A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems

This item was submitted to Loughborough University's Institutional Repository by the/an author.


Additional Information:

- This paper was accepted for publication in the journal International Journal of Bio-Inspired Computation and the definitive published version is available at http://dx.doi.org/10.1504/IJBIC.2015.073165

Metadata Record: https://dspace.lboro.ac.uk/2134/17676

Version: Accepted for publication

Publisher: © Inderscience

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
Biographical Notes: S. Karthikeyan received his graduation in Mechanical Engineering from University of Madras, Chennai, India and post-graduation in Manufacturing Technology from National Institute of Technology, Tiruchirappalli, India. Currently, he is pursuing PhD in the Department of Production Engineering, National Institute of Technology, Tiruchirappalli, India. His research interests focus on scheduling and manufacturing system optimization. He is an active life member of Indian Society of Technical Education.

P. Asokan received his graduation in Mechanical Engineering and post-graduation in Manufacturing Technology from Regional Engineering College, Tiruchirappalli, India. He received his PhD from Bharathidasan University, Tiruchirappalli, India in 1998. Presently he is serving as a Professor in Department of Production Engineering, National Institute of Technology, Tiruchirappalli, India. He has nearly 24 years of teaching experience. His area of specialization is operation management and optimization in manufacturing. He has published 55 papers in international journals and 33 papers in international conferences.

S. Nickolas is currently working as Associate Professor in the Department of Computer Applications, National Institute of Technology, Tiruchirappalli, India. He received his M.E. in the field of Computer Science from Regional Engineering College, Tiruchirappalli, India and PhD from National Institute of Technology, Tiruchirappalli, India in 2007. He has published 20 papers in International Conferences and 10 papers in International Journals. His area of interest includes Database systems, Data mining, Software metrics and Distributed systems.

Tom Page is a Senior Lecturer of Electronic Product Design at Loughborough University. He received his PhD from the University of Hertfordshire. His research interests are in engineering design, value management, technology education and electronic product design. He has been a consultant for many Small and Medium sized Enterprises in engineering design and electronic product design. His work has been widely published in the form of journal papers, book contributions, refereed proceedings, refereed conference papers and technical papers.
A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems

Abstract

Firefly Algorithm (FA) is a nature-inspired optimization algorithm that can be successfully applied to continuous optimization problems. However, lot of practical problems are formulated as discrete optimization problems. In this paper a hybrid discrete firefly algorithm (HDFA) is proposed to solve the multi-objective flexible job shop scheduling problem (FJSP). FJSP is an extension of the classical job shop scheduling problem that allows an operation to be processed by any machine from a given set along different routes. Three minimization objectives - the maximum completion time, the workload of the critical machine and the total workload of all machines are considered simultaneously. This paper also proposes firefly algorithm’s discretization which consists of constructing a suitable conversion of the continuous functions as attractiveness, distance and movement, into new discrete functions. In the proposed algorithm discrete firefly algorithm (DFA) is combined with local search (LS) method to enhance the searching accuracy and information sharing among fireflies. The experimental results on the well-known benchmark instances and comparison with other recently published algorithms shows that the proposed algorithm is feasible and an effective approach for the multi-objective flexible job shop scheduling problems.

Keywords: firefly algorithm; hybrid discrete firefly algorithm; HDFA; flexible job shop scheduling; FJSP; discrete firefly algorithm; DFA; multi-objective optimization

1. Introduction

Scheduling is the decision-making process which involves the allocation of resources over a period of time to perform a collection of tasks. Job-shop scheduling problem (JSP) is one of the hardest combinatorial optimization problems (Jain and Meeran, 1999), in the branch of production scheduling. The classical JSP consists of scheduling a set of jobs on a set of machines with the objective to minimize a certain criterion, subject to the constraint that each job has a specified processing order throughout. It is well known that this problem is NP-hard (Garey et al., 1976).

The flexible job shop scheduling problem (FJSP) is the generalization of the classical job shop problem, where operations are allowed to be processed on any machine from a given set along different routes. Flexible job shop scheduling problem possess many applications in competitive environment. For example, it is used in flexible manufacturing systems. A flexible manufacturing system consists of several computer numerical control (CNC) machines. A CNC machine is a multitasking machine. So flexible job shop scheduling is applicable for flexible manufacturing systems. Bruker and Schlie (1990) were among the first to address this problem. It is closer to the real manufacturing situation. It incorporates all difficulties and complexities of its predecessor JSP and is more complex than JSP due to the need to determine the assignment of operations to machines.

Due to the complexity of FJSP, no exact method has so far been introduced to tackle these problems within a reasonable amount of time. Hence, a variety of heuristic procedures such as dispatching rules, local search and meta-heuristic algorithms such as tabu search (TS), genetic algorithm (GA), particle swarm optimization (PSO), simulated
annealing (SA), variable neighbourhood search (VNS), artificial bee colony algorithm (ABC), artificial immune algorithm (AIA) and biogeography-based optimization algorithm (BBO) have been applied to solve these problems and find the optimal or near optimal schedule in a reasonable time. FJSP could be decomposed into two sub-problems of routing and scheduling. The routing sub-problem assigns each operation to a machine out of a set of capable machines authorized for each job. The scheduling sub-problem involves sequencing the operations assigned to the machines in order to obtain a feasible schedule that minimizes a predefined objective (Xia and Wu, 2005).

For solving the realistic case with more than two jobs, two types of approach have been used: hierarchical approach and integrated approach. Hierarchical approach was firstly proposed by Brandimarte (1993). In hierarchical approach, assigning operations to machines and the sequencing of operations on the machines are accomplished separately. Its basic idea is to divide a hard problem into two simpler sub-problems in order to decrease the complexity. Kacem et al. (2002a) proposed a localization approach to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for FJSP. Xia and Wu (2005) used particle swarm optimization (PSO) to assign operations to machines and simulated annealing (SA) algorithm to schedule operation on each machine for solving multi-objective FJSP. However, the integrated approach solves the assignment sub-problem and sequencing sub-problem simultaneously, such as greedy heuristics (Mati et al., 2001), simulated annealing (SA) algorithm (Hapke et al., 2000), genetic algorithm (Chan et al., 2006), tabu search (Scrigh et al., 2004) and particle swarm optimization (PSO) algorithm (Girish and Jawahar, 2009).

