbourhood from H to H_R, denoted MA(R); and combining both approaches denoted MA(R+LB). Again, changes w.r.t. GVPV08 do not concern makespan values and the parameters used for GVPV08 guaranteed convergence, so there is no point in comparing versions based on makespan values nor in prolonging computation time in an attempt to improve the makespan of the final solution. The interest is instead in evaluating the contribution of the proposals to reducing the number of evaluated neighbours and the CPU time required by local search.

Table 2 shows the number of evaluated neighbours in 30 runs of MA(R), MA(LB) and MA(R+LB) compared to MA; for the latter, the number of evaluated neighbours is the same as for GVPV08. Figure 5 illustrates that both proposals H_R and LB considerably reduce the number of evaluated neighbours. In general, the reduction is greater for LB used to discard neighbours (75% in average) than for H_R (57.02% in average), although there are differences depending on the problem structure. A common phenomenon for all problems, regardless of their structure, is the accumulative effect of LB and H_R: the proportion of evaluated neighbours for MA(R+LB) is approximately half of the minimum proportion of the two separate methods. This shows the synergy in the combined use of both proposals; the number of evaluated neighbours for MA(R+LB) is practically reduced in an order of magnitude with respect to the original MA.

Table 2 also shows the CPU time for a total of 30 runs of each algorithm, averaged across the ten fuzzy instances of each family of problems. Again, each proposal supposes a considerable reduction w.r.t. the MA, in general greater for MA(LB) (72.57% in average) than for MA(R) (44.59%). The synergy effect is not as striking as above; still CPU times for MA(R+LB) are sensibly smaller. For H_R, the proportion of time dedicated by the MA to evaluating neighbours is reduced, but the time necessary to compute neighbours is increased with the search of arcs in the extreme of critical blocks. In fact, the number of solutions evaluated in the local search and the required CPU time are in average 25.70% and 27.43% of the total for MA(LB), whilst for MA(R) they are 42.98% and 55.41% of the total. This shows that approximately 10% of computation time is dedicated to finding extremes of critical blocks, a search also performed in MA(R+LB), where the number of evaluated neighbours is reduced to 13.01% but CPU time is reduced to 23.51% in average.

Conclusions

We have considered the job shop scheduling problem with uncertain processing times modelled as triangular fuzzy numbers and where the objective is to minimise the expected makespan. With the aim of improving local search effi-

Table 2: Number of evaluated neighbours and CPU time (in seconds)

Problem	MA		MA(R)		MA(LB)		MA(R+LB)	
	Neigh.	CPU	Neigh.	CPU	Neigh.	CPU	Neigh.	CPU
FT10	2.83E+07	588.2	1.44E+07	372.90	6.25E+06	179.3	3.61E+06	166.5
FT20	6.78E+07	682.1	1.14E+07	283.1	2.25E+07	295.4	5.16E+06	211.0
La21	4.46E+07	1072.8	1.93E+07	610.9	1.07E+07	327.8	5.76E+06	292.2
La24	3.87E+07	950.1	2.04E+07	636.4	9.00E+06	304.5	5.59E+06	284.0
La25	4.28E+07	993.7	1.78E+07	555.3	1.03E+07	322.4	5.75E+06	289.5
La27	8.37E+07	2242.8	2.77E+07	1072.5	2.53E+07	641.9	9.79E+06	511.6
La29	7.85E+07	2071.7	2.44E+07	962.8	2.40E+07	609.8	8.90E+06	485.8
La38	5.16E+07	2556.7	3.10E+07	1732.8	1.11E+07	546.7	8.02E+06	514.0
La40	5.42E+07	2652.2	3.03E+07	1643.2	1.16E+07	565.1	8.42E+06	531.5
ABZ7	9.74E+07	5294.7	4.07E+07	2671.0	2.57E+07	1073.8	1.30E+07	898.4
ABZ8	1.08E+08	5780.9	4.60E+07	2936.5	2.87E+07	1168.0	1.48E+07	975.2
ABZ9	9.65E+07	5652.1	4.43E+07	3107.7	2.42E+07	1078.5	1.33E+07	925.8

ciency, we have proposed a new neighbourhood structure and have shown that it improves previous proposals, since it reduces the set of neighbours by excluding non-improving ones. Additionally, lower bound for the makespan has been proposed which allows to discard non-improving neighbours in an efficient way without evaluating them. The experimental results clearly show that the new neighbourhood structure and the lower bound, used independently but even more when combined, significantly reduce the computational load of local search and greatly improve its efficiency.

Acknowledgments

This work is supported by MEC-FEDER Grant TIN2007-67466-C02-01.

References

Applegate, D., and Cook, W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing* 3:149–156.

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

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

Fortemps, P. 1997. Jobshop scheduling with imprecise durations: a fuzzy approach. *IEEE Transactions of Fuzzy Systems* 7:557–569.

González Rodríguez, I.; Puente, J.; Vela, C. R.; and Varela, R. 2008a. 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.; Vela, C. R.; Puente, J.; and Varela, R. 2008b. A new local search for the job shop problem with uncertain durations. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 124–131. Sidney: AAAI Press.

González, M. A.; Vela, C. R.; and Varela, R. 2008. A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*. Sidney: AAAI Press.

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

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

Nowicki, E., and Smutnicki, C. 1996. A fast taboo search algorithm for the job shop scheduling problem. *Management Science* 42:797–813.

Petrovic, S.; Fayad, C.; Petrovic, D.; Burke, E.; and Kendall, G. 2008. Fuzzy job shop scheduling with lot-sizing. *Annals of Operations Research* 159:275–292.

Pinedo, M. L. 2008. *Scheduling. Theory, Algorithms, and Systems.* Springer, third edition.

Sakawa, M., and Kubota, R. 2000. 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.

Słowiński, R., and Hapke, M., eds. 2000. *Scheduling Under Fuzziness*, volume 37 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag.

Taillard, E. D. 1994. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing* 6(2):108–117.

Tavakkoli-Moghaddam, R.; Safei, N.; and Kah, M. 2008. Accessing feasible space in a generalized job shop scheduling problem with the fuzzy processing times: a fuzzy-neural approach. *Journal of the Operational Research Society* 59:431–442.

Van Laarhoven, P.; Aarts, E.; and Lenstra, K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40:113–125.