Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 101 – 114

doi:10.3906/elk-1701-22

An optimized multiobjective CPU job scheduling using evolutionary algorithms

Santhi VENKATRAMAN*, Dharshikha SELVAGOPAL
Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamil Nadu, India

Received: 03.01.2017 • **Accepted/Published Online:** 04.12.2017 • **Final Version:** 26.01.2018

Abstract: Scheduling in a multiprocessor parallel computing environment is an NP-hard optimization problem. The main objective of this work is to obtain a schedule in a distributed computing system (DCS) environment that minimizes the makespan and maximizes the throughput. We study the use of two of the evolutionary swarm optimization techniques, the firefly algorithm and the artificial bee colony (ABC) algorithm, to optimize the scheduling in a DCS. We also enhance the traditional ABC algorithm by merging the genetic algorithm techniques of crossover and mutation with the employed bee phase and the onlooker phase, respectively. The resulting enhanced ABC algorithm is used as the scheduling algorithm and is evaluated against the firefly and ABC algorithms. The results obtained show that in a distributed environment with a large number of jobs and resources, multiobjective scheduling using evolutionary algorithms can perform well in terms of minimizing makespan and maximizing throughput.

Key words: Firefly algorithm, makespan, throughput, artificial bee colony algorithm, crossover, mutation

1. Introduction

Day by day, data are becoming larger and more complex. The resources available for processing huge data are limited and there is a rising need for high processing capacity. Hence, the available resources should be used efficiently to handle the problem. Many computing systems like supercomputing, client server computing, and parallel computing have been proposed to handle huge data. Distributed computing is proving to be an efficient approach to handle the processing of large amounts of data. High processing capacity of low-cost computers and high-speed network technologies have driven the use of distributed computing environments to a very great extent. In a distributed computing system (DCS) many servers are integrated in such a manner that they appear as one system. There is always a master, which will delegate the tasks to the participating nodes equally. Here the main aim of the master is to distribute the jobs such that the overall time taken to complete all the jobs is minimal and no node is overloaded or underloaded. The key problem when using a DCS is to find a schedule to execute the tasks by assigning the resources in a smart way. Thus, scheduling in a DCS is an NP-hard problem [1]. Finding an optimized schedule plays an important role in determining the performance of the system. A typical scheduling algorithm should find a schedule such that the makespan is minimized and the response time is reduced. Tasks should also be scheduled in such a way that the throughput of the system is increased. Hence, scheduling in a DCS can be classified as a multimodal optimization problem [2].

Evolutionary algorithms are relatively very powerful techniques used to find solutions for many real-world search and optimization problems. For problems with multiple objectives, we need to obtain a set of optimal solutions, known as effective solutions. It has been found that using evolutionary algorithms is a highly effective

*Correspondence: sannthi@yahoo.com

way of finding multiple effective solutions in a single simulation run [3,4]. Starting with the traditional LPT algorithm [5], which is a heuristic approach, a number of nature-inspired evolutionary approaches have found applications in solving such multimodal optimization problems. For the genetic algorithm [6,7], though it uses an evolutionary approach to find the best schedule, the computation time required is higher. Particle swarm optimization (PSO) [8] and the artificial bee colony (ABC) algorithm [9,10] are metaheuristic approaches applied to various multiobjective problems. Unlike the ABC, the PSO algorithm cannot deal with larger instances. The firefly algorithm (FA) [11–13] outperforms both PSO and ABC algorithms for smaller instances, but when applied to obtain a schedule with a larger number of jobs and resources the result is not as optimal as that of ABC. The major drawback of the ABC algorithm is that when used with multimodal optimization problems, the complexity induced is very high. This drawback is overcome by the FA, which makes it more preferable with its ease of implementation.

Most of the previous research works done so far considered makespan, flow time, and latency as the criteria for the objective functions. In the multiobjective simulated annealing approach proposed by Varadharajan and Chandrasekaran [14], minimizing of total flow time and makespan are considered as the objectives. A multiobjective approach with PSO technique [15] was proposed by Zhang et al., where they minimized the makespan and the workload of the machines. Karthikeyan and Asokan et al. [16] proposed a discrete FA for solving multiobjective problems with some set of resource constraints. In another paper Marichelvam et al. used a discrete FA for multiobjective scheduling [17], which again considered the minimization of makespan and mean flow time. In this paper we have considered two objectives: one is minimizing the makespan and the other is to maximize the throughput. By throughput we mean the number of jobs completed within a given time unit.

