A PRIORITY-BASED GENETIC ALGORITHM FOR A FLEXIBLE JOB SHOP SCHEDULING PROBLEM

DIDEM CINAR
Department of Industrial Engineering, Istanbul Technical University
34367 Istanbul, Turkey
and
Center for Applied Optimization, Department of Industrial and Systems Engineering
University of Florida, Gainesville, United States

JOSÉ ANTÓNIO OLIVEIRA
ALGORITMI Research Centre, University of Minho, Campus Azurem
4800-058 Guimarães, Portugal

Y. ILKER TOPCU
Department of Industrial Engineering, Istanbul Technical University
34367 Istanbul, Turkey

PANOS M. PARDALOS
Center for Applied Optimization, Department of Industrial and Systems Engineering
University of Florida, Gainesville, United States
and
National Research University Higher School of Economics
Laboratory of algorithms and technologies for network analysis (LATNA), Russia

(Communicated by Kok Lay Teo)

Abstract. In this study, a genetic algorithm (GA) with priority-based representation is proposed for a flexible job shop scheduling problem (FJSP) which is one of the hardest operations research problems. Investigating the effect of the proposed representation schema on FJSP is the main contribution to the literature. The priority of each operation is represented by a gene on the chromosome which is used by a constructive algorithm performed for decoding. All active schedules, which constitute a subset of feasible schedules including the optimal, can be generated by the constructive algorithm. To obtain improved solutions, iterated local search (ILS) is applied to the chromosomes at the end of each reproduction process. The most widely used FJSP data sets generated in the literature are used for benchmarking and evaluating the performance of the proposed GA methodology. The computational results show that the proposed GA performed at the same level or better with respect to the makespan for some data sets when compared to the results from the literature.

2010 Mathematics Subject Classification. Primary: 58F15, 58F17; Secondary: 53C35.

Key words and phrases. Genetic algorithms, priority-based coding, flexible job shop scheduling problem, permutation coding, iterated local search.

This research is partially supported by LATNA Laboratory, NRU HSE, RF government grant, ag. 11.G34.31.0057.

* Corresponding author: Didem Cinar.
1. **Introduction.** Scheduling theory is one of the significant research areas in Operations Research. It has been the subject of research with techniques ranging from simple dispatching rules to sophisticated learning algorithms [38]. Job shop scheduling problem (JSP) is one of the most studied topics in scheduling theory which is known to be difficult to solve. Although the name job shop scheduling seems to refer to an industrial problem, the structure of JSP is related to various applications, such as management, computing, and public services. For detailed review of JSP, readers are referred to Jain and Meeran [38] and Lee and Pinedo [45].

In JSP, there are $n$ jobs planned to be scheduled on $m$ machines. Each job consists of consecutive operations such that the precedence relations should be fulfilled. Each operation has to be processed by a particular machine.

The flexible job shop scheduling problem (FJSP) is more complex and general than the classical JSP. Unlike the JSP, each operation can be processed by one of the machines in a given machine set. The aim of FJSP is to find both an assignment and a corresponding scheduling that minimize production time. The first study on FJSP was performed in 1990 by Brucker and Schlie [11] who developed a polynomial time algorithm for FJSP with two jobs. Since then, various approximation algorithms have been developed in FJSP literature to obtain good results for large size instances [64]. Furthermore, progressive hybrid methodologies have been developed to improve the performance of existing techniques.

FJSP is an NP-hard problem because a simpler problem, that is JSP, is NP-hard in the strong sense [30]. So, it is still a challenge to obtain good results for FJSP in a reasonable time. Genetic algorithms (GA), one of the well-known metaheuristic algorithms, have received remarkable interest in addressing the FJSP.

Oliveira et al. [61] used random key representation schema for JSP to represent the priorities in the Giffler and Thompson Algorithm (GTA), which is a constructive algorithm developed by Giffler and Thompson in 1960 for JSPs [31]. By this method, the search area consists of only active schedules which are represented by generated chromosomes. In this study, this chromosome structure is used to see the effect of the representation on FJSP. A constructive algorithm based on GTA, which can produce all active schedules for FJSP, is used for decoding. According to the representation used by Oliveira et al. [61], a solution can be represented by multiple chromosomes. In this study, permutation encoding is used to reduce the number of alternative chromosomes representing the same solution. In order to improve the diversity and quality of the chromosomes, problem-based operators are used for FJSP.

The organization of this paper is as follows. In the next section, a brief presentation of FJSP is given. A literature review on various algorithms developed for FJSP is presented in Section 3. In Section 4, the proposed priority-based GA is discussed. In Section 5 computational results are presented for well-known instances from the literature. Lastly, Section 6 will be reserved for the conclusion, discussions and suggestions for further studies.

2. **Flexible job shop scheduling problem.** The FJSP consists of $n$ jobs $\{J_1, J_2, ..., J_n\}$ and $m$ machines $\{M_1, M_2, ..., M_m\}$ including different tools. Each job $J_i$ contains $n_i$ operations, $O_{i1}, O_{i2}, ..., O_{in_i}$. There is a set $M_{ij} \subseteq \{M_1, ..., M_m\}$ for each operation $O_{ij}$ ($i = 1, ..., n; j = 1, ..., n_i$) which has to be assigned on a machine in this set for being processed without preemption. The $p_{ijk}$ is the processing time of operation $O_{ij}$ on machine $k$. Moreover, at most one operation can be
processed on a machine at the same time and a job cannot be processed on more than one machine at the same time [11].

Figure 1. A FJSP Example

A $2 \times 3$ example, with two jobs and three machines, is shown in Figure 1. Operation $O_{11}$ can be processed on $M_1$ or $M_2$, $O_{12}$ and $O_{21}$ can be processed on $M_2$ or $M_3$, and $O_{22}$ must be processed on $M_3$.

An assignment $\mu$ is feasible, if $\mu(O_{ij}) \in M_{ij}$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, nt$. A $\mu$-schedule is defined as a performance measure for assignment $\mu$. A $\mu$-schedule is feasible if and only if it is feasible for a JSP based on a $\mu$ assignment [11]. So, the problem is finding a feasible assignment and corresponding schedule that minimize a performance measure. When $|M_{ij}| = 1$ for all $O_{ij}$, the problem becomes a JSP. Because of the existent alternative machines to process one operation, the FJSP is more complicated than the JSP [10]. Only the sequence of the operations is decided in the JSP, since the assignment of operations to machines is fixed. On the other hand, both the assignment and the order of operations on each machine are decided in the FJSP [65]. If a $\mu$ assignment is given, finding an optimum $\mu$-schedule is a JSP [37].

Makespan, which corresponds to the longest completion time, is the most widely used performance measure in the FJSP literature. Other performance measures typically used in the literature are: flowtime, total workload, critical machine workload, total tardiness and mean tardiness. In the three-field notation scheme [33], the problem handled in this study is denoted by $FJ || C_{max}$ where $FJ$ denotes FJSP and $C_{max}$ represents makespan.

3. Literature review. In the scope of this study, the studies developing approximation algorithms for FJSP are investigated. The optimization methods used in FJSP literature are listed in Table 1. The studies on single and multi-objective FJSFs are given in Tables 2 and 3 as well as the methodologies utilized to solve the corresponding problems.

GA is one of the well known population based algorithms [7] which is also the most widely used approximation method in FJSP literature. A big part of these studies combine GA with a local search algorithm to utilize both the global search and the local search abilities. Gao et al. [26] hybridized the GA and bottleneck shifting procedure for multi-objective FJSP. Zribi et al. [88] solved the FJSP by considering assignment and sequencing problems hierarchically. Two methods, based on local search and the branch and bound algorithm, were developed for the assignment problem while a hybridized GA was proposed for the sequencing. Gao et al. [25] combined GA with two neighborhood structures developed as the local search to solve the FJSP with non-fixed availability constrains. Gao et al. [27] also combined GA with variable neighborhood search to improve the search ability.
Table 1. Algorithms used in the FJSP literature

| Abbreviation | Algorithm                                      |
|--------------|------------------------------------------------|
| AC           | Ant Colonies                                   |
| AIS          | Artificial Immune System                       |
| AL           | Approach by Localization                       |
| BBA          | Branch and Bound Algorithm                     |
| BSP          | Bottleneck Shifting Procedure                  |
| CPM          | Critical Path Method                           |
| DEA          | Differential Evolution Algorithm               |
| DSA          | Discrete firefly algorithm                     |
| FBS          | Filtered beam search                           |
| GA           | Genetic Algorithm                              |
| GRASP        | Greedy Randomised Adaptive Search Procedure    |
| HSA          | Harmony Search Algorithm                       |
| IA           | Insertion Algorithm                            |
| ILS          | Iterated Local Search                         |
| LNS          | Large Neighborhood Search                      |
| LS           | Local Search                                   |
| PDR          | Priority Dispatching Rules                     |
| PSO          | Particle Swarm Optimization                    |
| SA           | Simulated Annealing                            |
| TS           | Tabu Search                                    |
| VNS          | Variable Neighborhood Search                   |

Kacem et al. [41] developed an assignment and scheduling procedure, called approach by localization, which was used to assign each operation to the suitable machine considering the processing times and workloads of the machines. In order to find better results for many real problems, they hybridized the GA with approach by localization to combine the advantages of the methods. Pezzella et al. [65] integrated different strategies to improve the performance of GAs for the FJSP. They used approach by localization to generate the initial solution and dispatching rules to obtain the sequencing of the initial assignments.

