Comparative Study of Genetic and Discrete Firefly Algorithm for Combinatorial Optimization

Willian Tessaro Lunardi*
University of Luxembourg
Luxembourg, Luxembourg
willian.tessarolunardi@uni.lu

Holger Voos†
University of Luxembourg
Luxembourg, Luxembourg
holger.voos@uni.lu

ABSTRACT

Flexible job-shop scheduling problem (FJSP) is one of the most challenging combinatorial optimization problems. FJSP is an extension of the classical job shop scheduling problem where an operation can be processed by several different machines. The FJSP contains two sub-problems, namely machine assignment problem and operation sequencing problem. In this paper, we propose and compare a discrete firefly algorithm (FA) and a genetic algorithm (GA) for the multi-objective FJSP. Three minimization objectives are considered, the maximum completion time, workload of the critical machine and total workload of all machines. Five well-known instances of FJSP have been used to evaluate the performance of the proposed algorithms. Comparisons among our methods and state-of-the-art algorithms are also provided. The experimental results demonstrate that the FA and GA have achieved improvements in terms of efficiency. Solutions obtained by both algorithms are comparable to those obtained by algorithms with local search. In addition, based on our initial experiments, results show that the proposed discrete firefly algorithm is feasible, more effective and efficient than our proposed genetic algorithm for solving multi-objective FJSP.

CCS CONCEPTS

• **Computing methodologies → Heuristic function construction**;

KEYWORDS

Firefly algorithm, Genetic algorithm, Multi-objective optimization, Flexible job-shop scheduling

ACM Reference format:
Willian Tessaro Lunardi and Holger Voos. 2018. Comparative Study of Genetic and Discrete Firefly Algorithm for Combinatorial Optimization. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 9 pages.
https://doi.org/10.1145/3167132.3167160

*Also with Interdisciplinary Centre for Security, Reliability and Trust (SnT).
†Also with Interdisciplinary Centre for Security, Reliability and Trust (SnT).

1 INTRODUCTION

Production scheduling is one of the most important issues in the planning and scheduling of modern manufacturing systems. There are several workshop styles in the manufacturing system (including the Job Shop Scheduling Problem, JSP). In the classical JSP, a set of jobs must be processed on a set of machines. Each job consists of a sequence of operations, where each operation has to be processed without preemption on a machine which is determined in advance. The objective is to find a processing sequence for each machine that minimizes an objective. A typical objective is to minimize the completion time of the last operation or also referred as makespan [3]. In terms of computational complexity, JSP is one of the most challenging combinatorial optimization problems and has been proven to be an NP-hard problem [7, 13].

The inexistence of flexibility on the resources of each operation presented on JSP may meet the requirements of a traditional manufacturing system. However, with the ushering of the fourth industrial revolution (Industry 4.0) many computing devices, flexible manufacturing systems, and numerical control machines are introduced in order to achieve a higher level of autonomously, customization, and flexibility. Therefore, the assumption that one machine only processes one type of operation works in JSP but does not reflect what we face in modern manufacturing systems.

The Flexible Job Shop Problem (FJSP) is an extension of the classical JSP problem that allows an operation to be processed by any machine from a given set of machines. The FJSP can be decomposed into two sub-problems: the machine selection problem (MS) and the operations sequencing problem (OS). The JSP contains only the OS problem, while the FJSP deals with both the OS and the MS problem. In this way, FJSP is NP-hard and more difficult to solve than the JSP.

Since 1990 when this problem was first presented [2], many methods have been presented to solve it. The current approaches for solving FJSP mainly include exact algorithm [5], evolutionary algorithms (EA) [11, 14, 16, 17], swarm intelligence (SI) based approaches [12, 18, 19, 23], and local search (LS) algorithms [6].

In recent years multiple evolutionary algorithms (EA) were proposed to solve the FJSP. Recently, in [3], it was shown that hybrid techniques have been applied more often than other methods for solving FJSP. Furthermore, the technique most frequently chosen for performing exploration in hybrid algorithms is the genetic algorithm (GA). Although GA has powerful global searching ability [14], a recently developed algorithm called the Firefly Algorithm (FA), proposed in [20], has been shown [15, 20] to outperform Particle Swarm Optimization (PSO) and GA for continuous optimization problems. In this paper, we propose and compare a Genetic Algorithm and a discrete Firefly Algorithm for solving the FJSP. Five famous instances of FJSP are used to evaluate the performance of both methods. Through experimental studies, the merits of each algorithm are demonstrated clearly. Furthermore, the proposed algorithms are compared with other state-of-the-art algorithms.

The remainder of this paper is structured as follows. The problem formulation is discussed in Section 2. The solution representation for both algorithms is defined in the Section 3. The discrete FA is proposed in Section 4 and the GA is proposed in Section 5. Experimental results related to the proposed approaches are reported in Section 6. Section 7 addresses the conclusions and potential future works.

2 PROBLEM FORMULATION

A general multi-objective minimization can be defined as: minimize a function $F(x)$, with P ($P > 1$) decision variables and Q objectives ($Q > 1$), subject to several equality or inequality constraints.

$$\underset{x \in \Omega}{\text{Minimize}}\, F(x) = (f_1(x), ..., f_q(x), ...f_Q(x)) \tag{1}$$

where Ω is the feasible solutions space, $x = x_1, ..., x_p, ..., x_P$ is the set of p-dimensional decision variables (continuous, discrete or integer) ($1 \leq p \leq P$), i.e., a possible solution for the considered problem; $f_q(x)$ is the qth objective function ($1 \leq q \leq Q$). It is obvious that an exact solution to such a problem does not exist. However, the multi-objective optimization method should optimize the different objective functions simultaneously. More precise definitions of the terminology used in the field of multi-objective optimization can be found in [4].