Most of the research on FJSP has been concentrated on mono-objective. However several objectives must be considered simultaneously in the real world production situation and these objectives often conflict with each other. In the recent years, multi-objective FJSP (MOFJSP) has gained attention of some researchers. The methods to solve the MOFJSP can be roughly classified into two types: weighted summation approach and Pareto-based approach. The weighted summation approach solves the MOFJSP by transforming it to a mono-objective one by giving each objective a weight. The Pareto-based approach solves the MFJSP based on the Pareto optimality concept and aims at generating the set of Pareto optimal solutions. The existing algorithms belonging to the first type include localization approach to solve the resource assignment problem (Kacem et al., 2002a), hybrid localization approach with PSO and SA (Xia & Wu, 2005), hybrid genetic algorithm (GA) (Gao et al., 2008), hybrid PSO algorithm with tabu search (TS) (Zhang et al., 2009), efficient search method (Xing et al., 2009), effective hybrid tabu search algorithm (HTSA) (Li et al., 2010), hybrid PSO algorithm and data mining (Karthikeyan et al., 2012). The existing algorithms belonging to the second type include the Pareto based algorithm which combines fuzzy logic and evolutionary algorithms (Kacem et al., 2002b), multi-objective genetic algorithm (MOGA) (Wang et al., 2010), Memetic Algorithm based on NSGAII (Frutos et al., 2010), Pareto-based discrete artificial bee colony algorithm (Li et al., 2011), multi objective particle swarm optimization (Moslehli and Mahnam, 2011), multi objective evolutionary algorithm (Chiang and Lin, 2013), and hybrid shuffled frog-leaping algorithm (HSFLA) (Li et al., 2012).

Evolutionary Algorithms are stochastic search method that mimics the metaphor of natural biological evolution and/or the social behaviour of species. García-Gonzalo et al. (2012) presented a brief historical review of PSO, insisting in the importance of the stochastic stability analysis of the particle trajectories in order to achieve convergence. Bat algorithm (BA) is a bio-inspired algorithm developed by Yang, (2010) based on the echolocation features of microbats. Recently Yang and He (2013) provided a review of the bat algorithm and its new variants. Artificial plant optimization algorithm is a recent
proposed population based stochastic algorithm. It is inspired by the natural plant growing process. Bing Yu et al. (2013) used artificial plant optimization algorithm with correlation branches to test the performance of the unconstrained multi-modal benchmark problems. Firefly Algorithm (FA) was introduced by Yang, (2008). It is a meta-heuristic algorithm inspired by the social behaviour of fireflies. Yang (2009) proposed a firefly algorithm for multimodal optimization applications. Lukasik and Zak (2009) presented a further study on the firefly algorithm for constrained continuous optimization problems. Sayadi et al. (2010) presented a discrete firefly algorithm to minimize makespan for the flow shop scheduling problems. A discrete firefly algorithm was proposed by Jati, (2011) to solve the travelling salesman problem. Khadwillard et al. (2012) solved the job shop scheduling problems using firefly algorithm. They also investigated different parameters for the proposed algorithm and compared the performance with different parameters. Marichelvam et al. (2012) proposed a discrete firefly algorithm using the SPV rule for the multi-objective hybrid flow shop scheduling problems. The traditional firefly algorithm is a population based technique for solving continuous optimization problems, especially for continuous NP-hard problems. The learning process is based on the real number such that, the standard firefly algorithm cannot be directly applied to solve the discrete optimization problems.

To the best of our knowledge, there is no published work dealing with the multi-objective FJSP by using DFA. Thus, in this paper we proposed a hybrid algorithm combining DFA with a local search approach to solve the multi-objective FJSP. In the proposed algorithm, rules are presented for generating the initial population with a high level of quality. This paper describes firefly algorithm’s discretization, which consists of constructing a suitable conversion of the continuous functions such as attractiveness, distance and movement, into new discrete functions. DFA allows an extensive search for the solution space while the local search method is employed to reassign the machines to operations and to reschedule the results obtained from DFA, which will enhance the convergence speed. The objectives considered in this paper are to minimize maximal completion time, the workload of the critical machine and the total workload of machines simultaneously.

The remainder of this paper is organized as follows. The formulation and notation of multi-objective FJSP are introduced in Section 2. Section 3 describes the traditional firefly algorithm. In Section 4, the proposed approach is presented to solve the FJSP. Section 5 shows the computational results and its comparison with other algorithms. Finally, Section 6 provides conclusions and further research.

2. Problem Formulation

The flexible job-shop scheduling problem can be formulated as follows. There is a set of n jobs J (J = {J1, J2, ..., Jn}) supposed to be processed on a set of m machines M (M = {M1, M2, ..., Mk}). For each job Ji, consists of a sequence of ni operations. Each operation Oij (i = 1, 2, ..., m; j = 1, 2, ..., ni) of job (Ji) can be processed on any subset Mi,j ⊆ M of compatible machines. For job Ji, Pij denotes the processing time of operation j (Oi,j) on machine k. The FJSP is needed to determine both an assignment and sequence of the operations of the machines in order to satisfy the given criteria. However, the FJSP is more complex and challenging than the classical JSP because it requires a proper selection of machines from a set of available machines to process each operation of each job (Ho et al., 2007). The flexibility of problems can be categorised into partial flexibility and total flexibility. If there is Mi,j ⊂ M for at least one operation, it is partial flexibility FJSP (P-FJSP); while there is Mi,j = M for each operation, it is total flexibility FJSP (T-FJSP) (Kacem et al.
In this study, the following objectives are to be minimized:

1. Makespan ($C_m$) of the jobs, i.e. the completion time of all jobs
2. Maximal machine workload ($W_m$), i.e. the maximum working time spent on any machine
3. Total workload of the machines ($W_t$) which represents the total working time over all machines.

The following assumptions are also considered:

1. Move time between operations and setup time of machines are ignored.
2. Machines are independent from each other.
3. Jobs are independent from each other.
4. Pre-emption is not allowed, i.e., each operation cannot be interrupted before its completion on the assigned machine.
5. At a given time, a machine can execute only one operation.
6. There are no precedent constraints among the operations of different jobs.

The notations used in this study are listed as follows:

- $i, h$: index of jobs, $i, h = 1, 2, \ldots, n$
- $j, g$: index of operation sequence, $j, g = 1, 2, \ldots, n_i$
- $k$: index of machines, $k = 1, 2, \ldots, m$
- $n$: total number of jobs
- $n_m$: total number of machines
- $n_{ij}$: total number of operations of job $i$
- $O_{ij}$: the $j$th operation of job $i$
- $M_{ij}$: the set of available machines for the operation $O_{ij}$
- $P_{ijk}$: processing time of operation $O_{ij}$ on machine $k$
- $t_{ijk}$: start time of operation $O_{ij}$ on machine $k$
- $C_{ij}$: completion time of the operation $O_{ij}$
- $C_k$ is the completion time of $M_k$
- $W_k$ is the workload of $M_k$.

