Indonesian Journal of Electrical Engineering and Computer Science
Vol. 20, No. 3, December 2020, pp. 1556~1568
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v20.i3.pp1556-1568

New hybrid flower pollination algorithm with dragonfly algorithm and jaccard index to enhance solving university course timetable problem

Ma. Shiela C. Sapul[1], Rachsuda Setthawong[2], Pisal Setthawong[3]
[1,2]Computer Science Department, Assumption University, Thailand
[3]Management Information Systems Department, Assumption University, Thailand

Article Info

Article history:

Received Apr 3, 2020
Revised Jun 6, 2020
Accepted Jun 20, 2020

Keywords:

Course timetable
Dragonfly algorithm
Flower pollination algorithm
Jaccard index
Metaheuristic
Optimization

ABSTRACT

University course timetable problem (UCTP) is one of the problems on which many researches have been conducted over the years because of its importance in academic institutions. A nature-inspired metaheuristic optimization algorithm, flower pollination algorithm (FPA) has been adapted, so-called adapted FPA (AFPA), to cope with UCTP in the previous work. However, AFPA suffers from the stagnation problem because of the non-diversity in the population. To improve the diversity of the population, this work introduces new Hybrid FPA with two variants: JFPA provided the Jaccard index to determine similarities among categorical data and the greedy selection mechanism to improve the selection of the random solution, and DFPA applied the navigational characteristics of the dragonfly algorithm (DA) to help in the neighborhood relationship. The results in this study indicate that the proposed algorithms have better exploration ability and fast convergence rate in comparison to previous approaches; JFPA outperforms AFPA in 3 out of 4 datasets for both small and large datasets, and DFPA outperforms AFPA and GA in all datasets while it outperforms PSO in 3 out of 4 small datasets and 2 complex large datasets.

Corresponding Author:

Ma. Shiela C. Sapul,
Assumption University, Bangkok, Thailand.
Email: p5919398@au.edu

1. INTRODUCTION

In each academic semester, universities or colleges must face the problem of scheduling their resources. The scheduling problem is a broader class of combinatorial optimization problems of allocating resources to events. One typical scheduling problem is the timetable problem. Timetable scheduling is considered a difficult problem because they are made complicated by the variations in the constraints and due to the nature of the problem to solve. In solving optimization problems, the major goal is to search the best possible way to find the optimum solution, and the searching process should be completed in the shortest amount of iteration or shortest possible time. This paper focuses on automatically solving the university course timetable (UCTP) in the optimal time because traditionally academic institutions manually solve timetable problems, which takes days or weeks to satisfy the constraints and comply with the nature of the problem. The problem consists of assigning appropriate courses to lecturers and other resources (room, day and time) while maximizing given constraints. The proposed solution should satisfy both hard and soft constraints. Hard constraints are constraints that must be satisfied no matter what to obtain a feasible timetable. There are two main hard constraints: lecturer constraint and scheduling constraints. Lecturer constraint implies that a lecturer can teach a subset of courses. Scheduling constraints imply that two or more courses cannot be assigned to a lecturer at the same time, and two or more courses cannot be assigned to the same room at the same time. Soft constraints, on the other hand, are desired to be satisfied only, but if done so, it does not stop the timetable from being a valid solution, such soft constraint included in this study is to include the minimum or maximum number of courses should be taken or the total number of teaching load assignments a lecturer is allowed.

The existing solution techniques for timetable problems range from the manual generation of a timetable [1] to metaheuristic algorithms. The manual generation of the timetable is tedious and time-consuming and it does not guarantee that an optimal course timetable has been generated. Meta-heuristic algorithms, on the other hand, can find good solutions with less computational effort compared with traditional techniques. Metaheuristic algorithms such as genetic algorithm (GA) and particle swarm optimization (PSO) have been reported to be able to solve many optimization problems. GA was first used for a timetabling problem in 1992 [2] and has been applied to various scheduling problems [3-6]. PSO has ties with genetic algorithm and evolutionary algorithms [7] and has been used also for solving the various timetabling problems [8-10]. FPA is another meta-heuristic algorithm introduced to solve general optimization problems like finding solutions for mathematical equations [11]. It has been formulated for multi-objective optimization applications by mimicking the pollination process of flowering plants [12]. It has enticed the interest of several researchers because of its characteristics; like it uses fewer parameters and has shown good results when applied to various optimization problems. In the survey study of [13], they found out that the number of parameters can affect the complexity of an algorithm. Another similar survey study on metaheuristic algorithms that have adopted a large number of parameters can be a disadvantage because it requires that the parameters used must be tuned for the optimal task [14].

FPA was originally designed to solve continuous optimization problems, by modifying the FPA; it is possible to solve a subset of the UCTP problem and promising results proves that FPA can be used effectively to solve a combinatorial problem like UCTP [15]. FPA also suffers from a lack of a good balance between exploration and exploitation optimization process which may lead to a non-diverse population. A lot of researchers have proposed various strategies to enhance FPA performance and its implementation to various optimization problems; a study on flexible job shop scheduling problems wherein the crossover operator of GA and tabu method are applied to enhance the search capability of FPA [16]. Another study shows that when FPA is combined with other algorithms, it can outperform the combined algorithm and can improve its speed and convergence rate [17]. Because the population size may affect the results of the generation of the population, mutation operators were introduced in FPA [18]. Another study that suggests that FPA's performance can be improved by introducing the clonal operator from the Clonal Selection Algorithm (CSA) and replacing the levy flight using the uniform random distribution to improve the global pollination process and continuous checking of the best solution can avoid being trapped in local optima [19]. In the survey study on metaheuristic algorithms [14] also concluded in their findings that applying a combination of two metaheuristics can improve another metaheuristic algorithm's performance. To the best of our investigation, the majority of the studies presented for solving optimization problems using FPA are applied to engineering and industry-related problems [20-23]. Despite the enhancements and related works on FPA mentioned in this study, FPA has certain limitations or issues: FPA cannot deal directly with combinatorial problems like UCTP, the exploration ability of FPA is dependent on the levy flight which can be aggressive and can cause large steps and a non-diverse population may lead to local optima and the parameter settings in an FPA may affect also the performance of the algorithm.