Li et al. [49] developed a hybrid method combining variable neighborhood search and GAs to solve the multi-objective FJSP. Frutos et al. [24] used the GA with simulated annealing to combine local and global search for solving multi-objective FJSP. Wang et al. [76] and Gao et al. [29] used a GA which is based on immune and entropy principles to solve the multi-objective FJSP.

Al-Hinai and ElMekkawy [1] combined a heuristic used for initial population generation, a local search method, and a GA for the FJSP. Moradi et al. [57] developed and compared four multi-objective optimization methods, which combine the GA and priority dispatching rules to minimize makespan and system unavailability, for integrated FJSP with non-fixed preventive maintenance activities. Xing et al. [79] proposed a multi-population interactive coevolutionary algorithm to improve the performance of evolutionary algorithms for the FJSP. Both the artificial ant colonies and GA with different configurations were applied to evolve each population independently and the interaction, competition and sharing mechanism among populations were satisfied. Ho et al. [36] proposed a new coding technique for a learnable GA which provides an effective integration between evolution and learning.
## Table 2. Single objective FJSP studies

| Study               | Year | Problem  | Methodology               |
|---------------------|------|----------|---------------------------|
| Baykasoglu [5]      | 2002 | FJSP     | SA + PDR                  |
| Schrich et al. [70] | 2004 | FJSP     | TS + PDR                  |
| Ho et al. [36]      | 2007 | FJSP     | GA + PDR                  |
| Liouane et al. [51] | 2007 | FJSP     | TS + AC                   |
| Fattahi et al. [23] | 2007 | FJSP     | TS + SA                   |
| Pezzella et al. [65] | 2008 | FJSP     | GA + AL + PDR             |
| Moradi et al. [56]  | 2010 | FJSP with fixed maintenance | GA + PDR                  |
| Bozejko et al. [9]  | 2010 | FJSP     | TS + IA                   |
| Xing et al. [79]    | 2011 | FJSP     | GA + AC                   |
| Al-Hinai and ElMekkawy [1] | 2011 | FJSP     | GA + LS                   |
| Karimi et al. [43]  | 2012 | FJSP     | VNS                       |
| Roshanaei et al. [69]| 2013 | FJSP     | AIS + SA                  |
| Yuan and Xu [81]    | 2013 | FJSP     | DEA + LS                  |
| Wang et al. [77]    | 2013 | FJSP with non-fixed maintenance | GA                      |
| Farughi et al. [22] | 2013 | FJSP with overlapping operations | GA + CPM                 |
| Yuan and Xu [82]    | 2013 | FJSP     | HSA + LNS                 |
| Yuan et al. [84]    | 2013 | FJSP     | HSA + LS                  |
| Mousakhani [59]     | 2013 | FJSP with sequence dependent set-up times | ILS                     |
| Ziaee [87]          | 2014 | FJSP     | Heuristic                 |
| Driss et al. [19]   | 2015 | FJSP     | GA                       |

within a random search process. Moradi et al. [56] used a learnable GA to solve FJSP with preventive maintenance activities and the composite dispatching rule to generate population. Tay and Ho [74] used composite dispatching rules generated by genetic programming. They approved that no rule has remarkable performance on all criteria and the combination of the rules enhances the efficiency of the rules and quality of the results.

In Farughi et al. [22] hybridized GA and a heuristic based on critical path method for FJSP with overlapping operations. The heuristic is used to improve the results of the proposed methodology. Chiang and Lin [16] proposed an evolutionary algorithm for a multi-objective FJSP. The distinctive characteristics of the proposed algorithm are its simplicity with only two parameters, effective genetic operators and population diversity. Yuan and Xu [83] also proposed new memetic algorithms hybridizing GA and local search for the multi-objective FJSP.

Driss et al. [19] developed a GA with novel representation and operators for FJSP minimizing makespan. The proposed algorithm is validated on a series of benchmark data sets and tested on data from a drug manufacturing company. Wang et al. [77] proposed a novel chromosome encoding for FJSP and used just-in-time machine assignment and adapting scheduling rules to achieve the robustness and flexibility.

Rahmati et al. [66] developed non-dominated sorting GA and non-dominated ranking GA for multi-objective FJSP. They proposed new multi-objective Pareto-based modules and a new measure for the multi-objective evaluation. Rohaninejad
### Table 3. Multi-objective FJSP studies

| Study                  | Year | Problem | Methodology                  |
|------------------------|------|---------|------------------------------|
| Kacem et al. [41]      | 2002 | FJSP    | GA + AL                      |
| Baykasoglu et al. [6]  | 2004 | FJSP    | TS + PDR                     |
| Xia and Wu [78]        | 2005 | FJSP    | PSO + SA                     |
| Gao et al. [25]        | 2006 | FJSP with non-fixed maintenance | GA             |
| Gao et al. [26]        | 2007 | FJSP    | GA + BSP                     |
| Zribi et al. [88]      | 2007 | FJSP    | GA + BBA + LS                |
| Gao et al. [27]        | 2008 | FJSP    | GA + VNS                     |
| Tay and Ho [74]        | 2008 | FJSP    | GA + PDR                     |
| Wang et al. [75]       | 2008 | FJSP    | FBS + PDR                    |
| Zhang et al. [85]      | 2009 | FJSP    | PSO + TS                     |
| Li et al. [49]         | 2010 | FJSP    | GA + VNS                     |
| Frutos et al. [24]     | 2010 | FJSP    | GA + SA                      |
| Wang et al. [76]       | 2010 | FJSP    | GA + AIS                     |
| Gao et al. [29]        | 2010 | FJSP    | GA + AIS                     |
| Grobler et al. [34]    | 2010 | FJSP with sequence dependent set-up times | PSO + PDR     |
| Li et al. [47]         | 2010 | FJSP    | TS + VNS                     |
| Moradi et al. [57]     | 2011 | FJSP with non-fixed maintenance | GA + PDR     |
| Mosleh and Mahnam [58] | 2011 | FJSP    | PSO + LS                     |
| Li et al. [48]         | 2011 | FJSP    | PSO                          |
| Li et al. [46]         | 2011 | FJSP    | PSO                          |
| Rajkumar et al. [67]   | 2011 | FJSP with limited resources | GRASP         |
| Chiang and Lin [16]    | 2013 | FJSP    | GA                           |
| Rahmati et al. [66]    | 2013 | FJSP    | GAs                          |
| Shao et al. [71]       | 2013 | FJSP    | PSO + SA                     |
| Gao et al. [28]        | 2014 | FJSP    | HSA + LS                     |
| Jia and Hu [40]        | 2014 | FJSP    | TS                           |
| Karthikeyan et al. [44]| 2014 | FJSP    | DFA + LS                     |
| Li et al. [50]         | 2014 | FJSP with fixed maintenance | PSO + TS       |
| Rohaninejad et al. [68]| 2015 | FJSP with machine capacities | GA + ELECTRE   |
| Yuan and Xu [83]       | 2015 | FJSP    | GA + LS                      |

et al. [68] proposed a nonlinear IP model and hybridized GA with ELECTRE, which is a multi-attribute decision making method, for multi-objective FJSP with machines capacity constraints. The computational results obtained by well-known multi-objective algorithms from the literature showed that the proposed algorithm has a better performance especially in the closeness of the solutions to the Pareto optimal front.

The particle swarm optimization algorithm is another widely used approximation method in FJSP literature. Xia and Wu [78] combined simulated annealing
with particle swarm optimization to avoid becoming trapped in a local optimum for multi-objective FJSP. Grobler et al. [34] developed four particle swarm optimization based heuristic approaches for multi-objective FJSP with sequence-dependent setup times. The priority-based particle swarm optimization algorithm executed the best performance with respect to the quality of the solution and the computational complexity. Zhang et al. [85] hybridized the particle swarm optimization algorithm with tabu search algorithm, which was used as the local search strategy for every particle, for the multi-objective FJSP. Moslehi and Mahnam [58] combined the particle swarm algorithm and a local search algorithm for multi-objective FJSP with different release times. Shao et al. [71] developed an hybrid methodology, which uses discrete particle swarm optimization for global search and simulated annealing for local search, for multi-objective FJSP. Li et al. [46] and Li et al. [48] proposed hybridized artificial bee colony algorithm, which is a novel particle swarm methodology, for solving the multi-objective FJSP. Li et al. [50] combined artificial bee colony algorithm with tabu search for the multi-objective FJSP with maintenance activities. They used a self-adaptive strategy for producing neighboring solutions in different promising regions and designed an external Pareto archive set to record the obtained non-dominated solutions.