Decision variable

$$x_{ijk} = \begin{cases} 1, & \text{if machine } k \text{ is selected for the operation } O_{ij} \\ 0, & \text{otherwise} \end{cases}$$

Our model is presented as follows:

$$\min f_1 = \max_{1 \leq k \leq m} (C_k) \quad (1)$$

$$\min f_2 = \max_{1 \leq k \leq m} (W_k) \quad (2)$$

$$\min f_3 = \sum_{k=1}^{m} W_k \quad (3)$$

Subject to:
\[ C_{ij} - C_{i(j-1)} \geq P_{ijk} X_{ijk}, j = 2, \ldots, n; \forall i, j \] (4)

\[ [(C_{h,g} - C_{ij} - t_{jik}) X_{h,g,k} \geq 0] \lor [(C_{ij} - C_{h,g} - t_{ijk}) X_{h,j} X_{ijk} \geq 0], \forall (i,j), (h,g), k \] (5)

\[ \sum_{k \in M_{ij}} X_{ijk} = 1, \forall i, j \] (6)

Equation (1) ensures the minimization of maximal completion time of the machines. Equation (2) ensures the minimization of maximal machine critical workload among all the machines available. Equation (3) ensures the minimization of total workload of machines. Inequality (4) ensures the operation precedent constraint. Inequality (5) ensures that each machine could process only one operation at each time when the first or second condition mentioned in the constraint satisfies all stated elements. Equation (6) states that one machine could be selected from the set of machines for each operation.

Many approaches have been formulated to solve the multi-objective optimization. These approaches can be classified into three categories (Hsu et al., 2002).

1. Transform the multi-objective problem to a mono-objective problem by a weighted sum approach
2. The non-Pareto approach deals with different objectives in a separated way
3. The Pareto approach based on the Pareto optimality concept.

The objective function in this paper is based on the first type of approach described above. The weighted sum of the three objective values is taken as the objective function:

\[ \text{Minimize } F(c) = W_1 \times f_1 + W_2 \times f_2 + W_3 \times f_3 \] (7)

Subject to:

\[ W_1 + W_2 + W_3 = 1, \quad 0 \leq W_1, W_2, W_3 \leq 1 \] (8)

where \( F(c) \) denotes the combined objective function value of a schedule, \( f_1, f_2 \) and \( f_3 \) which denotes the makespan (\( C_m \)), maximal machine workload (\( W_m \)) and total workload of machines (\( W_t \)) respectively. \( W_1, W_2, \) and \( W_3 \) represent the weight coefficient for the three objective values, which could be set for different values depending upon the requirement. If the decision maker pays more attention to a certain objective, a large weight is defined to it. Otherwise, a small weight for the given objective can be defined. In this work the weight coefficients \( W_1, W_2, \) and \( W_3 \) for the five Kacem instances are set to 0.5, 0.3 and 0.2 according to Xing et al. (2009a, 2009b). The advantage of utilizing the weighted summation approach is its algorithmic actualization which is effortless and the users can change the weight of different objectives for satisfying the requirements of decision makers.

3. Firefly algorithm

Firefly Algorithm (FA) is a recently developed nature-inspired meta-heuristic algorithm. The firefly algorithm is inspired by the social behaviour of fireflies. Most of the fireflies produce short and rhythmic flashes and have different flashing behaviour. Fireflies use these flashes for communication and attracting the potential prey. The swarm of fireflies
will move to brighter and more attractive locations by the flashing light intensity that is associated with the objective function of problems considered, in order to obtain efficient optimal solutions.

## Insert Figure 1 ##

In essence, FA uses the following three idealized rules (Yang, 2008): (1) all fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex; (2) attractiveness of a firefly is proportional to their brightness, thus for any two flashing fireflies, the less brighter one will move towards the brighter one. The attractiveness is proportional to the brightness and they both increase as their distance decreases. If there is no brighter one than a particular firefly, it will move randomly; (3) the brightness of a firefly is determined by the value of the objective function. For a maximization problem, the brightness may be proportional to the objective function value. For minimization problem the brightness may be the reciprocal of the objective function value. The basic steps of the FA are summarized by the pseudo code shown in Figure 1 which consists of three rules discussed above.

Based on Yang (2009), FA is very efficient in finding the global optimal value with high success rates. Simulations and results indicate that FA is superior to both PSO and GA in terms of both efficiency and success rate. These facts give inspiration to investigate how to find optimal solution using FA in solving FJSP. The challenges are how to compute discrete distance between two fireflies and how they move in coordination. The following issues are important in this algorithm.

### 3.1 Attractiveness ###

The attractiveness of a firefly is determined by its light intensity. Each firefly has its distinctive attractiveness $\beta$ which implies how strong it attracts other members of the swarm. The form of attractiveness function of a firefly is the following monotonically decreasing function (Lukasik and Zak, 2009)

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

where $r$ is the distance between two fireflies, $\beta_0$ is the attractiveness at $r = 0$ and $\gamma$ is a fixed light absorption coefficient.

### 3.2 Distance ###

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

$$
r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2}
$$

where $x_{i,k}$ is the $k^{th}$ component of the spatial coordinate $x_i$ of the $i^{th}$ firefly and $d$ is the number of dimensions.

### 3.3 Movement ###

The movement of a firefly $i$ which is attracted by a more attractive (i.e., brighter) firefly $j$ is given by the following equation (Lukasik and Zak, 2009)

$$
x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha (rand - 1/2)
$$

where the first term is the current position of a firefly, the second term is used for considering a firefly’s attractiveness to light intensity seen by adjacent fireflies, and the third term is used for the random movement of a firefly if there is no brighter ones. The coefficient $\alpha$ is a randomization parameter determined by the problem of interest, while
rand is a random number generator uniformly distributed in the space [0, 1]. In this implementation of the algorithm, we will use $\beta_0 = 1.0$, $\alpha \in [0, 1]$ and the attractiveness or absorption coefficient $\gamma$ is in the interval [0.01, 0.15], which guarantees a quick convergence of the algorithm to the optimal solution.

4. Discrete Firefly Algorithm

The firefly algorithm has been originally developed for solving continuous optimization problems. The firefly algorithm cannot be applied directly to solve the discrete optimization problems. In this study, we propose a possible way that can be modified to solve the class of discrete problems, where the solutions are based on discrete job permutations. In DFA, the target individual is represented by two vectors: one is for the machine assignment and the other is for permutation of jobs. Hamming distance is used to measure the distance between two permutations. The Hamming distance has been already applied by the researchers in the scheduling problems. Movement is implemented by breaking the attraction step into two sub steps as $\beta$-step and $\alpha$-step.