The main contribution of this paper is to design and develop two new hybrid flower pollination algorithms: JFPA and DFPA, which improve the performance of adapted-flower pollination (AFPA) algorithm [15] in solving the lecturer-course assignment problem and extend our work to solving university course timetable problem (UCTP). AFPA has demonstrated that FPA can be adapted to solve a combinatorial problem. However, according to our observation, AFPA has limitations. It suffers from a non-diverse population when the population is too similar that may result in a low convergence rate or being restrained to the local optima. Another possible cause of the lack of diversity in the population is the ineffectiveness of the parameter settings. To overcome the limitations of AFPA, the two hybrid FPA algorithms: JFPA and DFPA, which are capable of diversifying the population, are introduced. A similarity factor is proposed in JFPA to facilitate the searching process when comparing data. This factor helps control diversity by measuring the similarity and dissimilarity of two individual solutions in addition to traditional fitness value [24]. Since the individual genomes of UCTP are presented as a categorical data, we introduced a Jaccard index to find the similarities of categorical data. Therefore, JFPA takes into account not only the fitness of the individual solutions but also the diversity among them. To improve the selection of the random solution in the local pollination a selection factor is embedded also in JFPA. According to [25] algorithms diversity problem occurs if the algorithm's optimization process cannot balance between exploration and exploitation. A good balance between exploration and exploitation is another important criterion for the performance of the algorithm [14]. Another reason for a non-diverse population is that the parameter choice can influence the algorithm's performance [26]. To balance AFPA's exploration and exploitation process, the other proposed algorithm DFPA utilizes the navigational behavioral characteristics of the Dragonfly algorithm that mimics the swarming behaviors of dragonflies, which are similar also to the exploration and exploitation phases of optimization [27-29].

2. RESEARCH METHOD

2.1. The adapted FPA (AFPA)

AFPA introduced in [13] aims to solve the lecturer-course assignment, which is considered as a combinatorial problem. It provided a discrete representation of the solution by defining the position of the flowers as a combinatorial set of resources assigned to an event. For the flower pollination process, it presents the redesigning of the step size or the scaling factor for updating the candidate solutions position. Also, it applies the GA operators to provide the flower constancy operators. The pseudocode of AFPA is given in Figure 1. In generating the initial population (line 1), the AFPA randomly assigns the resources to events like in the case of assigning courses to the lecturer and assigning room, day, and time to the course-lecturer assignment. Then, the algorithm utilizes the selection operator in getting the fittest flower solution (X_{best}), the current flower solution ($X_i^{\ t}$), and a random flower solution in the current iteration (X_j). Mutation (lines 10 and 22) and crossover (lines 12, 14, 16, 24, 26, and 28) operators of the GA are used as the flower constancy operators of AFPA.

```
1     Initialize a population of $n$ flowers/pollen gametes with random solutions
2     Find the best solution $X_{best}$ in the initial population
3     Define a switch probability $p \in [0,1]$
4     WHILE ($t < maxgeneration$)
5       FOR $i$=1: $n$
6         IF ($rand<p$) THEN
7         DO global pollination
8         $\Delta_{gp}=fv(X_{best})-fv(X_i^{\ t})$
9           IF ($\Delta_{gp}<pr_1$) THEN
10            $X_i^{\ t+1}$=mutationOneClass ($X_i^{\ t}$)
11          ELSE IF ($\Delta_{gp}<pr_2$) THEN
12            $X_i^{\ t+1}$=crossOverHalfWorkload($X_i^{\ t}$, $X_{best}$)
13          ELSE IF ($\Delta_{gp}<pr_3$) THEN
14            $X_i^{\ t+1}$=crossOverHalfnWorkload($X_i^{\ t}$, $X_{best}$)
15          ELSE
16            $X_i^{\ t+1}$=crossOvernWorkload($X_i^{\ t}$, $X_{best}$)
17          ENDIF
18        ELSE
19        DO local pollination
20        $\Delta_{lp}=fv(X_j^{\ t})-fv(X_i^{\ t})$
21          IF ($\Delta_{lp}<pr_1$) THEN
22            $X_i^{\ t+1}$=mutationOneClass ($X_i^{\ t}$)
23          ELSE IF ($\Delta_{lp}<pr_2$) THEN
24            $X_i^{\ t+1}$=crossOverHalfWorkload($X_i^{\ t}$, $X_j$)
25          ELSE IF ($\Delta_{lp}<pr_3$) THEN
26            $X_i^{\ t+1}$=crossOverHalfnWorkload($X_i^{\ t}$, $X_j$)
27          ELSE
28            $X_i^{\ t+1}$=crossOvernWorkload($X_i^{\ t}$, $X_j$)
29          ENDIF
30        END IF
31        IF ($fv(X_i^{\ t+1})>fv(X_i^{\ t})$) THEN
32          $fv(X_i^{\ t})=fv(X_i^{\ t+1})$
33        Find and update the best solution $X_{best}$
34      END FOR
35    END WHILE
```

Figure 1. Pseudocode of AFPA

The random value *rand* and the switch probability *p* are used to control the switching between the global and local pollination process, and at the same time determine the flower constancy operation that the individual flower serves (line 6). The fitness value *fv* as shown in 1 is derived by getting the number of violations in satisfying the constraints for each solution; the anticipated result equal to 1.0 indicating an optimal solution in the population.

$$fv = (1.0 - (\frac{\Sigma tconflict}{tcourse})) \qquad (1)$$

During global and local pollination, the step size for both pollination Δ_{gp} (line 8) *and* Δ_{lp} (line 20) is derived by getting the difference of the fitness values of two flower solutions; a set of parameters pr_1, pr_2, and pr_3 are used to determine how AFPA will behave, in other words, selecting a GA operator for pollination. AFPA has some drawbacks in getting the algorithm stuck in local optima since the FPA has the weakness of having to use a diverse population; in particular, both local (line 19) and global processes (line 7) use the same scaling factor. To solve this diversity problem, we should provide a good balance between the global and local process during the search process.