Many studies developed methodologies based on the tabu search algorithm, because the tabu search algorithm exhibits an effective local search. Schrich et al. [70] performed two methodologies, a hierarchical and multi-start tabu search, using priority dispatching rules to obtain the initial solution. Fattahi et al. [23] compared integrated and hierarchical solution approaches for FJSP. The computational results showed that the hierarchical algorithms have better performance than the integrated approaches. Among six different hybrid searching methodologies, including searching approach and heuristics, the algorithm combining the tabu search and simulated annealing algorithms for assignment and sequencing subproblems consecutively performs better than the other algorithms. Bozejko et al. [9] developed a two module algorithm consisting of a machine selection and the operation scheduling for solving FJSP. Tabu search was performed for the machine selection module. The insertion algorithm and the tabu search algorithm with backtracking, which were taken from the literature, were applied for the operation scheduling module. It is inferred that the exact algorithms can be used on both modules to obtain an optimal solution. Li et al. [47] developed a hybrid algorithm with two modules to handle the multi-objective FJSP. In the machine assignment module, a tabu search algorithm was used to produce neighboring solutions while a variable neighborhood search algorithm is applied to local search in the operation scheduling module. Liouane et al. [51] combined ant colony optimization metaheuristics with local search methods, which contain tabu search, and showed the efficiency of the local search methods with an ant system approach. Baykasoglu et al. [6] used grammar-form linguistics to represent data and the dispatching rules to order the operations. Baykasoglu [5] used a metaheuristic method based on simulated annealing for optimization while Baykasoglu et al. [6] used the tabu search algorithm for multi-objective optimization. Both studies considered process plan selection problem besides assignment and sequencing problems. Jia and Hu [40] proposed a path-relinking algorithm based on the tabu search algorithm with back-jump tracking to solve the multi-objective FJSP.

Wang et al. [75] developed a filtered beam search based heuristic algorithm to solve multiple-objective FJSP. They used dispatching rules based heuristics as local
and global evaluation functions to avoid useless paths and decrease the computational time. Ziaee [87] developed a heuristic algorithm for FJSP minimizing the makespan which can obtain high-quality solutions in very short time. Rajkumar et al. [67] proposed a GRASP algorithm to solve the multi-objective FJSP with limited resource constraints. Karimi et al. [43] proposed knowledge-based variable neighborhood search algorithm which searches the solution space to find good solutions, extracts the knowledge of good solution and feed it back to the algorithm. Mousakhani [59] proposed a MIP model and a metaheuristic algorithm based on iterated local search for the FJSP with sequence dependent set-up times to minimize total tardiness. The proposed model obtains better results in terms of both size and computational complexities than the model from the literature. Roshanaei et al. [69] developed two MIP models for FJSP to minimize total makespan. To solve the large size instances, a metaheuristic hybridizing artificial immune and simulated annealing algorithms was developed. The proposed hybrid algorithm was compared with the others from the literature using benchmark instances and an industrial problem of a mould and die manufacturing company. Yuan et al. [84] proposed a hybrid harmony search algorithm integrated with a local search procedure for solving the FJSP to minimize total makespan. Furthermore, Yuan and Xu [82] combined hybrid harmony search with large neighborhood search and Yuan and Xu [81] proposed hybrid differential evolution algorithms for the same problem. Gao et al. [28] used a discrete harmony search algorithm to solve the multi-objective FJSP. They used local search to initialize the harmony memory and enhance the exploitation capability. Karthikeyan et al. [44] combined hybridized discrete firefly algorithm with local search to enhance the exploitation capability.

In the inspected FJSP literature, the priority-based GA approach proposed in this study has not been encountered. A constructive algorithm which can generate active schedules for FJSP is used for decoding. Furthermore, iterated local search (ILS) algorithm is hybridized with GA to improve the effect of local search.

4. The priority-based genetic algorithm. In this study, the effect of a priority-based representation on the performance of GA is surveyed for FJSP. Before the main distinctive features of the algorithm is explained in further subsections, the general framework of the algorithm is represented. In the proposed GA, the initial population is generated randomly. A mating pool including the parents, which are used to generate offspring, is constituted by selection. A number of the best solutions are reserved for the mating pool by using the elitist rule. Elitism is used to maintain the best solution to satisfy monotonical improvement. The rest of the individuals in the mating pool are determined by roulette wheel selection [32]. After the mating pool is constructed, the randomly selected parents are matched to generate offspring. The number of offspring which is generated by individuals from the mating pool is determined by crossover probability. For each individual in the offspring pool, a mutation operator is applied with mutation probability. The ILS procedure is applied for the best individuals. If better individuals are obtained by ILS, the original chromosomes are replaced with them. Pairwise comparison between the individuals in the current population and the offspring is performed to construct the next generation. The chromosome having the better fitness value is transferred to the next generation and the other goes to the back-up matrix. The algorithm is terminated when the maximal number of generations is reached. The pseudocode of the proposed GA is given at Algorithm 1. The main components of
the proposed algorithm (representation, constructive algorithm, genetic operators and iterated local search) are explained in the following subsections.

**Algorithm 1** Priority Based Genetic Algorithm

\begin{verbatim}
P ← Generate_Initial_Population()
Evaluate(P)

while termination conditions not meet do
  if same fitness values occur in P then
    apply Immigrate
  end if
  P' ← Select(P)
P'' ← Crossover(P')
P'' ← Mutate(P'')
P'' ← ILS(P'')
P ← Pairwise_comparison(P, P'')
end while
\end{verbatim}

4.1. **Representation.** The priority-based representation used by Oliveira et al. [61] for JSP has an $n$-to-1 relation between corresponding chromosomes and schedules. In this study, permutation coding is performed to reduce the number of alternative chromosomes which can be used to represent the same schedule. Even if the number of alternative chromosomes representing a schedule is still high with permutation coding, priority-based representation can be effective for an FJSP problem because it uses a very simple constructive algorithm to generate an active schedule.

In the proposed priority-based GA, the number of genes of each chromosome is equal to the “total number of alternative operations of all jobs”. Each gene is represented by a permutation of the integers $0, 1, \ldots, \text{the total number of processable operations}-1$. A constructive algorithm is used to decode and evaluate the solutions.

A sample chromosome for a $2 \times 2$ FJSP is seen as follows. The data for this example is given in Tables 4 and 5. Each gene refers the priority value of an operation. For example, the priority degree for $O_{11}$ is 5 on $M_1$ and 3 on $M_2$. A higher priority value implies a higher priority during the scheduling process performed by the constructive algorithm which will be discussed in the following section.

**Table 4.** Alternative machines for a small FJSP example.

| Jobs | 1  | 2  |
|------|----|----|
| $J_1$ | $M_1, M_2$ | $M_2$ |
| $J_2$ | $M_1, M_2$ | $M_1, M_2$ |

chromosome = $[ O_{111} \ O_{112} \ O_{121} \ O_{122} \ O_{211} \ O_{212} \ O_{221} \ O_{222} ]$

4.2. **Constructive algorithm.** Although all active schedules can be generated by GTA for JSP, the same argument has not been valid for FJSP. A constructive algorithm which is a modified version of the GTA [31] is used in this study. The algorithm can generate all active schedules for FJSP.
Table 5. Duration for a small FJSP example.

| Operations | $M_1$ | $M_2$ |
|------------|-------|-------|
| $O_{11}$   | 3     | 2     |
| $O_{12}$   | -     | 5     |
| $O_{21}$   | 5     | 3     |
| $O_{22}$   | 6     | 8     |

Let $numop$ be the total number of operations. The proposed algorithm consists of the number of $numop$ iterations. In each iteration, according to the precedence and resource constraints, an operation $O_{ij}$ is assigned to a machine $k$, where $k \in M_{ij}$. In each iteration $a$, the set of assignable operations $S_a$ and the set of assigned operations $P_a$ are determined. Earliest starting ($s_{ijk}$) and completion ($c_{ijk}$) times are computed for each operation $O_{ijk} \in S_a$ satisfying both the precedence and resource constraints. The minimum earliest completion time is determined as $c^* = \min \{c_{ijk} | O_{ijk} \in S_a \}$. A conflict set, $CS_a$, is constituted at each iteration where $s_{ijk} < c^*$ for all $O_{ijk} \in CS_a$. The operation to be assigned is chosen from set $CS_a$ with respect to the priority values on the chromosome. Operation $O_{i^*j^*} \in CS_a$ which has the highest priority value is assigned on machine $k^*$. The starting and completion time of possible subsequent operations are computed. At the end of each iteration, the sets $S_a$ and $P_a$ are updated. The algorithm proceeds until no operations remained to be scheduled. At the end of the algorithm, the makespan of the schedule is determined by finding the maximum completion time among the last operations of each job. The pseudocode for the constructive algorithm is given in Algorithm 2.

This algorithm was firstly developed by Chang and Sullivan [14]. Nascimento [60] performed the same algorithm by generating all partial schedules at each iteration. In order to ensure the comprehensibility of the algorithm, an active schedule for the 2×2 FJSP, given above, is found by this algorithm. Figure 2 shows the iteration of the constructive algorithm that generates the schedule of the chromosome given in Section 4.1.

