Int J Adv Manuf Technol (2014) 72:1567–1579
DOI 10.1007/s00170-014-5753-3

ORIGINAL ARTICLE

A hybrid discrete firefly algorithm for multi-objective flexible job shop scheduling problem with limited resource constraints

S. Karthikeyan · P. Asokan · S. Nickolas

Received: 22 August 2013 / Accepted: 27 February 2014 / Published online: 28 March 2014

Abstract In this paper, a hybrid discrete firefly algorithm is presented to solve the multi-objective flexible job shop scheduling problem with limited resource constraints. The main constraint of this scheduling problem is that each operation of a job must follow a process sequence and each operation must be processed on an assigned machine. These constraints are used to balance between the resource limitation and machine flexibility. Three minimisation objectives—the maximum completion time, the workload of the critical machine and the total workload of all machines—are considered simultaneously. In this study, discrete firefly algorithm is adopted to solve the problem, in which the machine assignment and operation sequence are processed by constructing a suitable conversion of the continuous functions as attractiveness, distance and movement, into new discrete functions. Meanwhile, local search method with neighbourhood structures is hybridised to enhance the exploitation capability. Benchmark problems are used to evaluate and study the performance of the proposed algorithm. The computational result shows that the proposed algorithm produced better results than other authors' algorithms.

Keywords Flexible job shop scheduling · Hybrid discrete firefly algorithm · Multi-objective optimisation · Limited resource constraints · Local search

S. Karthikeyan (✉) · P. Asokan
Department of Production Engineering, National Institute of Technology, Tiruchirappalli 620015, Tamil Nadu, India
e-mail: karthikeyan5000@yahoo.com

S. Nickolas
Department of Computer Applications, National Institute of Technology, Tiruchirappalli 620015, Tamil Nadu, India

1 Introduction

Scheduling involves the allocation of resources over a period of time to perform a collection of tasks. It is a decision making process that plays an important role in most manufacturing and service industries [1]. One of the most difficult problems in the area of production scheduling is the job-shop scheduling problem (JSP). It is well known that this problem is nondeterministic polynomial-time (NP) hard [2]. The classical JSP consists of scheduling a set of jobs on a set of machines with the objective to minimise a certain criterion, subjected to the constraint that each job has a specific processing order through all machines, which are fixed and known in advance.

The flexible job-shop scheduling problem (FJSP) is an extension of the classical JSP that allows an operation to be processed on any machine from a given set of alternative machines. It is closer to the real manufacturing situation. FJSP is more complex than classical JSP because of the additional need to determine the assignment of machines for each operation.

FJSP reduces the constraint of a machine, but extends the search range for a solution. It is a more complex NP-hard problem [3], and it incorporates all the difficulties and complexities of JSP. The FJSP consists of two sub-problems of routing and scheduling. The routing sub-problem assigns each operation to a machine among a set of capable machines authorised for each job. The scheduling sub-problem involves sequencing the operations assigned to the machines in order to obtain a feasible schedule that minimises a predefined objective [4]. Since all machines can process all operations, it is logical to restrict this freedom, especially when resources are limited. Therefore, resource constraints are added to enhance the practicability of the algorithm. The wider availability of machines for all the operations gives flexibility to perform an operation by any available set of machines. However, if the degree of flexibility is too high, the machines will surely

process more operations. Since resources are limited, more flexible machines cannot guarantee that more operations will be executed on each machine. If more flexible machines are available, the workload of each machine tends to be small; hence, the balance between resource limitation and machine flexibility is more important [5].

In the scheduling literature, it is usually assumed that machines are available during the whole planning horizon. This assumption may no longer be true in actual workshop, since breakdowns can happen at any moment, making one or several machines unavailable for processing jobs. In fact, in manufacturing systems, machines are periodically submitted to washing, control, or setup operations [6]. Therefore, availability constraints of machines should be considered while formulating a realistic scheduling model. There are various kinds of availability constraints on production systems. Roughly, they can be categorised into two types: fixed and non-fixed. The unavailable period of a fixed availability constraint starts at a fixed time point. For non-fixed availability constraints, the starting time of the unavailable period is supposed to be flexible and must be determined during the production scheduling procedure [7].

Leon and Wu [8] considered single machine sequencing problem with ready time and due date constraints on jobs and vacation constraints on the machine to minimise maximum lateness. Qi et al. [9] used branch and bound algorithm to solve single machine problem with preventive maintenance. Schmidt [10] has reviewed most results related to deterministic scheduling problems with availability constraints. Gharbi and Haouari [11] and Liao et al. [12] addressed the parallel machine scheduling problem with availability constraints. Aggoune [13] proposed a heuristic approach based on genetic algorithm and tabu search to solve the flow shop scheduling problem with unavailability periods imposed on machines due to a preventive maintenance activity with the objective to minimise the makespan. Aggoune and Portman [6] addressed a general flow shop scheduling problem with limited machine availability. Allaoui and Artiba [14] investigated the two-stage hybrid flow shop scheduling problem, in which each machine is subject to at most one unavailability period. In Mauguière et al. [15], branch-and-bound algorithms are proposed to solve the job-shop problem with various types of unavailability periods. Chan et al. [5] has used GA for flexible job-shop scheduling problems under resource constraints and effectively minimised machine idle cost. Zirbi et al. [16] addresses a job shop with multi-purpose machine scheduling problem with availability constraints. Lei [17] presents a random key genetic algorithm to solve fuzzy job shop scheduling problem with availability constraints. Rajkumar et al. [18] proposed GRASP algorithm to solve FJSP with limited resource constraints.

Most of the research on FJSP has been concentrated on mono-objective alone. However, several objectives must be considered simultaneously in the real world production situation and these objectives often conflict with each other. In recent years, multi-objective FJSP (MOFJSP) has gained attention of some researchers. Kacem et al. [19, 20] developed an effective evolutionary algorithm controlled by an assigned model based on the approach of localisation to solve the FJSP. Xia and Wu [4] proposed a hybrid optimisation approach using a particle swarm optimisation (PSO) algorithm to assign operations on machines and a simulated annealing (SA) algorithm to schedule operations on each machine in solving MOFJSP. An algorithm hybridised with evolving dispatching rules and genetic programming was proposed by Tay and Ho [21]. Gao et al. [22] developed a hybrid genetic algorithm (GA) for the FJSP with three objectives. Zhang et al. [23] introduced a hybrid PSO and tabu search (TS) algorithm to solve the multi-objective FJSP. Xing et al. [24] proposed an efficient search method for the multi-objective flexible job-shop scheduling problems. An effective hybrid tabu search algorithm was introduced by Li et al. [25] for solving multi-objective flexible job shop scheduling problems. A Pareto approach to MOFJSP using PSO and local search was proposed by Moslehi and Mahnam [26]. Li et al. [27] proposed a hybrid shuffled frog-leaping algorithm for solving the MOFJSP. Rahmati et al. [28] developed two multi-objective evolutionary algorithms to solve the multi-objective FJSP with three objectives. A discrete particle swarm optimisation algorithm hybridised with SA algorithm to solve multi-objective FJSP was proposed by Shao et al. [29].

