A Genetic Algorithm for the Open Shop Problem with Uncertain Durations

Juan José Palacios[1], Jorge Puente[1], Camino R. Vela[1], and Inés González-Rodríguez[2]

[1] A.I. Centre and Department of Computer Science, University of Oviedo, Spain
{UO165228,puente,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 a variation of the open shop problem where task durations are allowed to be uncertain and where uncertainty is modelled using fuzzy numbers. We propose a genetic approach to minimise the expected makespan: we consider different possibilities for the genetic operators and analyse their performance, in order to obtain a competitive configuration. Finally, the performance of the proposed genetic algorithm is tested on several benchmark problems, modified so as to have fuzzy durations, compared with a greedy heuristic from the literature.

1 Introduction

The open shop scheduling problem is often regarded as a variation of the job shop scheduling problem, and traditionally it has received considerably less attention by researchers. However, its significantly larger solution space and the scarcity of specific methods to solve it make it an important problem in itself, with an increasing presence in the recent literature [1],[2],[3]. It is also a problem with clear applications. 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. To enhance the range of applications of scheduling, part of the research is devoted to model the uncertainty and vagueness pervading real-world situations. The approaches are diverse [4] and, among these, fuzzy sets have been used in a wide variety of approaches, ranging from representing incomplete or vague states of information to using fuzzy priority rules with linguistic qualifiers or preference modelling [5]. Incorporating uncertainty to scheduling usually requires a significant reformulation of the problem and solving methods, in order that the problem can be precisely stated and solved efficiently and effectively. Some attempts have been made to extend heuristic methods to shop scheduling problems where durations are modelled via fuzzy intervals, most commonly and successfully for the flow shop problem (for instance, in [6] and [7]) and also for the job shop [8], [9], [10] and [11].

J. Mira et al. (Eds.): IWINAC 2009, Part I, LNCS 5601, pp. 255–264, 2009.

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 mn operations, $\{O_k, 1 \leq k \leq mn\}$. 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, the objective will be minimising the makespan C_{max}, that is, the time lag from the start of the first operation until the end of the last one, a problem often denoted $O||C_{max}$ in the literature.

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. 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 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 $M \vee N \approx (m^1 \vee n^1, m^2 \vee n^2, m^3 \vee n^3)$, 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 [5]. 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 $E[M] \leq E[N]$.

Require: an instance of $FuzO||E[C_{max}]$, P
Ensure: a schedule for P
1. generate and evaluate the initial population;
while No termination criterion is satisfied **do**
2. select chromosomes from the current population;
3. apply recombination operators to the chromosomes selected at step 2.;
4. evaluate the chromosomes generated at step 3;
5. apply replacement using chromosomes from step 2. and step 3.;
return the schedule from the best chromosome evaluated so far;

Algorithm 1. Genetic Algorithm

2.2 Fuzzy Open Shop Scheduling

If processing times of operations are allowed to be imprecise and such imprecision or 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. Each TFN can be seen as a possibility distributions on the values that the time may take. Notice however that there is no uncertainty regarding the task processing ordering given by the schedule.

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 [9] and use the total ordering provided by the expected value and consider that the objective is to minimise the expected makespan $E[C_{max}]$. The resulting problem may be denoted $FuzO||E[C_{max}]$.

3 Genetic Algorithms for the FOSP

The open shop scheduling problem is NP-complete for a number of machines $m \geq 3$ [12].This motivates the use of metaheuristic techniques to solve the general m-machine problem. For instance, [1] proposes two heuristic methods to obtain a list of operation priorities later used in a basic algorithm of list scheduling. Also, local search and genetic algorithms are used, by themselves or combined with each other and with dispatching rules: [14] presents an iterative improvement algorithm with a heuristic dispatching rule to generate the initial solutions; [15] proposes a tabu search algorithm, [2] introduces a genetic algorithm hybridised with local search, and a genetic algorithm using heuristic seeding is proposed in [16]. In [17], a local search with constraint propagation and conflict-based heuristics framework is applied to OSP, and [3] proposes a solution based on particle swarm optimisation. However, to our knowledge, none of these metaheuristic techniques have been adapted to the case where durations are fuzzy numbers.