4.3. Genetic operators. Two well known crossover operators from the literature are used in this study: cycle crossover (CX) and job-based order crossover (JOX).
Algorithm 2 Constructive Algorithm for FJSP

\begin{verbatim}
input: FJSP data, chromosome
begin
    S_0 = \{O_{i1k} | k \in M_{i1}, \forall i,j\}, P_0 = \emptyset
    for a \leftarrow 1 \text{ to } numop do
        c^* = \min \{c_{ijk} | O_{ijk} \in S_{a-1}\}
        Determine conflict set
        CS_a = \{O_{ijk} | s_{ijk} < c^*, O_{ijk} \in S_{a-1}\}
        the operation with highest priority is selected
        O_{i^*j^*k^*} \leftarrow \arg\min \{\text{chromosome}_{ijk} | O_{ijk} \in CS_a\}
        if j^* < n_i^* then
            calculate s_{i^*,j^*+1,k} \forall k
            c_{i^*,j^*+1,k} = \max \{c_{i^*,j^*,k}, \max_{ij} \{c_{ijk}\}\} \forall k
            calculate c_{i^*,j^*+1,k} \forall k
            c_{i^*,j^*+1,k} = s_{i^*,j^*+1,k} + p_{i^*,j^*+1,k} \forall k
            update S_a
            S_a \leftarrow S_{a-1} \{O_{i^*,j^*,k} | k \in M_{i^*,j^*}\}
            S_a \leftarrow S_a \cup \{O_{i^*,j^*+1,k} | k \in M_{i^*,j^*}\}
        else
            S_a \leftarrow S_{a-1} \{O_{i^*,j^*,k} | k \in M_{i^*,j^*}\}
        end if
        update P_a
        P_a \leftarrow P_{a-1} \cup \{O_{i^*,j^*,k}\}
    end for
C_{max} \leftarrow \max_i \{c_{in_i}\}
end
output: an active schedule
\end{verbatim}

CX is one of the crossover operators used for path representation in the travelling salesman problem. In this study, it is applied for FJSP with priority-based permutation coding. An offspring generation using CX is illustrated in Figure 3. In CX, each gene is taken at the same position from one of its parents. The first offspring is generated by taking the first gene from the first parent. The next gene is 2 from the second parent that is just below the 1. Since the gene is at position 5, the fifth gene takes 2. Following this rule, 4 and 7 are placed at position 3 and 7, respectively. The cycle is completed with 1 which is below the gene with 7. The remaining genes are filled by the other parent. By the same way, the second offspring is found as in Figure 3. CX satisfies the inheritance of the absolute gene position [55].

JOX was developed by Ono et al. [62] for JSP to satisfy the inheritance from parents. They assigned the order of each job on all machines as a characteristic that children inherit from parents. Since this characteristic is indigenous to JSP, JOX can be considered as a problem based crossover operator. Ono et al. [62] claimed that JOX successfully satisfies characteristics-preserving because it can preserve the order of each job on all machines properly between parents and their children. Since the offspring generated by JOX may not be a feasible solution, a GTA based algorithm was used to transform an offspring into an active schedule.

In this study, as in Ono et al. [62], the priority order of each job on all machines is taken into account as a characteristic to be inherited. After the inherited jobs
are determined, the first offspring takes the inherited genes from the first parent. Among the rest of the genes, the ones that do not conflict are taken from the other parent. A repair mechanism is not needed after the implementation of JOX because of the proposed representation structure. Figure 4 shows an illustrative example for JOX. The order of operations is preserved by considering the value of priority as the inherited characteristic.

Since the assignment of the operations is not altered during crossover, three mutation operators are proposed for global search and reassignment of the operations: machine mutation, sequence mutation, and immigration. Machine mutation is used

| Parent 1 | Parent 2 |
|----------|----------|
| $M_1$    | $M_1$    |
| $O_{21}$ | $O_{21}$ |
| $O_{22}$ | $O_{32}$ |
| $O_{31}$ | $O_{33}$ |
| $M_2$    | $M_2$    |
| $O_{23}$ | $O_{23}$ |
| $O_{24}$ | $O_{34}$ |
| $M_3$    | $M_3$    |
| $O_{25}$ | $O_{25}$ |
| $O_{35}$ | $O_{35}$ |

**Figure 3. Illustration of CX**

| Offspring 1 | Offspring 2 |
|-------------|-------------|
| 1 4 2 7 6   | 2 3 5 4 6 1|

| Parent 1 |
|----------|
| $M_1$ |
| $O_{32}$ |
| $O_{33}$ |
| $M_2$ |
| $O_{23}$ |
| $O_{24}$ |
| $M_3$ |
| $O_{35}$ |

| Parent 2 |
|----------|
| $M_1$ |
| $O_{21}$ |
| $O_{31}$ |
| $M_2$ |
| $O_{22}$ |
| $O_{32}$ |
| $M_3$ |
| $O_{25}$ |

**Figure 4. Illustration of JOX with Gantt Charts**
to replace the operations on alternative machines. If the operation of which the position on the schedule is changed is not on critical path, then the makespan remains. In other words, to change the makespan, we need to alter the position of the operation on the critical path. Because of this reason, a mutation operator based on the changing the position of a randomly selected operation from the critical path on a selected machine is applied in this study.

In order to improve the neighborhood search, a mutation operator, called sequence mutation, based on the block structure is applied. It is used to relocate an operation on the current machine in two ways: (1) with 50% probability, a randomly selected operation can be processed before the first operation of the corresponding block. (2) the selected operation can be processed after the last operation of the corresponding block with probability 0.5. All neighborhoods of a solution can be generated by machine mutation and sequence mutation.

Immigration is a mutation operator for global search which is performed to enhance diversity in the gene pool by replacement of at least one chromosome by one randomly generated at each generation. In this study, immigration is used to prevent recurrences in the population. During the implementation of the proposed algorithm, a back-up matrix is defined beside the population matrix. If a parent is not transferred to the next generation, it passes through the back-up matrix. This matrix is used for individuals having the same fitness values. In each population, if the number of individuals having the same fitness value is greater than a predetermined number, the genes of an individual appearing after this number are replaced with those of the individual from the back-up matrix. If the parent in the back-up matrix and the offspring in the current population have the same fitness value, the genes of the offspring are replaced with a randomly generated chromosome. All these mutation operators can generate a solution with a better or worse fitness value than the mutated chromosome.

4.4. Iterated local search. The ILS algorithm iteratively applies a local search procedure to the perturbation of a solution. Since it is simple to implement and effective to obtain improved solutions [3], it provides a basis for several novel algorithms developing for challenging problems such as the quadratic assignment problem [72], [86], JSP [21], the single machine scheduling problem [18], the vehicle routing problem [35],[73], the graph coloring problem [63],[12] and the logistics network design problem [17].

In the proposed GA, ILS is applied to the chromosomes at the end of each reproduction process. Because of the computational effort, ILS is carried out on only a few individuals having the best fitness values in the population. The general procedure of the ILS algorithm performed in this study is given in Algorithm 3. It consists of four basic operators as follows:

- **Initial solution generation**: A good initial solution can be important if high-quality solutions are to be obtained as fast as possible. In the literature, a random generation procedure or a greedy construction heuristic have been used to obtain an initial solution [52]. In this study, the best individuals of each generation constitute initial solutions of ILS. Let $a$ be the predetermined number which refers the number of best individuals in each generation. In the proposed GA, the number of $a$ ILS procedures is applied at each generation. An individual belonging to the best $a$ individuals becomes the initial solution of ILS.
Local search: The aim of local search is to improve the current solution through seeking the neighborhoods. At each iteration, it searches the neighborhoods of the current solution to find a better solution. If a better solution is found, then it becomes the current solution. This iterative process continues until no better solution is found. When the algorithm terminates, the current solution is called the local optimum. In this study, two neighborhood structures are used. These structures are based on the same idea with machine and sequence mutations. An operation is removed from its original position and inserted on a machine. If the reassigned machine is the same as the original solution, then the operation is inserted at a different position on that machine. Since the makespan cannot be changed with the insertion of an operation which is not on the critical path, only the operations belonging to the critical path are considered during local search. Local search is applied until an improved solution is not found.

Perturbation: Perturbation is applied to escape from a local optimal solution [53]. After the local search procedure, the local optimal solution is perturbed by reassignment of a number of operations to different machines. The perturbation procedure exchanges $k$ randomly chosen operations corresponding to $k$ random insertions. The number of $k$ is determined by testing several fixed values. If the value of $k$ is too big, ILS may behave like a random restart, so better solutions will only be found with a very low probability. On the other hand, if it is too small, the local search will often fall back into the same local optimum and diversification of solution space will reduce [52]. According to the preliminary tests, the assignments and positions of three operations are changed during perturbation in this study. After the current optimal solution is perturbed, a new local search procedure is started with the aim of obtaining a better local optimum.

Acceptance criteria: The new local optimum is accepted if it is better than the current solution.

Algorithm 3 Iterated Local Search

\[
\begin{align*}
X_0 & \leftarrow \text{Initial Solution()} \\
X & \leftarrow \text{LocalSearch}(X_0) \\
\text{while acceptance criteria not achieved and all neighbors not searched do} & \\
& \quad X' \leftarrow \text{LocalSearch}(X) \\
& \quad X'' \leftarrow \text{Perturbation}(X') \\
\text{end while}
\end{align*}
\]

5. Computational results. The widely used benchmarking data sets are used to evaluate the proposed methodology. Because of the accessibility and frequency of usage in the literature, Kacem data [41, 42], BR data [10], BC data [13], and HU data [37] are preferred for measuring the performance of the methodology developed in this study. In order to obtain meaningful results, the number of runs fulfilled for each instance is determined according to the studies of which the results are used to compare.