## Insert Table 1 ##

4.1 Discrete firefly algorithm for multi-objective FJSP

4.1.1 Solution representation

The FJSP problem is a combination of machine assignment and operation scheduling decisions, so the solution can be expressed by the assignment of operations on machines and the processing sequence of operations on the machines. In this study, we used improved A-string (machine assignment) and B-string (operation scheduling) representation for each firefly that could be used to solve the multi-objective FJSP efficiently and avoid the use of a repair mechanism (Zhang et al., 2009).

**Machine assignment**: An array of integer values is used to represent A-string. The length of the array is equal to the sum of all operations of all jobs. Each integer value equals the index of the array of alternative machine set of each operation. An example of A-string from the problem stated in Table 1 is shown in Figure 2. The length of the A-string is 8. The value in each cell of the array represents that the particular operation is assigned to that machine number i.e., 2 is mentioned in the first cell of the array which means that the operation $O_{1,1}$ is assigned to the 2nd machine M2.

## Insert Figure 2 ##

**Operation Scheduling**: B-string has the same length as the A-string as shown in Figure 3. It consists of a sequence of job numbers in which the job number $i$ occurs $n_i$ times. It can avoid creating an infeasible schedule when replacing each operation by the corresponding job index. From Table 1, a possible B-string may be 2-1-3-2-3-1-1-2. Read the data from left to right, the B-string could be translated into a list of ordered operations: $O_{2,1}, O_{1,1}, O_{3,1}, O_{2,2}, O_{3,2}, O_{1,2}, O_{1,3}, O_{2,3}$. The operation sequence of B-String may be represented in terms of position like 4-1-7-5-8-2-3-6. When a particle is decoded, B-string is converted to a sequence of operations at first. Then each operation is assigned to a processing machine according to A-string as follows: $[(O_{2,1}, M_1), (O_{1,1}, M_2), (O_{3,1}, M_2), (O_{2,2}, M_3), (O_{3,2}, M_1), (O_{1,2}, M_4), (O_{1,3}, M_3), (O_{2,3}, M_4)]$.

## Insert Figure 3 ##

4.1.2 Population Initialization

The quality of the initial population has a greater effect on the performance of an algorithm. A good initial population locates promising areas in the search space and
provides enough diversity to avoid premature convergence.

**Machine assignment component initial rules:** Initiate the machine assignment component of the population using the following two rules: the operation minimum processing time rule (Pezzella et al., 2008) denoted by \( R_1 \), the global machine workload balance rule (Li et al., 2012) denoted by \( R_2 \).

**Scheduling component initial rules:** The scheduling component considers how to sequence the operations at each machine, i.e., to determine the start time of each operation. Following are initial approaches for scheduling component: the most work remaining (MWR) rule (Brandimarte, 1993) denoted by \( S_1 \), the most number of operations remaining (MOR) rule (Pezzella et al., 2008) denoted by \( S_2 \).

Finally, one part of the initial population is achieved by using the above initial approaches and the remaining population is generated by a simple random rule. To consider both the problem features and solution quality, in the first part of the population, machine assignment components and scheduling components are generated according to percentage of population size given for rules \( R_1 \), \( R_2 \), \( S_1 \), and \( S_2 \) respectively. All other solutions in the initial population are generated randomly to enhance the diversity of the population.

### 4.1.3 Firefly evaluation

Each firefly is represented by A-string and B-string. By using the permutation, each firefly is evaluated to determine the objective function. The objective function value of each firefly is associated with the light intensity of the corresponding firefly. In this work, the evaluation of the goodness of schedule is measured by the combined objective function which can be calculated using equation (7). Table 2 shows the firefly representation with combined objective function value for the example problem given in Table 1. The number of fireflies (population size) used in this problem is 10. The best firefly \( (P_{\text{best}}) \) based on the combined objective function value is population \( P-7 \).

### 4.1.4 Solution updation

In firefly algorithm, firefly movement is based on light intensity and comparing it between two fireflies. The attractiveness of a firefly is determined by its brightness which in turn is associated with the encoded objective function. Thus for any two fireflies, the less bright one will move towards the brighter one. If no one is brighter than a particular firefly, it will move randomly. In this work, discretization is done for the following issues:

**Distance:** There are two possible ways to measure the distance between two permutations: (a) Hamming distance and (b) the number of the required swaps of the first solution in order to get the second one. In A-string the distance between any two fireflies \( i \) and \( j \), at positions \( x_i \) and \( x_j \), respectively can be measured by using Hamming distance. The Hamming distance between two permutations is the number of non-corresponding elements in the sequence (Kuo et al., 2009). The example of Hamming distance is given as follows: If there are two A-strings in the FJSP solution space which are \( P = \{2 \ 4 \ 3 \ 1 \ 3 \ 4 \ 2 \ 1\} \) and \( P_{\text{best}} = \{1 \ 4 \ 1 \ 3 \ 2 \ 2 \ 3 \ 4 \} \), we compare every bit of two strings and record the number of bits whose machine indices are not equal. Hence, the Hamming distance \( (P, P_{\text{best}}) \) is 7. The distance between two permutations in the B-string can be measured by using swap distance. The swap distance is the number of minimal required swaps of one permutation in order to obtain the other one. For example let us consider B-string representation of two fireflies \( P = \{4 \ 1 \ 7 \ 5 \ 8 \ 2 \ 3 \ 6\} \) and \( P_{\text{best}} = \{1 \ 4 \ 7 \ 5 \ 2 \ 3 \ 6 \ 8\} \). Swap distance \( (P, P_{\text{best}}) \) is thereby, 4.
Attraction and Movement: Attraction and movement has to be implemented and interpreted for FJSP in the same way as it is intended for the continuous firefly algorithm. In this study, we can break up an attraction step given in equation (11) into two sub-steps: \(\beta\)-step and \(\alpha\)-step as given in equation (12) and equation (13) respectively. We can do this, since we know that the result will not change.

\[
x_i = \beta(r)(x_j - x_i) \tag{12}
\]

\[
x_i = x_i + \alpha(rand - 1/2) \tag{13}
\]

The attraction steps \(\alpha\) and \(\beta\) are not interchangeable. The \(\beta\)-step must be computed before the \(\alpha\)-step while finding the new position of the firefly.

