# Improving Local Search for the Fuzzy Job Shop using a Lower Bound

Jorge Puente[1], Camino R. Vela[1],
Alejandro Hernández-Arauzo[1], and Inés González-Rodríguez[2]

[1] A.I. Centre and Department of Computer Science,
University of Oviedo, (Spain) {puente,alex,crvela}@uniovi.es,
http://www.aic.uniovi.es/Tc
[2] Department of Mathematics, Statistics and Computing,
University of Cantabria, (Spain) ines.gonzalez@unican.es

**Abstract.** We consider the fuzzy job shop problem, where uncertain durations are modelled as fuzzy numbers and the objective is to minimise the expected makespan. A recent local search method from the literature has proved to be very competitive when used in combination with a genetic algorithm, but at the expense of a high computational cost. Our aim is to improve its efficiency with an alternative rescheduling algorithm and a makespan lower bound to prune non-improving neighbours. The experimental results illustrate the success of our proposals in reducing both CPU time and number of evaluated neighbours.

## 1 Introduction

Scheduling forms an important body of research since the late fifties, with multiple applications in industry, finance and science [1]. Traditionally, it has been treated as a deterministic problem that assumes precise knowledge of all data. However, modelling real-world problems often involves processing uncertainty, for instance in activity durations. In the literature we find different proposals for dealing with ill-known durations [2]. Perhaps the best-known approach is to treat them as stochastic variables. An alternative is to use fuzzy numbers or, more generally, fuzzy intervals in the setting of possibility theory, which is said to provide a natural framework, simpler and less data-demanding than probability theory, for handling incomplete knowledge about scheduling data (c.f. [3],[4]).

The complexity of scheduling problems such as job shop means that practical approaches to solving them usually involve heuristic strategies [5]. Extending these strategies to problems with fuzzy durations in general requires a significant reformulation of both the problem and solving methods. Proposals from the literature include a neural approach [6], genetic algorithms [7],[8],[9], simulated annealing [10] and genetic algorithms hybridised with local search [11],[12].

In the following, we consider a job shop problem with task durations given as triangular fuzzy numbers. Based on a definition of criticality and neighbourhood structure from [12], a new rescheduling algorithm is given and a lower bound for the makespan is defined and used to increase the efficiency of the local search. The potential of the proposals is illustrated by the experimental results.

## 2 Job Shop Scheduling with Uncertain Durations

The *job shop scheduling problem*, also denoted *JSP*, consists in scheduling a set of jobs $\{J_1, \ldots, J_n\}$ on a set of physical resources or machines $\{M_1, \ldots, M_m\}$, subject to a set of constraints. There are *precedence constraints*, so each job $J_i$, $i = 1, \ldots, n$, consists of $m$ tasks $\{\theta_{i1}, \ldots, \theta_{im}\}$ to be sequentially scheduled. Also, there are *capacity constraints*, whereby each task $\theta_{ij}$ 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.

### 2.1 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, and only some uncertain knowledge is available. 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. For a TFN $N$, denoted $N = (n^1, n^2, n^3)$, the membership function takes the following triangular shape:

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

In the 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, we follow [10] 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 may not be the case 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 [13], 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 [3]. It induces a total ordering $\leq_E$ in the set of fuzzy numbers [10], where for any two fuzzy numbers $M, N$ $M \leq_E N$ if and only if $E[M] \leq E[N]$.

### 2.2 Fuzzy Job Shop Scheduling

A job shop problem instance may be represented by a directed graph $G = (V, A \cup D)$. $V$ contains one node $x = m(i-1) + j$ per task $\theta_{ij}$, $1 \leq i \leq n$, $1 \leq$

$j \leq m$, plus two additional nodes 0 (or *start*) and $nm+1$ (or *end*), representing dummy tasks with null processing times. Arcs in $A$, called *conjunctive arcs*, represent precedence constraints (including arcs from node *start* to the first task of each job and arcs form the last task of each job to node *end*). Arcs in $D$, called *disjunctive arcs*, represent capacity constraints; $D = \cup_{j=1}^{m} D_j$, 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 $(x, y)$ is weighted with the processing time $p_x$ of the task at the source node (a TFN in our case). A feasible task processing order $\sigma$ is represented by a *solution graph*, an acyclic subgraph of $G$, $G(\sigma) = (V, A \cup R(\sigma))$, 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 $G(\sigma)$, it is possible to obtain the starting and completion times for all tasks and, therefore, the schedule and the makespan $C_{max}(\sigma)$.