The proposed GA was coded in Microsoft Visual C++ Version 7.0. The computational tests were performed on a portable workstation with a 1.73 GHz Intel Core i7 processor and 4 Gb of RAM. The parameters of the proposed GA are determined
Table 6. Parameters of proposed GA

| Parameters               | Value     |
|-------------------------|-----------|
| Population size         | 100       |
| Selection               | Elitist Rule (2%), Roulette Wheel (98%) |
| Crossover               | CX (50%), JOX (50%) |
| Mutation                | Machine (50%), Sequence (50%) |
| Crossover probability   | 0.5       |
| Mutation probability    | 0.5       |
| Number of generations   | 1000      |

by computational experiments and given in Table 6. In FJSP literature, although the population size ranges from 100 to 3000, most of the studies determine the population size to be smaller than 300. In this study, the population size is taken as 100 for each instance. Two individuals are selected by the elitist rule for the mating pool while the rest of them are determined by roulette wheel selection. If two individuals are selected for crossover, there is 0.5 probability of applying each of the operators (CX and JOX). This implementation is also the same for the mutation operators. Each individual is subjected to either crossover or mutation with the probability of 0.5. The algorithm terminates when the solution come up to the lower bound or the number of iterations reaches 1000. Relative deviation (RD) is used to compare the results of the proposed algorithm with the algorithms in the literature. RD is obtained as follows [2]:

$$RD = \frac{C_{max}^*(Alg) - C_{max}^*(pbGA)}{C_{max}^*(Alg)} \times 100\% \quad (1)$$

where $C_{max}^*(pbGA)$ is the best makespan obtained by the proposed priority-based GA (pbGA) and $C_{max}^*(Alg)$ is the best makespan of the algorithm that is used to compare with pbGA.

Due to technological differences between the computers used in the studies, comparison with respect to CPU time does not produce meaningful results. Therefore, some studies have not given the CPU times of their proposed algorithms. Moreover, the code could be enhanced by using the distributed computation or parallel computation to make better use of the cores of the processor. For these reasons, the computational time performance of the proposed algorithm is not compared with other algorithms in this study. The number of evaluations in each run is compared with the other studies to evaluate the computation performance. Although the number of evaluations would not give us the exact computational time, it allows us to compare different algorithms with respect to computer independent computation effort.

**Kacem data set:** Two instances from Kacem et al. [41],[42], ka08 and ka10, are used to test the pbGA and to compare it with the results from the literature. Problem ka08 is a partial flexibility instance where some operations can be processed on a subset of the machine set. It is formed by eight jobs with 27 operations and its optimal makespan is 14. Problem ka10 is a total flexibility instance where each operation can be processed by all machines. It contains 10 jobs with 30 operations and its optimal makespan is 7.

Table 7 compares the result of the pbGA with that of the algorithms developed by Gao et al. [27] and Amiri et al. [2]. 5 runs are fulfilled for this set of instances.
as the compared studies performed. The first column represents the name of the instance and the second column characterizes the size of the problem, in which $n$ refers to the number of jobs and $m$ embodies the number of given machines in the problem. In order to represent the size of the instances, the number of genes for each instance with respect to the proposed representation schema is given as “genes” in the third column. The fourth and fifth columns represent the best makespans resulting from Gao et al. [27] and Amiri et al. [2], respectively. The sixth column signifies the best makespan among the results of 5 runs for the pbGA, and the seventh column stands for the CPU time in terms of seconds for a run in average. The results show that the proposed algorithm has achieved the optimal solution for both instances in a very good computational time. Since the result of each run is equal to the optimal makespan, other explanatory results, such as average or the most recurring makespan, are also the same as the optimal makespan. Thus, it can be concluded that the proposed algorithm is robust for this data set.

Table 7. Results of priority based GA for Kacem data set

| inst. | $n \times m$ | genes | [27] | [2] | $C_{max}$ | CPU |
|-------|--------------|-------|------|-----|----------|-----|
| ka08  | $8 \times 8$ | 175   | 14   | 14  | 14       | 4.70|
| ka10  | $10 \times 10$ | 300   | 7    | 7   | 7        | 0.46|

Gao et al. [27] developed a hybridized GA with variable neighborhood search. The results obtained by proposed GA and Gao et al. [27] are the same for Kacem data set. In each run, the proposed GA requires 100,000 evaluations where the number of evaluations is figured out by multiplying the population size and the number of generation. Also, the local search, which is applied only for few chromosomes at each generation, takes approximately 70% of total computational time. In Gao et al. [27], the total number of evaluations in each run is 60,000 for this data set. Also, variable neighborhood search is performed for each offspring before it is inserted into the population with 300 chromosomes. Although the number of evaluations is less than the proposed GA, since the number of individuals applied local search is higher, there is no evidence to determine the most efficient algorithm with respect to computation effort.

**BR data set:** 10 problems with medium flexibility are taken from Brandimarte [10]. They were randomly generated using a uniform distribution. The number of jobs ranges from 10 to 20, the number of machines ranges from 4 to 15, the number of operations for each job ranges from 5 to 15 and the number of operations for all the jobs ranges from 55 to 240.

Table 8 indicates the result of pbGA for the BR data set. The first column represents the name of the instance, the second characterizes the size of the problem, the third column gives the flexibility index which refers the average number of equivalent machines per operation and the fourth column gives the number of genes for each instance. The fifth column refers the lower bound and upper bound which are indicated by LB and UB, respectively. They have the same value if the optimum makespan is known; otherwise they report the best lower bound and upper bound found up till now. From the sixth to eighth columns represent the explanatory results of the proposed algorithm for 5 runs. Similarly, the results for 10 runs are placed from the ninth to eleventh column. “min”, “av.” and “max” refers to the minimum, average and maximum values found by the proposed GA over 5 or 10
Table 8. Results of pbGA for BR data set

| instance | n x m | flex. genes (LB, UB) | \(C_{\text{max}}\)^a | \(C_{\text{max}}\)^b |
|----------|-------|----------------------|----------------------|----------------------|
|          |       |                      | min | av. | max | min | av. | max | CPU |
| Mk01     | 10x6  | 2.09 | 115 (36, 40) | 40 | 40 | 40 | 40 | 40 | 40 | 26.26 |
| Mk02     | 10x6  | 4.1  | 238 (24, 26) | 28 | 28.8 | 29 | 27 | 28.3 | 29 | 87.21 |
| Mk03     | 15x8  | 3.01 | 451 (204, 204) | 204 | 204 | 204 | 204 | 204 | 204 | 0.265 |
| Mk04     | 15x8  | 1.91 | 172 (48, 60) | 60 | 60.4 | 62 | 60 | 60.7 | 62 | 59.27 |
| Mk05     | 15x4  | 1.71 | 181 (168, 172) | 174 | 175.2 | 177 | 173 | 175.5 | 177 | 104.12 |
| Mk06     | 10x15 | 3.27 | 490 (33, 57) | 63 | 64.6 | 66 | 63 | 64.7 | 67 | 414.86 |
| Mk07     | 20x5  | 2.83 | 283 (133, 139) | 144 | 146.4 | 149 | 142 | 145.6 | 149 | 288.04 |
| Mk08     | 20x10 | 1.43 | 322 (523, 523) | 523 | 523 | 523 | 523 | 523 | 523 | 0.541 |
| Mk09     | 20x10 | 2.53 | 606 (299, 307) | 307 | 307.8 | 311 | 307 | 307.4 | 311 | 465.47 |
| Mk10     | 20x15 | 2.98 | 716 (165, 196) | 211 | 220.6 | 226 | 211 | 221.5 | 226 | 437.19 |

\(^a\) Obtained over 5 runs
\(^b\) Obtained over 10 runs

runs, respectively, which are considered in determining whether the algorithm is robust. Finally, the average computational time obtained over 10 runs is given as “CPU”. According to the minimum, maximum and average of 5 and 10 runs, the pbGA achieves robust results for all instances (except instance “Mk10”) in a reasonable time. The bold numbers represent the instances for which the upper bound value is found by the proposed algorithm. “Mk03” and “Mk08” are the instances for which the optimum solution is known. The pbGA reach the optimum values (lower bounds) of these instances in a very short time.

The RDs of the pbGA according to the best results of the other algorithms are reported in Table 9 and 10 in which the RD values are obtained over 5 runs and 10 runs, respectively. The number of runs is determined according to the compared studies. The studies in Table 9 obtained the minimum makespan over 5 runs. The last row refers the average RD for each algorithm. According to Table 9, the results of the other instances obtained over 5 runs reflect that the pbGA obtains better results for the makespan than the algorithm developed by Ennigrou and Ghedira [20]. On the other hand, Mastrolilli and Gambardella [54], Gao et al.[27], Pezzella et al.[65], Amiri et al. [2], Yazdani et al. [80], Yuan and Xu [82] and Driss et al. [19] have better results for the makespan than pbGA.

According to the comparison over 10 runs (Table 10), the pbGA finds better results than the algorithms of Chen et al. [15] and Jia et al. [39] for all instances but “Mk06”, and Xing et al. [79] for all instances but “Mk01”. The number of instances for which the pbGA finds better results is more than the algorithms of Chen et al. [15], Jia et al. [39] and Xing et al. [79]. So, it can be concluded from Table 9 and Table 10 that the pbGA has better performance than four of the algorithms with respect to the minimum makespan. On the other hand, the pbGA is dominated by some of the algorithms from the literature for the BR instances. According to the RD average, the algorithms with better results have performed with, at most, 3% higher performance.