\(\beta\) – Step: It brings the iterated firefly always closer to another firefly. In other words, after applying a \(\beta\)-step on a firefly towards the other firefly, their distance is always decreased, and the decrement is proportional to their former distance. The steps involved in \(\beta\)-step are as follows:

- \(d\) = the difference between the elements of best firefly and other firefly
- \(r\) = hamming distance for A-string and swap distance for B-string

An insertion mechanism and a pair-wise exchange mechanism are used to advance the A-string and B-string part of each firefly position towards the best firefly position. At first, the difference between the current position of the firefly and its best position is found by comparing their corresponding elements of the A-string part and B-string part separately. All necessary insertions in the A-string part and all necessary pair-wise exchanges in the B-string part, to make the elements of the current firefly equal to best firefly, are found and stored in \(d\). Counting the number of insertions and pair-wise exchanges between two firefly will give the hamming distance and swap distance respectively, which is stored in \(r\). Then we need to compute the \(\beta\) probability. Since it is often faster to calculate \(1 / (1 + r^2)\) than an exponential function (Pal et al., 2012), the equation (9) can be approximated as

\[
\beta(r) = \beta_0/(1 + \gamma r^2) \tag{14}
\]

Secondly, each insertion/pair-wise exchange in the set \(d\) is accessed and a random number \(rand(\cdot)\) is generated in the range \((0, 1)\). If \(rand(\cdot) \leq \beta\), then the corresponding insertion/pair-wise exchange is performed on the elements of the current firefly. This procedure moves the current firefly to the global best position, which is controlled by the \(\beta\) probability. This procedure is repeated for all the fireflies.

\(\alpha\) – Step: It is much simpler than the \(\beta\)-step. In equation (13) the random movement of firefly \(\alpha \cdot (rand - l/2)\) is approximated as \(\alpha \cdot (rand_{int})\) given in equation (15) which allows us to shift the permutation into one of the neighbouring permutations. \(\alpha\)-step is applied by

\[
x_i = x_i + \alpha(rand_{int}) \tag{15}
\]

choosing an element position using \(\alpha \cdot (rand_{int})\) and swap with another position in the string which is also chosen at random. The random number \(rand_{int}\) is a positive integer generated between the minimum and maximum number of elements in the string.

Table 3 illustrates the firefly position update procedure for population 1 in the first generation. The parameters used in this illustration are as follows: \(\beta_0 = 1\), \(\gamma = 0.1\), \(\alpha = 1\). The improvement of objective function value \(\text{F(c)}\) from 19.9 to 14.8 shows the movement of firefly from its current solution to the best firefly solution. The objective function of each firefly population obtained in the first generation replaces its previous value \((P)\), if its current solution is better than the previously stored firefly solution. The best solution of the first iteration replaces the global best firefly solution \((P_{best})\) if it is better than the
previously stored global best solution. The procedure is repeated until the termination criterion is satisfied. In this study, the termination criterion is the total number of generations.

4.2 Local search

In each generation of the discrete firefly algorithm, we improve the quality of the solution (firefly) using a local search mechanism. The local search process is often performed to explore the neighbourhood of the generated solution for better ones. To enhance both the search exploitation and exploration ability of the algorithm, that is, to increase the capability of convergence to the near optimal solution by keeping the population diversity, several neighbourhood structures are proposed in the algorithm, which either considers the problem constraints or the problem objectives (Li et al., 2012). Local search moves from one solution to another solution in the neighbourhood have good feature in exploiting the promising search space, so as to find optimal solutions around a near-optimal solution. Following are the neighbourhood structures for machine assignment and scheduling component.

4.2.1 Neighbouring structure for machine assignment

Random neighbourhood \((L_{ma1})\): The neighbouring structure for machine assignment is to find the alternative machine randomly. Wang et al. (2010) used this neighbourhood structure as a mutation operator in Genetic Algorithm. The neighbourhood is generated by following steps: (1) select operation \(O_{ij}\) randomly from machine assignment vector of solution; (2) detect machine \(M_k\) that is currently selected to process \(O_{ij}\); (3) detect a set of machines \(M\) that can process \(O_{ij}\); (4) select a random machine from machine set \(M\) (except \(M_k\)) to process \(O_{ij}\).

Critical operations neighbourhood \((L_{ma2})\): The neighbouring structure is used to find the machine with most critical operations, and then find some of the critical operations to assign another machine for the operation (Li et al., 2012). The steps are as follows: (1) get the number of critical operations for each machine; (2) sort all machines in non-increasing order according to their number of critical operations; (3) select the first machine with the most number of critical operations and denote as \(M_{old}\). Randomly search an operation which is scheduled on \(M_{old}\), and denote as \(O_{ij}\); (4) assign a machine with relatively less critical operations from other machines which is capable of processing \(O_{ij}\), denoted \(M_s\), and schedule operation \(O_{ij}\) on \(M_s\); (5) replace the machine assignment component of the current solution at position \(O_{ij}\) with the value of \(M_s\).

4.2.2 Neighbourhood structure for operation scheduling

Critical operation swap neighbourhood \((L_{swap})\): The neighbourhood solution is generated by moving two critical operations on the critical path. It was proposed by Nowicki and Smutnicki, (1996) for job shop scheduling. For FJSP we have to make an extra check that the two operations to be swapped do not belong to the same job. The rules of swapping two operations on the critical path are as follows: (1) if the first (last) block contains more than two operations, we only swap the last (first) two operations in the block. Otherwise, if the first (last) block contains only two operations, these operations are swapped; (2) in each critical block, we only swap the last two and first two operations; (3) if a critical block contains only one operation, then no swap is made. The scheme of swapping between the two operations is shown in Figure 4 (C. Zhang et al., 2007).
Critical operation insert neighbourhood ($L_{\text{insert}}$): Dell'Amico and Trubian (1993) proposed the neighbourhood by moving one critical operation on the critical path of job shop scheduling. In this paper, we adopted for our problem by moving one operation shown in Figure 5. The neighbourhood is generated by following steps: (1) randomly select a critical block with at least two critical operations; (2) in each critical block, the first (or last) operation is inserted into the internal operation within the critical block; (3) in each critical block, the internal operation is inserted at the beginning or the end of the critical block; (4) if a critical block contains only one operation, then no swap is made.

The local search approach is carried out by randomly selecting one approach each from machine assignment neighbourhood structure and operation scheduling neighbourhood structure.

