A Tabu Search-based Memetic Algorithm for the Multi-objective Flexible Job Shop Scheduling Problem

Marios Kefalas
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
m.kefalas@liacs.leidenuniv.nl

Steffen Limmer
Honda Research Institute Europe GmbH
Offenbach, Germany
Steffen.Limmer@honda-ri.de

Asteris Apostolidis
KLM Royal Dutch Airlines
Amstelveen, The Netherlands
Asteris.Apostolidis@klm.com

Markus Olhofer
Honda Research Institute Europe GmbH
Offenbach, Germany
markus.olhofer@honda-ri.de

Michael Emmerich
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
m.t.m.emmerich@liacs.leidenuniv.nl

Thomas Bäck
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

ABSTRACT

In this paper we propose a tabu search-based memetic algorithm (TSM) for the multi-objective flexible job shop scheduling problem (FJSSP), with the objectives to minimize the makespan, the total workload and the critical workload. The problem is addressed in a Pareto manner, which targets a set of Pareto optimal solutions. The novelty of our method lies in the use of tabu search (TS) as the local search method as well as a mutation operator and the use of the hypervolume indicator to avoid stagnation by increasing the flow of individuals in the local search. To the best of our knowledge, the use of TS in the context of multi-objective FJSSP has not been reported so far. We apply our algorithm on well known test instances and compare our results to state-of-the art algorithms. The results show that our approach yields competitive solutions in 6 of the 10 instances against two of their algorithms proving that the use of TS as a local search method can provide competitive results.

CCS CONCEPTS

• **Applied computing → Multi-criterion optimization and decision-making**;

KEYWORDS

scheduling, tabu search, memetic, flexible job shop, genetic algorithms, multi-objective optimization

ACM Reference Format:
Marios Kefalas, Steffen Limmer, Asteris Apostolidis, Markus Olhofer, Michael Emmerich, and Thomas Bäck. 2019. A Tabu Search-based Memetic Algorithm for the Multi-objective Flexible Job Shop Scheduling Problem. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3319619.3326817

1 INTRODUCTION

One of the most important industrial activities, especially in planning and managing of manufacturing processes, is the scheduling of operations. Already in the 1950s, this led to the formulation of one of the classical operations research problems [6], the job shop scheduling problem (JSSP). The JSSP, can be described a set of jobs which must be processed on a set of machines uninterruptedly, where each job is a sequence of consecutive operations. Each operation requires exactly one machine, and machines are continuously available and can process one operation in a given duration. A solution to this problem is a schedule. A typical performance indicator for the solution is the maximum completion time of all operations, also called the makespan. The usual objective, to find a schedule with minimum length (minimum makespan), was proven to be NP-hard [12] and belongs to the most intractable instances of NP-hard problems [18]. To deal with the combinatorial complexity, meta-heuristic techniques, such as evolutionary algorithms (EA) [27], particle swarm optimization (PSO) [17] and tabu search (TS) [19] can be used.

Instead of just the makespan though, several other performance measures can be used as well, such as the maximum machine workload or the total machine workload [25]. In this case, the problem automatically becomes a multi-objective optimization (MOO) problem, in which a variety of incomparable solutions exist. The set of such solutions, which cannot be improved with respect to one objective without making another objective worse, is called the Pareto set [10].

In this work, we focus on the flexible job shop scheduling problem (FJSSP), which is an extension of the job shop scheduling problem, and present a new approach using tabu search (TS) as the local search method for a multi-objective evolutionary algorithm (MOEA). We apply our algorithm to the famous Brandimarte datasets [4] and we compare ourselves to the algorithms of Yuan et al. [33]. We should note here that we do not intend to make a reference set as done by Yuan et al., as this would take enormous effort, but instead enrich this field by doing additional research on ways of determining new or even better solutions to this problem.

The main novelty of our work lies in the following two key aspects:

- Tabu search (TS) is used in two ways: As a local search method, as well as in the mutation operator.
- Stagnation avoidance based on the hypervolume indicator [9].

It is also our intention to focus on the use of memetic multi-objective approaches for the solution of this problem, as our work shows that combining local search with global search can have a significant positive impact for heuristic solvers for the FJSSP.

This paper is organized as follows: In Section 2, we formulate the FJSSP and discuss the state-of-the-art in Section 3. In Section 4, we describe TS and in Section 5 our algorithmic approach. Finally, in Section 6 we present our experimental setup and results, and in Section 7 we give concluding remarks and introduce directions for future research.

2 PROBLEM FORMULATION

An FJSSP instance can be described as a set $\mathcal{N}$ of N jobs that need to be processed on a set $\mathcal{M}$ of M machines. Each job $i \in \{1, \dots, N\}$ consists of a tuple of N_i operations O_{ij}, with $j \in \{1, \dots, N_i\}$, which have a predetermined sequence. This means that for job i to be completed, its N_i operations must be processed in their given order. This is called a precedence constraint. Furthermore, each operation O_{ij} has a predetermined set of machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$ which can process this operation. The processing time p_{ijk} of the process O_{ij} on machine $k \in \mathcal{M}_{ij}$ is also known a priori. Furthermore, the following assumptions are made:

- All machines are available at time 0.
- All jobs are released at time 0.
- Each machine can process one operation at a time.
- Jobs are independent between each other, i.e., there are no precedence constraints among the operations of different jobs.
- No interruption is allowed once a process has started (no pre-emption of operations is allowed).
- The setup times of machines and transfer times of operations are negligible.

The scheduling problem consists of two subproblems:

(1) The routing subproblem, i.e., assigning each operation O_{ij} to a machine $k \in \mathcal{M}_{ij}$.
(2) The sequencing subproblem that determines a sequence of operations on all machines, to obtain a feasible schedule which satisfies predefined objectives.

The classical JSP requires sequencing of operations on fixed machines, i.e., the machine assignment is pre-determined. Given that in FJSSP we not only deal with the sequencing of operations, but also with the machine assignments to the operations, it is by nature more complex than the classic JSP. The FJSSP is thus also NP-hard since it is an extension of the NP-hard JSP.