Many studies have been devoted to the subject of multi-objective meta-heuristic optimization and the developed methods to solve multi-objective optimization problems can be generally classified into three different types:

(1) The transformation towards a mono-objective problem consists of combining the different objectives into a weighted sum. Methods in this class are based on utility functions or E-Constraint and goal programming;
(2) The non-Pareto approach utilizes operators for processing the different objectives in a separated way;
(3) The Pareto approaches are directly based on the Pareto optimality concept. They aim at satisfying two goals: converging towards the Pareto front and also obtaining diversified solutions scattered all over the Pareto front.

In this study, in order to compare with the other state-of-the-art algorithm, the objective function is based on the first type where we transform the multi-objective into a mono-objective problem.

The FJSP can be formulated as follows. There is a set of n jobs and a set of m machines. M denotes the set of all machines. Each job i consists of a sequence of n_i operations. Each operation $O_{ij}(i = 1, 2, ..., n; j = 1, 2, ..., n_i)$ of job i has to be processed on a machine k out of a set of given compatible machines M_{ij} (for $k \in M_{ij}, M_{ij} \subseteq M$). FJSP can be classified further into two categories: partial flexibility (P-FJSP) when there is $M_{ij} \subset M$ for at least one operation, and total flexibility (T-FJSP) when there is $M_{ij} = M$ for each operation [10, 11].

In this paper, the following criteria are to be minimized:

(1) the makespan C_{max}, i.e., the maximal completion time;
(2) the maximal machine workload W_{max}, i.e., the maximum working time spent on any machine;
(3) the total workload of machines T, i.e., total working time over all machines.

During the process of solving the problem, the following assumptions are made:

(1) Each operation cannot be interrupted during its performance (non-preemptive condition);
(2) Each machine can perform at most one operation at any time (resource constraint);
(3) Each machine are continuously available;
(4) Setting up time of machines and move times between operations are negligible;
(5) Machines are independent of each other;
(6) Jobs are independent of each other.

The notation used in this paper is summarized in the following:

Indices

k :	index of machines, $k = 1, ..., m$;
i, h :	index of jobs, $i, h = 1, ..., n$;
j, g :	index of operation sequence, $j, g = 1, ..., n_i$;

Parameters

n :	total number of jobs;
n_i :	total number of operation of job i;
m :	total number of machines;
J_i :	the ith job;
O_{ij} :	the jth operation of job i;
M_{ij} :	machines able to perform operation O_{ij};
p_{ijk} :	processing time of O_{ij} on machine k;
λ :	an weight coefficient;

Decision variables

C_{max} :	maximal completion time of the machines;
W_{max} :	maximal workload of the machines;
W_k :	workload of the machine k;
T :	total workload of the machines;
c_{ij} :	completion time of the operation O_{ij};

$$v_{ijk} : \begin{cases} 1 & \text{if } O_{ij} \text{ is performed on machine } k \\ 0 & \text{otherwise;} \end{cases}$$

$$z_{ijhgk} : \begin{cases} 1 & \text{if } O_{ij} \text{ precedes operation } O_{hg} \text{ on machine } k \\ 0 & \text{otherwise.} \end{cases}$$

The mixed integer programming model can be given as follows:

$$\text{minimize } F(c) = \lambda_1 C_{max} + \lambda_2 W_{max} + \lambda_3 T \quad (2)$$

subject to

$$C_{max} \geq c_{ij} \qquad \forall i, j = n_i \quad (3)$$

$$W_{max} \geq W_k \qquad \forall k \quad (4)$$

$$W_k = \sum_{i=1}^{n} \sum_{j=1}^{n_i} p_{ijk} v_{ijk} \qquad \forall k \quad (5)$$

$$T = \sum W_k \qquad \forall k \quad (6)$$

$$c_{ij} - c_{i(j-1)} \geq p_{ijk} v_{ijk}, \qquad \forall i, k\ \forall j = 2, ..., n_i \quad (7)$$

$$c_{ij} \geq p_{ijk} v_{ijk} \qquad \forall i, j = 1, \forall k \in M_{ij} \quad (8)$$

$$(c_{hg} - c_{ij} - p_{hgk}) v_{hgk} v_{ijk} z_{ijhgk} \geq 0 \qquad \forall i, h, j, g, \forall k \in M_{ij} \cap M_{hg} \quad (9)$$

$$(c_{ij} - c_{hg} - p_{ijk}) v_{ijk} v_{hgk} z_{hgijk} \geq 0 \qquad \forall i, h, j, g, \forall k \in M_{ij} \cap M_{hg} \quad (10)$$

$$\sum\nolimits_{k \in M_{ij}} v_{ijk} = 1 \qquad \forall i, j \quad (11)$$

$$z_{ijhgk} + z_{hgijk} = v_{ijk} v_{hgk} \qquad \forall i, h, j, g, \forall k \in M_{ij} \cap M_{hg} \quad (12)$$

$$c_{ij} \geq 0 \qquad \forall i, j \quad (13)$$

$$v_{ijk} \in \{0, 1\} \qquad \forall i, j, k \quad (14)$$

Objective function (2) ensures the minimization of maximal completion time, maximal workload, and total workload of the machines and is supported by constraints (3), (4), (5), and (6). Constraints (7) enforces each job to follow a specific operation sequence. Constraint (8) ensures the completion time of first operation of job i equal to be at least the processing time of O_{ij}. Constraint (9) and (10) are disjunctive constraints. It represents that an operation must be completed before the starting of another operation if they are assigned on the same machine. Constraint (11) states that one machine must be selected from a set of available machines for each operation. Constraint (12) enforces to be chosen one of two precedence relationships.

3 ENCODING AND DECODING

The adopted solution representation for both algorithms is defined as follows. The FJSP contains two sub-problems, in this way, our representation will contain two strings. The first one is the MS string and the second is the OS string.