4.3 Hybrid algorithm framework

The proposed hybrid DFA could keep the balance of the global exploration and local exploitation, and it also stresses the diversity of the population during the searching process. Thus, it is expected to achieve good performances in solving the multi-objective flexible job shop scheduling problem. The framework of the proposed hybrid DFA is illustrated in Figure 6. During the operation process, the initialization is done with multiple strategies. Then, each individual in the population is evaluated. If the stop criterion is met, the non-dominated solution is output. Otherwise, update each firefly according to equations (12) and (15). In local search, neighbourhood structures are used to search the solution space. The algorithm is repeated until a termination criterion is met.

5. Computational results

This section describes the computational experiments used to evaluate the performance of the proposed algorithm. In order to conduct the experiment, we implement the algorithm in C++ on an Intel Core 2 Duo 2.0 GHz PC with 4 GB RAM memory. Two test instances are conducted and compared with other algorithms, i.e., five multi-objective Kacem instances from Kacem et al. (2002a), three multi-objective Kacem instances with release dates from Kacem et al. (2002a). The dimensions of the instances range from 4 jobs × 5 machines to 15 jobs × 10 machines. The best and average results of experiments from 20 different runs were collected for performance comparison.

5.1 Setting Parameter

Each instance can be characterized by the following parameters: number of jobs ($n$), number of machines ($m$), and the number of operations ($n_i$). The parameters of the hybrid algorithm consists of the initial population size $P$, rate of machine assignment initial rule $R_1$, rate of machine assignment initial rule $R_2$, rate of scheduling component initial rule $S_1$, rate of scheduling component initial rule $S_2$, maximum number of generations $\text{max}_\text{gen}$, neighbourhood size $\text{ite}_{\text{max}}$, attractiveness of fireflies $\beta_0$, light absorption coefficient $\gamma$, randomization parameter $\alpha$. The detailed parameters for the problem instances are presented in Table 4.
5.2 Performance comparison

Our proposed approach HDFA is compared with AL+CGA algorithm presented by Kacem et al. (2002b), the PSO+SA developed by Xia and Wu, (2005), the PSO+TS introduced by Zhang et al. (2009), the efficient search method (ESM) presented by Xing et al. (2009b) and the artificial immune algorithm (AIA) proposed by Bagheri et al. (2010). These algorithms are included in the group of approaches applied to FJSP that uses the weighted summation of objectives.

5.2.1 Test on Kacem instances

**Problem 4 × 5:** This is a small scale instance of total flexibility in which 4 jobs with 12 operations are to be performed on 5 machines. The computational results obtained by our proposed hybrid algorithm are characterized by the following values:

Solution

1. Makespan \((C_m) = 11\), Maximal workload \((W_m) = 10\), Total workload \((W_t) = 32\)
2. Makespan \((C_m) = 12\), Maximal workload \((W_m) = 8\), Total workload \((W_t) = 32\)

## Insert Figure 7 ##

The average computational time for 20 different runs is 0.13 s. In Figure 7 the Solution (1) is presented using Gantt chart. The representation \(J_{1,1}\) (job, operation) inside the block denotes the first operation of job 1, and so on. The hatched blocks represent the machine’s idle periods.

**Problem 8 × 8:** This is a middle scale instance of partial flexibility, in which 8 jobs with 27 operations are to be performed on 8 machines. The average computational time for 20 different runs is 3.53 s. The computational results obtained by our proposed hybrid algorithm are characterized by the following values:

Solution

1. Makespan \((C_m) = 14\), Maximal workload \((W_m) = 12\), Total workload \((W_t) = 77\)
2. Makespan \((C_m) = 15\), Maximal workload \((W_m) = 12\), Total workload \((W_t) = 75\)
3. Makespan \((C_m) = 16\), Maximal workload \((W_m) = 13\), Total workload \((W_t) = 73\)

## Insert Figure 8 ##

The schedule using Gantt chart representation corresponding to the Solution (1) is shown in Figure 8.

**Problem 10 × 7:** This is a middle scale instance of total flexibility in which 10 jobs with 29 operations are to be performed on 7 machines. The average computational time for 20 different runs is 2.36 s. The computational results obtained by our proposed hybrid algorithm are characterized by the following values:

Solution

1. Makespan \((C_m) = 11\), Maximal workload \((W_m) = 10\), Total workload \((W_t) = 62\)
2. Makespan \((C_m) = 11\), Maximal workload \((W_m) = 11\), Total workload \((W_t) = 61\)

## Insert Figure 9 ##

The Solution (1) representation in the form of a Gantt chart is shown in Figure 9.

**Problem 10 × 10:** This is a middle scale instance of total flexibility in which 10 jobs with 30 operations are to be performed on 10 machines. The average computation time for 20
different runs is 3.36 s. The computational results obtained by our proposed hybrid algorithm are characterized by the following values:

Solution

(1) Makespan \( (C_m) = 7 \), Maximal workload \( (W_m) = 5 \), Total workload \( (W_t) = 43 \)
(2) Makespan \( (C_m) = 7 \), Maximal workload \( (W_m) = 6 \), Total workload \( (W_t) = 42 \)
(3) Makespan \( (C_m) = 8 \), Maximal workload \( (W_m) = 7 \), Total workload \( (W_t) = 41 \)

Figure 10 shows the result of Solution 2 in the form of a Gantt chart.

Problem 15 × 10: This is a large scale instance of total flexibility, in which 15 jobs with 56 operations are to be performed on 10 machines. The average computational time for 20 different runs is 19.31 s. The computational results obtained by our proposed hybrid algorithm are characterized by the following values:

Solution

(1) Makespan \( (C_m) = 11 \), Maximal workload \( (W_m) = 11 \), Total workload \( (W_t) = 91 \)
(2) Makespan \( (C_m) = 11 \), Maximal workload \( (W_m) = 10 \), Total workload \( (W_t) = 93 \)

The Solution (1) representation in the form of a Gantt chart is shown in Figure 11.