2.2. Variations of FPA for UCTP

To address the limitations of AFPA as discussed in the previous section, this work proposes two hybridizations of AFPA: hybrid AFPA with similarity index using Jaccard index and selection factor (JFPA) and hybrid AFPA with navigational characteristics of the dragonfly (DFPA). JFPA applies the Jaccard index as another scaling factor to enhance the local pollination process. Jaccard index provides the similarity measurement of the information contained for each flower solution. This scaling factor dictates how the local pollination will behave and this helps the pollination process the chance that flower solutions may have the best information to exchange. The selection factor helps improve the selection of the best random flower. The pseudocode of JFPA is given in Figure 2.

```
1  Initialize a population of $n$ flowers/pollen gametes with random solutions
2  Find the best solution $X_{best}$ in the initial population
3  Define a switch probability $p \in [0,1]$
4  WHILE ($t<maxgeneration$)
5    FOR $i$=1: $n$
6     IF ($rand < p$) THEN
7      DO global pollination
8      $\Delta_{gp}=fv(X_{best})-fv(X_i^{\,t})$
9      IF ($\Delta_{gp} \leq pr$) THEN
10       $X_i^{\,t+1}$=mutationOneClass($X_i^{\,t}$)
11     ELSE
12       $X_i^{\,t+1}$=crossOverHalfWorkload($X_i^{\,t}$, $X_{best}$)
13      ENDIF
14    ELSE
15     DO local pollination
16     Randomly choose $X_j$ using the selection factor
17     $\Delta_{lp}=sf(X_j, X_i)$ // using Eq. 2
18     IF ($\Delta_{lp}>sim\_th$) THEN
19      $X_i^{\,t+1}$=mutationOneClass ($X_i^{\,t}$)
20     ELSE
21      $X_i^{\,t+1}$=crossOverHalfWorkload($X_i^{\,t}$, $X_j$)
22     ENDIF
23    END IF
24    IF ($fv(X_i^{\,t+1})>fv(X_i^{\,t})$) THEN
25     $fv(X_i^{\,t})=fv(X_i^{\,t+1})$
26   Find and update the best solution $X_{best}$
27   END FOR
28 END WHILE
```

Figure 2. Pseudocode of JFPA

During global and local pollination, the Δ_{gp} and Δ_{lp} are used to derive the step size of the pollination; Δ_{gp} uses the fitness value to determine the step size of the pollination (line 8), and Δ_{lp} uses the similarity factor to determine the step size of the pollination (line 17). The similarity factor *sf* is defined in (2).

$$sf = \frac{d}{(b+c+d)}, \qquad (2)$$

where d are those instances where X_i and X_j both have a value of 1, b are those instances where the attribute of X_i is 0 and the attribute of X_j is 1 and c are those instances when the attribute of X_i is 1 and the attribute of X_j is 0. Also, the parameters *pr* and *sim_th* are used to select a GA operator (swap mutation or a crossover) for pollination. It should be noted that JFPA excludes some operators of AFPA: crossOverHalfnWorkload() and crossOvernWorkload() because based on the experimental results observed, these operators do not help a lot in providing a diverse population.

For DFPA, neighborhood selection is important because it will improve search efficiency and increase population diversity. To accomplish this DFPA embeds the DA's navigational behavior with FPA and also divides the population into a set of subpopulations by finding the similarity between all solutions to distribute the solutions into different neighbors. As shown in Figure 3, the algorithm initially finds the similarity of the solutions to create or update the radius of the clusters or neighbors (lines 7 and 8). During global and local pollination, the update of the new flower solution X_i^{t+1} depends on the neighborhood relationship of X_i. If X_i has neighbors, then both the direction ΔX, and the position of X_i are updated (lines 14 and 21). Otherwise, only the position is updated.

```
1   Initialize a population of $n$ flowers/pollen gametes with random solutions
2   Find the best solution $X_{best}$ in the initial population
3   Define a switch probability $p \in [0,1]$
4   Determine how many clusters $C_n$
5   WHILE ($t<maxgeneration$)
6     Randomly select the vector or center point for each cluster $C_1, C_2, \ldots, C_n$
7     Determine the similarity values $sv(X_i)$ in $C_1, C_2, \ldots, C_n$
8     Update the neighboring radius by assign $X_i$ to a cluster $C_i$
9     Determine the $X_{best}, X_{worst}, X_{flowbest}$
10    FOR $i$=1:$n$
11       IF ($rand<p$) THEN
12        DO global pollination
13        IF the $X_i$ has at least one neighbor
14              Update $\Delta X_i^{t+1}$ and $X_i^{t+1}$//using 8 and 9, respectively
15              ELSE
16              $X_i^{t+1}$=mutationOneClass ($X_i^t$)
17              END IF
18       ELSE
19              DO local pollination
20              IF the $X_i$ has at least one neighbor
21          Update $\Delta X_i^{t+1}$ and $X_i^{t+1}$//using 8 and 9, respectively
22              ELSE
23              $X_i^{t+1}$= mutationOneClass ($X_i^t$)
24              END IF
25       END IF
26       IF ($fv(X_i^{t+1})>fv(X_i^t)$) THEN
27              $fv(X_i^t)=fv(X_i^{t+1})$
28    END FOR
29  END WHILE
```

Figure 3. Pseudocode of DFPA

Due to space limitation, we do not provide the traditional operators of DA, but it can be referred to in [27]. Dragonfly Algorithm (DA) updates the direction and position of the individual flowers depends on five navigational behavioral characteristics of the dragonflies: separation S, alignment A, cohesion C, attraction F, and destruction E as determined in 3–7, respectively. The Separation S operator provides collision avoidance of the solutions found in the neighbor, and each neighbor avoids colliding with others in the neighborhood by determining the similarity of their solutions. This operation will be repeated between X_i and its entire neighbor X_j. If similarity value $sv(X_i, X_j)$ is greater than 0.5, then perform the 3.

$$\mathrm{S} = \mathrm{crossoverHalfWorkloadSwap}\ (X_i, X_j), \tag{3}$$

where crossoverHalfWorkloadSwap(X_i, X_j) selects randomly half of the genomes from the workload from X_i's neighbor X_j and if they are found in X_i, then they are swapped. 4 provides the Alignment A operator to ensure that solutions are moving towards the same best flower solution in the neighborhood, wherein the information from the best flower $X_{flowbest}$ in the current cluster is shared with the current flower X_i of the same cluster.

$$A = \text{crossoverHalfWorkloadA}(X_i, X_{flowbest}), \tag{4}$$

where crossoverHalfWorkloadA(X_i, $X_{flowbest}$) selects randomly half of the genomes from the workload from the best solution $X_{flowbest}$ in the neighborhood and locates the occurrence in X_i, and replace half of the genomes at the same random position as indicated in $X_{flowbest}$. 5 provides the Cohesion *C* operator, it maintains a form of neighborhood, all the solutions in the same neighborhood exchange information.

$$C = \text{crossoverHalfWorkloadC}(X_i, X_j), \tag{5}$$

where the crossoverHalfWorkloadC(X_i, X_j) selects randomly half of the genomes from the workload from the neighborhood X_j and locate the occurrence in X_i, and replaces half of the genomes at the same random position as indicated in X_j. 6 provides the Attraction *F* operator *crossover*, ensures that the current flower solution X_i tends to move towards the food source or the best flower X_{best} in the entire population by exchanging information from the best flower in the entire population.