The MS string denotes the selected machine for the corresponding operations of each job. The length of this string is equal to $\sum_{i=1}^{n} n_i$. The hth part of the MS string can assume any value $k \in M_{ij}$ and represents the assigned machine for operation O_{ij}. The operation number does not change throughout the whole searching process.

We use the operation-based representation method for the OS string, which is comprised of the jobs number. This representation uses an unpartitioned permutation with $\sum_{i=1}^{n} n_i$ repetitions of the job numbers. In this representation, each job J_i number appears n_i times in the OS string. By scanning the OS string from left to right, the fth appearance of a job number refers to the fth operation of this job. In this way, any permutation of the OS string can be decoded into a feasible solution and avoid the use of a repair mechanism.

When an individual or a firefly is decoded, the OS string is converted into a sequence of operation at first. Given the problem instance shown in Table 1 and the strings shown in Figure 1, the OS string can be translated into a list of ordered operations: [O_{21}, O_{11}, O_{31}, O_{22}, O_{32}, O_{12}, O_{13}, O_{23}]. Then each operation is assigned to a machine according to the MS string as follows: [(O_{21}, M_1), (O_{11}, M_2), (O_{31}, M_2), (O_{22}, M_3), (O_{32}, M_1), (O_{12}, M_4), (O_{13}, M_3), (O_{23}, M_4)].

Job	*Operation*	M_1	M_2	M_3	M_4
J_1	O_{11}	1	3	4	1
	O_{12}	3	8	2	1
	O_{13}	3	5	4	7
J_2	O_{21}	4	1	1	4
	O_{22}	2	3	9	3
	O_{23}	9	1	2	2
J_3	O_{31}	8	6	3	5
	O_{32}	4	5	8	1

Table 1: FJSP with three jobs and four machines.

4 FIREFLY ALGORITHM

The firefly algorithm is a nature-inspired meta-heuristic for solving continuous NP-hard problems and has been motivated by the simulation of the social behavior of fireflies. The two fundamental functions of its flashing lights are to attract mating partners (communication), and to attract potential prey.

In essence, FA uses the following three idealized rules [20]:

- All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex;
- Attractiveness β is proportional to their brightness, thus for any two flashing fireflies, the less bright one will move towards the brighter one. The attractiveness is proportional to the brightness and they both decrease as their distance increases. If there is no brighter one than a particular firefly, it will move randomly;
- The brightness of a firefly is affected or determined by the landscape of the objective function.

For a maximization problems, the brightness may be proportional to the objective function value. For minimization problems, the

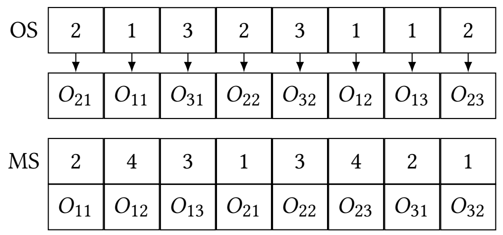

OS
2 1 3 2 3 1 1 2
O_{21} O_{11} O_{31} O_{22} O_{32} O_{12} O_{13} O_{23}
MS
2 4 3 1 3 4 2 1
O_{11} O_{12} O_{13} O_{21} O_{22} O_{23} O_{31} O_{32}

Figure 1: OS and MS strings example.

Current Position	MS string	OS string
(P)	2 4 3 1 3 4 2 1	2 1 3 2 3 1 1 2
(P_{best})	1 4 1 3 2 2 3 4	1 2 3 2 1 1 2 3
d_{ms} and d_{os}	{(1,1), (3,1), (4,3), (5,2), (6,2), (7,3), (8,4)}	{(1,2), (5,6), (6,7), (7,8)}
$\|d_{ms}\|$ and $\|d_{os}\|$	7	4
Attractiveness $\beta(r)$	0.17	0.38
$rand \in [0,1]$	{0.35, 0.1, 0.09, 0.14, 0.33, 0.49, 0.32}	{0.52, 0.05, 0.12, 0.69}
Movement β-step	{(3,1), (4,3), (5,2)}	{(5,6), (6,7)}
Position after β-step	2 4 1 3 2 4 2 1	2 1 3 2 1 1 3 2
Position after α-step	2 4 1 3 2 2 4 1	2 1 3 2 1 1 2 3

Table 2: Movement of firefly towards the global best.

brightness may be the reciprocal of the objective function value. The pseudo code shown in Algorithm 1 abstracts the basic steps of the FA which consists of the three rules discussed above.

4.1 Distance, Attraction and Movement

In the firefly algorithm, there are two important issues: the variation of light intensity and formulation of the attractiveness. For simplicity, we can always assume that the attractiveness of a firefly is determined by its brightness, which in turn is associated with the encoded objective function.

The attractiveness function $\beta(r)$ can be any monotonically decreasing functions such as the following generalized form

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \quad m \geq 1, \tag{15}$$

where β_0 is the attractiveness at $r = 0$, and r is the distance between two fireflies. As it is often faster to calculate $1/(1 + r^2)$ than an exponential function [20], the Equation (15) can be approximated as

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}. \tag{16}$$

The distance between any two fireflies i and j, at position x_i and x_j, respectively can be defined as a Cartesian distance:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_{ik} - x_{jk})^2}, \tag{17}$$

where x_{ik} is the kth component of the spatial cordinate x_i of ith firefly.