Regarding the flexibility of the problem, there are two classifications based on [16]. These are:

(1) Total flexibility, where each operation can be processed by any of the M machines ($\mathcal{M}_{ij} = \mathcal{M}, \forall i \in \{1, .., n\}$ and $j \in \{1, .., N_i\}$.
(2) Partial flexibility where some operations can only be processed on a subset of the available m machines in the shop.

Finally, let C_i be the completion time of job i. W_k is the summation of processing times of operations that are processed on machine k. In this paper the three objectives makespan C_{max}, total workload W_T and maximum or critical workload W_{max} are to be minimized. These are defined as follows:

$$C_{max} = \max\{C_i \mid i \in \{1, .., N\}\}, \tag{1}$$

$$W_T = \sum_{k=1}^{M} W_k, \tag{2}$$

$$W_{max} = \max\{W_k \mid k \in \{1, .., M\}\}. \tag{3}$$

3 RELEVANT LITERATURE

Due to its high relevance, the last three decades have seen extensive development of efficient techniques to solve the flexible job shop scheduling problem (FJSSP) [6]. Between 2010 and 2013, a considerable increase in the number of publications addressing the problem can be observed, with almost 50% of those contributions using multi-objective performance measures [6]. Regarding the latter, the performance measures mostly used are makespan, total workload and critical workload. Moreover, emphasis has been given to the use of hybrid techniques, i.e., techniques that combine one or more heuristics or metaheuristics [6]. The most common form of hybridization is local search [1]. The term memetic algorithm (MA) is often used synonymously for hybrid evolutionary algorithms [21], [20]. Memetic algorithms combine evolutionary algorithms with local search operators and are widely used in combinatorial optimization. In this view, in [7] the authors introduce a multi-objective memetic algorithm (MA) with an embedded variable neighborhood descent procedure, and in [33] the authors propose new memetic algorithms for the multi-objective flexible job shop scheduling problem (MO-FJSSP) with the objectives to minimize the makespan, total workload, and critical workload, by adapting the NSGA-II optimizer [8] through a well-designed chromosome encoding/decoding scheme and genetic operators. They also develop a novel local search method based on critical operations, using a hierarchical strategy to handle multiple objectives, emphasizing on makespan. To the best of our knowledge these two papers are the most recent in the field that deal with multiple objectives using a memetic approach. Most researches with a hybrid/memetic structure usually deal with one objective, most often makespan (i.e., see [30], [32], [5]), or other forms of hybridization such as in [31] and [22].

4 TABU SEARCH

Tabu search (TS) is a metaheuristic, developed by Glover [13], that guides a local heuristic search procedure to explore the solution space beyond local optimality, in mathematical optimization. TS is based on the assumption that problem solving, to qualify as intelligent, must incorporate adaptive memory and responsive exploration [6].

Local search methods have a tendency to become stuck in suboptimal regions (local optima) or on plateaus where many solutions are equally fit. Tabu search overcomes this pitfall of local search by relaxing its basic rule. First, at each step a worse move can be accepted if no improving move is available. In addition, prohibitions (hence the term tabu) are introduced to discourage the search from coming back to previously visited solutions. These prohibitions are facilitated through a memory structure called the tabu list. In its simplest form, a tabu list is a short-term set of the solutions that have been visited in the recent past, i.e., within less than a certain number of iterations which is called the tabu list size $|T|$. In this list one can alternatively store characteristics or attributes of the forbidden moves [14]. In this approach, we used solutions, instead of attributes or moves. Furthermore, these memory structures can be divided into three categories:

(1) Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an expiration point, which usually means $|T|$ iterations. This is the approach used in this paper.
(2) Intermediate-term: Intensification rules which intend to bias the search towards promising areas of the search space.
(3) Long-term: Diversification rules that drive the search into new regions.

An aspiration criterion is introduced in tabu search to determine when the tabu restriction can be overridden, thus removing a tabu classification. A solution is above the current aspiration level if it is better than any solution met before.

The main decisions to be made are:

- The specification of a neighborhood structure.
- The move attributes (if used).
- The tabu list length (or tenure).
- The aspiration criterion (if used).
- The stopping rule.

A typical pseudocode for TS, and the one used in this paper, is presented in Algorithm 1, where we consider the minimization of an objective function f. In detail, lines 2 through 4 initialize a solution x_0 and set it as the best solution found so far and as the current seed/candidate to continue the search from. In line 4, the tabu_list is initialized to an empty list. In line 5 the search begins. Line 6 finds the neighbors of the current candidate and lines 7 to 9 check if the neighborhood is empty. This can be the case, if for example the number of blocks that generate the moves is not larger than 1 [24]. If it is, the search stops and returns the best solution so far. Lines 10 to 12 check if all elements of the neighborhood belong in the tabu list and if they do, the search stops and returns the best solution so far. In line 13 the best neighbor in the generated neighborhood is found. If the best neighbor is in the tabu list (line 14) then we check if it passes the aspiration criterion (lines 15 to 20) and if not we find a new best neighbor, discarding the previous one (lines 22 and 23). If the best neighbor does not pass the check in line 14, then it is added to the tabu list, the new candidate is updated and a check takes place to see whether the new candidate has a better fitness value than the best solution found so far (lines 29 to 33). We continue like this until a termination criterion is met. In our case, no aspiration criterion was needed since we used solutions instead of moves in the tabu list.