2. Problem formulation

A set of $n$jobs is to be processed on a set of $m$machines under the following basic assumptions: preemption of operations is not allowed; each machine can process only one job at a time; each job may be processed by only one machine at a time. The processing of a job j on machine k is referred to as an operation. The processing time (P_{jk}- processing time taken by job j on machine k) of a job is not known at the time of scheduling. Therefore, we assume that the processing times are independent random variables with uniform/normal/exponential distribution [18].

Makespan is defined as the completion time of the last job to leave the system. The completion time includes the processing time and the waiting time. Makespan is important for effective utilization of resources and it should be minimized for effective performance. If$C_{\max}$ is the makespan that is to be minimized, at any instance, the completion time of the jth job using the kth resource should be less than or equal to it. Hence,

$$C_{\max} \geq C_{jk} \tag{1}$$

If Z_i denotes the makespan for a scheduleσ_i, C_j is the completion time of job j, and if P_j is the processing time of job j, which is a random variable, then

$$Zi = \sum_{j=1}^{n} C_j, \tag{2}$$

$$Zi = \sum_{j=1}^{n} (n+1-j)\, P_j \tag{3}$$

The problem is formulated mathematically as

$$f = min(Z). \tag{4}$$

Here Z = $\{Z_i\}$ where i = 1 to size. Size denotes the population size, i.e. number of fireflies in a population. Since processing time P_{jk} is a random variable, the completion time C^{σ} for a schedule σ is also a random variable. A minimum makespan, $C^{\sigma}_{\max}$, which is also a random variable, can be achieved for each realization of P_{jk}. Therefore, the objective function is expressed in the form of an expectation. The obtained makespan value should be less than or equal to the expected makespan value. The objective function f can be rewritten as:

$$f \leq minE(C_{\max}). \tag{5}$$

The other objective function considered is the throughput. The schedule that is obtained should be achieving high throughput along with minimal make span. If k is the number of jobs executed for a given time duration, where k $\leq$ n, the objective function is given by:

$$g = max(k). \tag{6}$$

For finding the best schedule in a population, the intensity value is calculated for each of the fireflies in the population. This intensity depends on both throughput and makespan, which are calculated for every schedule in a population. Among the calculated values, the minimum makespan is taken as the optimal makespan and the maximum throughput is taken as the optimal throughput. The efficiency of each schedule (σ) is calculated based on the optimal makespan and optimal throughput using Eqs. (7) and (8), respectively:

$$Eff^{\sigma}_{makespan} = \frac{Makespan_{\sigma}}{OptimalMakespan} \times 100, \tag{7}$$

$$Eff^{\sigma}_{throughput} = \frac{Throughput_{\sigma}}{OptimalThroughput} \times 100. \tag{8}$$

$Eff^{\sigma}_{makespan}$ is the efficiency of the schedule with respect to makespan and $Eff^{\sigma}_{throughput}$ is the efficiency with respect to throughput. The intensity ($Intensity_{\sigma}$) of the schedule is then calculated by finding a cumulative of the efficiency values obtained. Eq. (9) is used to find the intensity of a firefly/schedule:

$$Intensity_{\sigma} = (Eff^{\sigma}_{makespan} \times \pi) + (Eff^{\sigma}_{throughput} \times \rho). \tag{9}$$

Here π and ρ refer to the weightage allocated to makespan and throughput, respectively.

Since in this paper we are trying to achieve an optimal schedule considering both the minimization of makespan as in Eq. (4) and maximization of throughput as in Eq. (6), the problem can be classified as a multiobjective optimization problem.

3. Firefly algorithm

The FA is a metaheuristic optimization algorithm based on the flashing characteristics of fireflies [19]. The algorithm has been formulated with three main assumptions: 1) all fireflies are unisexual, which eliminates the possibility of attraction based on sex, i.e. each firefly will be attracted by all other fireflies; 2) attraction depends on the amount of brightness where a less bright firefly is attracted to a brighter one and the brightest firefly will move randomly; 3) the brightness of the firefly is determined by the objective functions.