The random movement of a firefly i towards another more brighter firefly j is determined by

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_i - x_j) + \alpha\, \epsilon_i, \tag{18}$$

where the second term considers a firefly's attractiveness, the third term is randomization with α being the randomization parameter, and ϵ_i is a vector of random numbers drawn from a Gaussian distribution or uniform distribution. In a simplest form, ϵ_i can be replaced by **rand** $- 1/2$, where **rand** is a random number generator uniformly distributed in [0,1]. For most applications we can take $\beta_0 = 1$, $\alpha \in [0, 1]$. In this implementation of the algorithm we used $\beta_0 = 1.0$, $\alpha \in [0, 1]$ and $\gamma \in [0.01, 0.15]$.

4.2 Discretization

The FA has been originally developed for solving continuous optimization problems and cannot be directly applied to solve discrete optimization problems. The main challenges for using the FA to solve FJSP are computing the discrete distance between two fireflies, and how they move in the coordination. In this work, we extend the discretization proposed in [12], and is done for the following issues described below.

4.3 Distance

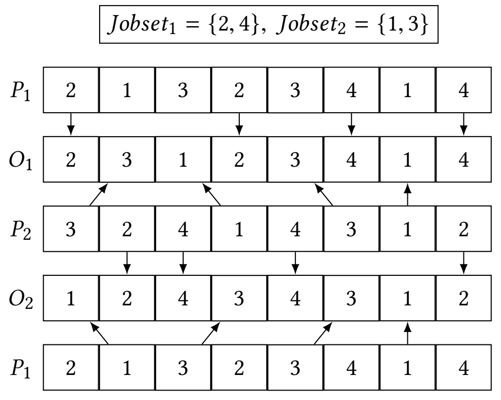
The distance between two fireflies is defined by the distance between the permutation of its strings. There are two possible ways to measure the distance between two permutations:

(a) Hamming Distance;
(b) The number of the required swaps of the first solution in order to get the second one.

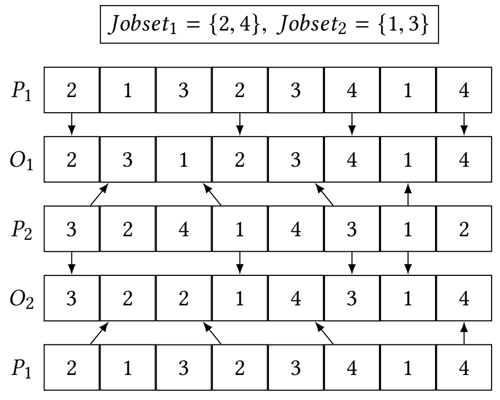
The distance between the MS string of two fireflies can be measured by using Hamming distance. The Hamming distance between two permutations is the number of non-corresponding elements in the sequence. The example of Hamming distance is given as follows: given two MS strings where $P^{ms} = \{2\ 4\ 3\ 1\ 3\ 4\ 2\ 1\}$ and $P^{ms}_{best} = \{1\ 4\ 1\ 3\ 2\ 2\ 3\ 4\}$, we compare every item and record the number of bits whose machine indices are not equal. Hence, the Hamming distance $(P^{ms}, P^{ms}_{best}) = 7$. The distance between two OS

Algorithm 1 Firefly Algorithm

```
1:  Objective function $f(x),\ x = (x_1, ..., x_d)^T$
2:  Generate initial pop. $P$ of fireflies $x_i (i = 1, 2, ..., n)$
3:  Light intensity $I_i$ at $x_i$ is determined by $f(x_i)$
4:  Define light absorption coefficient $\gamma$
5:  **while** ($t < MaxGeneration$) **do**
6:      **for each** $x_i \in P$ **do**
7:          **for each** $x_j \in P$ **do**
8:              **if** ($I_i < I_j$) **then** Move $x_i$ towards $x_j$ **end if**
9:              Vary $\beta$ with distance $r$ via $exp[-\gamma r]$
10:             Evaluate solutions and update light intensity
11:         **end for** $j$
12:     **end for** $i$
13:     Rank fireflies and find the current global best
14: **end while**
```



(a) POX crossover



(b) JBX crossover

Figure 2: Crossover operators for OS string.

strings of two fireflies can be measured with the so called swapping distance. The swapping distance is the number of minimal required swaps of one permutation in order to obtain the other one [12]. Given two OS strings P^{os} = {2 1 3 2 3 1 1 2} and P^{os}_{best} = {1 2 3 2 1 1 2 3}, the swapping distance (P^{os}, P^{os}_{best}) = 4.

4.4 Attraction and movement

In this study we break up the movement given in equation (18) into two sub-steps: β-step and α-step as given in equation (19) and equation (20) respectively.

$$x_i = \beta(r)(x_j - x_i) \tag{19}$$

$$x_i = x_i + \alpha(rand - 1/2) \tag{20}$$

The attraction steps β and α are not interchangeable, thereby, β-step must be computed before α-step while finding the new position. Both steps are explained in details bellow. Table 2 illustrates a firefly position update towards the global best. The parameters used in this illustration are as follows: $\beta_0 = 1, \gamma = 0.1, \alpha = 1$.

4.4.1 β-step. In this procedure, an insertion mechanism and a pair-wise exchange mechanism are used to advance the MS string and OS string of a firefly towards the global best firefly position. At first, all necessary insertions in the MS string and all pair-wise exchanges in the OS string, to make the elements of the current firefly equal to the best firefly, are found and store in d_{ms} and d_{os} respectively. The hamming distance and swap distance are respectively defined by $|d_{ms}|$ and $|d_{os}|$, and stored in r. The β probability is computed using equation (16). Furthermore, we have to define which elements of d_{ms} and d_{os} will be used to change the current solution. In this way, for each element of d_{ms} and d_{os} is accessed and a random number $rand \in [0, 1]$ is generated. In this way, if $rand \leq \beta$, then the corresponding insertion/pair-wise exchange is performed on the elements of the current Firefly.