Table 5 shows the comparison of the results on the above five Kacem instances. The three objectives are considered simultaneously, i.e. minimization of the makespan \( (C_m) \), the maximal workload \( (W_m) \), and the total workload \( (W_t) \). It can be seen from Table 5, that HDFA is comparable to other algorithms for solving the five Kacem instances. The computational results of the proposed algorithm dominate the results of the AL + CGA for solving all the four instances, i.e., problem 4 × 5, 8 × 8, 10 × 10, and 15 × 10. For comparison with PSO + SA algorithm, HDFA can either be obtained by more non-dominated solutions or can be obtained by superior results in solving the three instances (i.e., 8 × 8, 10 × 10, and 10 × 15). In comparison with PSO + TS algorithm for solving the four instances (i.e., 4 × 5, 8 × 8, 10 × 10, and 15 × 10), our approach also obtain richer optimal solutions or dominated solutions. For solving the large scale instances such as 10 × 10 and 15 × 10, HDFA obtained superior result than PSO + TS. In comparison with the SME algorithm for solving the five instances (i.e., 4 × 5, 8 × 8, 10 × 7, 10 × 10, and 15 × 10), HDFA either obtains superior solutions or obtained non-dominated solutions with the same weightage set. The proposed algorithm obtained richer non-dominated solutions than the SME algorithm for middle scale instances 8 × 8 and 10 × 10. For solving three instances (i.e., 8 × 8, 10 × 10, and 15 × 10), our approach obtained three non-dominated solutions, while AIA can only be obtained one non-dominated solution for the considered instance. Table 5 shows that the proposed algorithm performed at the same level or better with respect to three objective functions in a very short time for the all five instances, when compared to the results obtained from the other methods. It proves that HDFA is efficient and effective.

5.2.2 Test on Kacem instances with release time

The second test instance compares the performances of HDFA with the FL + EA algorithm presented by Kacem et al. (2002b), the MOPSO + LS proposed by Moslehi and Mahnam (2011) on three FJSP instances with release time for each job. Kacem’s instances of 4 × 5,
10 \times 7, and 15 \times 10 with different release times are given as follows:

**Instance 1 (4 \times 5):** \( r_1 = 3, r_2 = 5, r_3 = 1, r_4 = 6 \).

**Instance 2 (10 \times 7):** \( r_1 = 2, r_2 = 4, r_3 = 9, r_4 = 6, r_5 = 5, r_6 = 7, r_7 = 4, r_8 = 1, r_9 = 0 \).

**Instance 3 (15 \times 10):** \( r_1 = 5, r_2 = 3, r_3 = 6, r_4 = 4, r_5 = 9, r_6 = 7, r_7 = 2, r_8 = 8, r_9 = 0, r_{10} = 0, r_{11} = 14, r_{12} = 13, r_{13} = 11, r_{14} = 12, r_{15} = 5 \).

## Insert Table 6 ##

It can be seen from Table 6 that our algorithm is comparable to other algorithms for solving the above three instances. The computational results of the proposed algorithm dominate the results of the FL+ EA for solving all the three instances, i.e., problem 4 \times 5, 10 \times 7, and 15 \times 10. For solving the three instances, HDFA can obtain the same multi set of optimal solutions with MOPSO + LS in a very short computational time.

6. Conclusions

In this paper, an efficient hybrid discrete firefly algorithm (HDFA) is proposed to solve the multi-objective flexible job shop scheduling problems where the objective function include the minimization of makespan, maximal workload and total workload of machines. In our algorithm, we proposed the discrete versions of the continuous function such as distance, attractiveness and movement to update a firefly position. A combination of rules is utilized for generating the initial population. In addition, two neighbourhood structures in relation to machine assignment and operation sequence were used in the algorithm to direct the local search to the more promising search space. Experimental results on two test instances show that our algorithm is comparable to other recently published algorithms for solving the flexible job shop scheduling problems. In future, flexible job shop scheduling problems with additional constraints such as maintenance requirements or breakdown could be solved by HDFA. Furthermore, the proposed HDFA can also be applied to other kinds of combinatorial optimization problems.

References


Figure 1  Pseudo code of the firefly algorithm

Firefly Algorithm

Objective function $f(x)$, $x = (x_1, ..., x_d)^T$

Generate initial population of fireflies $x_i (i = 1, 2, ..., n)$

Light intensity $I_i$ at $x_i$ is determined by $f(x_i)$

Define light absorption coefficient $\gamma$

while ($t < \text{Max Generation}$)
    for $i = 1 : n$ all n fireflies
        for $j = 1 : i$ all n fireflies
            if ($I_j > I_i$), Move firefly i towards j in d-dimension; end if
            Attractiveness varies with distance $r$ via $\exp[-\gamma r]$
            Evaluate new solutions and update light intensity
        end for
    end for
    Rank the fireflies and find the current best
end while

Postprocess results and visualization

Figure 2  A-String representations

<table>
<thead>
<tr>
<th>Operation Indicated</th>
<th>O_{1,1}</th>
<th>O_{1,2}</th>
<th>O_{1,3}</th>
<th>O_{2,1}</th>
<th>O_{2,2}</th>
<th>O_{2,3}</th>
<th>O_{3,1}</th>
<th>O_{3,2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - String</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>M_1</td>
<td>M_2</td>
<td>M_3</td>
<td>M_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3  B-String representations

<table>
<thead>
<tr>
<th>Operation Sequence</th>
<th>O_{2,1}</th>
<th>O_{1,1}</th>
<th>O_{3,1}</th>
<th>O_{2,2}</th>
<th>O_{3,2}</th>
<th>O_{1,2}</th>
<th>O_{1,3}</th>
<th>O_{2,3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
Figure 4  Critical operation swap neighbourhood

Figure 5  Critical operation insert neighbourhood
Figure 6  Framework of the proposed hybrid algorithm

Figure 7  Gantt chart of problem 4 × 5  \((C_m=11, W_m=10, W_t = 32)\)
Figure 8  Gantt chart of problem 8 × 8 \((C_m=14, W_m=12, W_t = 77)\)

Figure 9  Gantt chart of problem 10 × 7 \((C_m=11, W_m=10, W_t = 62)\)
Figure 10  Gantt chart of problem 10 × 10 ($C_m=7$, $W_m=6$, $W_f = 42$)

Figure 11  Gantt chart of problem 15 × 10 ($C_m=11$, $W_m=11$, $W_f = 91$)
Table 1: Example of FJSP with 3 jobs and 4 machines

<table>
<thead>
<tr>
<th>Job</th>
<th>Position</th>
<th>Operation</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>1</td>
<td>O1,1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>O1,2</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>O1,3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>O2,1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>J2</td>
<td>5</td>
<td>O2,2</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>O2,3</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>J3</td>
<td>7</td>
<td>O3,1</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>O3,2</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2: Illustration of initial firefly generation with objective function