The firefly algorithm (FA) is a novel metaheuristic algorithm inspired by the social behaviour of fireflies. By idealising some of the flashing characteristics of fireflies, the firefly-inspired algorithm was presented by Yang in 2008 [30]. Yang [31] proposed a firefly algorithm to solve nonlinear design problems. Lukasik and Zak [32] developed the firefly algorithm for the continuous constrained optimisation task. Their experimental evaluation demonstrates the efficiency of the firefly algorithm. Sayadia et al. [33] presented a discrete firefly algorithm for makespan minimisation in permutation flow shop scheduling problems. A discrete firefly algorithm was proposed by Jati [34] to solve the travelling salesman problem. Khadwilard et al. [35] solved the job shop scheduling problems using firefly algorithm. Marichelvam et al. [36] proposed a discrete firefly algorithm using the SPV rule for the multi-objective hybrid flow shop scheduling problems. Yang [37] proposed multi-objective firefly algorithm to solve multi-objective design optimisation problems. Vahedi Nouri et al. [38] presented a hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. The traditional firefly algorithm is a population based technique for solving continuous optimisation 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 optimisation problems.

In this paper, we propose a hybrid algorithm combining discrete firefly algorithm (DFA) with a local search approach

to solve MOFJSP with limited resource constraints in which each operation of a job must be followed by a process sequence and each operation must be processed on an assigned machine. There are no precedence constraints among the operations of different jobs. This paper describes firefly algorithm's discretisation, 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 for operations and to reschedule the results obtained from DFA, which will enhance the convergence speed. The objectives considered in this paper are to minimise maximal completion time, the workload of the critical machine and the total workload of machines simultaneously.

The remainder of this paper is organised as follows. The problem formulation and notation of multi-objective FJSP are introduced in Section 2. Section 3 describes the standard firefly algorithm. In Section 4, the proposed approach is presented to solve the FJSP. Section 5 shows the computational results and present study on well-known FJSP with multi-constraints problems. Finally, Section 6 provides conclusions and scope for further research.

2 Problem formulation

The flexible job-shop scheduling problem with limited resource constraints can be described as follows. There are m machines and n jobs. Each job J_i ($1 \le i \le n$) consists of a sequence of n_i operations. Each operation O_{ij} ($i=1,2,\ldots,n$; $j=1,2,\ldots,n_i$) of job (J_i) can be processed by one machine m_{ij} in the set of eligible machines M_{ij}. P_{ijk} denotes the processing time of operation O_{ij} on machine $k \in M_{ij}$. At the starting time of scheduling, it is not that all the machines are unoccupied. The role of scheduling is that the starting time of machine for different operations are determined, and the maximum process time is minimised.

The FJSP with limited resource constraints is needed to assign operations to a machine and schedule job operations subject to the constraint that [19]:

1. The operation sequence for each job is prescribed and the appointed process order must be followed.
2. Each machine can process only one operation at a time.
3. Each operation must be processed on an appointed machine.

For example, FJSP with three jobs and four machines is shown in Table 1, the numbers in the table present the processing time of operations, and the symbol '–' means the operation cannot be processed on the corresponding machine.

In this study, the following objectives are to be minimised:

Table 1 Example of FJSP with three jobs and four machines

Job	Position	Operation	M_1	M_2	M_3	M_4
J_1	1	$O_{1,1}$	2	–	1	6
	2	$O_{1,2}$	5	3	–	2
	3	$O_{1,3}$	–	2	4	–
J_2	4	$O_{2,1}$	7	–	–	11
	5	$O_{2,2}$	4	4	12	8
J_3	6	$O_{3,1}$	2	–	7	9
	7	$O_{3,2}$	3	5	8	1
	8	$O_{3,3}$	4	3	–	5

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 represent the total working time over all machines.

The hypotheses of this scheduling are as follows: an operation cannot be interrupted during the process; there is no operation order constraint between different jobs; different jobs have the same priority; all the jobs can be processed at the starting time of scheduling; and setup time and transfer time are included in the processing time.

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
m	total number of machines
n_i	total number of operations of job i
O_{ij}	the jth 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

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

Our model is presented as follows:

$$\min\ f_1 = \max_{1 \le k \le m} (C_k) \tag{1}$$

$$\min\ f_2 = \max_{1 \le k \le m} (W_k) \tag{2}$$

$$\min\ f_3 = \sum_{k=1}^{m} W_k \tag{3}$$

Subject to:

$$C_{ij} - C_{i(j-1)} \geq P_{ijk} X_{ijk}, j = 2, \ldots \ldots n_i; \forall i, j \tag{4}$$

$$\left[\left(C_{hg} - C_{ij} - t_{hjk}\right) X_{hgk} X_{ijk} \geq 0\right] \text{ v } \left[\left(C_{ij} - C_{hg} - t_{ijk}\right) X_{hji} X_{ijk} \geq 0\right], \quad \forall (i,j), (h,g), k \tag{5}$$

$$\sum\nolimits_{k \in M_{ij}} X_{ijk} = 1, \forall i, j \tag{6}$$

Equation (1) ensures the minimisation of maximal completion time of the machines. Equation (2) ensures the minimisation of maximal machine critical workload among all the machines available. Equation (3) ensures the minimisation of total work load 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 the 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 optimisation. These approaches can be classified into three categories [39].

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. The weighted sum of the three objective values is taken as the objective function:

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

Subject to:

$$W_1 + W_2 + W_3 = 1, \qquad 0 \leq W_1, W_2, W_3 \leq 1 \tag{8}$$

where $F(c)$ denotes the combined objective function value of a schedule; f_1, f_2 and f_3 denote 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 of 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. The advantage of utilising the weighted summation approach is its algorithmic actualisation, which is effortless and the users can change the weight of different objectives for satisfying the requirements of decision makers.