The schedule will be fuzzy in the sense that the starting and completion times of all tasks and the makespan are TFNs, interpreted as possibility distributions on the values that the times may take. However, the task processing ordering $\sigma$ that determines the schedule is crisp; there is no uncertainty regarding the order in which tasks are to be processed.

Given that the makespan is a TFN and neither the maximum nor its approximation define a total ordering in the set of TFNs, it is necessary to reformulate what is understood by "minimising the makespan". 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. Durations are kept fuzzy, which contributes to the resulting schedule's robustness [14].

Another concept that needs some reformulation in the fuzzy case is that of criticality, an issue far from being trivial. In [10], an arc $(x, y)$ in the solution graph is taken to be critical if and only if the completion time of $x$ and the starting time of $y$ coincide in any of their components. In [12], it is argued that this definition yields some counterintuitive examples and a more restrictive notion is proposed. From the solution graph $G(\sigma)$, three *parallel solution graphs* $G^i(\sigma)$, $i = 1, 2, 3$, are derived with identical structure to $G(\sigma)$, but where the cost of arc $(x, y) \in A \cup R(\sigma)$ in $G^i(\sigma)$ is $p_x^i$, the $i$-th component of $p_x$. Each parallel solution graph $G^i(\sigma)$ is a disjunctive graph with crisp arc weights, so in each of them a critical path is the longest path from node *start* to node *end*. For the fuzzy solution graph $G(\sigma)$, a path will be considered to be *critical* if and only if it is critical in some $G^i(\sigma)$. Nodes and arcs in a critical path are termed critical and a critical path is naturally decomposed into critical blocks, these being maximal subsequences of tasks requiring the same machine.

## 3 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 *simple hill-climbing*, evaluation stops as soon as a first improving neighbour is found, which will then replace the original solution. Local search starts again from that improving neighbour, so the procedure finishes when no neighbour satisfies the acceptation criterion.

### 3.1 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 [15], extended to the fuzzy case in [12] using the given definition of criticality:

**Definition 1.** *Given a task processing order $\pi$ and an arc $v = (x, y) \in R(\pi)$, let $\pi_{(v)}$ denote the processing order obtained from $\pi$ after an exchange in the order of tasks in arc $v$. Then, the* neighbourhood structure *obtained from $\pi$ is given by $H(\pi) = \{\pi_{(v)} : v \in R(\pi)$ is critical$\}$.*

It can be proved that if $\pi$ is a feasible task processing order, then all elements in its neighbourhood $H(\pi)$ are feasible. This feasibility property limits the 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.

The proposal to extend the neighbourhood structure proposed in Van Laarhoven et al. [15] to the fuzzy case originally stems from [10], but using the earlier definition of critical arc for fuzzy durations. Let this neighbourhood be denoted by $H'$. The set of critical arcs, according to the definition based on parallel graphs, is a strict subset of the critical arcs according to [10]. Thus the neighbourhood $H$ is strictly included in $H'$. It can be shown that those neighbours from $H' - H$ can never improve the makespan. Indeed, this is a consequence of the fact that, for a feasible processing order $\pi$, if $\sigma = \pi_{(v)}$ where $v$ is not critical (in the sense of $H$) in $G(\pi)$, then $\forall i, \quad C^i_{max}(\pi) \leq C^i_{max}(\sigma)$ and hence $E[C_{max}(\pi)] \leq E[C_{max}(\sigma)]$.

Neighbourhood structures have been used in different metaheuristics to solve the fuzzy job shop. In [10], neighbourhood $H'$ is used in a simulated annealing algorithm. The same neighbourhood is used in [11] for a memetic algorithm (MA) hybridising a local search procedure (LS) with a genetic algorithm (GA) using permutations with repetition as chromosomes. Combining LS and GAs was an approach already successful for fuzzy flow shop [16]; the results in [11] show a clear synergy between the GA and the LS, with the hybrid method also comparing favourably with the simulated annealing from [10] and a GA from [7]. The same memetic algorithm is used in [12], but here the local search procedure uses the neighbourhood based on parallel graphs, $H$. The experimental results reported in [12] show that this new memetic algorithm performs better than state-of-the-art algorithms. Despite 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

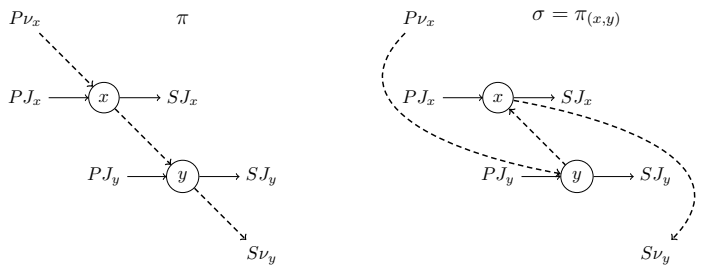


**Fig. 1.** Situation before ($\pi$) and after ($\sigma$) the reversal of a critical arc $(x, y)$.

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 improve the local search efficiency in two 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 actually avoid the evaluation of certain neighbours by means of makespan lower bounds.