4.4.2 α-step. In this procedure, a swapping mechanism is used to shift the permutation into one of the neighbouring permutations. Equation (21) approximate equation (20), where $rand_{int} \in [0, \sum_{i=1}^{n} n_i]$.

$$x_i = x_i + \alpha(rand_{int}) \tag{21}$$

The α-step is applied by choosing an element position and swap with another non-equal position in the string which is also chosen at random. This procedure is similar to the swapping mutation shown in the Figure 4.

5 GENETIC ALGORITHM

Genetic algorithm (GA), proposed in [9], is a class of algorithms based on the abstraction of Darwinian evolution of biological systems. Starting from an initial population, the algorithm applies genetic operators in order to produce offspring. At each generation, every new individual corresponds to a solution, i.e., a schedule of the given FJSP instance.

5.1 Genetic Operators

In this paper, each string is subjected to its own genetic operators. Three commonly used genetic operators, selection, crossover, and mutation, will be briefly discussed in the following sub-sections.

5.1.1 Selection. In GAs, the selection operator is used to select the individuals according to their fitness and maintain the highest quality chromosomes and characteristics within the population. In our algorithm, the selection strategy includes two parts: the method of keeping the best individuals and tournament selection. The method of keeping the best individuals is to copy the 1% of the best individuals for the next generation. The tournament selection strategy, proposed in [8], works as follows: two solutions are selected randomly as the parent solutions, if a random number generated between 0 and 1 is smaller than the probability r which usually is set to 0.8, then we select the better one; otherwise, we select the other one.

5.1.2 Crossover. Crossover is the recombination of two parent chromosomes through the exchange of a part of one chromosome with a corresponding part of another in order to produce offspring. In this paper, two crossover operators: (a) precedence operation crossover (POX), proposed in [21]; (b) job-based crossover (JBX), proposed in [22]; are adopted for the OS string. During the OS string crossover procedure, one crossover operator is selected randomly.

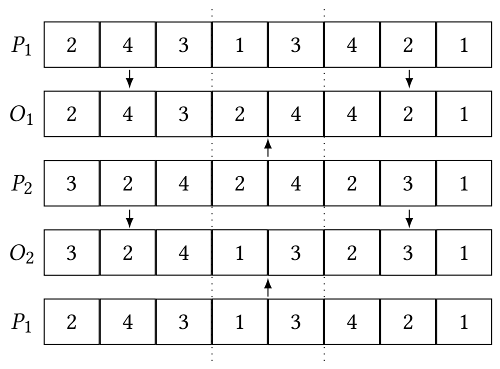


Figure 3: Two point crossover.

The basic working procedure of POX is described as bellow (two parents are denoted as P_1 and P_2; two offspring are denoted as O_1 and O_2). Figure 2 shows an example of POX crossover operator.

(1) The Job set $J = \{J_1, J_2, J_3, ..., J_n\}$ is divided into two groups $Jobset_1$ and $Jobset_2$ randomly;
(2) Any element in P_1 which belongs to $Jobset_1$ are appended to the same position in O_1 and deleted in P_1; any element in P_2 which belongs to $Jobset_1$ are appended to the same position in O_2 and deleted in P_2;
(3) the remaining elements in P_2 are appended to the remaining empty positions in O_1 seriatim; and the remaining elements in P_1 are appended to the remaining empty positions in O_2 seriatim.

The second crossover operator for OS string is the job-based crossover (JBX). The basic working procedure of JBX is described below. Figure 2 shows an example of JBX crossover operator.

(1) The Job set $J = \{J_1, J_2, J_3, ..., J_n\}$ is divided into two groups $Jobset_1$ and $Jobset_2$ randomly;
(2) Any element in P_1 which belongs to $Jobset_1$ are appended to the same position in O_1; any element in P_2 which belongs to $Jobset_2$ are appended to the same position in O_2;
(3) Any element in P_2 which belongs to $Jobset_2$ are appended to the remaining empty positions in O_1 seriatim; and any element in P_1 which belongs to $Jobset_1$ are appended to the remaining empty positions in O_2 seriatim.

For the MS string, a standard two-point crossover has been adopted as the crossover operation. In this operation, two positions are selected at random. Based on the selected positions, two children strings are created by swapping all elements between the positions of the two parent strings. Figure 3 shows an example of two-point crossover.

5.1.3 Mutation. The mutation can provide some extra variation into the current population, which can enhance the diversity. The mutation probability should be small as a large one will be adverse for the information preservation of the good chromosomes. In this paper, the swapping mutation [22], has been adopted for the OS string. In this procedure, two positions are selected and its respective elements are swapped. A single point mutation is used for the MS string. In this procedure, a position of the MS string is

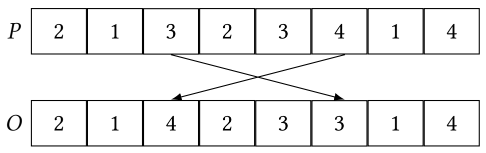


Figure 4: Swapping mutation.

Problem	P_{fa}/P_{ga}	G_{fa}/G_{ga}	γ	p_c	p_m
4×5	50/100	100/80	0.05	0.85	0.01
8×8	50/250	300/80	0.05	0.85	0.01
10×7	50/400	400/80	0.05	0.85	0.01
10×10	50/500	500/80	0.05	0.85	0.01
15×10	100/2000	1000/100	0.05	0.85	0.01

Table 3: Parameters of FA and GA.

selected, and a new machine is assigned for its respective operation. Figure 4 shows an example of swapping mutation.