3 Firefly algorithm

The firefly algorithm is a novel population based technique for solving continuous optimisation problems, especially for continuous NP-hard problems and has been motivated by the simulation of the social behaviour of fireflies. It is possible to formulate optimisation algorithms because the flashing light can be formulated in such a way that it is associated with the objective function of problems considered, in order to obtain efficient optimal solutions [30].

Firefly-inspired algorithm uses the following three idealised rules [30, 31]: (1) all fireflies are unisex, which means that they are 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 maximisation problem, the brightness may be proportional to the objective function value and inverse for the minimisation problem. The basic steps of the FA are summarised by the pseudocode shown in Fig. 1, which consists of three rules discussed above.

Based on Yang [40], 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 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 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:

$$r_{ij} = \left\| x_i - x_j \right\| = \sqrt{\sum\nolimits_{k=1}^{d} \left(x_{i,k} - x_{j,k}\right)^2} \tag{9}$$

Firefly Algorithm

Objective function f(x*),* x = $(x_1, \ldots, x_d)^T$
Generate initial population of fireflies x_i $(i = 1, 2, \ldots, n)$
Light intensity I_i *at* x_i *is determined by f* (x_i)
Define light absorption coefficient γ
while $(t <$ *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 *j*
  end for *i*
  Rank the fireflies and find the current best
end while
Postprocess results and visualization

Fig. 1 Pseudo code of the firefly algorithm

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

3.2 Attractiveness

The attractiveness of a firefly is determined by its light intensity. Each firefly has its distinctive attractiveness β, 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 [32]

$$\beta(r) = \beta_0 e^{-\gamma r^m}, (m \geq 1) \tag{10}$$

where r is the distance between two fireflies, β_0 is the attractiveness at $r=0$ and γ is a fixed light absorption coefficient.

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 [32]

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}\left(x_j - x_i\right) + \alpha(rand - 1/2) \tag{11}$$

where the first term is the current position of a firefly, the second term is due to attraction and the third term is a randomisation with α being the randomisation parameter, while *rand* is a random number generator uniformly distributed in the space [0, 1].

The γ value ensures the firefly algorithm to search for the global solution in the large search space and is explained below [30, 31].

From Eq. (11), it is easy to see that there exist two limit cases when γ is small or large, respectively. When $\gamma \to 0$, the attractiveness is constant $\beta(r) = \beta_0$. This is equivalent to saying that the light intensity does not decrease in an idealised sky. Thus, a flashing firefly can be seen anywhere in the domain. This limiting case can lead to a variant of standard particle swarm optimisation (PSO) algorithm. Here, $\beta(r)$ is always the largest; it could possibly be, therefore, fireflies attempt to advance towards other fireflies with the largest possible steps. The exploration–exploitation, in this case, is out of balance because exploitation is maximal and exploration is normal. On the other hand, when $\gamma \to \infty$, the attractiveness is almost zero in the sight of other fireflies. This is equivalent to the case where the fireflies are surrounded with very thick fog and could not see any of the other fireflies. The only movement fireflies would be doing is the α-steps; this means that all fireflies move almost randomly, which corresponds to a random search technique. Here, the exploration–exploitation is out of the balance as well, as fireflies would do only the exploration, with no exploitation in the search space.

In general, the firefly algorithm corresponds to the situation between these two limit cases, and it is thus possible to fine-tune these parameters so that FA can outperform both PSO and random search. In fact, FA can find the global optima as well as all the local optima simultaneously in a very effective manner. The most efficient searching is when the exploration–exploitation is in balance, or first to emphasise the exploration, and then the exploitation. The parameter γ now characterises the variation of the attractiveness, and its value is crucially important in determining the speed of the convergence and how the FA algorithm behaves. In theory, γ can range in the interval $(0, \infty)$. However, its value depends on the characteristic length of γ of the system to be optimised: In most applications, it typically varies from 0.01 to 100.

Based on the effectiveness of the firefly algorithm in optimising continuous problems, it is predicted that this algorithm will be impressive in solving discrete optimisation problems. The firefly algorithm cannot be applied directly to solve the discrete optimisation problems.

4 Hybrid discrete firefly algorithm

The firefly algorithm has been originally developed for solving continuous optimisation problems. In order to make it applicable for solving the problem, considered a novel hybrid discrete firefly algorithm, named hybrid discrete firefly algorithm (HDFA), is proposed in this section.

4.1 Solution representation

Solution representation is one of the most important issues in designing a DFA. 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 vector) and B-string (operation scheduling vector) representation for each firefly that could be used to solve the multi-objective FJSP efficiently and avoid the use of a repair mechanism to maintain feasibility [23]. Machine assignment vector tells the machine assigned for each operation, while the operation scheduling vector displays the sequence of the operations on each machine.

Machine assignment An array of integer values is used to represent a machine assignment vector. 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.

Operation scheduling Operation scheduling vector has the same length as the machine assignment vector. It consists of a sequence of job numbers in which job number i occurs n_i times. It can avoid creating an infeasible schedule when replacing each operation by the corresponding job index.

The candidate solution generated at each level of iteration has to satisfy the constraints stated in the problem formulation model discussed in Section 2, and thus, feasibility of the solution is checked and ensured.

Table 1 gives a processing time table for an example problem with three jobs, four machines and eight operations. The machine assignment vector and operation scheduling vector of a solution for the given problem is shown in Fig. 2. The solution in Fig. 2 in this study is represented by a vector {3,4,2,1,1,1,4,2 | 2,3,1,1,2,3,3,1}. The first part {3,4,2,1,1,1,4,2} denotes the machine assigned to the operation, i.e. $\{(O_{11},M_3),(O_{12},M_4),(O_{13},M_2),(O_{21},M_1),(O_{22},M_1),(O_{31},M_1),(O_{32},M_4),(O_{33},M_2)\}$. The second part {2,3,1,1,2,3,3,1} is the scheduling component tells that the scheduling sequence is $\{O_{21}>O_{31}>O_{11}>O_{12}>O_{22}>O_{32}>O_{33}>O_{13}\}$. When a firefly solution is decoded, operation scheduling vector is converted to a sequence of operations at first. Then each operation is assigned to a processing machine according to machine assignment vector as follows: $[(O_{2,1}, M_1),(O_{3,1}, M_1),(O_{1,1}, M_3),(O_{1,2}, M_4),(O_{2,2}, M_1),(O_{3,2}, M_4),(O_{3,3}, M_2),(O_{1,3}, M_2)]$. Gantt chart of the decoded solution is shown in Fig. 3.