$$F = \text{crossoverHalfWorkloadF}(X_i, X_{best}), \tag{6}$$

where crossoverHalfWorkloadF(X_i, X_{best}) selects randomly half of the genomes from the workload from the best flower X_{best} in the population and locate the occurrence in X_i, and replaces half of the genomes at the same random position as indicated in X_{best} 7 provides the Destruction *D* operator; it ensures that the current flower solution is kept away from the worst solution.

$$D = \text{crossoverHalfWorkloadRandomSwap}(X_i, X_{worst}), \tag{7}$$

where crossoverHalfWorkloadRandomSwap(X_i, X_{worst}) selects randomly half of the genomes from the workload from the worst solution X_{worst} in the population and if they are found in X_i then they are swapped by selecting a new random genome in X_i. During the global and local pollination, the ΔX serves as step vector for modifying direction of the flower solutions as shown in 8, and the new position of X_i of the flowers can be derived in 9.

$$\Delta X_i^{t+1} = S_i + A_i + C_i + F_i + E_i \tag{8}$$

$$X_i^{t+1} = X_i + \Delta X_i^{t+1} \tag{9}$$

3. RESULTS AND ANALYSIS

To assess the performance of the proposed improvements of FPA, we compared the results with other meta-heuristic algorithms: Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The performance of the algorithms was examined concerning different levels of constraint complexity from low to high. We provided four sets of constraints with complexity equal 10%, 30%, 50%, and 60% for the lecturer-course assignment, respectively; the lower the complexity of the problem was, the higher the percentage of feasible courses that a lecturer could teach to. The dataset consists of two categories: one for small scale sample size (60 courses offered in a semester, 15 lecturers), and another one for large scale sample size (200 courses offered in a semester, 50 lecturers), and a set of constraints. For each test, the maximum iteration numbers were set to 1000 and 3000, respectively; the numbers of population *n* were set to 10, 50 and 100, respectively. The results are evaluated in terms of the following: fitness value, max generations or iterations, how the sizes of the datasets affect the performance of the algorithm, how the population can affect the performance in terms of complexity, and how the complexity constraint can affect the behavior of the algorithm.

Tables 1 and 2 summarize the convergence rate of the 3 benchmarked algorithms: GA, PSO, and AFPA, and the two proposed algorithms: JFPA and DFPA. It should be noted that the numbers listed in Tables 1 and 2 are the maximum fitness value *(fv)* generated at the corresponding iteration (*gen).* The results that are optimal for each test case are highlighted in bold to help display the best approach. Experiment results in Tables 1 and 2 shows that DFPA tends to find the optimal solution faster than the benchmarked algorithms and of the other proposed algorithm JFPA at higher complexity. We further perform the detailed analysis and comparison of the 4 algorithms: PSO, AFPA, JFPA, and DFPA as depicted in Figure 4 to 23. It is remarked that the experimental results of GA are excluded from the analysis and comparison because it does not reach the global optimal value (*fv*=1.0).

Table 1. Summary of fitness value and generation number (small scale dataset)

Pop	Complexity	GA		PSO		AFPA		JFPA		DFPA	
Size	Level	F_V	Gen	F_V	Gen	F_V	Gen	F_V	Gen	F_V	Gen
10	10	1.00	707	1.0	18	1.00	22	1.0	23	**1.0**	**14**
	30	0.88	199	**1.0**	**52**	1.00	175	1.0	110	1.0	56
	50	0.70	860	1.0	332	1.00	360	1.0	335	**1.0**	**191**
	60	0.58	979	1.0	632	1.00	932	**1.0**	**554**	1.0	587
50	10	1.00	117	**1.0**	**2**	1.00	4	1.0	4	**1.0**	**2**
	30	0.90	121	**1.0**	**40**	1.00	122	1.0	63	1.0	42
	50	0.73	115	1.0	295	1.00	210	1.0	299	**1.0**	**147**
	60	0.62	129	1.0	497	1.00	506	1.0	356	**1.0**	**293**
100	10	1.00	53	**1.0**	**1**	1.00	2	**1.0**	**1**	**1.0**	**1**
	30	0.93	789	**1.0**	**19**	1.00	52	1.0	52	1.0	34
	50	0.77	316	1.0	154	1.00	199	1.0	169	**1.0**	**132**
	60	0.65	337	1.0	488	1.00	338	1.0	295	**1.0**	**252**

Table 2. Summary of fitness value and generation number (large scale dataset)

Pop	Complexity	GA		PSO		AFPA		JFPA		DFPA	
Size	Level	F_V	Gen	F_V	Gen	F_V	Gen	F_V	Gen	F_V	Gen
10	10	0.97	12	**1.00**	**97**	1.00	495	1.0	433	1.0	127
	30	0.80	2256	**1.00**	**527**	1.00	1343	1.0	885	1.0	768
	50	0.62	1062	1.00	1813	1.00	2362	1.0	2119	**1.0**	**1676**
	60	0.52	838	0.98	1822	0.98	2709	1.0	2784	**1.0**	**2776**
50	10	0.98	2751	**1.00**	**65**	1.00	349	1.0	287	1.0	98
	30	0.82	742	**1.00**	**417**	1.00	954	1.0	789	1.0	516
	50	0.64	1600	1.00	1483	1.00	1964	1.0	1925	**1.0**	**1188**
	60	0.53	873	0.99	2662	0.99	2852	**1.0**	**2572**	**1.0**	**2615**
100	10	0.98	1773	**1.00**	**40**	1.00	293	1.0	212	1.0	69
	30	0.83	430	**1.00**	**235**	1.00	754	1.0	748	1.0	375
	50	0.66	314	1.00	1197	1.00	1686	1.0	1883	**1.0**	**824**
	60	0.56	2035	0.99	2545	0.99	2642	1.0	2836	**1.0**	**1995**

Figures 4 to 6 show a comparison of how the algorithms behave (complexity behavior) for small dataset; it can be observed in the results that increasing the population size improves the convergence (less number of iterations. DFPA tends to improve its number of iterations despite increasing the level of complexity and since the iterations are improving in DFPA it also finds the optimal solution faster than the benchmarked algorithms and of JFPA.

Figures 7 to 9 show that the PSO and AFPA suffers when it is tested on large data set, both algorithms were not able to achieve the global optimum value (fv=1.0) in the most constraint complexity level (60%); increasing the population size did not improve the number of iterations (decreasing), and increasing the number of iterations did not also help in achieving the maximum value. DFPA, on the other hand, increasing the size of the dataset and increasing the constraint complexity level tend to improve the number of iterations; as the population size is increased, DFPA can find the maximum value faster than the benchmarked algorithms.