3.1. Pseudocode

The following steps are taken in the FA:

1. Define the representation of a single firefly, $\mathrm{X}_p = (\mathrm{x}_1 \ldots \mathrm{x}_d)$, which is a vector. 'd' denotes the dimension, which is the size of the vector. 'x_i' denotes the value of a single element in the vector where $1 \leq i \leq d$ and p is the index number of a single firefly.
2. Initialize population of fireflies $\mathrm{X} = \{\mathrm{X}_i\}$ where (i = 1 to size). 'size' refers to the size of the population.
3. Calculate intensity using the objective functions of makespan and throughput.
4. While t < size,

   For i = 1: n all n fireflies,

   For j = 1: i all n fireflies,

   If $(\mathrm{Intensity}_j > \mathrm{Intensity}_i)$, move fireflies i and j according to attractiveness.

   Evaluate new solutions. Update light intensity for next iteration. End if.

   End for j. End for i.

   Sort the fireflies to find the brightest one.

   End while.
5. The brightest firefly (optimal schedule) obtained is saved.

3.2. Phases in the algorithm

The algorithm includes three main phases: representation, initialization, and updating.

3.2.1. Representation

Let N refer to the population size and k refer to the index of the iteration; the firefly population is defined as $X^k = \left(X_1^k, X_2^k, \ldots, X_N^k\right)$ where X_i^k denotes the firefly i in the kth iteration. Each firefly is a possible schedule; length of each firefly is a vector of length 'n' where 'n' is number of jobs; each element inside the vector may take random values between 1 and 'm', where 'm' is the number of resources. The continuous position vector X_i^k is converted to a discrete permutation S_i^k based on the shortest position value (SPV) rule [20]. The resource values (R_i^k) are determined using the permutation function:

$$R_i^k = \left(S_i^k mod m\right) + 1. \tag{10}$$

Eq. (10) determines the resource number to which a particular job has been assigned. For example, for 5 jobs and 3 resources, Table 1 denotes the firefly representation. In Table 1, the first row denotes the continuous position values X_i^k generated in the initialization process. The SPV rule is applied to these continuous values. The corresponding discrete values generated are given in the second row, S_i^k. These discrete values represent the job numbers. The third row, R_i^k, denotes the resource values allocated for the corresponding job values. The sequence of the jobs is first-in-first-out, so the jobs are executed in the order they enter the system.

Table 1. Solution representation example.

Jobs (dimension)	1	2	3	4	5
X_i^k	5.14	2.54	3.85	–0.9	–2.56
S_i^k	5	3	4	2	1
R_i^k	3	1	2	1	3

3.2.2. Initialization

It takes the following steps:

1. Primary population is generated randomly.
2. The processing times are generated randomly based on the distribution.
3. The initialized fireflies are continuous values given by Eq. (11):

$$X_{i,j}^0 = X_{\min} + (X_{\max} - X_{\min}) \times U(0,1). \tag{11}$$

Here $X_{i,j}^0$ stands for the initial population generated. This will be a set of vectors with continuous values. $X_{\min} = 0.4$ and $X_{\max} = 4.0$ and U (0, 1) is a random variable between 0 and 1.

3.2.3. Updating

The brightness β_0 of each firefly is calculated using the fitness functions f and g. The distance (r_{ij}) between any two fireflies is calculated using the Cartesian distance given in Equation (12):

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}. \tag{12}$$

The attractiveness (β) of a firefly is given as follows, where β_0 is the initial brightness:

$$\beta = \beta_0 e^{-\gamma r_{ij}^2}, m \geq 1. \tag{13}$$

Based on the brightness function and the distance, the fireflies in the population show movements. The random position value for a firefly is obtained using the following equation:

$$x_i(t+1) = x_i(t) + \beta e^{-\gamma r_{ij}^2}(x_i - x_j) + \alpha(rand - \frac{1}{2}). \tag{14}$$

4. Artificial bee colony algorithm: the original version

The ABC algorithm is an optimization algorithm based on the foraging behavior of honey bees [21]. The main steps of the algorithm are:

1. Initialization: The food sources are initialized for the employed bees to go explore.
2. Iterate:

i. The employed bees go to their respective food source and explore the neighbor sources.

ii. They share the information with the onlookers.

iii. The onlookers select the sources based on the information and go evaluate the source.

iv. The abandoned sources are replaced with the new sources found by the scouts.