Algorithm 1 Tabu search.

```
1:  Input: $x_0, T$
2:  $bestSolution \leftarrow x_0$
3:  $bestCandidate \leftarrow bestSolution$
4:  $tabu\_list \leftarrow []$
5:  while termination criterion not met do
6:      $sNeighborhood \leftarrow getNeighbors(bestCandidate)$
7:      if not $sNeighborhood$ then
8:          Return $bestSolution$
9:      end if
10:     if all $x$ in $sNeighborhood$ is in $tabu\_list$ then
11:         Return $bestSolution$
12:     end if
13:     $bestNeighbor \leftarrow getBestNeighbor(sNeighborhood)$
14:     while $bestNeighbor$ in $tabu\_list$ do
15:         if $f(bestNeighbor) < f(bestSolution)$ then
16:             if $length(tabu\_list) == |T|$ then
17:                 $tabu\_list.pop(0)$
18:             end if
19:             $tabu\_list \leftarrow tabu\_list \cup \{bestNeighbor\}$
20:             $bestSolution \leftarrow bestNeighbor$
21:         else
22:             $sNeighborhood.remove(bestNeighbor)$
23:             $bestNeighbor \leftarrow getBestNeighbor(sNeighborhood)$
24:         end if
25:     end while
26:     if $length(tabu\_list) == |T|$ then
27:         $tabu\_list.pop(0)$
28:     end if
29:     $tabu\_list \leftarrow tabu\_list \cup \{bestNeighbor\}$
30:     $bestCandidate \leftarrow bestNeighbor$
31:     if $f(bestCandidate) < f(bestSolution)$ then
32:         $bestSolution \leftarrow bestCandidate$
33:     end if
34: end while
35: Return $bestSolution$
```

5 A NEW MEMETIC GENETIC ALGORITHM

Memetic algorithms combine evolutionary algorithms with local search operators and are widely used in combinatorial optimization [20]. Our algorithmic approach to the multi-objective nature of this problem combines a genetic algorithm (GA) with local search (here, with tabu search). As such, it can be considered as a memetic multi-objective algorithm. The proposed approach, memetic genetic algorithm (TSM), is outlined in Algorithm 2. Details of each step are given in the following subsections.

5.1 Representation

For the chromosome representation we follow the approach presented in [29]. In this representation each individual is a tuple (u, v), where u represents the operation sequences and v the machine assignment for operations. In detail, u is a vector of integers in which the operations of each job is denoted by the corresponding job number. Thus the $k-$th occurrence of a job number refers to the $k-$th operation in the sequence of this job. For example, the operation sequence $[1, 2, 1, 2, 1]$ represents the operation sequence $[0_{11}0_{21}0_{12}0_{22}0_{13}]$. For the machine assignment vector v, each number represents the machine assigned for each operation successively. For example, for a two job problem with 3 and 2 operations, respectively, and 3 machines, the vector [[132][12]], means that 0_{11} is

Algorithm 2 TSM.

```
1: Initialize population
2: Apply local search on population
3: Evaluate population
4: Hypervolume calculation of the Pareto front
5: while Termination criterion not met do
6:     Mutate individuals mapping to the same value and evaluate them
7:     Create offspring (parent selection, reproduction, mutation)
8:     Apply local search to offspring individuals
9:     Evaluate offspring population
10:    Merge parent population with offspring population
11:    Select new parent population for next generation
12:    Compute hypervolume indicator, check for stagnation and adjust parameters accordingly
13: end while
```

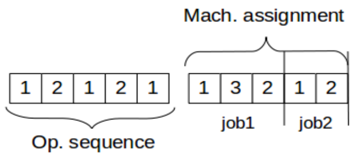

Figure 1: Individual representation.

sequenced on machine 1, 0_{12} on machine 3 and operation 0_{13} on machine 2 and so on for the other job. In Figure 1 we see an illustration of the example above, which represents the following operation sequence and their assigned machines: $(0_{11}, M_1)$, $(0_{21}, M_1)$, $(0_{12}, M_3)$, $(0_{22}, M_2)$, $(0_{13}, M_2)$.

5.2 Initialization

For the initialization of the initial population (line 1 in Algorithm 2) we follow the procedure introduced in [26] for both the machine assignment sequence and the operation sequence. For the machine assignment we switch between two assignment approaches. Assignment rule 1 starts from the operation that corresponds to the minimum in the processing time table. Assignment rule 2 permutes randomly the jobs in the table, before applying the approach by *localization*, described in [16]. This approach takes into account both the processing times and the workload of the machines, i.e., the sum of the processing times of all the operations assigned to each of the machines. The procedure then, consists in finding, for each operation, the machine with the minimum processing time, fixing that particular assignment, and then adding this minimum processing time to every subsequent entry in the same column (machine workload update) [26]. The initialization with the minimum method has a rate of 10% and the initialization with permutation 90%, based on [26]. After the machine assignment is settled, we move on to the operation sequencing. The sequencing of the initial assignments is obtained by a mix of three known dispatching rules:

- Randomly select a job. In this method, a job is randomly selected to be put into the chromosome.
- Most work remaining. In this method before selecting an operation, the remaining processing times of all jobs are calculated respectively, and the first unselected operation sequence of the job with the highest remaining processing time is placed into the chromosome.
- Most number of operations remaining. In this method before selecting an operation, the number of succeeding operations of all jobs are calculated respectively, and the first unselected operation sequence of the job with the highest number is placed into the chromosome.

The three dispatching rules above, are used interchangeably with rates 20%, 40%, 40%, respectively, based on [26].

5.3 Parent Selection and Offspring Generation

For the parent selection we use tournament selection, i.e., the individual for reproduction is chosen to be the one with the smallest makespan among a particular number q of randomly selected individuals. Once the individuals for reproduction have been selected, the crossover and mutation operators are applied to produce the offspring (line 7 in Algorithm 2). The crossover operator is applied to pairs of chromosomes, while the mutation operator is applied to single individuals. We distinguish between two kinds of operators:

- Assignment operators, referring to the machine assignment of individuals.
- Sequencing operators, referring to the sequencing of operations of individuals.

Assignment operators only change the machine assignment of the individuals, i.e., the sequencing of operations is preserved in the offspring. Assignment crossover generates the offspring by exchanging the assignment of a subset of operations between the two parents. Assignment mutation, on the other hand, only exchanges the assignment of a single operation in a single parent. In this paper for the machine assignment operators we used the recombination operator as in [29]. For the mutation operator for the assignment we used a mixture of the approach of [29] and TS. In this TS we used as neighborhood structure of the assignment of the individual the use of random selection of two operations. With these operations in hand we then randomly exchange the machine already assigned for these operations, with another one from the set of available machines of those operations. We did this 10 times to define the neighborhood around the current seed. To save time we only selected operations that have more than one machine available. We used a neighborhood of steady size equal to 10, and used as the tabu list tenure the closest integer to the squared root of the size of the neighborhood. We also used as a stopping rule 20 repetitions of the TS and a limit of 5 repetitions without improvement. Moreover, we used a 50% probability for the switch between these two mutation methods. The parameters selected here were based on preliminary results.