<table>
<thead>
<tr>
<th>Population</th>
<th>Initial Firefly generation</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>A-string</td>
<td>B-string</td>
</tr>
<tr>
<td>1</td>
<td>2 4 3 1 3 4 2 1</td>
<td>4 1 7 5 8 2 3 6</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>13 35 19.9</td>
</tr>
<tr>
<td>2</td>
<td>2 3 1 4 2 1 4 3</td>
<td>1 4 2 7 3 5 6 8</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>14 39 21.5</td>
</tr>
<tr>
<td>3</td>
<td>3 1 2 4 1 2 3 4</td>
<td>7 1 4 2 8 3 5 6</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>7 25 15.6</td>
</tr>
<tr>
<td>4</td>
<td>2 3 2 4 1 2 3 4</td>
<td>1 7 2 4 8 3 5 6</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>9 22 12.6</td>
</tr>
<tr>
<td>5</td>
<td>3 1 1 4 2 1 3 4</td>
<td>7 4 5 8 1 2 6 3</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>18 33 24.5</td>
</tr>
<tr>
<td>6</td>
<td>1 3 1 2 4 2 3 4</td>
<td>1 4 5 2 7 6 3 8</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5 16 8.2</td>
</tr>
<tr>
<td>7</td>
<td>1 4 1 3 2 2 3 4</td>
<td>1 4 7 5 2 3 6 8</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5 15 7.5*</td>
</tr>
<tr>
<td>8</td>
<td>1 3 1 2 2 3 3 4</td>
<td>1 4 7 8 2 3 5 6</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>7 17 10.0</td>
</tr>
<tr>
<td>9</td>
<td>1 3 1 3 2 4 2 4</td>
<td>4 1 7 2 5 8 3 6</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>9 22 12.6</td>
</tr>
<tr>
<td>10</td>
<td>4 4 3 2 1 2 3 4</td>
<td>4 1 2 5 7 8 3 6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7 15 8.6</td>
</tr>
</tbody>
</table>

\(w_1 = 0.5, w_2 = 0.3, w_3 = 0.2\)

* Best firefly solution
### Table 3: Solution updation for population 1 in first generation

<table>
<thead>
<tr>
<th>Current firefly position ( (P) )</th>
<th>A-string</th>
<th>B-string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 4 3 1 3 4 2 1</td>
<td>4 1 7 5 8 2 3 6</td>
</tr>
</tbody>
</table>

| Best firefly position \( (P_{best}) \) | 1 4 1 3 2 2 3 4 | 1 4 7 5 2 3 6 8 |

| Difference between the elements \( (d) \) | \( \{(1,1), (3,1), (4,3), (5,2), (6,2), (7,3), (8,4)\} \) | \( \{(1,2), (5,6), (6,7), (7,8)\} \) |

| Hamming distance \( (r) \) | 7 | 4 |

| Attractiveness \( \beta \) – step \( \beta (r) = \frac{\beta_0}{(1 + \gamma \cdot r^2)} \) | 0.17 | 0.38 |

| \( rand \) ( ) between (0,1) | \{0.35, 0.09, 0.14, 0.33, 0.49, 0.32\} | \{0.52, 0.05, 0.12, 0.69\} |

| Movement \( \beta \) – step | (3,1), (4,3), (5,2) | (5,6), (6,7) |

| Firefly position after \( \beta \) – step | 2 4 1 3 2 4 2 1 | 4 1 7 5 2 3 8 6 |

| Attractiveness \( \alpha \) – step \( \alpha (rand_{int}) \) | 2 4 1 3 2 2 4 1 | 4 1 7 5 2 3 6 8 |

| Combined objective function | | \( F(c) = 14.80 \) |

### Table 4: Parameters of HDFA

<table>
<thead>
<tr>
<th>Problem ( (n \times m) )</th>
<th>( P_{size} )</th>
<th>( max_{gen} )</th>
<th>( ite_{max} )</th>
<th>( R_1 ) %</th>
<th>( R_2 ) %</th>
<th>( S_1 ) %</th>
<th>( S_2 ) %</th>
<th>( \beta_0 )</th>
<th>( \gamma )</th>
<th>( \alpha )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 5</td>
<td>50</td>
<td>50</td>
<td>25</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1.0</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>8 x 8</td>
<td>100</td>
<td>50</td>
<td>50</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1.0</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>10 x 7</td>
<td>200</td>
<td>100</td>
<td>50</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1.0</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>10 x 10</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1.0</td>
<td>0.15</td>
<td>1.0</td>
</tr>
<tr>
<td>15 x 10</td>
<td>300</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1.0</td>
<td>0.15</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Table 5: Comparison of results with five Kacem instances

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Problem Size</th>
<th>4 x 5</th>
<th>8 x 8</th>
<th>10 x 7</th>
<th>10 x 10</th>
<th>15 x 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C_m</td>
<td>W_m</td>
<td>W_t</td>
<td>C_m</td>
<td>W_m</td>
<td>W_t</td>
</tr>
<tr>
<td>AL + CGA</td>
<td>1</td>
<td>16</td>
<td>10</td>
<td>34</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>n/a</td>
<td>16</td>
<td>13</td>
<td>75</td>
<td>n/a</td>
</tr>
<tr>
<td>PSO + SA</td>
<td>1</td>
<td>n/a</td>
<td>15</td>
<td>12</td>
<td>75</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>n/a</td>
<td>16</td>
<td>13</td>
<td>73</td>
<td>n/a</td>
</tr>
<tr>
<td>PSO + TS</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>32</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>n/a</td>
<td>15</td>
<td>12</td>
<td>75</td>
<td>n/a</td>
</tr>
<tr>
<td>ESM</td>
<td>1</td>
<td>12</td>
<td>8</td>
<td>32</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>n/a</td>
<td>15</td>
<td>12</td>
<td>76</td>
<td>11</td>
</tr>
<tr>
<td>AIA</td>
<td>1</td>
<td>n/a</td>
<td>14</td>
<td>12</td>
<td>77</td>
<td>n/a</td>
</tr>
<tr>
<td>HDFA</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>32</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>7</td>
<td>33</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>n/a</td>
<td>16</td>
<td>13</td>
<td>73</td>
<td>12</td>
</tr>
</tbody>
</table>

The weight coefficients are: (w_1 = 0.5, w_2 = 0.3, w_3 = 0.2)
n/a – not available

Table 6: Comparison of results on Kacem instances with release times

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Problem Size</th>
<th>4 x 5</th>
<th>10 x 7</th>
<th>15 x 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C_m</td>
<td>W_m</td>
<td>W_t</td>
<td>C_m</td>
</tr>
<tr>
<td>FL + EA</td>
<td>1</td>
<td>18</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>18</td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>16</td>
<td>9</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16</td>
<td>10</td>
<td>34</td>
</tr>
<tr>
<td>MOPSO + LS</td>
<td>1</td>
<td>16</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HDFA</td>
<td>1</td>
<td>16</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>