3. Until the stop conditions are met.

4.1. Phases in the algorithm

The three main phases in the search process are the employed bees phase, onlookers phase, and scouts phase. The parameters used in the algorithm are the number of food sources, termination criteria, and the limit after which the source is to be abandoned.

4.1.1. Initialization

Each food source is generated randomly using the below equation:

$$X_{ij} = lb_{ij} + rand(0,1)\,(ub_{ij} - lb_{ij})\,. \tag{15}$$

Here $\vec{X_i}$ is a vector in a population (i = 1... size, size: population size). X_i contains n variables, which are denoted by X_{ij} where j = 1...n (n is the vector size, i.e. number of food sources). 'rand' is a random variable between 0 and 1 and ub_{ij} and lb_{ij} are the upper bound and lower bound of the variable X_{ij}.

4.1.2. Employed bee phase

The employed bees will update the present solution based on the fitness value of the new solution. The position update equation is given by:

$$V_{ij} = X_{ij} + \varphi_{ij}\,(X_{ij} - X_{kj})\,. \tag{16}$$

Here $\vec{V_i}$ is the new food source vector and V_{ij} is a variable in the vector. $\vec{X_i}$ is the randomly selected food source vector and X_{ij} is a value in the vector (i = 1... size, size: population size and j=1...n). φ_{ij} is a random number within –1 and 1. Once $\vec{V_i}$ is generated, its fitness is compared with its parent, $\vec{X_i}$. A greedy selection is applied between them.

4.1.3. Onlooker bee phase

In this phase the employed bees share the fitness information about the new food sources with the onlookers. The onlookers evaluate the food source and calculate the selection probability (P_i) of each source using the below probabilistic selection function:

$$P_i = \frac{fit_i}{\sum_j fit_j}. \tag{17}$$

Here fit_i is the fitness value of the ith solution $\vec{X_i}$ and $\sum_j fit_j$ is the sum of the fitness values of all the solutions in the population (j = 1...size, size: population size).

4.1.4. Scouts phase

If a position cannot be improved over a predefined number of cycles determined by the parameter limit, then the food source is abandoned. A new food source is then discovered by the scouts using Eq. (15).

5. Artificial bee colony algorithm: the enhanced version

In the enhanced ABC algorithm (E-ABC), the genetic algorithm (GA) operators of crossover [22] and mutation [23] are included in the original ABC algorithm. The main steps of the algorithm are:

1. Initialization: The food sources are generated randomly using Eq. (15) for the employed bees to go explore.
2. Evaluate the fitness value of each food sources using Eq. (16).
3. Iterate:
   i) The employed bees go to their respective food source and explore the neighbor sources.
   ii) Apply the crossover operation to the selected food sources.
   iii) Calculate the probability for each source using Eq. (17).
   iv) The employed bees share the information with the onlookers.
   v) The onlookers select the sources based on the information and go evaluate the source and then update the probability for each source.
   vi) Apply the mutation operation on the selected food source.
   vii) The abandoned sources are replaced with the new sources found by the scouts.
4. Until the stop conditions are met.

5.1. Crossover

Crossover is the process of producing an offspring from more than one parent solution. Here two parent solutions are selected and one-point crossover is applied where the median position is selected as the single point of crossover. Then the resource values beyond that point for both parents are swapped. This operation is applied after the employed bee phase.

5.2. Mutation

Mutation is the process of altering one or more values in the solution. In E-ABC, uniform mutation is applied. A uniform random value is chosen between the upper and lower bounds and the value is used to replace the chosen resource value. This operation is applied after the onlooker phase.