Sequencing operators only change the sequence of the operations in the parent chromosomes, i.e., the assignment of operations to machines is preserved in the offspring. In applying the sequencing operators, we must respect the precedence constraints among operations of the same job. We followed the suggestion of [29] for both the crossover and mutation for the sequencing operator.

Finally, non-dominated sorting and the crowding distance operator from NSGA-II [8] are applied for parent selection for the next generation, after merging the offspring with the current parent population.

5.4 Local Search

We decided to hybridize genetic algorithms with tabu search (TS) due to the fact that in many combinatorial optimization problems TS can be locally more exhaustive than genetic search [15], as it prevents premature convergence in sub-optimal regions, such as local optima. The local search we decided to use is a TS. For the local search we focused on the minimization of the makespan.

In detail, in every iteration we applied TS to 10% of the individuals after the genetic operators are applied and before the merging between the parent population and offspring population takes place. Since we focused on the minimization of the makespan, based on [24] we made adjustments only on particular parts of the critical path of the individual-schedule, called blocks. The critical path is the longest path on the disjunctive graph representation of a schedule [28], [3]. Blocks can be considered as the maximal subset of the critical path, which contains operations processed on the same machine. The TS parameters used were: 7 for the tabu list size, 5 for the maximum number of iterations without improvement and 20 for the maximum overall number of iterations. The parameters selected here were based on preliminary results. The neighborhood structure was based on the notion of critical paths and blocks, and as a result the neighborhood size varied per iteration.

5.5 Problem Specific Hypervolume Calculation

In each iteration we calculate the hypervolume indicator [9] of the Pareto front of the solutions. To counter the possible stagnation we entered a switch. If the hypervolume is stagnant for more than 3 generations, we increase the number of the possible solutions entering the local search to 0.5. A problem-specific conservative choice of a reference point is used, as follows: The basic idea is to find a point which will bound from above the Pareto front and as such we decided to go with the summation of the predefined processing times on all capable machines of all processes overall the jobs. In detail we used as a reference point the triple (x, x, x), where:

$$x = \sum_{i=1}^{N} \sum_{j=1}^{N_i} \sum_{k \in \mathcal{M}_{ij}} p_{ijk} \quad , \tag{4}$$

where p_{ijk} is the process time of 0_{ij} on machine k from its set of machines able to process it. Obviously, this point varies per problem instance, as it is directly related to its input data.

5.6 Solution Redundancy

One issue that is common for the FJSSP is solution redundancy. With this we mean that more than one solution in the decision space maps to the same value in the objective space. That is, the mapping is not injective. There is also the chance that after several generations some individuals in the decision space are duplicates. We tackled the first of the two issues by inserting (line 6 in Algorithm 2) an operator which determines the individuals that map to the same objective values. After that, it selects the largest subset of individuals which map to the same value and mutates them.

6 EXPERIMENTAL SETUP AND RESULTS

We tested the performance of our algorithm on the 10 instances Mk01-10 taken from Brandimarte [4]. Table 1 summarizes the parameter settings of our new algorithm as used for these runs. We ran our algorithm on each benchmark 30 times and aggregated the results keeping in the end the non-dominated solutions from the aggregated ones. We used as a termination criterion 500, 000 examined solutions, as proposed in [33]. Furthermore, we executed our experiments on the DAS-4 (Distributed ASCII Computer) [2], with 16 dual quad-core at 2.4GHz with 48GB RAM, in Python 3. We furthermore used the DEAP [11] framework to build TSM. To the best of our knowledge, this is the first research written in Python on multi-objective scheduling [1].

We compared our algorithm to the state-of-the-art algorithms MA-1, MA-2, MA-1-NH, MA-2-NH, MRLS-1, MRLS-2, NSGA-II variant, from Yuan et. al, [33]. Their parameter settings, as well as their solutions and reference set can be found in the same paper. We compared our results with the aggregated results over 30 runs of each of their algorithm, and we did this for each benchmark. We report the results found between our algorithm and theirs as well as the hypervolume indicator difference between their reference set (after having added our solutions) and our solutions, for each benchmark. The results are summarized in Table 2 and Table 3.

For saving space we only report our results for each benchmark and the dominated solutions of the algorithms we compare to, if they exist, otherwise there is a '–'. Furthermore, we also report on the new solutions found, indicating this with the phrase *Extended by*. For some benchmarks, specifically for Mk06, Mk07 we report only part of the results of TSM due their big number, and also part of the new solutions found against MRLS-1 on Mk06 and Mk10 [2].

From the results we can see that TSM performs well in terms of diversity of solutions in most benchmarks, although we can also detect a slight bias on detecting points with big values of makespan. We furthermore, see that TSM performs well against both MRLS-1 and MRLS-2 in all instances by either dominating some of their solutions or by determining new points that extened the Pareto front found by MRLS-1 and MRLS-2. Specifically, TSM partially dominates the solutions of MRLS-1 in 8 out of 10 benchmarks (Mk01, Mk02, Mk03, Mk04, Mk05, Mk07, Mk08, Mk09) and finds new Pareto solutions in 2 out of 10 benchmarks (Mk06 and Mk10). Regarding MRLS-2, TSM partially dominates the solutions returned by MRLS-2 in 7 out of 10 cases (Mk01, Mk03, Mk04, Mk05, Mk06, Mk08, Mk10) and identifies new solutions in 3 out of 10 cases (Mk02, Mk07, Mk09). Furthermore, we see that for instance Mk06, TSM identifies a new solution to the results returned by MA-1, MA-2 and NSGA-II, however in most cases MA-1, MA-2, NSGA-II dominate our solutions. Similarly we did not find any new or dominating solutions compared to the solutions returned by MA-1-NH and MA-2-NH in any case.