Through this representation, every solution could decode a feasible FJSP schedule effectively.

Machine assignment vector

Position	1	2	3	4	5	6	7	8
Operation	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{31}	O_{32}	O_{33}
Machine	3	4	2	1	1	1	4	2

Operation scheduling vector

Sequence	2	3	1	1	2	3	3	1
Operation	O_{21}	O_{31}	O_{11}	O_{12}	O_{22}	O_{32}	O_{33}	O_{13}
Position	4	6	1	2	5	7	8	3

Fig. 2 Illustration of the firefly representation

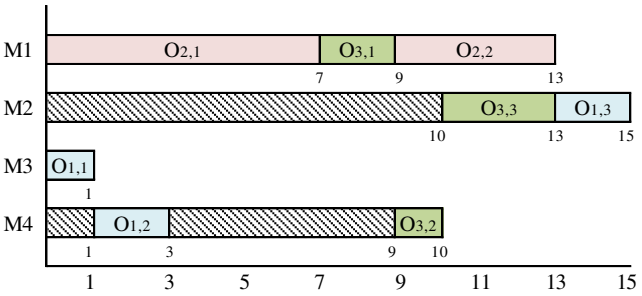

Fig. 3 Gantt chart (C_m=15, W_m=13, W_t=22)

4.2 Population initialisation

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.

4.2.1 Machine assignment component initial rules

Initiate the machine assignment component of the population using the following two rules: the operation minimum processing time rule [41] denoted by R_{ma1}, the global machine workload balance rule [27] denoted by R_{ma2}.

4.2.2 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 [42] denoted by R_{sc1}, the most number of operations remaining rule [41] denoted by R_{sc2}.

To consider both the problem feature 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 by rules R_{ma1}, R_{ma2}, R_{sc1} and R_{sc2}, respectively. All other solutions in the initial population are generated randomly to enhance the diversity of the population.

4.3 Firefly evaluation

Each firefly is represented by a machine assignment vector and an operation scheduling vector. Using the permutation of these vectors in the population 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 Eq. (7). Table 2 shows the initial firefly generation with calculated combined objective function value for the example problem given in Table 1.

Table 2 Illustration of initial firefly generation with objective function

Population P	Initial firefly generation		Objective function			
	A-string	B-string	f_1	f_2	f_3	$F(c)$
1	3 4 2 1 1 1 4 2	4 6 1 2 5 7 8 3	15	13	22	15.8
2	3 4 2 1 1 1 4 2	4 6 1 2 5 7 3 8	13	13	22	14.8
3	3 4 2 1 2 1 4 2	4 6 1 5 7 2 8 3	16	9	22	15.1
4	3 2 2 4 2 1 4 1	1 6 2 4 7 3 5 8	16	12	28	17.2
5	3 4 3 1 2 1 4 2	6 7 1 2 4 5 3 8	16	9	24	15.5
6	3 4 3 1 2 1 4 2	4 6 1 2 7 5 8 3	14	9	24	14.5
7	3 4 3 1 2 1 4 2	1 4 2 6 7 8 3 5	17	9	24	16.0
8	3 4 2 1 2 1 4 2	6 4 7 1 8 5 2 3	15	9	22	14.6
9	3 4 2 1 2 1 4 2	1 2 4 6 3 5 7 8	14	9	22	14.1[a]
10	1 4 2 1 1 1 4 2	6 4 7 1 5 8 2 3	15	15	23	16.6

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

[a] P_{best} best firefly solution

The number of fireflies (population size) used in this problem is 10. The best firefly (P_{best}) based on the combined objective function value is population P-9.

4.4 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 none is brighter than a particular firefly, it will move randomly. In this work, discretisation is done for the following issues:

4.4.1 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 using Hamming distance. The Hamming distance between two permutations is the number of non-corresponding elements in the sequence [43]. The distance between two permutations in the B-string can be measured using swap distance. The swap distance is the number of minimal required swaps of one permutation in order to obtain the other one.

4.4.2 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 Eq. (11) into two sub-steps: β-step and α-step as given in Eqs. (12) and (13), respectively. We can do this, since we know that the result will not change.

$$x_i = \beta(r)\left(x_j - x_i\right) \tag{12}$$

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

The attraction steps α and β are not interchangeable. The β-step must be computed before the α-step while finding the new position of the firefly.

β-Step It brings the iterated firefly always closer to another firefly. In other words, after applying β-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 β-step are as follows:

Let

- d the difference between the elements of best firefly and other firefly
- r Hamming distance for A-string and swap distance for B-string

Step 1: All necessary insertions in the machine assignment vector and all necessary pair-wise exchanges in the operation scheduling vector, to make the elements of the current firefly solution equal to best firefly solution are found and stored in d.

Step 2: Counting the number of insertions in machine assignment vector and number of pair-wise exchanges in operation scheduling vector between two firefly solutions will give the hamming distance and swap distance respectively, which is stored in r.

Step 3: Compute β probability using Eq. (14). Since it is often faster to calculate $1/(1+r^2)$ than an exponential function, the Eq. (10) can be approximated as mentioned below

$$\beta(r) = \beta_0/\left(1 + \gamma r^2\right) \tag{14}$$

Step 4: Random number *rand ( )* is generated in the range (0, 1) according to the number of counts in the set d of machine assignment vector and operation scheduling vector.

Step 5: If *rand ( )*≤β, then the corresponding insertion in the machine assignment vector and pair-wise exchange in the operation scheduling vector 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 β probability.

α-Step It is much simpler than the *β*-step. In Eq. (13) the random movement of firefly α (*rand*− 1/2) is approximated as α ($rand_{\text{int}}$) given in Eq. (15), which allows us to shift the permutation into one of the neighbouring permutations.

$$x_i = x_i + \alpha(rand_{\text{int}}) \tag{15}$$

In machine assignment vector *α*-step is applied by randomly choosing an element position using *α* ($rand_{\text{int}}$) and the machine assignment component of that position is replaced with a new machine having minimum processing time among the set of available machines. In operation scheduling vector *α*-step is applied by randomly choosing an element position by using α ($rand_{\text{int}}$) and swap with another position in the string which is also chosen at random. The random number $rand_{\text{int}}$ 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: β_0=1, γ=0.1, α=1. The improvement of objective function value *F(c)* from 17.2 to 15.1 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.