In BRdata set, the worst RD values are obtained in the comparison with Gao et al.[27]. Gao et al.[27] used a hybrid GA in which the number of evaluations ranges from 60,000 to 600,000 (for pbGA it is 100,000) for BRdata and BCdata sets. Also, a variable neighborhood search is applied for every offspring at each population. So, in terms of the number of evaluation, the proposed algorithm finds the corresponding results in less computational effort than Gao et al.[27]. Moreover, the GA developed
Table 9. Comparison of algorithms with respect to RD for BR data set (5 runs)

| Inst. | [54] | [27] | [20] | [65] | [2] | [80] | [82] | [19] |
|-------|------|------|------|------|----|------|------|------|
| Mk01  |  0   |  0   |  0   |  0   |  0 |  0   |  0   | -8.11|
| Mk02  | -7.69| -7.69| 12.50| -7.69| -7.69| -7.69| -7.69|
| Mk03  |  0   |  0   |  0   |  0   |  0 |  0   |  0   |  0   |
| Mk04  |  0   |  0   | 10.45|  0   |  0 |  0   |  0   |  0   |
| Mk05  | -0.58| -1.16|  7.45| -0.58| -0.58| -0.58| -1.16| -0.58|
| Mk06  | -8.62| -8.62| 25.88|  0   | -6.78| -5  | -8.62|  5.97|
| Mk07  |  0   | -3.60|  6.49| -3.60| -2.86| -2.13| -3.60|  2.70|
| Mk08  |  0   |  0   |  0   |  0   |  0 |  0   |  0   |  0   |
| Mk09  |  0   |  0   |  0   |  1.29|  0 |  0   |  0   |  0   |
| Mk10  | -6.57| -7.11|  0   | -0.47| -1.93| -1.44| -6.57|  0.47|
| Av. RD| -2.35| -2.82|  8.97| -1.01| -1.98| -1.68| -2.76| -0.72|

Table 10. Comparison of algorithms with respect to RD for BR data set (10 runs)

| Inst. | [4]  | [15] | [39] | [79] |
|-------|------|------|------|------|
| Mk01  |  0   |  0   |  0   | -2.56|
| Mk02  | -3.85|  6.90|  3.57|  6.90|
| Mk03  |  0   |  0   |  0   |  0   |
| Mk04  |  0   |  4.76|  1.64|  7.69|
| Mk05  |  0   |  4.42|  1.70|  0   |
| Mk06  |  0   | -5.00| -1.61|  5.97|
| Mk07  | -1.43|  4.05|  2.07|  1.39|
| Mk08  |  0   |  0   |  0   |  0   |
| Mk09  |  1.60|  0.32|  0.97|  1.29|
| Mk10  |  1.40|  0.47|  2.31|  7.86|
| Av. RD| -0.23|  1.59|  1.07|  2.85|

by Pezzella et al. [65] obtains better result that the proposed algorithm. The number of total evaluations in each run is 5,000,000 (for pbGA it is 100,000) and the initial population is generated using the approach by localization.

Bagheri et al. [4] obtained better results than proposed GA over 10 runs. They proposed an artificial immune algorithm in which 7,500,000 evaluations (for pbGA it is 100,000) occur at each run for BR data set. So, although the minimum makespans attained over 5 or 10 runs are worse, the computational effort to obtain these results seems less than these algorithms.

**BC data set:** 21 problems are taken from Chambers and Barnes [13], in which the number of jobs ranges from 10 to 15, the number of machines ranges from 11 to 18, the number of operations for each job ranges from 10 to 15 and the number of operations for all the jobs ranges from 100 to 225.

The lower bounds of all instances computed in a straightforward way are available and they are generally not very close to the optimum. Table 11 provides a
The computational results show that the pbGA achieves better results than the hybrid algorithm of Gao et al. [27], which combines the GA and the variable neighborhood descent algorithm. The number of better results found by pbGA and Mastrolilli and Gambardella [54] are much the same and the average RD is 0.02%. The most important contribution of the pbGA is that two new better upper bounds for “setb4xy” and “seti5c12” are found.

HU data set: This data set contains 195 problems from Hurink et al. [37]. The number of jobs ranges from 6 to 30, the number of machines ranges from 5 to 15, the number of operations for each job ranges from 5 to 15 and the number of operations for all the jobs ranges from 36 to 300. A data set for classical JSP from Hurink et al. [37] is adapted to FJSP as edata, rdata and vdata by enlarging the set of alternative machines for every operation. Flexibility is one of the complexity indices for FJSP which represent the average number of equivalent machines per operation. Flexibility is one of the complexity indices for FJSP which represent the average number of equivalent machines per operation. In edata, few operations are assigned to more than one machine with flexibility ranging from 1.13 to 1.21. The rdata set contains the instances in which most of the operations may be assigned to some machines, with flexibility between 1.88 and 2.08. In the HU data, the vdata set has the highest flexibility ranging from 2.09 to 6.70, where all operations may be assigned to several machines. The largest instance has 341 genes for edata, 592 for rdata and 1994 for vdata.

The best results found over all 12 runs for pbGA and ILOG-CP (ILOG Constraint Programming) [8] are compared in Table 12 with respect to RD. Since there are 195 instances in the HU data set, only the summary results table is shown. The first

Table 11. Results of algorithms for BC data set

| inst. | n × m | flex. | genes (LB, UB) | min | av. | max | CPU | RD |
|-------|-------|-------|----------------|-----|-----|-----|-----|-----|
| mt10c1 | 10x11 | 1.1   | 110 (655, 927) | 927 | 929 | 934 | 69.31 | 0.11 | 0  |
| mt10cc | 10x12 | 1.2   | 120 (655, 910) | 910 | 910.4 | 929 | 73.74 | 0 | 0  |
| mt10x  | 10x11 | 1.1   | 110 (655, 918) | 918 | 920.6 | 931 | 68.00 | 0 | 0  |
| mt10xxx | 10x12 | 1.2   | 120 (655, 918) | 918 | 921.8 | 929 | 79.80 | 0 | 0  |
| mt10xy | 10x12 | 1.2   | 120 (655, 905) | 905 | 906 | 908 | 86.04 | 0.11 | 0  |
| mt10xyz | 10x13 | 1.3   | 130 (655, 847) | 849 | 854.8 | 863 | 88.22 | -0.24 | 0  |
| setb4c9 | 15x11 | 1.1   | 165 (857, 914) | 914 | 922.6 | 927 | 113.24 | 0.54 | 0  |
| setb4cc | 15x12 | 1.2   | 180 (857, 909) | 909 | 912 | 914 | 126.92 | 0 | 0.55 |
| setb4x  | 15x11 | 1.1   | 165 (846, 925) | 925 | 928.6 | 932 | 113.74 | 0 | 0  |
| setb4xx | 15x12 | 1.2   | 180 (846, 925) | 925 | 934 | 944 | 133.50 | 0 | 0  |
| setb4xxx | 15x13 | 1.3   | 195 (846, 918) | 925 | 928.8 | 933 | 140.34 | 0 | 0  |
| setb4xy | 15x12 | 1.2   | 180 (845, 916) | 912* | 913.8 | 915 | 147.42 | 0.44 | 0.44 |
| setb4xyz | 15x13 | 1.3   | 195 (838, 905) | 905 | 907.8 | 909 | 158.44 | 0 | 0  |
| seti5c12 | 15x16 | 1.07  | 240 (1027, 1174) | 1172* | 1186 | 1197 | 176.85 | 0.17 | 0.26 |
| seti5cc | 15x17 | 1.13  | 255 (955, 1136) | 1136 | 1145.4 | 1154 | 208.40 | 0 | 0.18 |
| seti5x  | 15x16 | 1.07  | 240 (955, 1201) | 1204 | 1216 | 1228 | 193.46 | -0.25 | 0  |
| seti5xx | 15x17 | 1.13  | 255 (955, 1199) | 1199 | 1209.4 | 1232 | 192.63 | 0 | 0.25 |
| seti5xxx | 15x18 | 1.2   | 170 (955, 1197) | 1199 | 1207.4 | 1223 | 207.63 | -0.17 | 0.42 |
| seti5xy | 15x17 | 1.13  | 255 (955, 1136) | 1136 | 1138.8 | 1146 | 217.357 | 0 | 0  |
| seti5xyz | 15x18 | 1.2   | 270 (955, 1125) | 1128 | 1134 | 1145 | 232.66 | -0.27 | -0.18 |
three rows indicate the minimum, maximum and average results of the makespans obtained over 12 runs. The fourth row states the number of instances having RD with 0, that is, the number instances in which the pbGA finds the same results with ILOG-CP. The fifth row states the number of instances having a positive RD, that is, the number instances in which the pbGA finds better results than ILOG-CP. The sixth row gives the number of instances in which the pbGA obtains worse results than ILOG-CP.