We furthermore computed the difference between the hypervolume indicator of the reference set (after having added our solutions)

[1]The source code will be provided in http://liacs.leidenuniv.nl/~csmoda/

[2]The full list of solutions will be made available in http://liacs.leidenuniv.nl/~csmoda/

Table 1: Parameter settings of the TSM.

Parameter	Value
Population size	300
Crossover probability	0.5
Mutation probability	1
Tournament size	3
Local search probability	0.1
Mutation tabu tenure	3
Mutation tabu search maximum number of no progress	5
Mutation tabu search maximum number of iterations	20
Local search tabu tenure	7
Local search maximum number of no progress	5
Local search maximum number of iterations	20

and the hypervolume indicator of the augmented solutions (union over 30 runs) of all algorithms, for each benchmark. We normalized the reference sets and the obtained sets by using the nadir point of the reference set multiplying by 1.1 [23]. The results can be seen in Table 4. It is clear from the results that algorithms MA-1 and MA-2 have the best results. Nevertheless our approach gives (see TSM results in bold), when compared to MRLS-1 and MRLS-2, competitive results on Mk01, better results in Mk03 and Mk04, Mk05 (compared to MRLS-2), Mk07, Mk08 and Mk09. In Mk08 our algorithm is able to determine the reference set.

In Table 5 we present the median of the difference between the hypervolume indicator of the reference set (after having added our solutions) and the hypervolume indicator of each algorithm on each benchmark on 30 trials. We used the Wilcoxon rank sum test, with a significance level of $\alpha = 0.01$, to see whether the hypervolume difference values obtained with the TSM strategy are significantly better than those obtained with one of the other strategies. We used also the Bonferroni correction, which means, that for each individual test the significance level α is divided by the number of tests per test instance. For us this is 7. The superscripts in the bold TSM indicate from which of the other 7 algorithms the TSM performed, on average, significantly better. From the table we see that, on average, TSM performed significantly better than both MRLS-1 and MRLS-2 on Mk03, Mk04 and Mk08 and significantly better than MRLS-2 on Mk09.

Finally, the average time (in seconds) is 994 for Mk01, 1057 for Mk02, 2880 for Mk03, 1798 for Mk03, 2094 for Mk05, 2743 for Mk06, 1888 for Mk07, 5546 for Mk08, 5277 for Mk09 and 5176 for Mk10.

7 CONCLUSIONS AND FUTURE WORK

To conclude we designed a memetic multi-objective algorithm for the flexible job shop scheduling problem (FJSSP). We used tabu search (TS) as the local search method, emphasizing on the makespan. Although there have been many publications on the FJSSP, to our knowledge there haven't been any publications that use TS in the context of multi-objective FJSSP. The main novelties of our work are the use of TS as the local search method and mutation operator and the use of the hypervolume indicator for stagnation check. We tested our approach on the famous Brandimarte dataset [4] and compared our solutions to the solutions of the algorithms by Yuan et al. [33]. Although, our approach did not yield overall better solutions, it was able to find better solutions and dominate several points found by two of their algorithms (MRLS-1 and MRLS-2) and find solutions that were not contained in and not dominated by the Pareto front approximations found by other algorithms. In one of the cases (Mk06) we also found a new solution on the Pareto front returned by three of their algorithms (MA-1, MA-2, NSGA-II). We furthermore compared the hypervolume indicator of the reference set in [33] and our solutions for each benchmark and we see that, when compared to two of [33] algorithms (MRLS-1 and MRLS-2) we get competitive (Mk01) or better results (Mk03, Mk04, Mk05, Mk07, Mk09, Mk08). Finally, we performed the Wilcoxon rank sum test to see whether the hypervolume difference values obtained with the TSM strategy are significantly better than those obtained with one of the other strategies. The results showed us that, on average, TSM performed significantly better than both MRLS-1 and MRLS-2 on Mk03, Mk04 and Mk08 and significantly better than MRLS-2 on Mk09. In summary the TSM algorithm is an interesting alternative to the algorithms of Yuan et al., in terms of quality. Besides, it is made available as an open source Python implementation, which makes multi-objective FJSSP available to the big community of Python programmers.

This paper is intended to be a stepping stone towards our future research directions on robust and online scheduling. There we will deal with unexpected events, such as sudden jobs appearances, where optimal scheduling should take place on-the-fly.

ACKNOWLEDGMENTS

This work is part of the research programme Smart Industry SI2016 with project name CIMPLO and project number 15465, which is partly financed by the Netherlands Organisation for Scientific Research (NWO).

REFERENCES

[1] Hasan Ali, Rashedul Haque, and Fashiar Rahman. 2017. Chronological Evolution Of Flexible Job Shop Scheduling: A Review. *Wiley Encyclopedia of Electrical and Electronics Engineering* (2017), 7.

[2] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer* 49, 5 (May 2016), 54–63. https://doi.org/10.1109/MC.2016.127

[3] Egon Balas. 1969. Machine Sequencing Via Disjunctive Graphs: An Implicit Enumeration Algorithm. *Operations Research* 17, 6 (Dec. 1969), 941–957. https://doi.org/10.1287/opre.17.6.941

[4] Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 41, 3 (Sept. 1993), 157–183. https://doi.org/10.1007/BF02023073

[5] Hao-Chin Chang, Yeh-Peng Chen, Tung-Kuan Liu, and Jyh-Horng Chou. 2015. Solving the Flexible Job Shop Scheduling Problem With Makespan Optimization by Using a Hybrid Taguchi-Genetic Algorithm. *IEEE Access* 3 (2015), 1740–1754. https://doi.org/10.1109/ACCESS.2015.2481463

[6] Imran Ali Chaudhry and Abid Ali Khan. 2016. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research* 23, 3 (May 2016), 551–591. https://doi.org/10.1111/itor.12199

[7] Tsung-Che Chiang and Hsiao-Jou Lin. 2012. Flexible Job Shop Scheduling Using a Multiobjective Memetic Algorithm. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, De-Shuang Huang, Yong Gan, Phalguni Gupta, and M. Michael Gromiha (Eds.). Vol. 6839.

Table 2: Results.