6. Experimental setup and evaluation

The three algorithms are coded in JAVA and are run with NetBeans 8.0 on a system with the following configuration: processor: Intel CORE i3 2.13 GHz; RAM: 3.00 GB; Java Version: JDK1.8. The simulation parameters used for analysis of the FA are given in Table 2. Alpha denotes the randomization factor. It is kept at a medium level (0.5). The attractiveness of a firefly (beta = 1) is kept at a high level and the light absorption coefficient (gamma = 0.1) is kept at a low level in order to promote more movement among the fireflies and also to speed up the process. The simulation parameter values for the ABC algorithm are taken based on [9]. With θ = 0.1, 0.2, and 0.3 three distribution patterns are considered for generating the random values for processing times: normal distribution, uniform distribution, and exponential distribution. Here θ denotes the level of variability. In all cases, the mean values are generated from the uniform distribution $U(1, 99)$.

Table 2. Simulation parameters for firefly algorithm.

Parameters	Definitions		
	Algorithmic description	Symbols used	Values
Brightness	Objective functions	f and g	Already defined
Alpha	Randomization parameter	α	0.5
Beta	Attractiveness	β	1
Gamma	Absorption coefficient	γ	0.1
Population	Number of solutions per iteration	max	100
Dimension	Problem dimension, size of the vector (number of jobs 'n')	d	n (number of jobs)

Theoretically for an optimal makespan, we consider that there is no waiting time between jobs, so the makespan is calculated by only using the expected value of the processing times in each distribution. In the uniform distribution the processing times are taken in the range of [1, 99]. The expected value of a single variable (X) of range [a, b] in a uniform distribution is given by the below formula:

$$E(X) = \frac{b+a}{2}.$$

The expected processing time is 50 in a uniform distribution. With this the makespan is calculated for each instance. For example, in a 5 $\times$ 3 instance the makespan will be 5 $\times$ 50, which equals 250. For a normal distribution the expected value of a random variable is given by its mean valueμ, which is 49.5. The makespan value is calculated in a similar way as for uniform distribution. For exponential distribution the expected value is given by $1/\lambda$, which is the mean value μ. Based on the expected values of processing times the expected value of makespan is calculated for each distribution. Table 3 gives the expected values of makespan min (E ($C_{\max}$)), which is calculated from the expected values of processing times in each distribution. The experimental value obtained should be less then the expected value of makespan obtained in this theoretical evaluation.

The algorithms are run in multiple instances with 5 jobs and 3 resources (5 $\times$ 3), 30 jobs and 20 resources (30 $\times$ 20), and 80 jobs and 60 resources (80 $\times$ 60). Each algorithm is run ten times for a single instance and for a single input file. For throughput calculation the unit time taken in 5 $\times$ 3 instances is 100. For 30 $\times$ 20 the unit time taken is 1000 and for 80 $\times$ 60 instances the unit time taken for calculating the throughput is 3000. That is, if the time taken is in seconds, for 3000 s how many jobs in a schedule have completed their execution

Table 3. Expected makespan values in each distribution.

$\min(E(C_{max}))$	Uniform	Normal	Exponential
5 × 3	250	247.5	247.5
30 × 20	1500	1485	1485
80 × 60	4000	3960	3960

gives the throughput for that schedule. The results of individual runs are not shown due to space limitations. The values computed in each run are consolidated and the best (Bst), average (Avg), and worst (Wrst) values are computed for each algorithm.

Table 4 shows the computational results of the three algorithms under uniform distribution with varying levels of variability (θ) as 0.1, 0.2, and 0.3. Similarly, Table 5 shows the computational results of the algorithms under normal distribution also with varying levels of variability. Finally, Table 6 shows the performance of the algorithms with respect to the exponential distribution. The makespans obtained from the results are compared with the expected makespan values given in Table 3 to evaluate the performance of these algorithms. In the 5 × 3 instance the expected value for makespan is 250 and all three algorithms give a makespan that is less than this expected value. Similarly, in the 30 × 20 instance the expected value of makespan (=1500) is greater than the obtained makespan values in all three algorithms. In the 80 × 60 instance the obtained makespan values are much less than the expected value of makespan, which is 4000.

Now we have three different samples: FA, ABC, and E-ABC. The dependent variable, makespan, is measured at a continuous level. The independent variable, instance (jobs × resources), is measured as a categorical variable. To obtain the best performing algorithm among the three, we need to prove that the makespan obtained from one algorithm is less than the makespan obtained from the others. The Kolmogorov–Smirnov (KS) test is performed to determine whether a parametric or a nonparametric test is to be conducted on the samples. The null hypothesis for the KS test is "data follow a normal distribution". If the test is statistically significant, i.e. $P < 0.05$, then the null hypothesis can be rejected. From the tests conducted on all three samples, the P-value obtained is less than 0.05. Thus, it is proven that the data are not normally distributed. Hence, a nonparametric statistical test is to be performed to compare the three different populations.