Table 12. Comparison of algorithms with respect to RD for HU data set

|        | edata | rdata | vdata |
|--------|-------|-------|-------|
| min RD | -1.98 | -2.16 | -5.20 |
| max RD | 5.76  | 2.23  | 0.22  |
| av. RD | 0.43  | 0.01  | -0.27 |
| # of 0 | 30    | 22    | 47    |
| # of better | 26 | 26 | 5 |
| # of worse | 9 | 17 | 13 |

Comparing the pbGA with ILOG-CP [8] with respect to the minimum makespan, in 65 test instances of the edata set, 26 better results are obtained by pbGA while in only nine cases the best solution values of pbGA are worse. Finding better results in the edata set seems to be more difficult than in the rdata and vdata sets because some instances in the edata set have poor lower bound quality [54].

Although the results are not as good as for the edata set, the pbGA obtains better results than ILOG-CP for the rdata set. In 65 test instances of the rdata set, 26 better results are obtained by the pbGA while in 17 cases the best solution values of the pbGA are worse than ILOG-CP. On the other hand, ILOG-CP has better performance for the vdata set. In 65 test instances of the vdata set, 5 better results are obtained by the pbGA while in 13 cases the best solution values of the pbGA are worse than ILOG-CP. The vdata set has better lower bounds than the edata and rdata sets.

6. Conclusions and further suggestions. In this study, a pbGA approach is developed to obtain a near optimum solution for FJSP. The GA is the most widely used metaheuristic approach in the FJSP literature. The proposed representation schema is one of the contributions of this study. The performance of the pbGA is evaluated in comparison with the results obtained by the other algorithms from the literature for four instance sets. The computational results showed that the proposed algorithm performed at the same level or better with respect to the makespan in the Kacem and BC data sets when compared to the results from the other alternative solution methods. Two new better upper bounds for two instances in BRdata are attained by pbGA. For the BR data set, the pbGA performs worse than some of the algorithms developed in other studies. Among the HU data sets, the instances in the edata set seem to be more difficult than those in the rdata and vdata sets because, in some instances, the edata has poor lower bound quality. The proposed GA performs better than the other benchmarking algorithms for the edata and rdata sets. However, the results found by the pbGA are not as good as the benchmarking algorithm for the vdata set. The reason that the proposed algorithm achieves worse performance might be the parameters which are fixed for all
the data sets. Different parameters for each data set may give better performance. The size and flexibility of the problem could be considered to be determinants of the parameters in further studies in order to improve the results for the data sets in which the proposed algorithm has worse performance. For example, a higher population size may be used to improve the results of the high flexibility instances.

Generally, it can be claimed that the proposed GA is efficient in terms of the computational effort and effective with respect to the average RD in obtaining near optimal results for FJSPs, especially for the four data sets (Kacem, BC, edata and rdata). Variations in the structure of the GA should be implemented to improve the results for the BR data and vdata sets. All results were achieved in a reasonable computational time less than 10 minutes.

In further studies, the number of replaced operations in machine mutation could be altered, according to the difficulty of the instances, in order to avoid local optima. In machine mutation, sequence mutation and the ILS algorithm, all neighbors of a schedule construct the solution space. In order to reduce the solution space, the neighborhood function developed by Mastrolilli and Gambardella [54] could be implemented. Since the search area would be shrunk, the number of individuals applied in the local search could be augmented. In the ILS algorithm used in the pbGA, the number of operations repositioned during perturbation is designated as a fixed value. In further studies, the best number of operations being perturbed could be changed with respect to the particular instance, as in Stützle [72].

Acknowledgments. This research is partially supported by LATNA Laboratory, NRU HSE, RF government grant, ag. 11.G34.31.0057.

REFERENCES

[1] N. Al-Hinai and T. Y. ElMekkawy, An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem, Flexible Services and Manufacturing Journal, 23 (2011), 64–85.
[2] M. Amiri, M. Zandieh, M. Yazdani and A. Bagheri, A variable neighbourhood search algorithm for the flexible job-shop scheduling problem, International Journal of Production Research, 48 (2010), 5671–5689.
[3] J. Arroyo, G. Nunes and E. Kamke, Iterative local search heuristic for the single machine scheduling problem with sequence dependent setup times and due dates, in Hybrid Intelligent Systems, 2009. HIS ’09. Ninth International Conference on, 1 (2009), 505–510.
[4] A. Bagheri, M. Zandieh, I. Mahdavi and M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem, Future Generation Computer Systems-the International Journal of Grid Computing-Theory Methods and Applications, 26 (2010), 533–541.
[5] A. Baykasoglu, Linguistic-based meta-heuristic optimization model for flexible job shop scheduling, International Journal of Production Research, 40 (2002), 4523–4543.
[6] A. Baykasoglu, L. Ozbakir and A. I. Sonmez, Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems, Journal of Intelligent Manufacturing, 15 (2004), 777–785.
[7] J. E. Beasley, Population heuristics, in Handbook of Applied Optimization (eds. P. M. Pardalos and M. G. Resende), Oxford University Press, 2002.
[8] D. Behnke and M. J. Geiger, Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers, Technical report, Helmut-Schmidt-University, Logistics Management Department, Hamburg, Germany, 2012.
[9] W. Bozejko, M. Uchronski and M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, Computers and Industrial Engineering, 59 (2010), 323–333.
[10] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research, 41 (1993), 157–183.
[11] P. Brucker and R. Schlie, Job-shop scheduling with multipurpose machines, Computing, 45 (1990), 369–375.
[12] M. Caramia and P. Dell’Olmo, Coloring graphs by iterated local search traversing feasible and infeasible solutions, *Discrete Applied Mathematics*, **156** (2008), 201–217.
[13] J. B. Chambers and J. W. Barnes, Flexible job shop scheduling by tabu search, 1996.
[14] Y.-L. Chang and R. S. Sullivan, Schedule generation in a dynamic job shop, *International Journal of Production Research*, **28** (1990), 65–74.
[15] H. Chen, J. Ihlow and C. Lehmann, A genetic algorithm for flexible job-shop scheduling, in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 2 (1999), 1120–1125.
[16] T.-C. Chiang and H.-J. Lin, A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling, *International Journal of Production Economics*, **141** (2013), 87–98.
[17] J.-F. Cordeau, G. Laporte and F. Pasin, An iterated local search heuristic for the logistics network design problem with single assignment, *International Journal of Production Economics*, **113** (2008), 626–640.
[18] M. den Besten, T. Stützle and M. Dorigo, Design of iterated local search algorithms, in *Applications of Evolutionary Computing (ed. E. Boers)*, vol. 2037 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, 441–451.
[19] I. Driss, K. Mous and A. Laggoun, A new genetic algorithm for flexible job-shop scheduling problems, *Journal of Mechanical Science and Technology*, **29** (2015), 1273–1281.
[20] M. Ennigrou and K. Ghedira, New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach, *Autonomous Agents and Multi-Agent Systems*, **17** (2008), 270–287.
[21] I. Essafi, Y. Mati and S. Dauzère-Pérès, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, *Computers & Operations Research*, **35** (2008), 2599–2616.
[22] H. Farughi, B. Yousefi Yegane and M. Fathian, A new critical path method and a memetic algorithm for flexible job shop scheduling with overlapping operations, *Simulation*, **89** (2013), 264–277.
[23] P. Fattahi, M. S. Mehrabad and F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, *Journal of Intelligent Manufacturing*, **18** (2007), 331–342.
[24] M. Frutos, A. C. Olivera and F. Tohme, A memetic algorithm based on a NSGAII scheme for the flexible job-shop scheduling problem, *Annals of Operations Research*, **181** (2010), 745–765.
[25] J. Gao, M. Gen and L. Y. Sun, Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm, *Journal of Intelligent Manufacturing*, **17** (2006), 493–507.
[26] J. Gao, M. Gen, L. Y. Sun and X. H. Zhao, A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems, *Computers and Industrial Engineering*, **53** (2007), 149–162.
[27] J. Gao, L. Y. Sun and M. S. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, *Computers and Operations Research*, **35** (2008), 2892–2907.
[28] K. Gao, P. Suganthan, Q. Pan, T. Chua, T. Cai and C. Chong, Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling, *Information Sciences*, **289** (2014), 76–90.
[29] L. Gao, C. Y. Zhang and X. J. Wang, An improved genetic algorithm for multi-objective flexible job shop scheduling problem, *Advanced Materials Research*, **97** (2010), 2449–2454.
[30] M. R. Garey, D. S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research*, **1** (1976), 117–129.
[31] B. Giffler and G. Thompson, Algorithms for solving production scheduling problems, *Operations Research*, **8** (1960), 487–503.
[32] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
[33] R. Graham, E. Lawler, J. Lenstra and A. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics*, **5** (1979), 287–326.
[34] J. Grobler, A. P. Engelbrecht, S. Kok and S. Yadavalli, Metaheuristics for the multi-objective FJSP with sequence-dependent set-up times, auxiliary resources and machine down time, *Annals of Operations Research*, **180** (2010), 165–190.
[35] H. Hashimoto, M. Yagiura and T. Ibaraki, An iterated local search algorithm for the time-dependent vehicle routing problem with time windows, *Discrete Optimization*, 5 (2008), 434–456.

[36] N. B. Ho, J. C. Tay and E. M. K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, *European Journal of Operational Research*, 179 (2007), 316–333.

[37] J. Hurink, B. Jurisch and M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, *OR Spectrum*, 15 (1994), 205–215.