4.5 Local search

In each generation of the discrete firefly algorithm, we improve the quality of the solution (firefly) using a local search mechanism which explore the neighbourhood of the generated solution for better ones. Neighbourhood structures are proposed 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 diversity of the solution population. 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. The following are the neighbourhood structures for machine assignment and scheduling component.

4.5.1 Neighbourhood structure for machine assignment

Random neighbourhood (L_{ma1}) The neighbouring structure of machine assignment is to find the alternative machine randomly. Wang et al. [44] used this neighbourhood structure as a mutation operator in genetic algorithm. The neighbourhood is generated by following steps: (1) select operation $O_{i,j}$ randomly from machine assignment vector of solution; (2) detect machine M_k that is currently selected to process $O_{i,j}$; (3) detect a set of machines *M* that can process $O_{i,j}$; and (4) select a random machine from machine set *M* (except M_k) to process $O_{i,j}$.

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 [27]. The steps are as follows: (1) get the number of critical operations for each machine; (2) sort all machines in non-increasing order

Table 3 Illustration of firefly solution updation

Solution vector	Machine assignment	Operation scheduling
Current firefly position (P_4)	3 2 2 4 2 1 4 1	1 6 2 4 7 3 5 8
Combined objective function for P_4	$F(c)$=17.2	
Best firefly position (P_{best})	3 4 2 1 2 1 4 2	1 2 4 6 3 5 7 8
Combined objective function for P_{best}	$F(c)$=14.1	
Difference between the elements (*d*)	{(2,4), (4,1), (8,2)}	{(2,3), (3,4), (5,6), (6,7)}
Hamming distance (*r*)	3	4
Attractiveness *β*-step $\beta(r) = \frac{\beta_0}{(1+\gamma.\ r^2)}$	0.53	0.38
Random number generation *rand ( )* between (0,1)	{0.72, 0.52, 0.06}	{0.46,0.66, 0.08, 0.76}
Movement *β*-step	(4,1), (8,2)	(5,6)
Firefly position after *β*-step	3 2 2 1 2 1 4 2	1 6 2 4 3 7 5 8
Attractiveness *α*-step *α* ($rand_{\text{int}}$)	3 4 2 1 2 1 4 2	1 6 2 4 3 5 7 8
Combined objective function after movement	$F(c)$=15.1	

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 that is scheduled on M_{old}, and denote as $O_{i,j}$; (4) assign a machine with relatively less critical operations from other machines which is capable of processing $O_{i,j}$, denoted M_s, and schedule operation $O_{i,j}$ on M_s ; (5) replace the machine assignment component of the current solution at position $O_{i,j}$ with the value of M_s.

4.5.2 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, [45] for job shop scheduling. For FJSP, we have to make an extra check that the two operations to be swapped does 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.

Critical operation insert neighbourhood (L_{insert}) Dell'Amico and Trubian [46] proposed the neighbourhood by moving one critical operation on the critical path to job shop scheduling. Hence we adopt by moving one operation. The neighbourhood is generated by the 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 at the end of the critical block; and (4) if a critical block contains only one operation, then no swap is made.

The local search approach is carried out by selecting randomly one approach each from machine assignment neighbourhood structure and operation scheduling neighbourhood structure. Figure 4 illustrates the neighbourhood generated using this approach for the decoded solution of the example problem given in Fig. 3. In machine assignment, operation J2,2 is assigned to machine M_2 from M_1 based on the steps involved in critical operations neighbourhood. Then, according to critical operation swap neighbourhood the operations are swapped which in turn gives the better solution (C_m=13, W_m=9, W_t=22) than the previous one (C_m=15, W_m=13, W_t=22).

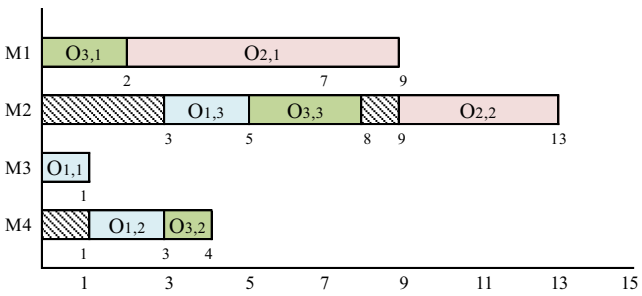

Fig. 4 Gantt chart (C_m=13, W_m=9, W_t=22)

4.6 The framework of HDFA

The proposed HDFA could keep the balance of both 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 performance in solving the multi-objective flexible job shop scheduling problem. The framework of the proposed HDFA is illustrated in Fig. 5. During the operation process, the initialisation is done with multiple strategies. Then, each individual in the population is evaluated. If the stop criterion is met, the non-dominated solution is the output. Otherwise, update each firefly according to Eqs. (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.

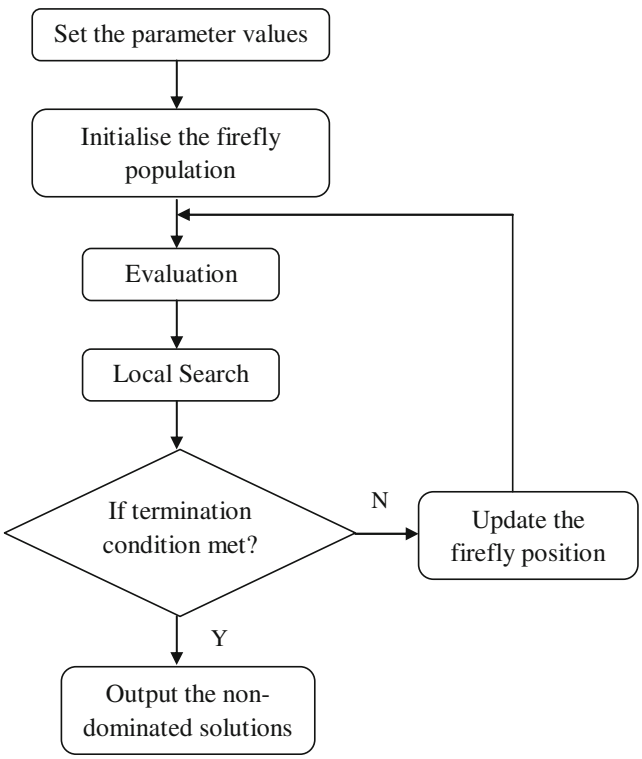

Fig. 5 Framework of the proposed hybrid algorithm

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. Three representative instances based on the practical data have been considered. Each instance can be characterised by the following parameters: number of jobs (n), number of machines (m) and the number of operations (n_i). Two problem instances (problem 8×5 and problem 12×5) are taken from Du et al. [3] and the third problem instance (problem 8×8) from Rajkumar et al. [18]. The best and average results of experiments from 20 different runs were collected for performance comparison.