The Mann–Whitney U test [24] is a nonparametric test performed to compare the makespan of the computational results statistically. The algorithms are run 10 times each. They are compared with each other: FA against ABC, FA against E-ABC, and ABC against E-ABC. The null hypothesis taken is "there is no difference in performance between the two algorithms". The critical value of U is 23 at significance level $P \leq 0.05$, so if the obtained U value is less than 23 at the 5% level then the null hypothesis can be rejected. The obtained values of U are listed in Table 7. From the U test we can see that in a smaller instance the U values obtained in FA vs. ABC and FA vs. E-ABC are less than 23. We can conclude that in a smaller instance with fewer number of jobs, FA has a significant performance difference when compared to ABC and E-ABC. However, when performing the test between ABC and E-ABC, the null hypothesis is proved, which denotes no performance difference between those algorithms. In larger instances, both ABC and E-ABC outperform FA by overruling the null hypothesis. The E-ABC algorithm gives optimal results in smaller and larger instances, but the complexity induced is high owing to the additional GA operators.

The histogram comparison of throughput of the three algorithms obtained from the results of the individual runs is shown in the Figure. The first three boxes correspond to the instance 5 × 3, the second three boxes correspond to the instance 30 × 20, and the third three boxes correspond to the instance 80 × 60. The Figure shows that the FA works better than ABC and E-ABC.

Table 4. Computational results under uniform distribution with θ= 0.1, 0.2, and 0.3.

Instances		Uniform distribution																	
		$\theta = 0.1$						$\theta = 0.2$						$\theta = 0.3$					
		Makespan			Throughput			Makespan			Throughput			Makespan			Throughput		
		Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst
Firefly	5 × 3	183	200	219	4	3.5	3	190	209	233	4	3.6	3	181	202	221	4	3.8	3
	30* × 20	1347	1675	1862	28	26	24	1276	1670	1806	30	26.7	25	1340	1658	1881	29	26.8	25
	80 × 60	3821	4259	5732	76	71	66	3721	4001	5477	75	72	69	3680	5125	5268	77	74.5	69
ABC	5 × 3	187	213	232	4	3.9	3	186	212	248	4	3.5	3	169	201	236	4	3.9	3
	30 × 20	1383	1487	1670	29	28.3	27	1308	1461	1545	29	28.2	27	1330	1508	1632	29	28	27
	80 × 60	3072	4703	5014	74	71	68	3453	4322	4822	77	71	66	4000	4819	5388	78	72	68
E-ABC	5 × 3	183	205	223	4	3.8	3	186	207	236	4	4	4	166	190.6	234	4	3.6	3
	30 × 20	1137	1277.9	1355	29	28.4	28	1178	1283.5	1322	29	28.8	28	1219	1293.7	1402	30	28.1	27
	80 × 60	3141	4424.2	4644	76	73	68	3163	3557.8	4920	78	72.5	67	3422	4616.7	4956	78	74	66

Table 5. Computational results under normal distribution with θ= 0.1, 0.2, and 0.3.

Instances		Normal distribution																	
		$\theta = 0.1$						$\theta = 0.2$						$\theta = 0.3$					
		Makespan			Throughput			Makespan			Throughput			Makespan			Throughput		
		Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst	Bst	Avg	Wrst
Firefly	5 × 3	196	229	290	4	3.3	3	213	245	302	4	3.2	3	209	263	365	4	3.3	3
	30 × 20	1412	1737.6	1852	28	25.2	23	1416	1613.5	1897	26	24.2	23	1386	1414.7	1519	24	22.9	22
	80 × 60	3472	5322.4	6038	75	69	61	3527	4668.4	5138	73	69.5	64	4004	5580.2	6082	74	69	62
ABC	5 × 3	195	242.5	289	4	3.3	3	199	225.6	265	4	3.4	3	209	256.6	340	4	3.5	3
	30 × 20	1364	1522.2	1655	28	27.5	27	1407	1532.6	1722	29	26.7	26	1323	1443.9	1522	28	27.4	26
	80 × 60	3580	5028	5571	74	71	69	5183	4477.6	5367	75	71	68	3076	4665.7	6367	77	73	68
E-ABC	5 × 3	183	233.9	293	4	3.6	3	199	246.5	280	4	3.8	3	25	220.4	305	4	3.2	3
	30 × 20	1213	1396.5	1531	29	28.9	28	1304	1416.7	1676	30	29.3	28	1304	1503.3	1776	30	28.1	27
	80 × 60	3562	4856.8	5006	75	70	64	3720	4970.4	5880	76	70.5	65	3092	4046.8	4880	74	72	69