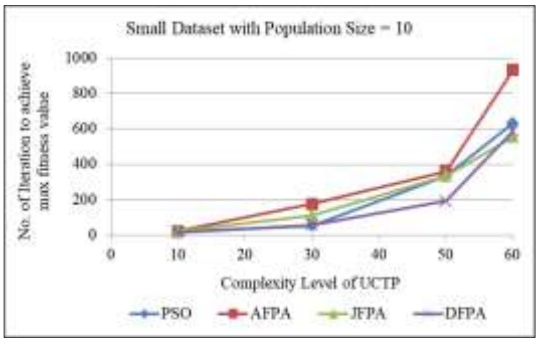

Figure 4. Complexity behavior
(small scale data set, population size=10)

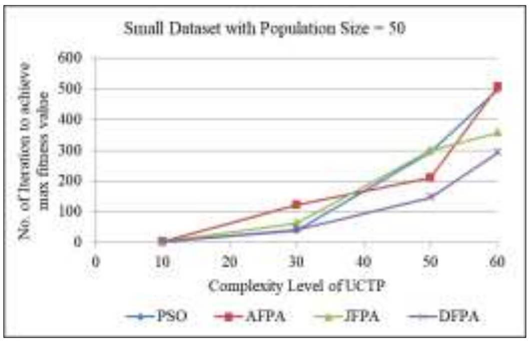

Figure 5. Complexity behavior
(small scale data set, population size=50)

The results can be showed in Figures 10 to 13 that the population size influences the algorithm's performance as indicated by the number of iterations. Increasing the population may help, but when the complexity constraints are added, it may not help improve the number of iterations as the complexity level is increased. For all the test results shown in Figures 10 to 13, DFPA performs better whether its population size and complexity level are increased. As indicated, it has less iteration compared with the algorithms mentioned in the figures.

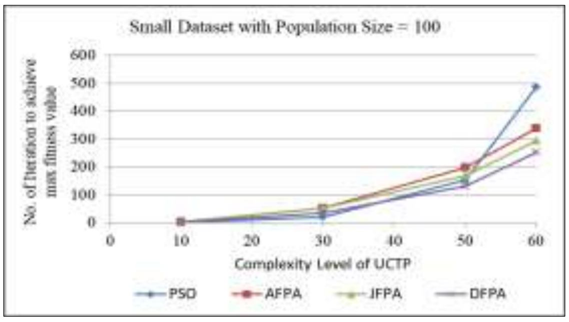

Figure 6. Complexity behavior (small scale data set, population size=100)

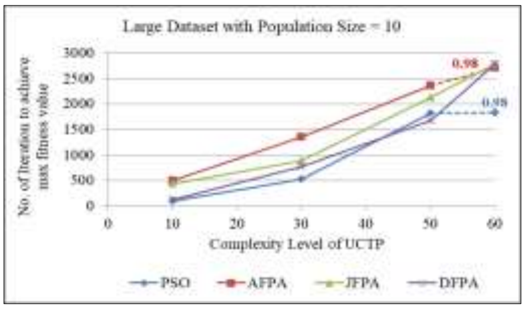

Figure 7. Complexity behavior (large scale data set, population size=10)

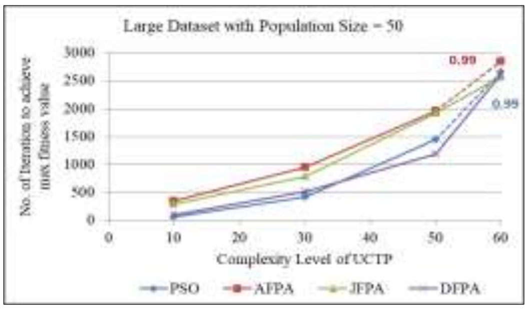

Figure 8. Complexity behavior (large scale data set, population size=50)

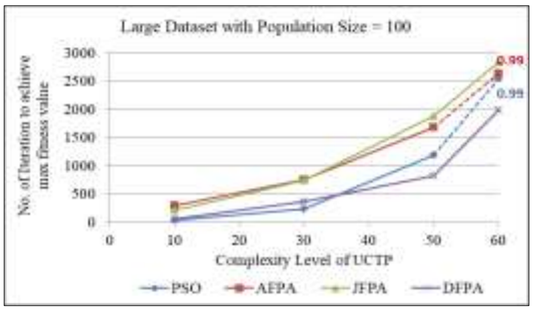

Figure 9. Complexity behavior (large scale data set, population size=100)

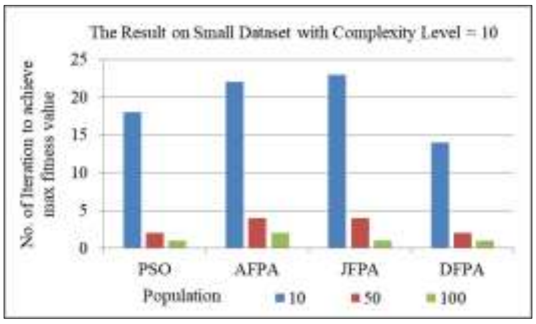

Figure 10. Algorithms behavior for various population sizes–complexity level=10 (small scale data set)

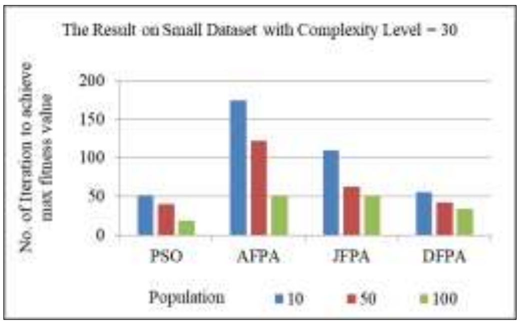

Figure 11. Algorithms behavior for various population sizes–complexity level=30 (small scale data set)

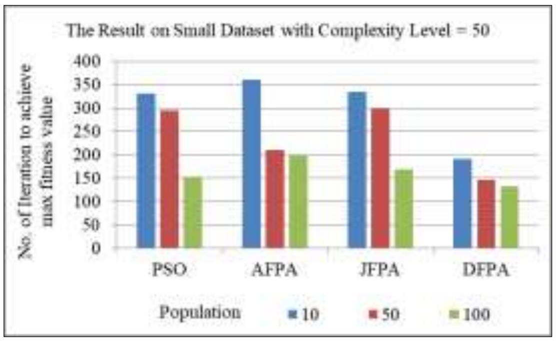

Figure 12. Algorithms behavior for various population sizes–complexity level=50 (small scale data set)