The parameters of the hybrid algorithm are as follows. Set the population size P_{size}=200, rate of machine assignment rule R_{ma1} and scheduling component rule R_{sc1}=20 %, rate of machine assignment rule R_{ma2} and scheduling component rule R_{sc2}=30 %, the maximum number of generations max_{gen}= 100, maximum local search iteration ite_{max}=50, attractiveness of fireflies β_0=1.0, light absorption coefficient γ=0.1 and randomisation parameter α=1.0.

5.1 Problem 8×5

A small scale instance 8×5 with 20 operations, which is taken from Du et al. [3], is shown in Table 4, and it is taken to evaluate the efficiency of our proposed hybrid algorithm. In the data of Table 4, symbol '–' indicates that the machine is not available for the corresponding operation. The computational results obtained by the proposed hybrid algorithm are shown as follows:

$$\text{Makespan } (C_m) = 24, \text{maximal workload } (W_m) = 24, \text{total workload } (W_t) = 101.$$

Table 4 Problem 8×5 with 20 operations

Job	O_{ij}	M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	5	3	–	–	–
	$O_{1,2}$	–	7	–	–	–
J_2	$O_{2,1}$	–	–	6	–	–
	$O_{2,2}$	–	–	–	3	4
J_3	$O_{3,1}$	–	4	6	–	–
	$O_{3,2}$	7	–	–	–	–
	$O_{3,3}$	–	–	–	7	–
J_4	$O_{4,1}$	–	–	–	–	10
J_5	$O_{5,1}$	–	–	–	5	–
	$O_{5,2}$	4	5	8	–	–
	$O_{5,3}$	–	–	–	6	5
	$O_{5,4}$	–	3	–	–	4
J_6	$O_{6,1}$	–	2	6	–	–
	$O_{6,2}$	–	-	8	–	–
J_7	$O_{7,1}$	–	–	3	8	–
	$O_{7,2}$	–	–	–	7	4
J_8	$O_{8,1}$	3	–	5	–	–
	$O_{8,2}$	–	–	-	9	6
	$O_{8,3}$	–	–	7	–	–
	$O_{8,4}$	–	3	–	–	–

Table 5 Comparison of results on problem 8×5

	GA	GRASP	DFA	HDFA
Makespan (C_m)	27	24	28	24
Maximal machine workload (W_m)	27	24	25	24
Total workload of the machines (W_t)	109	101	102	101
F(0.5–0.3–0.2)	43.4	39.4	41.9	39.4
F(0.3–0.2–0.5)	68	62.5	64.4	62.5
F(0.2–0.5–0.3)	51.6	47.1	48.7	47.1
Computational time (s)	–	–	0.18	0.45

The comparison of the results of the proposed HDFA and DFA with other algorithms is shown in Table 5. It also shows that the combined objective function value $F(c)$ for different weights assigned to makespan, maximum workload and total workload. From Table 5, it can be seen that the proposed HDFA outperforms DFA. The column labelled 'GA' refers to genetic algorithm proposed by Du et al. [3], and the next column labeled 'GRASP' refers to greedy randomised adaptive search procedure proposed by Rajkumar et al. [18]. The comparison result shows that the proposed algorithm is more suitable for solving problems with multi constraints. In Fig. 6, the solution is shown by Gantt chart. The representation $O_{1,1}$ (job, operation) inside the block denotes the first operation of job 1, and so on. The hatched blocks represent the machine's idle periods.

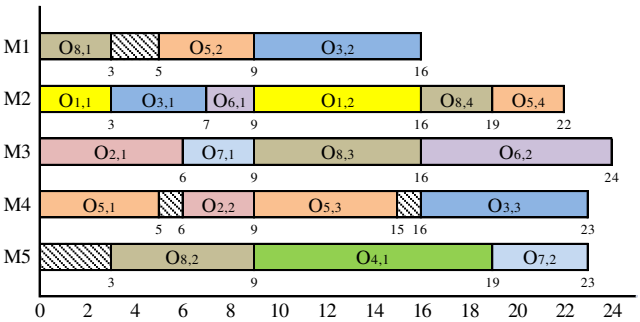

Fig. 6 Gantt chart of problem 8×5 with 20 operations (C_m=24, W_m=24, W_t=101)

Table 6 Problem 12×5 with 30 operations

Job	O_{ij}	M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	5	3	–	–	–
	$O_{1,2}$	–	7	–	–	–
J_2	$O_{2,1}$	–	–	6	–	–
	$O_{2,2}$	–	–	–	3	4
J_3	$O_{3,1}$	–	4	6	–	–
	$O_{3,2}$	7	–	–	–	–
	$O_{3,3}$	–	–	–	7	–
J_4	$O_{4,1}$	–	–	–	–	10
J_5	$O_{5,1}$	–	–	–	5	–
	$O_{5,2}$	4	5	8	–	–
	$O_{5,3}$	–	–	–	6	5
	$O_{5,4}$	–	3	–	–	4
J_6	$O_{6,1}$	–	2	6	–	–
	$O_{6,2}$	–	–	8	–	–
J_7	$O_{7,1}$	–	–	3	8	–
	$O_{7,2}$	–	–	–	7	4
J_8	$O_{8,1}$	3	–	5	–	–
	$O_{8,2}$	–	–	–	9	6
	$O_{8,3}$	–	–	7	–	–
	$O_{8,4}$	–	3	–	–	–
J_9	$O_{9,1}$	–	4	–	–	–
	$O_{9,2}$	4	–	–	–	3
J_9	$O_{10,1}$	–	–	4	–	–
	$O_{10,2}$	–	–	–	5	–
	$O_{10,3}$	–	3	–	–	–
J_{10}	$O_{11,1}$	–	4	–	–	4.5
	$O_{11,2}$	–	–	6	–	4
	$O_{11,3}$	3	4	–	–	–
J_{11}	$O_{12,1}$	–	5	–	4	–
	$O_{12,2}$	–	–	4	3	–