Table 6. Computational results under exponential distribution.

Instances		Exponential distribution					
		Makespan			Throughput		
		Best	Average	Worst	Best	Average	Worst
Firefly	5 × 3	185	209	243	4	3.2	3
	30 × 20	1248	1398.7	1452	27	25.8	24
	80 × 60	3883	4283.4	5675	77	75	73
ABC	5 × 3	204	225.6	262	4	3.7	3
	30 × 20	1263	1434.1	1584	29	27.6	27
	80 × 60	3580	4102.9	5542	76	72.7	69
E-ABC	5 × 3	181	207.5	241	4	3.7	3
	30 × 20	1128	1378.9	1545	29	26.8	25
	80 × 60	3325	4008.1	4512	78	77	72

Table 7. Mann–Whitney U test results under uniform, normal, and exponential distributions.

Instance		Uniform			Normal			Exponential
		0.1	0.2	0.3	0.1	0.2	0.3	
5 × 3	FA & ABC	22.5	23	22	22	11	21.5	21.5
	ABC & E-ABC	35.5	30	36.5	35.5	34	35.5	29
	FA & E-ABC	11	27.5	22	14	11.5	22.5	17.5
30 × 20	FA & ABC	11	10	16	7.5	8	0	17
	ABC & E-ABC	0	3	4	15	1	10	17.5
	FA & E-ABC	0	7	0	1	0	0	5
80 × 60	FA & ABC	17	7	17	24	24	25	32
	ABC & E-ABC	2	0	0	0	0	0	0
	FA & E-ABC	0	0	3	0	0	0	0

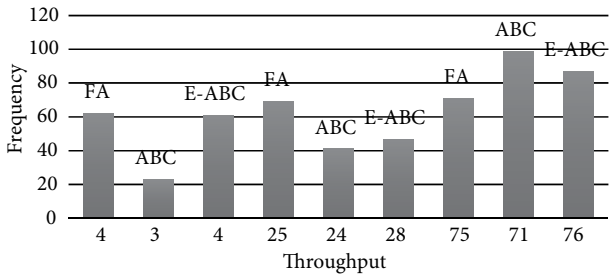

Figure. This histogram shows the frequency of throughput values obtained when the three algorithms are run in 5 × 3, 30 × 20, and 80 × 60 instances.

7. Conclusion and future work

The FA uses the process of attraction based on the brightness of fireflies to optimize an objective function. Prior research has shown that the algorithm can solve both continuous and discrete optimization problems. Given the algorithm's capability to be useful in both continuous and discrete domains, it is used to solve the task allocation problem, which falls under the category of NP-hard. The algorithm is also compared with two other algorithms,

the ABC and the enhanced ABC. The makespan values are evaluated using the Mann–Whitney U test and the throughput is evaluated with the help of histograms. Though the performance of E-ABC is significant when compared to the other two algorithms, owing to its complexity, the FA gives a better performance overall. Our future work lies in experimenting and making use of the many new evolutionary algorithms that have been proposed to improve the performance of distributed system environments.

References

[1] Drozdowski M. Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. Poznan, Poland: Politechnika Poznańska, Monografie, 1997.

[2] Xia W, Wu Z. An effective hybrid optimization approach for multiobjective flexible job-shop scheduling problems. Comput Ind Eng 2005; 48: 409-425.

[3] Deb K. Multi-Objective Optimization Using Evolutionary Algorithms. New York, NY, USA: John Wiley & Sons, 2011.