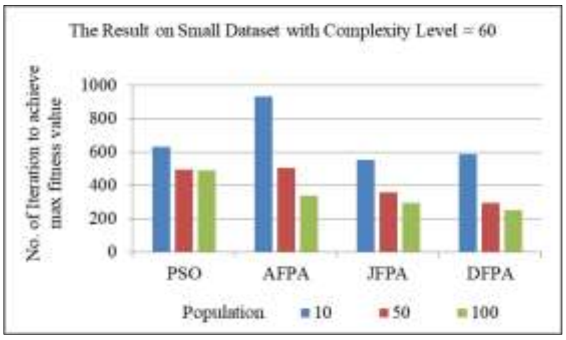

Figure 13. Algorithms behavior for various population sizes–complexity level=60 (small scale data set)

One way to test the performance of the algorithm for various population sizes is to test whether the size of the dataset affects its performance. Results in Figures 14 to 17 shows that a large dataset may affect the performance of the algorithm as indicated by a higher number of iterations required to reach the global maximum value. It should be addressed that in Figure 17 PSO and AFPA did not provide an optimal solution for the large scale dataset with complexity level equal 60 (as referred in detailed result in Table 2), therefore, the figure shows the maximum number of iteration that they can achieve their highest fitness values (less than 1.0). The dash bar in Figure 17 indicates non-optimal results (fv<1.0). Increasing the size of the population may help achieve the optimal solution, but it requires several iterations. It can also be observed from the test results that increasing the population size may not help at all some algorithms (PSO and AFPA) achieve the optimal solution because of the constraint complexity. Despite the constraint factor in improving the results by increasing the population, among the algorithms presented in the Figures, DFPA provided better results in the most constraint complexity level. Referring to the results in Tables 1 and 2, Figures 4 to 17 it can be observed that increasing the population size significantly improves the number of iterations, and referring to Table 2, increasing also the number of iterations allows the algorithm to achieve the maximum fitness value for large scale dataset. It has been observed in Figures 10 to 16, increasing the population size increases the chance of finding the optimal solution. However, for some algorithms (PSO and AFPA) increasing the population size may not improve the solution or the convergence rate because population size depends on other parameters like the difficulty of the problem or the complexity level. Also, increasing the number of iterations did not make some algorithms (PSO and AFPA) to find the maximum fitness value in the large scale dataset especially for the most complex dataset.

The performance of an algorithm can be tested in terms of the execution time. Figures 18 to 20 describe the performance of the algorithms for small scale data set at 1000 iterations. In particular, increasing the population size will result in an algorithm's execution time longer. Increasing the population size and increasing the complexity level tend not to improve the execution time. The two proposed two-hybrid FPA relatively takes a longer time compared with the AFPA, this is due to the additional operations to be performed during local pollination in JFPA, and as the optimization progresses in DFPA there is a constant update of the neighborhood in both local and global pollination.

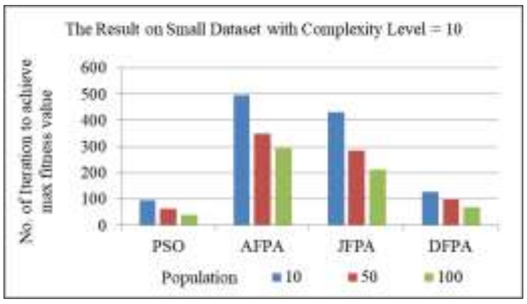

Figure 14. Algorithms behavior for various population sizes–complexity level=10 (large scale data set)

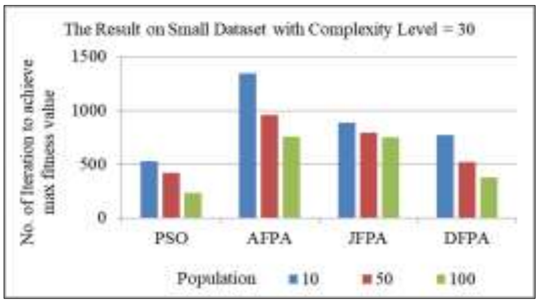

Figure 15. Algorithms behavior for various population sizes–complexity level=30 (large scale data set)

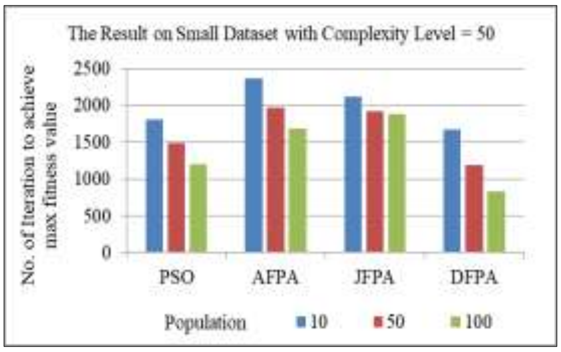

Figure 16. Algorithms behavior for various population sizes–complexity level=50 (large scale data set)

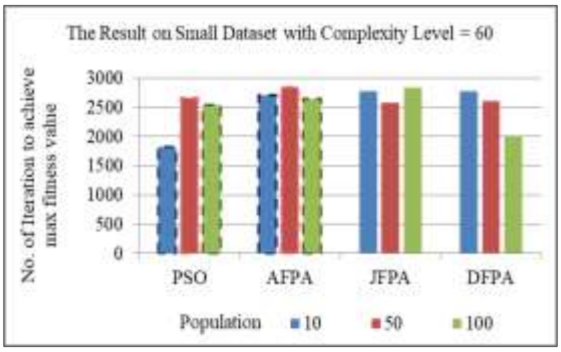

Figure 17. Algorithms behavior for various population sizes–complexity level=60 (large scale data set)

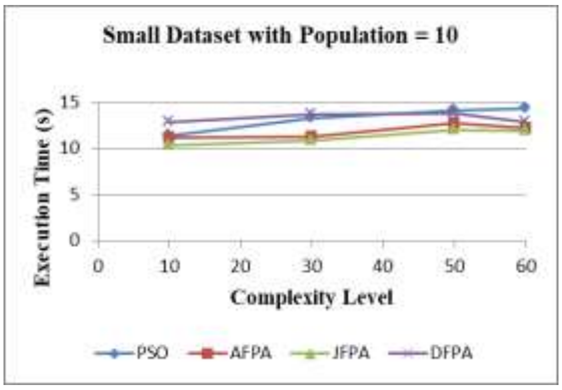

Figure 18. Execution time (small scale data set, population size=10)