Algorithm	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
TSM	(40, 168, 37), (41, 167, 36), (41, 162, 39), (41, 165, 37), (42, 159, 39), (42, 160, 38), (40, 174, 36), (41, 164, 38), (43, 155, 40), (42, 163, 37), (42, 165, 36), (43, 158, 39), (44, 154, 40), (46, 153, 42)	(29, 150, 26), (29, 144, 28), (29, 145, 27), (30, 143, 29), (31, 141, 31), (31, 142, 30), (33, 140, 33)	(204, 864, 204), (206, 857, 204), (210, 855, 204), (213, 852, 204), (215, 849, 213), (216, 848, 213), (222, 847, 222), (223, 847, 213), (224, 851, 204), (226, 843, 222), (230, 842, 222), (234, 846, 213), (237, 844, 213), (240, 850, 204), (246, 841, 231), (247, 849, 210), (248, 848, 210), (249, 840, 249), (256, 838, 249), (256, 840, 222), (262, 838, 231), (274, 839, 222), (275, 838, 222), (282, 837, 231), (297, 843, 221)	(68, 355, 68), (68, 376, 60), (69, 360, 60), (69, 351, 63), (71, 353, 62), (72, 347, 66), (72, 357, 61), (73, 342, 72), (73, 348, 63), (75, 344, 66), (75, 347, 65), (77, 340, 72), (78, 337, 78), (79, 343, 67), (84, 334, 84), (90, 331, 90), (98, 330, 98), (106, 329, 106), (114, 328, 114), (122, 327, 122), (130, 326, 130), (138, 325, 138), (146, 324, 146)	(174, 687, 173), (176, 686, 173), (177, 685, 173), (178, 683, 175), (178, 682, 176), (179, 684, 174), (179, 680, 179), (180, 682, 175), (180, 681, 178), (181, 684, 173), (181, 679, 179), (181, 680, 178), (182, 683, 173), (182, 687, 172), (183, 677, 183), (185, 676, 185), (191, 675, 191), (197, 674, 197), (203, 673, 203), (209, 672, 209)	(91, 474, 57), (91, 453, 66), (92, 436, 60), (93, 480, 54), (95, 456, 55), (96, 434, 60), (96, 428, 61), (99, 432, 60), (99, 427, 71), (100, 476, 54), (100, 450, 57), (102, 455, 54), (103, 452, 54), (103, 446, 59), (104, 451, 54), (105, 449, 55), (106, 420, 74), (107, 423, 63), (108, 421, 69), (108, 447, 56), (109, 421, 66), (109, 441, 59), (110, 442, 55), (110, 421, 60), (112, 417, 67), (113, 411, 74), (115, 414, 68), (115, 415, 63), (122, 439, 56), ... (129, 437, 57), (130, 434, 58), (131, 440, 55), (131, 413, 63), (136, 449, 54), (139, 407, 69), (140, 444, 54), (141, 438, 56), (141, 439, 55), (142, 411, 65), (143, 402, 82), (144, 406, 67), (154, 434, 54), (158, 473, 53)	(144, 690, 144), (148, 685, 144), (150, 690, 143), (150, 684, 149), (153, 680, 150), (153, 683, 147), (154, 673, 150), (156, 682, 147), (157, 683, 145), (157, 691, 142), (158, 670, 156), (158, 679, 145), (158, 690, 140), (160, 675, 147), (160, 671, 150), (160, 677, 144), (161, 673, 144), (162, 668, 156), (163, 666, 162), (163, 667, 157), (166, 664, 157), ... (172, 687, 143), (174, 688, 140), (175, 686, 140), (176, 660, 174), (178, 668, 152), (179, 657, 170), (182, 684, 143), (185, 665, 156), (191, 660, 169), (192, 661, 162), (193, 659, 162), (194, 655, 190), (197, 655, 176), (206, 653, 202), (220, 658, 166), (221, 654, 190), (227, 653, 187), (241, 652, 209), (244, 657, 166), (265, 651, 209), (268, 651, 205), (277, 652, 202)	(523, 2524, 523), (524, 2519, 524), (533, 2514, 533), (542, 2509, 542), (551, 2504, 551), (560, 2499, 560), (569, 2494, 569), (578, 2489, 578), (587, 2484, 587)	(369, 2711, 328), (372, 2493, 310), (373, 2452, 299), (377, 2415, 300), (379, 2396, 299), (386, 2375, 320), (389, 2387, 299), (393, 2365, 315), (394, 2376, 299), (396, 2368, 299), (399, 2364, 307), (401, 2336, 331), (401, 2364, 299), (410, 2340, 316), (414, 2361, 315), (419, 2352, 304), (424, 2361, 299), (427, 2359, 300), (427, 2360, 299), (432, 2341, 299), (448, 2331, 328), (468, 2322, 307), (493, 2339, 299), (507, 2338, 303), (523, 2338, 299), (534, 2335, 301), (543, 2311, 320), (559, 2321, 310), (563, 2335, 299), (567, 2327, 299)	(300, 2157, 224), (311, 2128, 256), (313, 2190, 220), (313, 2127, 242), (313, 2132, 241), (314, 2133, 230), (315, 2156, 220), (316, 2128, 220), (317, 2127, 211), (318, 2113, 239), (318, 2125, 230), (321, 2101, 259), (322, 2122, 223), (323, 2113, 224), (324, 2112, 217), (325, 2094, 220), (326, 2090, 221), (331, 2109, 214), (332, 2171, 210), (333, 2137, 210), (335, 2106, 218), (336, 2087, 233), (336, 2112, 208), (339, 2082, 229), (343, 2109, 213), (345, 2107, 216), (353, 2105, 215), (357, 2082, 220), (358, 2111, 212), (359, 2069, 253), (359, 2091, 208), (362, 2080, 250), (362, 2081, 236), (363, 2057, 242), (364, 2054, 210), (364, 2128, 205), (368, 2115, 206), (390, 2092, 205), (397, 2050, 248), (416, 2084, 206), (427, 2127, 204), (452, 2082, 206), (460, 2078, 209), (515, 2132, 202)
MA-1	-	-	-	-	-	Extended by: (158, 473, 53)	-	-	-	-
MA-2	-	-	-	-	-	Extended by: (158, 473, 53)	-	-	-	-
MA-1-NH	-	-	-	-	-	-	-	-	-	-
MA-2-NH	-	-	-	-	-	-	-	-	-	-
MRLS-1	(43, 163, 37), (43, 156, 40), (42, 166, 36), (46, 153, 46)	(33, 142, 30)	(212, 932, 204), (204, 956, 204), (207, 947, 204)	(79, 338, 78), (84, 335, 84), (78, 339, 78)	(186, 676, 186), (192, 675, 192), (181, 679, 181)]	Extended by: (91, 474, 57), (92, 436, 60), (93, 480, 54), (95, 456, 55), ... (139, 407, 69), (140, 444, 54), (141, 438, 56), (141, 439, 55), (142, 411, 65), (144, 406, 67), (154, 434, 54), (158, 473, 53)	(157, 673, 150), (150, 688, 144), (149, 689, 144)	(555, 2531, 542), (523, 2542, 523), (524, 2541, 524), (533, 2532, 533), (530, 2540, 524)	(387, 2382, 320)	Extended by: (300, 2157, 224), (313, 2190, 220), (314, 2133, 230), (315, 2156, 220), (316, 2128, 220), (317, 2127, 211), (318, 2113, 239), (318, 2125, 230), (322, 2122, 223), (323, 2113, 224), (324, 2112, 217), ... (390, 2092, 205), (397, 2050, 248), (416, 2084, 206), (427, 2127, 204), (452, 2082, 206), (460, 2078, 209), (515, 2132, 202)