From all heuristic strategies, genetic algorithms (GAs) have proved effective techniques to solve scheduling problems, specially when hybridised with other strategies, such as local search. The structure of a standard genetic algorithm is described in Algorithm 1. First, the initial population is generated and evaluated. Then the genetic algorithm iterates for a number of steps or generations and in each iteration, a new population is built from the previous one by applying the genetic operators such as selection, mutation, crossover, etc,. Clearly, the choice of chromosome codification and genetic operators is decisive in order to obtain a competitive GA.

3.1 Chromosome Codification and Evaluation

Following [2], we use operation-based chromosomes. This representation encodes a schedule as an ordered sequence of operations, with one gene per operation. Operations are listed in the order in which they are scheduled, so a chromosome is just a permutation of numbers from 1 to nm, where number i corresponds to operation O_i. Such a permutation expresses partial orders among operations in each job and each machine.

Decodification of a chromosome may be done in different ways. Here, we consider two approaches. The first one is to schedule every operation so its starting time is the maximum of the completion times of the previous operation in its machine and the previos operation in its job, according to the ordering provided by the chromosome. This strategy produces *semi-active* schedules, meaning that for any operation to start earlier, the relative ordering of at least two operations must be swapped.

The second approach consists in using an insertion strategy, scheduling operations directly from the order expressed in the chromosome: the starting time of each operation is obtained as the earliest possible time given the operations which are already scheduled (previous in the chromosome) both on the same machine and on the same job. It is easy to check that this strategy yields active schedules, where a schedule is *active* if one operation 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 [12].

To obtain the initial population, chromosomes are obtained as random permutations of (1 2 3 4 5 6 7 8 9). Notice that, given the codification schema, this initialisation method, albeit simple, always produces feasible individuals.

3.2 Genetic Operators

Selection. In order to select chromosomes from one population to be combined and generate the next population, we propose to group the chromosomes in pairs at random, so as to give every individual the opportunity to contribute to the next generation.

Crossover. Once we have a pair of chromosomes, we consider several crossover operators proposed in the literature (c.f. [18]):

First, we consider the *Partially-Mapped Crossover* or PMX, which builds an offspring by choosing a subsequence from one parent and preserving the order and position of as many operations as possible from the other parent. A subsequence is selected by choosing two random cut points, which serve as boundaries for swapping operations, for instance, we may have as parents $p_1 = (1\,2\,3|4\,5\,6\,7|8\,9)$, $p_2 = (4\,5\,2\,|1\,8\,7\,6|\,9\,3)$ with two cut points marked by |. The segment between cut points in the first parent is copied in the same positions onto the first offspring: $o_1 = (x\,x\,x\,|4\,5\,6\,7|\,x\,x)$ and similarly with the segment from the second parent onto the second offspring $o_2 = (x\,x\,x\,|1\,8\,7\,6|\,x\,x)$ (the symbol x may be interpreted as 'unknown at present'). The cut also defines a mapping between the operations in the same position in both parents: $1 \leftrightarrow 4$, $8 \leftrightarrow 5$, $7 \leftrightarrow 6$, $6 \leftrightarrow 7$. Next, we fill further operations in the first (resp. second) offspring from the second (resp. first) parent for which there is no conflict: $o_1 = (x\,x\,2|4\,5\,6\,7|9\,3)$ and $o_2 = (x\;2\;3\;|1\;8\;7\;6|\;x\;9)$. Finally, we resolve the conflicts by replacing the operation from the second (resp. first) parent which already appears between the cut points with the corresponding operation from the first (resp. second) parent, according to the obtained mapping: $o_1 = (1\,8\,2\,4\,5\,6\,7\,9\,3)$ and $o_2 = (4\,2\,3|1\,8\,7\,6|5\,9)$.

A second operator is *Linear-Order Crossover* or LOX in short, a modification of the well-known order crossover so as not to treat chromosomes as circular. This operator selects a subsequence of operations at random and copies this subsequence from the first (resp. second) parent onto the first (resp. second) offspring. It then completes each offspring by placing operations from left to right in the same order as they appear in the other parent. For instance, for the same parents as above, $p_1 = (1\,2\,3\,|4\,5\,6\,7|\,8\,9)$, $p_2 = (4\,5\,2\,|1\,8\,7\,6|\,9\,3)$ and the same selected subsequence, the offsprings would be $o_1 = (2\,1\,8\,4\,5\,6\,7\,9\,3)$ and $o_2 = (2\,3\,4\,|1\,8\,7\,6|\,5\,9)$. LOX is designed so as to preserve as much as possible both the relative positions between genes and the absolute positions relative to the extreme operations in the parents, which correspond to the high and low priority operations.