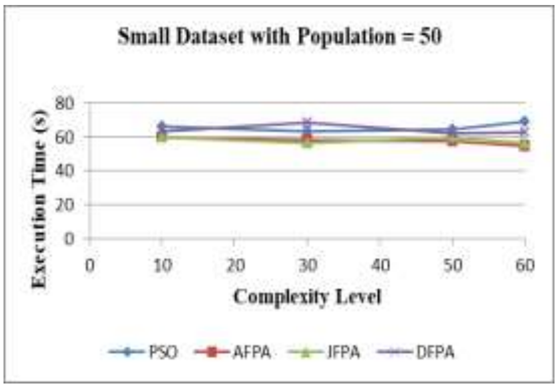

Figure 19. Execution time (small scale data set, population size=50)

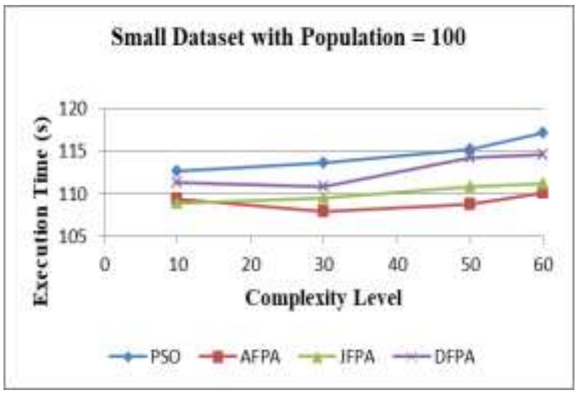

Figure 20. Execution time (small scale data set, population size=100)

Another way to test the performance of the algorithm in terms of the execution time is for various population sizes is to test whether the size of the dataset affects its performance. Results in Figures 21 to 23 shows that a large dataset may affect the execution time as indicated by the higher number of the execution time. Generally, increasing the population size as well as the size of the data and the number of iterations can increase the execution time for all algorithms. The two proposed two-hybrid FPA relatively takes a longer time compared with the AFPA; this is due to the same reason as stated in Figures 18 to 20.

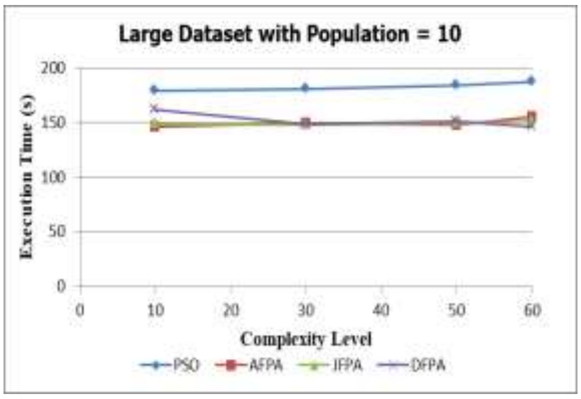

Figure 21. Execution time
(large scale data set, population size=10)

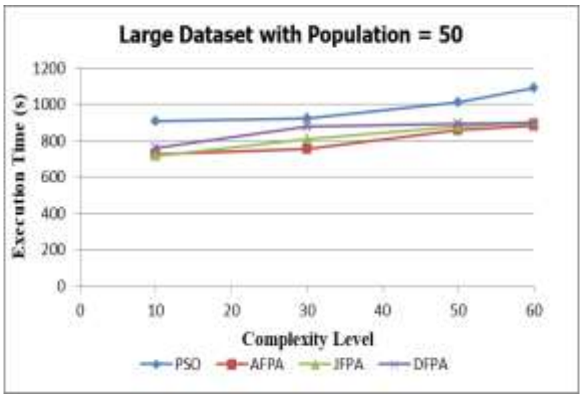

Figure 22. Execution time
(large scale data set, population size=50)

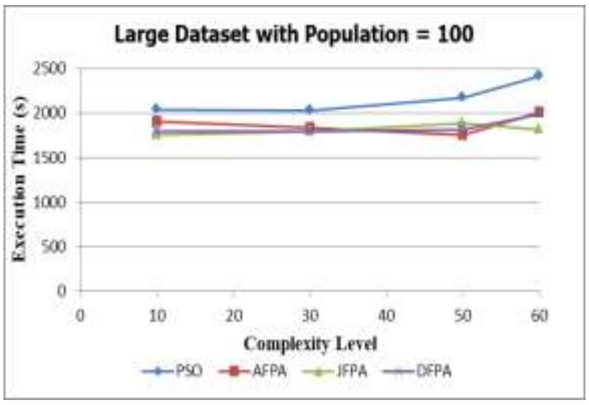

Figure 23. Execution time (large scale data set, population size=100)

3.1. Parameter tuning of FPA

The behavior of the original FPA is determined by the number of flowers in the population *n*, switch probability *p*, scaling factor, step size **Δ,** and the number of iterations *i*. In this study we fix the value of the switch probability (0.80), for the scaling factor we applied the Jaccard index to enhance the local pollination process in JFPA, and to increase the diversity of the population; we applied DA's navigational behavior with DFPA. For the size of the population and the number of iterations parameters, we tried different values. By providing different values for the population size and the number of iterations allows us to test if increasing the population size can improve the algorithm's performance. Referring to the results in Tables 1 and 2, it can be observed that increasing the number of population size, it helps the performance of the algorithm even in the most constraint complexity level. It can be seen also in Figures 4 to 17 that using a different scaling factor helped the algorithm exchange the best information.

4. CONCLUSION

In the previous work Adapted-FPA (AFPA), FPA has been modified to address the limitation of FPA to solve combinatorial problems. However, when solving constrained problems, FPA suffers from slow convergence and easy to fall to local optima because of the diversity problem. This happens when the solutions generated are too similar. Another cause of the diversity problem is the ineffectiveness of the parameter settings applied. To cope up with the observed limitation in AFPA, our work proposed Hybrid AFPA with two improvements. In JFPA, the algorithm firstly improved the diversity of the population by introducing the Jaccard index in the local pollination process a strategy that can measure the similarity of the solutions for categorical datasets. In the previous work of AFPA, the fitness value was used as the scale factor to determine the similarity of the solutions. However, this is not a feasible measurement or parameter when determining the similarities for categorical data; this technique gives the local pollination a better judgment on how the solutions are similar.