Table 3: Results

Algorithm	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
MRLS-2	(47, 153, 42), (48, 165, 36), (43, 163, 37), (46, 166, 36), (46, 153, 46)	Extended by: (33, 140, 33)	(204, 931, 204)	(78, 339, 78), (84, 336, 84), (72, 360, 61)	(198, 674, 198), (186, 676, 186)	(92, 439, 62), (92, 477, 61), (94, 475, 61), (99, 486, 60)	Extended by: (150, 690, 143), (157, 691, 142), (158, 679, 145), (158, 690, 140), (160, 675, 147), (160, 671, 150), (160, 677, 144), (161, 673, 144), (166, 670, 150), (168, 689, 142), (169, 688, 141), (169, 663, 162), (170, 662, 157), (171, 661, 169), (172, 667, 156), (172, 687, 143), (174, 688, 140), (175, 686, 140), (176, 660, 174), (178, 668, 152), (179, 657, 170), (182, 684, 143), (185, 665, 156), (191, 660, 169), (192, 661, 162), (193, 659, 162), (194, 655, 190), (197, 655, 176), (206, 653, 202), (220, 658, 166), (221, 654, 190), (227, 653, 187), (241, 652, 209), (244, 657, 166), (265, 651, 209), (268, 651, 205), (277, 652, 202)	(560, 2528, 560), (523, 2537, 523), (524, 2532, 524), (543, 2530, 542), (569, 2525, 569)	Extended by: (373, 2452, 299), (377, 2415, 300), (379, 2396, 299), (386, 2375, 320), (389, 2387, 299), (393, 2365, 315), (394, 2376, 299), (396, 2368, 299), (399, 2364, 307), (401, 2336, 331), (401, 2364, 299), (410, 2340, 316), (414, 2361, 315), (419, 2352, 304), (424, 2361, 299), (427, 2359, 300), (427, 2360, 299), (432, 2341, 299), (448, 2331, 328), (468, 2322, 307), (493, 2339, 299), (507, 2338, 303), (523, 2338, 299), (534, 2335, 301), (543, 2311, 320), (559, 2321, 310), (563, 2335, 299), (567, 2327, 299)	(330, 2100, 239)
NSGA-II	-	-	-	-	-	Extended by: (158, 473, 53)	-	-	-	-

Table 4: Hypervolume indicator difference from reference set (smaller is better).

Algorithms	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
TSM	0.000263	0.002660	**0.002046**	**0.004647**	**0.000247**	0.056141	**0.002001**	**0.000000**	**3.509746e-03**	0.021632
MA-1	0.000000	0.000000	0.000000	0.000000	0.000000	0.003675	0.000000	0.000000	5.414126e-07	0.000915
MA-2	0.000000	0.000000	0.000000	0.000000	0.000000	0.002644	0.000000	0.000000	1.598030e-09	0.000412
MA-1-NH	0.000040	0.000043	0.000000	0.000189	0.000000	0.002143	0.000000	0.000000	2.404332e-05	0.002283
MA-2-NH	0.000000	0.000047	0.000000	0.000076	0.000000	0.002596	0.000000	0.000000	2.520381e-05	0.002037
MRLS-1	0.000131	0.000226	0.016691	0.004766	0.000266	0.042805	0.003620	0.000409	9.736424e-03	0.019498
MRLS-2	0.000113	0.000217	0.020501	0.004812	0.000219	0.043104	0.003403	0.000355	9.138569e-03	0.019221
NSGA-II	0.000040	0.000138	0.000000	0.000564	0.000000	0.007663	0.000000	0.000000	3.174112e-04	0.003024

Table 5: Median of hypervolume indicator difference from reference set.