A modification of LOX is the PBX or *Position-Based Crossover* where, instead of selecting one subsequence of operations to be copied, several operations are selected at random for that purpose.

Mutation. Mutation is applied to a chromosome by randomly modifying its features, thus helping to preserve population diversity and providing a mechanism to escape local minima. Among the several mutation operators proposed for permutation-based codification [18], we consider insertion and swap mutations. The *insertion* operator selects an operation at random and then inserts it into another random position. The *swap* or *gene swapping* operator selects two positions at random and then swaps the operations in these positions.

Replacement. From each pair of parents we obtain offsprings by applying crossover and mutation. This forms a set of four solutions from which two will be accepted as members of the next generation using tournament: the two old chromosomes and the new ones are put together and the best two are selected to replace the parents in the next generation. Here, we have two possibilities, namely, accept two

chromosomes with the same fitness (same expected makespan), or force the two accepted chromosomes to have different fitness. Notice that with this replacement scheme, inferior solutions are eliminated only through newborn superior solutions. Therefore, both the best and worst makespans of the solutions in the population at each generation are non-increasing and there is an implicit elitism.

As we can see, by selecting one operator or another from those explained herein, we obtain a wide range of configurations for a GA. Our goal is to choose the best possible configuration based on thorough experimentation.

4 Experimental Results

For the experimental study, we follow [8] and generate a set of fuzzy problem instances from well-known benchmark problems. In particular, we consider a subset of the problems proposed by Taillard [19], consisting of four families of problems of sizes 4×4, 5×5, 7×7 and 10×10 and where each family contains 10 problem instances. From each of these crisp problem instances, we generate 10 fuzzy instances, so we have 400 fuzzy problem instances in total. To generate a fuzzy instance from a crisp one, we transform each crisp processing time x into a symmetric fuzzy processing time $p(x)$, so its modal value is $p^2 = x$ 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. By doing this, the optimal solution to the crisp problem provides a lower bound for the fuzzified version [8]. The obtained benchmarks for the fuzzy open shop are available at `http://www.aic.uniovi.es/tc/spanish/repository.htm`

4.1 Configuration and Parameter Setting

A first set of experiments is conducted to choose the GA's configuration, in an incremental approach, where the first step is to fix the population size and the number of generations, to ensure convergence. We outline the experimentation process and give a summary of results, but we omit a more detailed exposition, due to lack of space and the large number (400) of problem instances used.

The **convergence** study is performed using the following base configuration: fitness, active scheduling; crossover, LOX with probability $p_x = 0.70$; mutation, insertion with probability $p_m = 0.05$; selection, random pairs; replacement, tournament with repetition. For this base configuration, the GA has been run varying the number of maximum iterations depending on the problem sizes: 1000 (4×4), 5000 (5×5), 10000 (7×7 and 10×10). Tests have been repeated with two different population sizes: 100 and 200 and for each possibility we have considered the average performance across ten runs. The quality of a solution is assessed using the following makespan relative error w.r.t. a lower bound: $(E[C_{max}] - LB)/LB$ with $LB = \max(\max_j\{\sum_{i=1}^{n} p_{ij}\}, \max_i\{\sum_{j=1}^{m} p_{ij}\})$, where p_{ij} is the crisp processing time of the operation belonging to job J_i which has to be processed on machine M_j. Here, we have considered the relative makespan error of the best individual in each generation and have followed the error evolution along

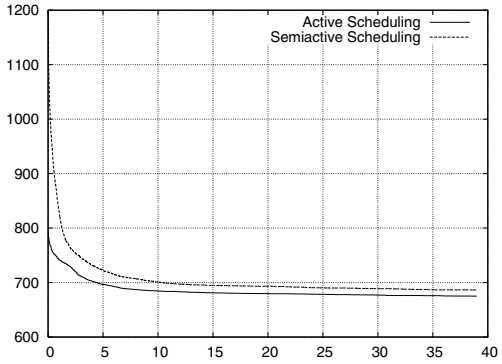

Fig. 1. Convergence of base configuration of GA for problem tai10_1_05

the generations for both population sizes, establishing as convergence point the point where the difference between the generation's error and the minimum error obtained with the maximum number of generations is less than 1%. With this criterion, we set the population size as 100 for all problems, with the number of generations depends on the size as follows: $4 \times 4 : 40$, $5 \times 5 : 130$, $7 \times 7 : 1600$, $10 \times 10 : 3000$.