6 EXPERIMENTAL RESULTS

This section describes the computational experiments used to evaluate the performance of the proposed algorithms. In order to conduct the experiment, we implement the algorithm in C++ on an Intel Core i7 2.70GHz × 8, with 8 GB memory. To test the performance of the algorithms, five representative instances have been taken into this experiment. The dimensions of the instances range from 4 jobs × 5 machines to 15 jobs × 10 machines. In order to fairly compare our proposed work with other state-of-the-art algorithms, we adopted a common set of weight coefficients, define by: $\lambda_1 = 0.5, \lambda_2 = 0.3, \lambda_3 = 0.2$. The best and average results of experiments from 50 different runs were collected for performance comparison.

6.1 Setting parameter

Each instance can be characterized by the following parameters: number of jobs (n), number of machines (m), and operation $O_{i,j}$ of job i. The parameters of the FA consist of the population size P_{fa}, maximum number of generations G_{fa}, attractiveness of fireflies β_0, light absorption coefficient γ, and randomization α. In this experiments, the β_0 and α parameters were kept at 1. The parameters of the GA consist of population size P_{ga}, maximum number of generations G_{ga}, crossover probability p_c and mutation probability p_m. The detailed parameters of the proposed FA and GA for the problem instances are presented in the Table 3.

6.2 Problem 4 × 5

This is a T-FJSP small-scale instance in which 4 jobs with 12 operations are to be performed on 5 machines. Both proposed approaches obtained the same solutions characterized by the following values:

(1) makespan $C_{max} = 11$, maximal workload $W_{max} = 10$, total workload $T = 32$.
(2) makespan $C_{max} = 12$, maximal workload $W_{max} = 8$, total workload $T = 32$.

Algorithm	4 × 5				8 × 8				10 × 7				10 × 10				15 × 10			
	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU
AL+CGA	16	10	34	—	15	13	79	—		—		—	7	5	45	—	23	11	93	—
		—			16	13	75			—				—			24	11	91	
PSO+SA		—		—	15	12	75	—		—		—	7	6	44	—	12	11	91	—
		—			16	13	73			—				—				—		
PSO+TS	12	8	32	0.34	14	12	77	1.67		—		—	7	6	43	2.05	11	11	93	10.88
		—		—	15	12	75			—				—		—		—		—
AIA		—		—	14	12	77	0.76		—		—	7	5	43	8.97	11	11	93	109.22
MOGA	11	10	32	5.8	15	11	81	9.5		—		—	7	5	45	14.2	11	11	91	87.5
	12	8	32		15	12	75			—			7	6	42		11	10	98	
Prop. FA	11	10	32	0.11	14	12	77	0.64	11	10	62	0.84	7	5	43	1.07	11	11	93	6.86
	12	8	32		15	12	75		11	11	61		7	6	42		12	11	91	
Prop. GA	11	10	32	0.29	14	12	77	0.90	11	10	62	1.57	7	6	42	2.02	12	12	93	18.76
	12	8	32		15	12	75		11	11	61		8	5	42		12	10	95	

$f_1 = C_{max}$, $f_2 = W_{max}$, and $f_3 = T$. — equals not available.

Table 4: Comparison of results with five Kacem instances.

The average computational time for the FA and GA are respectively 0.11 and 0.29 seconds. In Figure 5 the solution (1) is presented using Gantt chart.

6.3 Problem 8 × 8

This is a P-FJSP instance, in which 8 jobs with 27 operations are to be performed on 8 machines. The best solutions obtained by both proposed approaches are the same and characterized by the following values:

(1) makespan $C_{max} = 14$, maximal workload $W_{max} = 12$, total workload $T = 77$.
(2) makespan $C_{max} = 15$, maximal workload $W_{max} = 12$, total workload $T = 75$.

The average computational time for the FA and GA are respectively 0.64 and 0.90 seconds. In Figure 6 the solution (1) is presented using Gantt chart.

6.4 Problem 10 × 7

This is a T-FJSP instance, in which 10 jobs with 29 operations are to be performed on 7 machines. The best solutions obtained by both proposed approaches are the same and characterized by the following values:

(1) makespan $C_{max} = 11$, maximal workload $W_{max} = 10$, total workload $T = 62$.
(2) makespan $C_{max} = 11$, maximal workload $W_{max} = 11$, total workload $T = 61$.

The average computational time for the FA and GA are respectively 0.84 and 1.57 seconds. In Figure 7 the solution (1) is presented in a Gantt chart.

6.5 Problem 10 × 10

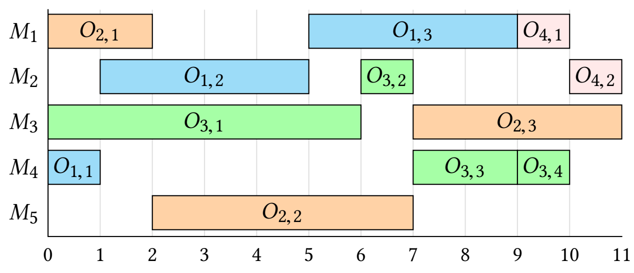
This is a T-FJSP instance, in which 10 jobs with 30 operations are to be performed on 10 machines. The best solutions obtained by the proposed FA can be characterized by the following values:

(1) makespan $C_{max} = 7$, maximal workload $W_{max} = 5$, total workload $T = 43$.
(2) makespan $C_{max} = 7$, maximal workload $W_{max} = 6$, total workload $T = 42$.