Algorithms	Mk01	Mk02	Mk03	Mk04	Mk05	Mk06	Mk07	Mk08	Mk09	Mk10
TSM	0.000998	0.003774	**0.003641**^{5,6}	**0.007081**^{5,6}	0.000772	0.061426	0.005147	**0.000020**^{5,6}	**0.010655**^{6}	0.022215
MA-1	0.000000	0.000050	0.000000	0.000295	0.000000	0.005438	0.000000	0.000000	0.000005	0.001809
MA-2	0.000000	0.000045	0.000000	0.000342	0.000000	0.005230	0.000000	0.000000	0.000003	0.001432
MA-1-NH	0.000042	0.000185	0.000000	0.000400	0.000000	0.007017	0.000000	0.000000	0.000112	0.003695
MA-2-NH	0.000040	0.000142	0.000000	0.000381	0.000000	0.006897	0.000000	0.000000	0.000110	0.003174
MRLS-1	0.000504	0.001232	0.020065	0.010253	0.000375	0.049451	0.005101	0.000571	0.010895	0.020479
MRLS-2	0.000420	0.001167	0.021041	0.009804	0.000326	0.048765	0.004778	0.000521	0.010474	0.020179
NSGA-II	0.000108	0.000905	0.000000	0.001873	0.000105	0.010133	0.000715	0.000000	0.000359	0.004454

Springer Berlin Heidelberg, Berlin, Heidelberg, 49–56. https://doi.org/10.1007/978-3-642-25944-9_7

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

[9] Michael Emmerich, Nicola Beume, and Boris Naujoks. 2005. An EMO Algorithm Using the Hypervolume Measure as Selection Criterion. In *Evolutionary Multi-Criterion Optimization*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Carlos A. Coello Coello, Arturo HernÃąndez Aguirre, and Eckart Zitzler (Eds.). Vol. 3410. Springer Berlin Heidelberg, Berlin, Heidelberg, 62–76. https://doi.org/10.1007/978-3-540-31880-4_5

[10] Michael T. M. Emmerich and Andrè H. Deutz. 2018. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing* 17, 3 (Sept. 2018), 585–609. https://doi.org/10.1007/s11047-018-9685-y

[11] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[12] M. R. Garey, D. S. Johnson, and Ravi Sethi. 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research* 1, 2 (May 1976), 117–129. https://doi.org/10.1287/moor.1.2.117

[13] Fred Glover. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* 13, 5 (Jan. 1986), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

[14] Alain Hertz, Eric Taillard, and Dominique de Werra. 1996. A Tutorial on Tabu Search. (1996), 13.

[15] H. Ishibuchi, T. Yoshida, and T. Murata. 2003. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation* 7, 2 (April 2003), 204–223. https://doi.org/10.1109/TEVC.2003.810752

[16] I. Kacem, S. Hammadi, and P. Borne. 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 32, 1 (Feb. 2002), 1–13. https://doi.org/10.1109/TSMCC.2002.1009117

[17] Grecia Lapizco-Encinas, Carl Kingsford, and James Reggia. 2009. A cooperative combinatorial Particle Swarm Optimization algorithm for side-chain packing. In *2009 IEEE Swarm Intelligence Symposium*. IEEE, Nashville, 22–29. https://doi.org/10.1109/SIS.2009.4937840

[18] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys. 1993. Chapter 9 Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science*. Vol. 4. Elsevier, 445–522. https://doi.org/10.1016/S0927-0507(05)80189-6

[19] Sanghyup Lee, Ilkyeong Moon, Hyerim Bae, and Jion Kim. 2012. Flexible job-shop scheduling problems with 'AND'/'OR' precedence constraints. *International Journal of Production Research* 50, 7 (April 2012), 1979–2001. https://doi.org/10.1080/00207543.2011.561375

[20] Pablo Moscato. 1989. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts Towards Memetic Algorithms. (1989), 68.

[21] Pablo Moscato and Carlos Cotta. 2003. A Gentle Introduction to Memetic Algorithms. In *Handbook of Metaheuristics*, Fred Glover and Gary A. Kochenberger (Eds.). Vol. 57. Kluwer Academic Publishers, Boston, 105–144. https://doi.org/10.1007/0-306-48056-5_5

[22] Ghasem Moslehi and Mehdi Mahnam. 2011. A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics* 129, 1 (Jan. 2011), 14–22. https://doi.org/10.1016/j.ijpe.2010.08.004

[23] B. Naujoks, N. Beume, and M. Emmerich. 2005. Multi-objective optimisation using S-metric selection: application to three-dimensional solution spaces. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 2. 1282–1289 Vol. 2. https://doi.org/10.1109/CEC.2005.1554838

[24] Eugeniusz Nowicki and Czeslaw Smutnicki. 1996. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science* 42, 6 (1996), 797–813. http://www.jstor.org/stable/2634595

[25] E O Oyetunji. 2009. Some Common Performance Measures in Scheduling Problems: Review Article. (2009), 5.

[26] F. Pezzella, G. Morganti, and G. Ciaschetti. 2008. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research* 35, 10 (Oct. 2008), 3202–3212. https://doi.org/10.1016/j.cor.2007.02.014

[27] Seyed Habib A. Rahmati, M. Zandieh, and M. Yazdani. 2013. Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 64, 5-8 (Feb. 2013), 915–932. https://doi.org/10.1007/s00170-012-4051-1

[28] Yves Tabourier. 1969. Probème d'ordonnancement è contraintes purement disjonctives. *Revue française d'informatique et de recherche opérationnelle. Série verte* 3, V3 (1969), 51–65. https://doi.org/10.1051/ro/196903V300511

[29] Xiaojuan Wang, Liang Gao, Chaoyong Zhang, and Xinyu Shao. 2010. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 51, 5-8 (Nov. 2010), 757–767. https://doi.org/10.1007/s00170-010-2642-2

[30] Wenchao Yi, Xinyu Li, and Baolin Pan. 2016. Solving flexible job shop scheduling using an effective memetic algorithm. 53, 2 (2016).

[31] Jian Xiong, Xu Tan, Ke-wei Yang, Li-ning Xing, and Ying-wu Chen. 2012. A Hybrid Multiobjective Evolutionary Approach for Flexible Job-Shop Scheduling Problems. *Mathematical Problems in Engineering* 2012 (2012), 1–27. https://doi.org/10.1155/2012/478981

[32] Wenchao Yi, Xinyu Li, and Baolin Pan. 2016. Solving flexible job shop scheduling using an effective memetic algorithm. *International Journal of Computer Applications in Technology* 53, 2 (2016), 157. https://doi.org/10.1504/IJCAT.2016.074454

[33] Yuan Yuan and Hua Xu. 2015. Multiobjective Flexible Job Shop Scheduling Using Memetic Algorithms. *IEEE Transactions on Automation Science and Engineering* 12, 1 (Jan. 2015), 336–353. https://doi.org/10.1109/TASE.2013.2274517