[38] A. S. Jain and S. Meeran, Deterministic job-shop scheduling: Past, present and future, *European Journal of Operational Research*, 113 (1999), 390–434.

[39] H. Jia, A. Nee, J. Fuh and Y. Zhang, A modified genetic algorithm for distributed scheduling problems, *Journal of Intelligent Manufacturing*, 14 (2003), 351–362.

[40] S. Jia and Z.-H. Hu, Path-relinking tabu search for the multi-objective flexible job shop scheduling problem, *Computers & Operations Research*, 47 (2014), 11–26.

[41] I. Kacem, S. Hammadi and P. Borne, 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 (2002), 1–13.

[42] I. Kacem, S. Hammadi and P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic, *Mathematics and Computers in Simulation*, 60 (2002), 245–276.

[43] H. Karimi, S. H. A. Rahmati and M. Zandieh, An efficient knowledge-based algorithm for the flexible job shop scheduling problem, *Knowledge-Based Systems*, 36 (2012), 236–244.

[44] S. Karthikeyan, P. Asokan and S. Nickolas, A hybrid discrete firefly algorithm for multi-objective flexible job shop scheduling problem with limited resource constraints, *The International Journal of Advanced Manufacturing Technology*, 72 (2014), 1567–1579.

[45] C.-Y. Lee and M. Pinedo, Optimization and heuristics in scheduling, in *Handbook of Applied Optimization* (eds. P. M. Pardalos and M. G. Resende), Oxford University Press, 2002, 569–584.

[46] J. Li, Q. Pan, S. Xie and S. Wang, A hybrid artificial bee colony algorithm for flexible job shop scheduling problems, *International Journal of Computers Communications and Control*, 6 (2011), 280–296.

[47] J. Q. Li, Q. K. Pan and Y. C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, *Computers and Industrial Engineering*, 59 (2010), 647–662.

[48] J. Q. Li, Q. K. Pan, P. N. Suganthan and T. J. Chua, A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, *International Journal of Advanced Manufacturing Technology*, 52 (2011), 683–697.

[49] J. Q. Li, Q. K. Pan and S. X. Xie, A hybrid variable neighborhood search algorithm for solving multi-objective flexible job shop problems, *Computer Science and Information Systems*, 7 (2010), 907–930.

[50] J.-Q. Li, Q.-K. Pan and M. F. Tasgetiren, A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities, *Applied Mathematical Modelling*, 38 (2014), 1111–1132.

[51] N. Liouane, I. Saad, S. Hammadi and P. Borne, Ant systems and local search optimization for flexible job shop scheduling production, *International Journal of Computers Communications and Control*, 2 (2007), 174–184.

[52] H. Lourenço, O. Martin and T. Stützle, Iterated local search, in *Handbook of Metaheuristics* (eds. F. Glover and G. Kochenberger), vol. 57 of International Series in Operations Research & Management Science, Springer US, 2003, 320–353.

[53] H. Lourenço, O. Martin and T. Stützle, Iterated local search: Framework and applications, in *Handbook of Metaheuristics* (eds. M. Gendreau and J.-Y. Potvin), vol. 146 of International Series in Operations Research & Management Science, Springer US, 2010, 363–397.

[54] M. Mastrolilli and L. Gambardella, Effective neighborhood functions for the flexible job shop problem, *Journal of Scheduling*, 3 (2000), 3–20.

[55] Z. Michalewicz and D. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, Berlin, 2000.

[56] E. Moradi, S. Ghomi and M. Zandieh, An efficient architecture for scheduling flexible job-shop with machine availability constraints, *International Journal of Advanced Manufacturing Technology*, 51 (2010), 325–339.
[57] E. Moradi, S. Ghomi and M. Zandieh, Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem, *Expert Systems with Applications*, 38 (2011), 7169–7178.

[58] G. Moslehi and M. Mahnam, A pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search, *International Journal of Production Economics*, 129 (2011), 14–22.

[59] M. Mousakhani, Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness, *International Journal of Production Research*, 51 (2013), 3476–3487.

[60] M. A. Nascimento, Giffler and thompson’s algorithm for job shop scheduling is still good for flexible manufacturing systems, *The Journal of the Operational Research Society*, 44 (1993), 521–524.

[61] J. A. Oliveira, L. Dias and G. Pereira, Solving the job shop problem with a random keys genetic algorithm with instance parameters, in *2nd International Conference on Engineering Optimization, Lisbon, Portugal*, 2010.

[62] I. Ono, M. Yamamura and S. Kobayashi, A genetic algorithm for job-shop scheduling problems using job-based order crossover, in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, 1996, 547–552.

[63] L. Paquete and T. Stützle, An experimental investigation of iterated local search for coloring graphs, in *Applications of Evolutionary Computing (eds. S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. Raidl), vol. 2279 of Lecture Notes in Computer Science, Springer Berlin Heidelberg*, 2002, 122–131.

[64] P. M. Pardalos and O. V. Shylo, An algorithm for the job shop scheduling problem based on global equilibrium search techniques, *Computational Management Science*, 3 (2006), 331–348.

[65] F. Pezzella, G. Morganti and G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, *Computers and Operations Research*, 35 (2008), 3202–3212.

[66] S. Rahmati, M. Zandieh and M. Yazdani, Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 64 (2013), 915–932.

[67] M. Rajkumar, P. Asokan, N. Anilkumar and T. Page, A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints, *International Journal of Production Research*, 49 (2011), 2409–2423.

[68] M. Rohaninejad, A. Kheirkhah, P. Fattahi and B. Vahedi-Nouri, A hybrid multi-objective genetic algorithm based on the Electre method for a capacitated flexible job shop scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 77 (2015), 51–66.

[69] V. Roshanaei, A. Azab and H. ElMaraghy, Mathematical modelling and a meta-heuristic for flexible job shop scheduling, *International Journal of Production Research*, 51 (2013), 6247–6274.

[70] C. R. Schrich, V. A. Armentano and M. Laguna, Tardiness minimization in a flexible job shop: A tabu search approach, *Journal of Intelligent Manufacturing*, 15 (2004), 103–115.

[71] X. Shao, W. Liu, Q. Liu and C. Zhang, Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 67 (2013), 2885–2901.

[72] T. Stützle, Iterated local search for the quadratic assignment problem, *European Journal of Operational Research*, 174 (2006), 1519–1539.

[73] L. Tang and X. Wang, Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem, *The International Journal of Advanced Manufacturing Technology*, 29 (2006), 1246–1258.

[74] J. C. Tay and N. B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, *Computers and Industrial Engineering*, 54 (2008), 453–473.

[75] S. J. Wang, B. H. Zhou and L. F. Xi, A filtered-beam-search-based heuristic algorithm for flexible job-shop scheduling problem, *International Journal of Production Research*, 46 (2008), 3027–3058.

[76] X. J. Wang, L. Gao, C. Y. Zhang and X. Y. Shao, A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem, *International Journal of Advanced Manufacturing Technology*, 51 (2010), 757–767.
Y. Wang, H. Yin and K. Qin, A novel genetic algorithm for flexible job shop scheduling problems with machine disruptions, *The International Journal of Advanced Manufacturing Technology*, 68 (2013), 1317–1326.

W. J. Xia and Z. M. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, *Computers and Industrial Engineering*, 48 (2005), 409–425.

L. N. Xing, Y. W. Chen and K. W. Yang, Multi-population interactive coevolutionary algorithm for flexible job shop scheduling problems, *Computational Optimization and Applications*, 48 (2011), 139–155.

M. Yazdani, M. Amiri and M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, *Expert Systems with Applications*, 37 (2010), 678–687.

Y. Yuan and H. Xu, Flexible job shop scheduling using hybrid differential evolution algorithms, *Computers & Industrial Engineering*, 65 (2013), 246–260.

Y. Yuan and H. Xu, An integrated search heuristic for large-scale flexible job shop scheduling problems, *Computers & Operations Research*, 40 (2013), 2864–2877.

Y. Yuan and H. Xu, Multiobjective flexible job shop scheduling using memetic algorithms, *Automation Science and Engineering, IEEE Transactions on*, 12 (2015), 336–353.

Y. Yuan, H. Xu and J. Yang, A hybrid harmony search algorithm for the flexible job shop scheduling problem, *Applied Soft Computing*, 13 (2013), 3259–3272.

G. H. Zhang, X. Y. Shao, P. G. Li and L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, *Computers and Industrial Engineering*, 56 (2009), 1309–1318.

Q. Zhang and J. Sun, Iterated local search with guided mutation, *2006 IEEE International Conference on Evolutionary Computation*, 924–929.

M. Ziaee, A heuristic algorithm for solving flexible job shop scheduling problem, *The International Journal of Advanced Manufacturing Technology*, 71 (2014), 519–528.

N. Zribi, I. Kacem, A. El Kamel and P. Borne, Assignment and scheduling in flexible job-shops by hierarchical optimization, *Ieee Transactions on Systems Man and Cybernetics Part C-Applications and Reviews*, 37 (2007), 652–661.

Received January 2015; revised June 2015.

E-mail address: cinard@itu.edu.tr
E-mail address: jose.oliveira@algoritmi.uminho.pt
E-mail address: topcuil@itu.edu.tr
E-mail address: pardalos@ufl.edu