### 3.2 Scheduling 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 $G^i(\pi)$.

As explained above, for a task processing order $\pi$ 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 1. The schedule after the move may be calculated as for any solution, using forward propagation from 0 onwards in the graph $G(\sigma)$. This has been the method used in [12]. Alternatively, the evaluation of neighbouring solutions may be done very quickly (in time $O(N)$), as shown in [17] for the classical JSP.

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 $\pi$, $r'_a = r_a$ and for every task $b$ posterior to $y$ in $\pi$, $q'_b = q_b$. The heads and tails for $x$ and $y$ after the move (see Figure 1) are given by the following:

$$r'_y = \max\{r_{PJ_y} + p_{PJ_y}, r_{P\nu_x} + p_{P\nu_x}\}, \qquad r'_x = \max\{r_{PJ_x} + p_{PJ_x}, r'_y + p_y\}$$
$$q'_x = \max\{q_{SJ_x} + p_{SJ_x}, q_{S\nu_y} + p_{S\nu_y}\} \qquad q'_y = \max\{q_{SJ_y} + p_{SJ_y}, q'_x + p_x\}$$

Therefore, we need only re-calculate the heads of tasks from $x$ onwards and the tails of tasks previous to $y$ in the graph $G(\sigma)$. We propose to incorporate this quick way of evaluating neighbours in $H$ to the local search algorithm, an idea which, albeit simple, may prove a considerable reduction in computational load.

### 3.3 Makespan Lower Bound

At each iteration of the local search, only those neighbours with improving makespan are of interest. Hence, another way of reducing computational cost is to foresee, by means of a makespan lower bound, that certain neighbours are certainly not improving, thus avoiding unnecessary calculations. A well-known and inexpensive lower bound for the makespan in the crisp case was proposed by Taillard in [17]. In the following, we generalise this bound to the fuzzy case.

For a processing order $\pi$ and tasks $x$ and $y$, let $P_\pi(x \vee y)$ denote the set of all paths in the solution graph $G(\pi)$ containing $x$ or $y$, $P_\pi(x \wedge y)$ denote the set of all paths in $G(\pi)$ containing both $x$ and $y$ and let $P_\pi(\neg x)$ denote de set of all paths in $G(\pi)$ 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 1.** *Let $\sigma = \pi_{(v)}$, where $v = (x, y)$ is an arc in $G(\pi)$. Then, the makespan for the new solution is given by:*

$$C_{max}(\sigma) = \max\{D[P_\sigma(x \vee y)], D[P_\pi(\neg x)]\} \tag{2}$$

*Proof.* For every $i = 1, 2, 3$, $C^i_{max}(\sigma) = \max\{D^i[P_\sigma(x \vee y)], D^i[P_\sigma(\neg x \wedge \neg y)]\}$. 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 $C^i_{max}(\sigma) = \max\{D^i[P_\sigma(x \vee y)], D^i[P_\pi(\neg x \wedge \neg y)]\}$.

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 $(y, S\nu_y)$ 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 write:

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

□

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

**Corollary 1.** *Let $c(\pi, x, y)$ and $LB_F$ be two TFNs defined by:*

$$c(\pi, x, y)^i = \begin{cases} 0 & \text{if } (x, y) \text{ is critical in } G^i(\pi), \\ C^i_{max}(\pi) & \text{otherwise.} \end{cases}, i = 1, 2, 3 \quad (3)$$

$$LB_F = \max\{c(\pi, x, y), r'_x + p_x + q'_x, r'_y + p_y + q'_y\} \quad (4)$$

*Then, $LB^i_F \leq C^i_{max}(\sigma)$ for all $i$ and hence $LB_F \leq_E C_{max}(\sigma)$, thus providing a* lower bound *for the makespan $C_{max}(\sigma)$.*