Having fixed the base configuration for population size and number of generations, we decide on the type of **chromosome evaluation**: active or semi-active scheduling. To compare both approaches, we extend the running time for the GA with semi-active scheduling so it is approximately the same as that of the GA with active scheduling. The conclusion is that, under equivalent conditions, the relative makespan error for semi-active scheduling is between 3.074% and 4.438% worse than for active scheduling. We therefore opt for the latter. Figure 1 illustrates the convergence of the GA with both active and semi-active scheduling for problem `tai10_1_05`, that is, the fifth fuzzy instance obtained from the first problem proposed by Taillard of size 10×10. The x axis represents the time taken in seconds and the y axis represents the expected makespan.

To perform **replacement**, we only need choose whether to filter those chromosomes with the same expected makespan or not. Again, with the base configuration and using active scheduling, the experimental results show that the cost of filtering individuals with identical makespan is insignificant w.r.t. the total running time of the algorithm and it always generates slightly better results.

Using the GA with active scheduling and replacement with no repetition, we analyse the performance of the three different **crossover** operators using as crossover probability values $0.7, 0.8, 0.9$. Again, the conclusions are drawn based on the relative makespan error w.r.t. LB and also on computational cost. The results show that there is no significant difference between PMX and LOX, both for the makespan error and running times, being 0.7 the best value for the crossover probability. PMX is always better, although the improvement w.r.t. LOX is less than 0.4%. PMX is also preferred to PBX, since the latter takes approximately 28.38% more running time and the makespan does not always improve. For the

Table 1. Average relative makespan error (in %) for all families of problems

Problem	GA		Random Pop.	DS/LRPT
	B	A		
4×4	3.002	4.020	4.774	12.473
5×5	3.432	6.263	9.840	16.100
7×7	1.480	4.123	11.698	10.938
10×10	1.382	3.624	13.723	7.069

ten sets of fuzzy problems of size 10×10, PBX with $p_x = 0.9$ (the best of the three values) only improves the makespan obtained with PMX with $p_x = 0.7$ in three of the sets and this improvement is less than 0.9%. We conclude that the crossover for the final configuration will be PMX with $p_x = 0.7$ (although LOX is very similar).

As above, we try the two **mutation** operators and adjust the mutation probability choosing from $0.05, 0.10, 0.15$. Here the best results for insertion are obtained with $p_m = 0.05$ and the best results for swap are obtained with $p_m = 0.10$; overall, insertion behaves better than swap.

After this experimental analysis, the chosen configuration for the GA will be: fitness, active scheduling; crossover, PMX with probability $p_x = 0.70$; mutation, insertion with probability $p_m = 0.05$; selection, random pairs; replacement, tournament without repetition.

4.2 Performance

To evaluate the performance of the proposed GA, we run the GA 30 times for each problem instance and consider the average value across these 30 runs of the relative makespan error for the best individual (B) and the average (A) in the last generation. Table 1 shows these values averaged across the 100 problem instances of each size, as a summary of the GA's behaviour, compared to the makespan relative error obtained by a dispatching rule DS/LRPT proposed in [14], adapted to fuzzy durations, and also compared to relative error of the best individual from a population of random permutations, with as many individuals as the total number of chromosomes evaluated by the GA. No other comparisons are made since, to our knowledge, there are no heuristic methods proposed in the literature for $FuzO||E[C_{max}]$.