5.2 Problem 12×5

Table 6 displays the middle scale instance of 12×5 with 30 operations, which is taken from Du et al. [3]. The best

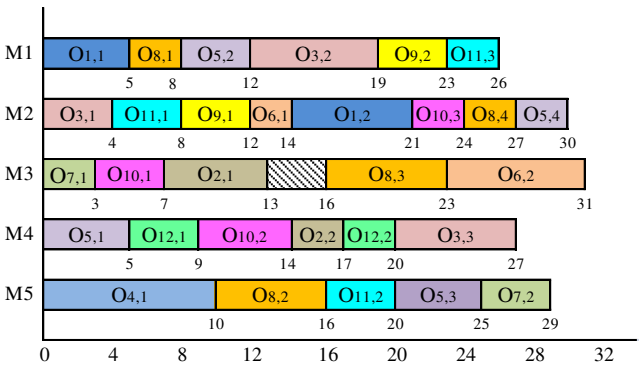

Fig. 7 Gantt chart of problem 12×5 with 30 operations (C_m=31, W_m=30, W_t=140)

Table 7 Comparison of results on problem 12×5

	GA	GRASP	DFA	HDFA
Makespan (C_m)	33	33	34	31
Maximal machine workload (W_m)	33	33	32	30
Total workload of the machines (W_t)	145	138	139	140
F(0.5–0.3–0.2)	55.4	54	54.4	52.5
F(0.3–0.2–0.5)	89	85.5	86.1	85.3
F(0.2–0.5–0.3)	66.6	64.5	64.5	63.2
Computational time (s)	–	–	0.37	0.93

computational results of our hybrid algorithm are shown as follows:

$$\text{Makespan } (C_m) = 31, \text{maximal workload } (W_m) = 30, \text{total workload } (W_t) = 140.$$

The schedule using Gantt chart representation corresponding to the optimal solution is shown in Fig. 7. The comparison

Table 8 Problem 8×8 with 27 operations

Job	O_{ij}	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J_1	$O_{1,1}$	5	–	7	–	3	–	–	9
	$O_{1,2}$	–	–	–	4	3	–	6	–
	$O_{1,3}$	–	–	–	5	–	2	4	–
J_2	$O_{2,1}$	4	–	3	–	8	–	9	–
	$O_{2,2}$	–	8	–	2	6	–	–	–
	$O_{2,3}$	–	–	–	5	–	4	1	7
	$O_{2,4}$	10	–	9	–	4	7	–	–
J_3	$O_{3,1}$	–	–	–	–	–	5	2	4
	$O_{3,2}$	–	5	6	4	–	9	–	–
	$O_{3,3}$	1	–	–	6	–	10	–	7
J_4	$O_{4,1}$	–	1	6	–	–	–	8	–
	$O_{4,2}$	–	11	–	8	9	5	6	–
	$O_{4,3}$	–	–	2	–	3	–	5	7
J_5	$O_{5,1}$	3	–	7	–	9	–	–	–
	$O_{5,2}$	–	–	7	4	–	–	6	–
	$O_{5,3}$	–	9	–	–	4	2	–	–
	$O_{5,4}$	–	–	–	6	7	–	3	6
J_6	$O_{6,1}$	–	–	1	–	3	–	–	–
	$O_{6,2}$	11	–	9	–	–	–	6	4
	$O_{6,3}$	–	5	9	10	–	–	–	–
J_7	$O_{7,1}$	5	–	2	–	–	–	3	–
	$O_{7,2}$	–	9	–	–	11	9	–	5
	$O_{7,3}$	–	–	–	3	–	6	–	7
J_8	$O_{8,1}$	2	8	–	9	–	–	–	–
	$O_{8,2}$	–	4	7	–	9	–	10	–
	$O_{8,3}$	–	9	–	8	5	–	–	1
	$O_{8,4}$	9	–	3	–	1	5	–	–

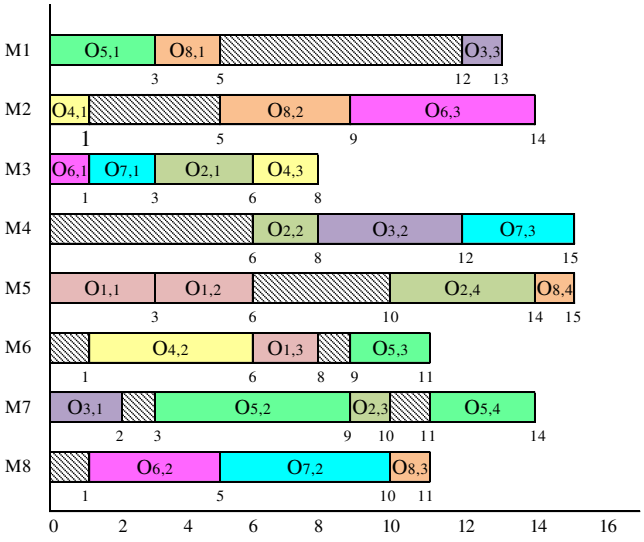

Fig. 8 Gantt chart of problem 8×8 with 27 operations (C_m=15, W_m=12, W_t=75)

of our hybrid algorithm and DFA with other algorithms is shown in Table 7. The column labeled 'GA' refers to genetic algorithm proposed by Du et al. [3] and the next column labeled 'GRASP' refers to greedy randomised adaptive search procedure proposed by Rajkumar et al. [18]. Table 7 shows our hybrid algorithm's performance and effectiveness over DFA and other algorithms.

5.3 Problem 8×8

This is a middle scale instance taken from Rajkumar et al. [18] given in Table 8, in which 8 jobs with 27 operations are to be performed on 8 machines. The obtained solutions by our hybrid algorithm are characterised by the following values:

$$\text{Makespan } (C_m) = 15, \text{maximal workload } (W_m) = 12, \text{total workload } (W_t) = 75$$

Figure 8 shows the result of solution in the form of a Gantt chart. From Table 9, by comparison with DFA and GRASP algorithm, the effectiveness of our proposed hybrid algorithm is shown obviously.