*Proof.* Using heads and tails, by the propoposition above, $C^i_{max}(\sigma) \leq D^i[P_\sigma(x \vee y)] = (\max\{r'_x + p_x + q'_x, r'_y + p_y + q'_y\})^i$. Suppose now that the reversed arc $(x, y)$ is critical in $G^i(\pi)$ but is not critical in $G^j(\sigma)$. If none of the arcs that change after a move is contained in a critical path of $G^j(\pi)$, then that path, with length $C^j_{max}(\pi)$, remains unchanged in $G^j(\sigma)$. If it contains arc $(P\nu_x(\pi), x)$ then the resulting path in $G(\sigma)$ is longer. Analogously, if the critical arc in $G^j(\pi)$ contains $(y, S\nu_y(\pi))$, then the resulting path in $G(\sigma)$ is longer. Therefore, in every component $j$ where $(x, y)$ is not critical there is always a path in $G^j(\sigma)$ with length greater or equal than $C^j_{max}(\pi)$. □

The lower bound $LB_F$ may be used in the acceptance criterion of the LS to decide whether a neighbour $\sigma$ is chosen or not without any loss in makespan quality: if $E[C_{max}(\pi)] \leq E[LB_F]$, we may discard $\sigma$ as a non-improving neighbour without evaluating it, since $E[LB_F] \leq E[C_{max}(\sigma)]$.

## 4 Experimental Results

We now consider 12 benchmark problems for job shop: the well-known FT10 and FT20, and the set of 10 problems identified in [18] as hard to solve for classical JSP: La21, La24, La25, La27, La29, La38, La40, ABZ7, ABZ8, and ABZ9. Ten fuzzy versions of each benchmark are generated following [10] and [12], 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. In total, we consider 120 fuzzy job shop instances, 10 for each of the 12 crisp benchmark problems.

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 [12], denoted GVPV08 in the following, which improved previous approaches from the literature in terms of makespan optimisation. GVPV08 combines a genetic algorithm with a simple hill-climbing local search procedure based on the neighbourhood structure $H$, with high computational load for the local search. We shall use GVPV08 as a baseline algorithm and introduce the different improvements proposed herein in the local search module, to evaluate their contribution towards improving efficiency.

**Table 1.** Rescheduling algorithm: CPU time of MA vs. GVPV08

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

The first experiment consists in incorporating the new rescheduling algorithm to GVPV08, obtaining a new hybrid algorithm denoted MA, which is run with the same parameters as GVPV08 (population size 100 and 200 generations). 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 is identical in terms of makespan. Table 1 shows the average across each family of ten fuzzy instances of the total CPU time (in seconds) taken by 30 runs of GVPV08 and MA. It shows a clear reduction in time for MA, ranging from a minimum (15.83%–17.64%) for the square problems of size $15 \times 15$ and a maximum (59.73%) obtained for FT20 of size $20 \times 5$; for identical number of jobs, the greater the number of resources, the greater the reduction. In any case, the rescheduling 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 increase in efficiency with the new rescheduling algorithm, we proceed to analyse the contribution of the lower bound. We consider a variation of the most efficient algorithm MA, denoted MA(LBF), which incorporates the lower bound $LB_F$ to avoid unnecessary evaluations. 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 comparisons 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 average across the ten fuzzy instances of each family of problems of the total number of evaluated neighbours and CPU time (in seconds) for 30 runs of both MA(LBF) and MA (notice that the latter evaluates the same neighbours than GVPV08). The average reduction for MA(LBF) w.r.t. MA and GVPV8 is 87.09%, with a standard deviation of 6.25%; the minimum (84.81%, 84.98%) is obtained for square problems of size $15 \times 15$ and the maxi-

**Table 2.** Comparison of number of evaluated neighbours and CPU time (in seconds).

| Problem | No. Neighbours | | | CPU time | | |
|---|---|---|---|---|---|---|
| | **MA** | **MA(LBF)** | **red.%** | **MA** | **MA(LBF)** | **red.%** |
| FT10 | 2.83E+07 | 3.49E+06 | 87.67 | 588.2 | 183.1 | 68.87 |
| FT20 | 6.78E+07 | 5.11E+06 | 92.47 | 682.1 | 257.2 | 62.29 |
| La21 | 4.46E+07 | 5.61E+06 | 87.41 | 1072.8 | 331.4 | 69.11 |
| La24 | 3.87E+07 | 5.37E+06 | 86.13 | 950.1 | 315.1 | 66.84 |
| La25 | 4.28E+07 | 5.68E+06 | 86.73 | 993.7 | 328.9 | 66.90 |
| La27 | 8.37E+07 | 9.55E+06 | 88.59 | 2242.8 | 593.2 | 73.55 |
| La29 | 7.85E+07 | 8.74E+06 | 88.87 | 2071.7 | 562.5 | 72.85 |
| La38 | 5.16E+07 | 7.74E+06 | 84.98 | 2556.7 | 570.9 | 77.67 |
| La40 | 5.42E+07 | 8.22E+06 | 84.81 | 2652.2 | 595.7 | 77.54 |
| ABZ7 | 9.74E+07 | 1.36E+07 | 85.99 | 5294.7 | 1053.9 | 80.10 |
| ABZ8 | 1.08E+08 | 1.53E+07 | 85.91 | 5780.9 | 1138.5 | 80.31 |
| ABZ9 | 9.65E+07 | 1.40E+07 | 85.50 | 5652.1 | 1081.9 | 80.86 |