Each row of Table 1 summarises the information relative to 100 problem instances: 10 fuzzy versions of each of the 10 crisp instances of one size. More detailed results are presented in Table 2, where each row corresponds to one set of 10 fuzzy versions of a crisp instance of size 10×10. As expected, the GA compares favourably with the random population. Concentrating on the second table, the makespan for the latter is between 9.259% and 10.904% worse (it also takes between 36.238% and 37.959% more running time). The GA also performs better than the priority rule DS/LRPT with, of course, smaller difference in makespan error (between 2.263% and 4.564% compared to the average performance of

Table 2. Average relative makespan error (in %) for sets of problems of size 10×10

Problem	GA		Random Pop.	DS/LRPT
	B	A		
tai10_1	2.743	5.245	16.044	7.508
tai10_2	0.923	2.842	12.904	5.940
tai10_3	2.140	4.591	14.101	7.738
tai10_4	0.568	2.404	11.659	5.884
tai10_5	1.687	3.987	13.559	8.160
tai10_6	0.520	2.608	14.150	6.018
tai10_7	1.088	3.497	12.890	6.920
tai10_8	1.361	3.839	14.063	7.815
tai10_9	1.366	3.541	14.513	8.105
tai10_10	1.422	3.686	13.343	6.607

the GA). Notice as well that the relative errors for the best (B) and average (A) solution do not differ greatly, which means that the GA is quite stable. Also, relative errors are relatively small, showing that the GA either obtains an optimum solution or is quite close, even for the problems of greater size.

5 Conclusions and Future Work

We have considered an open shop problem with uncertain durations modelled as triangular fuzzy numbers, $FuzO||E[C_{max}]$, and have proposed a genetic algorithm to solve this problem. Using a permutation-based codification, we have considered several genetic operators and have conducted a thorough experimentation in order to select operators and set GA parameters to obtain a final competitive configuration. The performance of the GA has been assessed on a set of problems obtained from classical ones in what would constitute a first benchmark for $FuzO||E[C_{max}]$. The GA has obtained good results both in terms of relative makespan error and also in comparison to a priority rule and a population of random solutions. These promising results suggest directions for future work. First, the GA should be tested on more difficult problems, fuzzy versions of other benchmark problems from the literature. Also, the GA provides a solid basis for the development of more powerful hybrid methods, in combination with local search techniques, an already successful approach in classical shop problems [15],[2] and also in fuzzy open shop [7] and fuzzy job shop [9],[20].

References

1. Guéret, C., Prins, C.: Classical and new heuristics for the open-shop problem: A computational evaluation. European Journal of Operational Research 107, 306–314 (1998)
2. Liaw, C.F.: A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research 124, 28–42 (2000)

3. Sha, D.Y., Cheng-Yu, H.: A new particle swarm optimization for the open shop scheduling problem. Computers & Operations Research 35, 3243–3261 (2008)
4. Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165, 289–306 (2005)
5. 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)
6. Celano, G., Costa, A., Fichera, S.: An evolutionary algorithm for pure fuzzy flowshop scheduling problems. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 11, 655–669 (2003)
7. 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), 392–403 (1998)
8. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions of Fuzzy Systems 7, 557–569 (1997)
9. 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, London, pp. 692–697. IEEE, Los Alamitos (2007)
10. 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)
11. 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, 431–442 (2008)
12. Pinedo, M.L.: Scheduling, 3rd edn. Theory, Algorithms, and System. Springer, Heidelberg (2008)
13. Liu, B., Liu, Y.K.: Expected value of fuzzy variable and fuzzy expected value models. IEEE Transactions on Fuzzy Systems 10, 445–450 (2002)
14. Liaw, C.F.: An iterative improvement approach for the nonpreemptive open shop scheduling problem. European Journal of Operational Research 111, 509–517 (1998)
15. Liaw, C.F.: A tabu search algorithm for the open shop scheduling problem. Computers and Operations Research 26, 109–126 (1999)
16. Puente, J., Diez, H., Varela, R., Vela, C., Hidalgo, L.: Heuristic Rules and Genetic Algorithms for Open Shop Scheduling Problem. In: Conejo, R., Urretavizcaya, M., Pérez-de-la-Cruz, J.-L. (eds.) CAEPIA/TTIA 2003. LNCS(LNAI), vol. 3040, pp. 394–403. Springer, Heidelberg (2004)
17. Jussien, N., Lhomme, O.: Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence 139, 21–45 (2002)
18. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd revised and extended edn. Springer, Heidelberg (1996)
19. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
20. 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, pp. 124–131. AAAI Press, Menlo Park (2008)