The best solutions obtained by the proposed GA can be characterized by the following values:

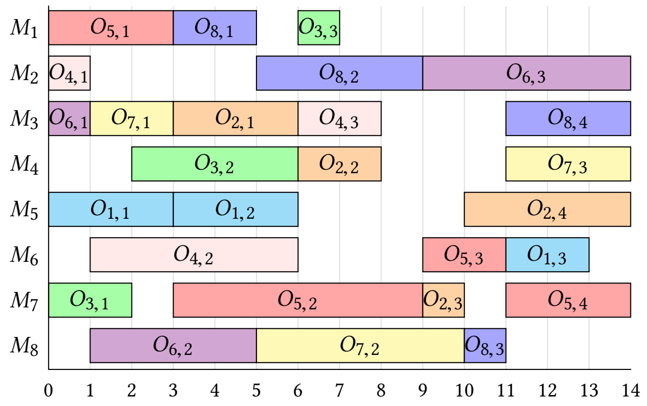
(1) makespan $C_{max} = 7$, maximal workload $W_{max} = 6$, total workload $T = 42$.
(2) makespan $C_{max} = 8$, maximal workload $W_{max} = 5$, total workload $T = 42$.

The average computational time for the FA and GA are respectively 1.07 and 2.02 seconds. In Figure 8 the solution (1) obtained by the FA is presented in a Gantt chart.



$C_{max} = 11, W_{max} = 10, T = 32.$

Figure 5: Gantt chart of the problem 4 × 5.



$C_{max} = 14, W_{max} = 12, T = 77.$

Figure 6: Gantt chart of the problem 8 × 8.

6.6 Problem 15 × 10

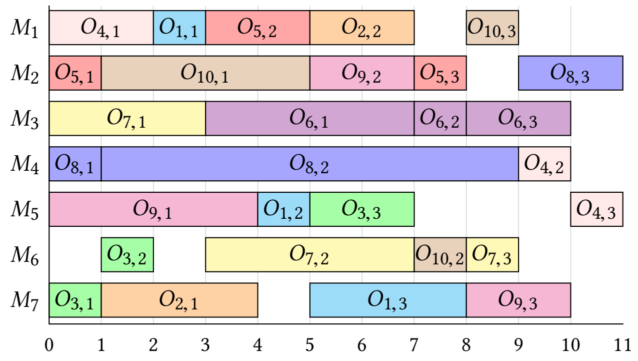
This is a large-scale T-FJSP instance, in which 15 jobs in a total of 56 operations are to be performed on 10 machines. The best solutions obtained by the proposed FA can be characterized by the following values:

(1) makespan $C_{max} = 11$, maximal workload $W_{max} = 11$, total workload $T = 93$.
(2) makespan $C_{max} = 12$, maximal workload $W_{max} = 11$, total workload $T = 91$.

The best solutions obtained by the proposed GA can be characterized as follows:

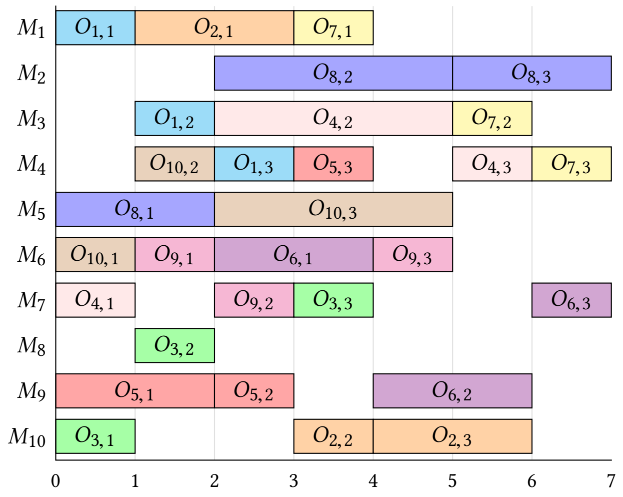
(1) makespan $C_{max} = 12$, maximal workload $W_{max} = 12$, total workload $T = 93$.
(2) makespan $C_{max} = 12$, maximal workload $W_{max} = 10$, total workload $T = 95$.

The average computational time for the FA and GA are respectively 6.86 and 18.76 seconds. In Figure 9 the solution (1) obtained by the FA is presented in a Gantt chart.



$C_{max} = 11, W_{max} = 10, T = 62.$

Figure 7: Gantt chart of the problem 10 × 7.

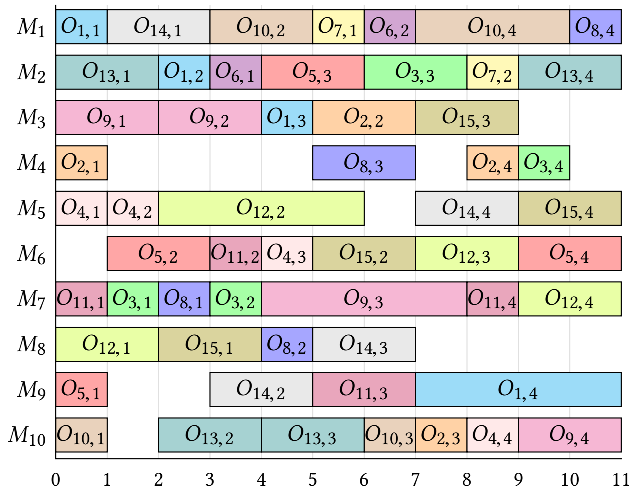


$C_{max} = 7, W_{max} = 5, T = 43.$

Figure 8: Gantt chart of the problem 10 × 10.

6.7 Performance Comparison

In addition to the comparison made among the proposed FA and GA, our approaches are compared with AL+CGA [11], PSO+SA [18], PSO+TS [23], MOGA [17], AIA [1]. MOGA is based on Pareto-optimality and the other use the same weighted summation of objectives. Table 4 presents the comparison of the results on the five cases. The three objectives are considered simultaneously. It can be seen from Table 4 that the proposed algorithms are comparable to the other algorithms and more efficient for small instances. The computational results of the proposed algorithm dominate the AL+GCA and the PSO+SA for solving the four instances, (i.e., 4 × 5, 8 × 8, 10 × 10, and 15 × 10). In comparison with the PSO+TS algorithm for solving the four instances, our approaches are more



$C_{max} = 11, W_{max} = 11, T = 93.$

Figure 9: Gantt chart of the problem 15 × 10.