[4] Zitzler E, Thiele L. Multiobjective optimization using evolutionary algorithms — a comparative case study. In: Eiben AE, Back T, Schoenauer M, Schwefel H, editors. Parallel Problem Solving from Nature. Berlin, Germany: Springer, 1998. pp. 292-301.

[5] Koulamas C, Kyparisis GJ. An improved delayed-start LPT algorithm for a partition problem on two identical parallel machines. Eur J Oper Res 2008; 187: 660-666.

[6] Muthiah A, Rajkumar R. A comparison of artificial bee colony algorithm and genetic algorithm to minimize the makespan for job shop scheduling. In: 12th Global Congress on Manufacturing and Management; 8–10 December 2014; Vellore, India. Amsterdam, the Netherlands: Elsevier. pp. 1745-1754.

[7] Zalzala AMS, Fleming PJ. Genetic Algorithms in Engineering Systems. London, UK: Institution of Electrical Engineers, 1997.

[8] Singh MR, Mahapatra SS. A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks. Int J Adv Manuf Tech 2012; 62: 267-277.

[9] Li JQ, Xie S, Pan Q, Wang S. A hybrid artificial bee colony algorithm for flexible job shop scheduling problems. Int J Comput Commun 2011; 6: 286-296.

[10] Zhang R, Wu C. An artificial bee colony algorithm for the job shop scheduling problem with random processing times. Entropy 2011; 13: 1708-1729.

[11] Khadwilard A, Chansombat S. Thepphakorn T, Thapatsuwan P, Chainate W, Pongcharoen P. Application of firefly algorithm and its parameter setting for job shop scheduling. J Ind Tech 2012; 8: 1-10.

[12] Udaiyakumar KC,Chandrasekaran C. Application of firefly algorithm in job shop scheduling problem for minimization of makespan. In: 12th Global Congress on Manufacturing and Management; 8–10 December 2014; Vellore, India. Amsterdam, the Netherlands: Elsevier. pp. 1798-1807.

[13] Yousif A, Abdulah AH, Nor SM, Bashir MB. Optimizing job scheduling for computational grid based on firefly algorithm. In: IEEE Conference on Sustainable Utilization and Development in Engineering and Technology; 6–9 October 2012; Kuala Lumpur, Malaysia. New York, NY, USA: IEEE. pp. 97-101.

[14] Varadharajan TK, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. Eur J Oper Res 2005; 167: 772-795.

[15] Zhang G, Shao X, Li P, Gao L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 2009; 46: 1309-1318.

[16] Karthikeyan S, Asokan P, Nickolas S, Page T. A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. Int J Bioisp Comp 2015; 7: 386-401.

[17] Marichelvam MK, Prabaharan T, Yang XS. A discrete firefly algorithm for the multi-objective hybrid flow shop scheduling problems. IEEE T Evolut Comput 2014; 18: 301-305.

[18] Tavakkoli-Moghaddam R, Jolai F, Vaziri F, Ahmed PK, Azaron A. A hybrid method for solving stochastic job shop scheduling problems. App Math Comput 2005; 170: 185-206.

[19] Yang XS. Firefly algorithms for multimodal optimization. In: Watanabe O, Zeugmann T, editors. Stochastic Algorithms: Foundations and Applications. Berlin, Germany: Springer, 2009. pp. 169-178.

[20] Tasgetiren F, Chen A, Gencyilmaz G, Gattoufi S. Smallest position value approach. In: Godfrey C Onwubolu, Davendra D, editors. Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. Berlin, Germany: Springer, 2009. pp. 121-138.

[21] Karaboga D, Gorkemli B, Ozturk C, Karaboga N. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 2014; 42: 21-57.

[22] Kumar S, Sharma VK, Kumari R. A novel hybrid crossover based artificial bee colony algorithm for optimization problem. Int J Comput Applic 2013; 82: 18-25.

[23] Singh A, Gupta N, Sinhal A. Artificial bee colony algorithm with uniform mutation. In: Proceedings of the International Conference on Soft Computing for Problem Solving; 20–22 December 2011; Roorkee, India. Berlin, Germany: Springer. pp. 503-511.

[24] Nachar N. The Mann-Whitney U: a test for assessing whether two independent samples come from the same distribution. Tutorials in Quantitative Methods for Psychology 2008; 4: 13-20.