Table 9 Comparison of results on problem 8×8

	GRASP	DFA	HDFA
Makespan (C_m)	16	16	15
Maximal machine workload (W_m)	13	13	12
Total workload of the machines (W_t)	73	73	75
F(0.5–0.3–0.2)	26.5	26.5	26.1
F(0.3–0.2–0.5)	43.9	43.9	44.4
F(0.2–0.5–0.3)	31.6	31.6	31.5
Computational time (s)	–	0.41	0.85

6 Conclusions

In this paper, an effective HDFA is proposed for multi-objective flexible job shop scheduling with limited resource constraints. The objective function considered is minimisation of makespan, maximal workload and total workload of machines. Instead of applying the standard firefly algorithm, we proposed the discrete version of the continuous function such as distance, attractiveness and movement to update a firefly position. A combination of rules is utilised 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. The performance of the presented approach is evaluated in comparison with the results obtained from other authors' algorithms for three representative instances. The obtained computational results and time demonstrated the effectiveness of the proposed approach. The future work is to enhance the convergence capability of the algorithm and to generalise the application of the proposed HDFA for other combinatorial optimisation problems.

References

1. Pinedo M (2002) Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood Cliffs
2. Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1(2):117–129
3. Du, X., Li, Z., & Xiong, W. (2008) Flexible job shop scheduling problem solving based on genetic algorithm with model constraints. IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2008, Singapore, pp 1239–1243
4. Xia WJ, Wu ZM (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 48(2):409–425
5. Chan FTS, Wong TC, Chan LY (2006) Flexible job-shop scheduling problem under resource constraints. Int J Prod Res 44(11):2071–2089
6. Aggoune R, Portmann MC (2006) Flow shop scheduling problem with limited machine availability: a heuristic approach. Int J Prod Econ 99(1):4–15
7. Gao J, Gen M, Sun L (2006) Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. J Intell Manuf 17(4):493–507
8. Leon VJ, Wu SD (1992) On scheduling with ready times, due dates and vacations. Nav Res Logist 39(1):53–65
9. Qi X, Chen T, Tu F (1999) Scheduling the maintenance on a single machine. J Oper Res Soc 1071–1078
10. Schmidt G (2000) Scheduling with limited machine availability. Eur J Oper Res 121(1):1–15
11. Gharbi A, Haouari M (2005) Optimal parallel machines scheduling with availability constraints. Discret Appl Math 148:63–87

12. Liao CJ, Shyur DL, Lin CH (2005) Makespan minimization for two parallel machines with an availability constraint. Eur J Oper Res 160: 445–456
13. Aggoune R (2004) Minimizing the makespan for the flow shop scheduling problem with availability constraints. Eur J Oper Res 153:534–543
14. Allaoui H, Artiba A (2006) Scheduling two-stage hybrid flow shop with availability. Comput Oper Res 33(5):1399–1419
15. Mauguière P, Billaut JC, Bouquard JL (2005) New single machine and job-shop scheduling problems with availability constraints. J Sched 8(3):211–231
16. Zribi N, El Kamel A, Borne P (2008) Minimizing the makespan for the MPM job-shop with availability constraints. Int J Prod Econ 112(1):151–160
17. Lei D (2010) Fuzzy job shop scheduling problem with availability constraints. Comput Ind Eng 58(4):610–617
18. Rajkumar M, Asokan P, Anilkumar N, Page T (2011) A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. Int J Prod Res 49(8):2409–2423
19. Kacem I, Hammadi S, Borne P (2002) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Syst Man Cybern 32(1):1–13
20. Kacem I, Hammadi S, Borne P (2002) Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60(3):245–276
21. Tay JC, Ho NB (2008) Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput Ind Eng 54(3):453–473
22. Gao J, Sun L, Gen M (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 35(9):2892–2907
23. Zhang G, Shao X, Li P, Gao L (2009) An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 56(4):1309–1318
24. Xing LN, Chen YW, Yang KW (2009) An efficient search method for multi-objective flexible job shop scheduling problems. J Intell Manuf 20:283–293
25. Li JQ, Pan QK, Liang YC (2010) An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 59(4):647–662
26. Moslehi G, Mahnam M (2011) A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. Int J Prod Econ 129(1): 14–22
27. Li J, Pan Q, Xie S (2012) An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems. Appl Math Comput 218(18):9353–9371
28. Rahmati SHA, Zandieh M, Yazdani M (2013) Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem. Int J Adv Manuf Technol 64(5–8):915–932
29. Shao X, Liu W, Liu Q, Zhang C (2013) Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem. Int J Adv Manuf Technol 67(9–12):2885–2901
30. Yang X (2008) Nature-inspired metaheuristic algorithm. Luniver Press, Bristol
31. Yang XS (2010) Firefly algorithm, stochastic test functions and design optimization. Int J Bio-Inspired Comput 2(2):78–84
32. Łukasik S, Żak S (2009) Firefly algorithm for continuous constrained optimization tasks. In Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, Springer Berlin Heidelberg pp 97–106
33. Sayadi MK, Ramezanian R, Ghaffari-Nasab N (2010) A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. Int J Ind Eng Comput 1(1):1–10
34. Jati GK (2011) Evolutionary discrete firefly algorithm for travelling salesman problem. In Adaptive and intelligent systems. Springer Berlin Heidelberg, pp 393–403
35. Khadwilard A, Chansombat S, Thepphakorn T, Chainate W, Pongcharoen P (2012) Application of firefly algorithm and its parameter setting for job shop scheduling. J Ind Technol 8(1):49–58
36. Marichelvam MK, Prabaharan T, Yang XS (2012) A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. IEEE Transactions on Evolutionary Computation 99
37. Yang XS (2013) Multiobjective firefly algorithm for continuous optimization. Eng Comput 29(2):175–184
38. Vahedi Nouri B, Fattahi P, Ramezanian R (2013) Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. Int J Prod Res 51(12): 3501–3515
39. Hsu T, Dupas R, Jolly D, Goncalves G (2002) Evaluation of mutation heuristics for solving a multiobjective flexible job shop by an evolutionary algorithm. IEEE Syst Man Cybern 5:6
40. Yang XS (2009) Firefly algorithm for multimodal optimization. In stochastic algorithms: Foundations and applications, 2009. Lecture notes in computer science vol 5792. Springer Berlin Heidelberg, pp 169–178
41. Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35(10): 3202–3212
42. Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41(3):157–183
43. Kuo I, Horng SJ, Kao TW, Lin TL, Lee CL, Terano T, Pan Y (2009) An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. Expert Syst Appl 36(3):7027–7032
44. Wang X, Gao L, Zhang C, Shao X (2010) A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. Int J Adv Manuf Technol 51(5–8):757–767
45. Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Management science 42(6):797–813
46. Dell'Amico M, Trubian M (1993) Applying tabu search to the job-shop scheduling problem. Ann Oper Res 41(3):231–252