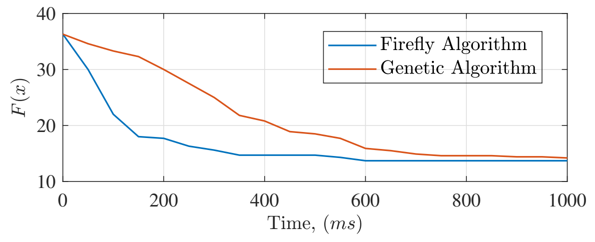


Figure 10: Convergence for the instance 10 × 10.

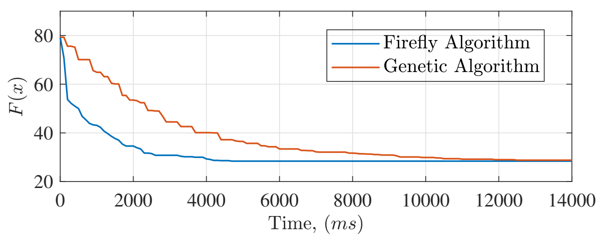


Figure 11: Convergence for the instance 15 × 10.

efficient and obtained richer optimal solutions. In comparison with the AIA algorithm for solving the four instances, our approaches obtained richer solutions with less computational resources. In comparison with the Pareto-based MOGA algorithm, our proposed approaches are more efficient, however less effective for larger instances. Table 4 shows that our proposed discrete FA when compared to the results obtained from the other algorithms that also uses weighted summation approach, it performs at the same level or better with respect to three objective functions in a very short time, for all five instances.

Based on the results of the above five instances, it can be seen that the proposed discrete firefly algorithm is more effective and efficient than the proposed genetic algorithm. In Figure 10 and 11, we draw the decrease of the average $F(x)$ over five runs for the instances 10×10 and 15×10. Note that the FA converges faster when compared to the GA.

7 CONCLUSION AND FUTURE WORK

The multi-objective flexible job-shop scheduling problem has attracted several researcher's attention. The complexity of this problem leads to the appearance of many meta-heuristics approaches. Currently, the research concentrates on the hybrid algorithms. In this paper, we put forward a comparison among a discrete firefly algorithm and a genetic algorithm for solving multi-objective FJSP. We combine the different objective functions into a weighted sum, where the objective function includes the minimization of makespan, maximal workload and total workload of machines. Experimental results on five instances show that when compared to the results obtained from the state-of-the-art methods that used weighted summation approach, the proposed discrete FA is more efficient for small instances and performed at the same level or better with respect to three objective functions. In addition, based on our experiment, the fact that firefly algorithm converges faster than the genetic algorithm, support the premise that the combination of FA with a local search method (e.g., tabu search) with an effective neighborhood function could generate better hybrids than those which uses GA for exploration.

Future research directions include combining the FA and local search methods to construct a hybrid algorithm to solve scheduling problems.

REFERENCES

[1] A Bagheri, Mostafa Zandieh, Iraj Mahdavi, and Mehdi Yazdani. 2010. An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems* 26, 4 (2010), 533–541.

[2] Peter Brucker and Rainer Schlie. 1990. Job-shop scheduling with multi-purpose machines. *Computing* 45, 4 (1990), 369–375.

[3] 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 (2016), 551–591.

[4] Kalyanmoy Deb and Deb Kalyanmoy. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA.

[5] Yunus Demir and S Kürşat İşleyen. 2013. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling* 37, 3 (2013), 977–988.

[6] LM Gambardella and M Mastrolilli. 1996. Effective neighborhood functions for the flexible job shop problem. *Journal of scheduling* 3, 3 (1996), 3–20.

[7] Michael R Garey, David S Johnson, and Ravi Sethi. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research* 1, 2 (1976), 117–129.

[8] David E Goldberg and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms* 1 (1991), 69–93.

[9] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA.

[10] Imed Kacem, Slim Hammadi, and Pierre 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 (2002), 1–13.

[11] Imed Kacem, Slim Hammadi, and Pierre Borne. 2002. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and computers in simulation* 60, 3 (2002), 245–276.

[12] S Karthikeyan, P Asokan, S Nickolas, and Tom Page. 2015. A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. *International Journal of Bio-Inspired Computation* 7, 6 (2015), 386–401.

[13] Eugene L Lawler, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science* 4 (1993), 445–522.

[14] Xinyu Li and Liang Gao. 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics* 174 (2016), 93–110.

[15] Michael Lohrer. 2013. *A comparison between the firefly algorithm and particle swarm optimization.* Ph.D. Dissertation. Oakland University.

[16] F Pezzella, G Morganti, and G Ciaschetti. 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* 35, 10 (2008), 3202–3212.

[17] 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 (2010), 757–767.

[18] Weijun Xia and Zhiming Wu. 2005. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering* 48, 2 (2005), 409–425.

[19] Li-Ning Xing, Ying-Wu Chen, Peng Wang, Qing-Song Zhao, and Jian Xiong. 2010. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing* 10, 3 (2010), 888–896.

[20] Xin-She Yang. 2010. *Nature-inspired metaheuristic algorithms.* Luniver press.

[21] Chaoyong Zhang, Peigen Li, Yunqing Rao, and Shuxia Li. 2005. A new hybrid GA/SA algorithm for the job shop scheduling problem. *Evolutionary computation in combinatorial optimization* (2005), 246–259.

[22] Guohui Zhang, Liang Gao, and Yang Shi. 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications* 38, 4 (2011), 3563–3573.

[23] Guohui Zhang, Xinyu Shao, Peigen Li, and Liang Gao. 2009. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering* 56, 4 (2009), 1309–1318.