mum (92.47%) is obtained for FT20 of size $20 \times 5$. As above, for identical number of jobs, the greater the number of resources, the greater the reduction. It is remarkable that the number of evaluated neighbours using the lower bound is practically reduced in an order of magnitude. CPU time is also considerably reduced, 73.07% in average, although this rate is not linear in neighbour reduction. The reason is the extra cost in MA(LBF) of marking those components where the arc is critical. For small-size problems in the first rows, the cost of labelling arcs means a CPU time reduction which is in average 15% smaller than the reduction in number of neighbours (with the exception of FT20, where the labelling proves most expensive). As the problem size increases (and we descend in Table 2) the computational cost becomes less significant, with a loss in CPU time reduction w.r.t. the neighbours reduction of less than 7.5% in average. This suggests that the advantage of using the lower bound is greater as the problem size and difficulty increase. Notice that the labelling process could also be used in order to incorporate neighbour-ordering heuristics to the local search.

## 5 Conclusions

We have considered a job shop problem with uncertain durations modelled as TFNs and have proposed two changes in a local search method from the literature to improve its efficiency. We have first proposed a different algorithm to reschedule neighbours and then a lower bound for neighbours' makespan which allows to prune the local search and avoids unnecessary calculations of makespan. The results show that the proposals greatly increase the efficiency of the local search, with considerable reductions in the number of evaluated neighbours and CPU times, without affecting the makespan of the final solution. This allows for future extensions of the criticality model and therein-based search methods to more general and expressive fuzzy representations.

**Acknowledgements** This work supported by MEC-FEDER Grants TIN2007-67466-C02-01 and MTM2007-62799.

## References

1. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems. Third edn. Springer (2008)
2. Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research **165** (2005) 289–306
3. Dubois, D., Fargier, H., Fortemps, P.: Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. European Journal of Operational Research **147** (2003) 231–252
4. Słowiński, R., Hapke, M., eds.: Scheduling Under Fuzziness. Vol. 37 of Studies in Fuzziness and Soft Computing. Physica-Verlag (2000)
5. Brucker, P., Knust, S.: Complex Scheduling. Springer (2006)
6. Tavakkoli-Moghaddam, R., Safei, N., Kah, M.: 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** (2008) 431–442
7. 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** (2000) 393–407
8. Petrovic, S., Fayad, S., Petrovic, D.: Sensitivity analysis of a fuzzy multiobjective scheduling problem. International Journal of Production Research **46**(12) (2007) 3327–3344
9. 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) (2008) 655–666
10. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions of Fuzzy Systems **7** (1997) 557–569
11. González Rodríguez, I., Vela, C.R., Puente, J.: A memetic approach to fuzzy job shop based on expectation model. In: Proc. of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007, London, IEEE (2007) 692–697
12. González Rodríguez, I., Vela, C.R., Puente, J., Varela, R.: A new local search for the job shop problem with uncertain durations. In: Proc. of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008), Sidney, AAAI Press (2008) 124–131
13. Liu, B., Liu, Y.K.: Expected value of fuzzy variable and fuzzy expected value models. IEEE Transactions on Fuzzy Systems **10** (2002) 445–450
14. González Rodríguez, I., Puente, J., Varela, R., Vela, C.R.: A study of schedule robustness for job shop with uncertainty. In: Proc. of IBERAMIA 2008. Vol. 5290 of LNAI, Germany, Springer (2008) 31–41
15. Van Laarhoven, P., Aarts, E., Lenstra, K.: Job shop scheduling by simulated annealing. Operations Research **40** (1992) 113–125
16. Ishibuchi, H., Murata, T.: A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews **67**(3) (1998) 392–403
17. Taillard, E.D.: Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing **6**(2) (1994) 108–117
18. Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA Journal of Computing **3** (1991) 149–156