Another improvement included in the JFPA is applying a selection factor that allows the selection of the local pollination for the best random flower. Therefore, JFPA can ensure that only the best random flower is used in the pollination process. As a result, this gives a chance that using the best random flower will also produce better new solutions. Another improvement is by improving the exploration and exploitation process of the AFPA, wherein we applied the navigational characteristics of the DA, which provides the DFPA a selection mechanism for making sure that solutions in the same neighborhood do not collide with each other, and is moving towards the best solution and avoiding worst solution. Likewise, the similarity factor is also used to provide the neighborhood solution to distribute the solutions into different solution subsets.

The results in both Tables and Figures 4 to 17 show that both JFPA and DFPA outperform AFPA, DFPA's performance proves its ability to adaptively search the solution space for the most promising solution for solving UCTP when compared to similar approaches and better than the other variants of the AFPA. DFPA converges faster but has a higher execution time compared with AFPA and JFPA as reported in Figures 18 to 23. It can be also considered that execution time is affected by several factors, such as algorithms can work differently on different input size or the size of the data, CPU waiting time and other programs that might be running in parallel. Our future work will focus on improving the performance of Hybrid FPA by further researching on developing the parallel version of the Hybrid FPA to speed up computation time. Also, it is feasible to include an internal memory for the personal best and neighbor best that is available in PSO to help improve the search process in DFPA.

REFERENCES

[1] Murphy, J., & Sutter, R., "School Scheduling by Computer," the Story of GASP, 1964.

[2] Colorni, A., Dorigo, M., & Maniezzo, V., "Genetic algorithms: A new approach to the timetable problem," In *NATO ASI Series, Combinatorial Optimization*, vol. F82, pp. 235-239, Springer, Berlin, Heidelberg, 1992.

[3] Li, D. C., Hsu, P. H., & Chang, C. C., "A genetic algorithm-based approach for single-machine scheduling with learning effect and release time," *Mathematical Problems in Engineering*, vol. 2014, pp. 1-12, 2014.

[4] Poorjafari, V., Yue, W. L., & Holyoak, N., "A comparison between genetic algorithms and simulated annealing for minimizing transfer waiting time in transit systems," *International Journal of Engineering and Technology*, vol. 8, no. 3, pp. 216-221, 2016.

[5] Budhi, G. S., Gunadi, K., & Wibowo, D. A., "Genetic Algorithm for Scheduling Courses," In *International Conference on Soft Computing, Intelligence Systems, and Information Technology*, pp. 51-63, Springer, Berlin, Heidelberg, 2015.

[6] Kadam, V. J., Upadhye, S. V., & Laddha, M. D., "Implementation of University Sports Time table Scheduling using Genetic Algorithm-A Case Study," *International Journal of Advance Foundation and Research in Science & Engineering (IJAFRSE)*, vol. 1, 2015.

[7] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, pp. 39-43, 1995. doi: 10.1109/MHS.1995.494215.

[8] S. F. H. Irene, S. Deris and M. H. S. Zaiton, "A Study on PSO-Based University Course Timetabling Problem," *2009 International Conference on Advanced Computer Control*, Singapore, pp. 648-651, 2009, doi: 10.1109/ICACC.2009.112.

[9] Shu-Chuan Chu, Yi-Tin Chen and Jiun-Huei Ho, "Timetable Scheduling Using Particle Swarm Optimization," *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*, Beijing, pp. 324-327, 2006. doi: 10.1109/ICICIC.2006.541.

[10] Chen, R. M., & Shih, H. F., "Solving university course timetabling problems using constriction particle swarm optimization with local search," *Algorithms*, vol. 6, no. 2, pp. 227-244, 2013.

[11] Yang, X. S., "Flower pollination algorithm for global optimization. In *International conference on unconventional computing and natural computation*, pp. 240-249, Springer, Berlin, Heidelberg, 2012.

[12] Yang, X. S., Karamanoglu, M., & He, X., "Flower pollination algorithm: a novel approach for multiobjective optimization," *Engineering Optimization*, vol. 46, no. 9, pp. 1222-1237, 2014.

[13] BoussaïD, I., Lepagnot, J., & Siarry, P., "A survey on optimization metaheuristics," *Information sciences*, vol. 237, pp. 82-117, 2013.

[14] Dokeroglu, T., et al., "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, 2019.

[15] M. S. C. Sapul, R. Setthawong and P. Setthawong, "Adapted Flower Pollination Algorithm for Lecturer-Class Assignment," *2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT)*, Yogyakarta, Indonesia, pp. 315-321, 2019. doi: 10.1109/ICAIIT.2019.8834620.

[16] A. Phuang, "The flower pollination algorithm with disparity count process for scheduling problem," *2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Phuket, pp. 1-5. 2017. doi: 10.1109/ICITEED.2017.8250497.

[17] D. Chakraborty, S. Saha and O. Dutta, "DE-FPA: A hybrid differential evolution-flower pollination algorithm for function minimization," *2014 International Conference on High Performance Computing and Applications (ICHPCA)*, Bhubaneswar, pp. 1-6, 2014. doi: 10.1109/ICHPCA.2014.7045350.

[18] Salgotra, R., & Singh, U., "Application of mutation operators to flower pollination algorithm," *Expert Systems with Applications*, vol. 79, no. 112-129, 2017.

[19] Nabil, E., “A modified flower pollination algorithm for global optimization,” *Expert Systems with Applications*, vol 57, pp. 192-203, 2016.
[20] S. Sutradhar, N. B. D. Choudhury and N. Sinha, "Hydrothermal scheduling using Modified Flower Pollination Algorithm: A parallel approach," *2016 IEEE Region 10 Conference (TENCON)*, Singapore, pp. 1696-1700, 2016. doi: 10.1109/TENCON.2016.7848307.
[21] Gupta, I., Kaswan, A., & Jana, P. K., “A Flower Pollination Algorithm Based Task Scheduling in Cloud Computing,” In *International Conference on Computational Intelligence, Communications, and Business Analytics*, pp. 97-107, Springer, Singapore, 2017.
[22] Abdel-Baset, M., & Hezam, I., “A hybrid flower pollination algorithm for engineering optimization problems. *International Journal of Computer Applications*, vol. 140, no. 12, pp. 10-23, 2016.
[23] Łukasik, S., & Kowalski, P. A., “Study of flower pollination algorithm for continuous optimization. In *Intelligent Systems' 2014*, Springer, Cham, pp. 451-459, 2015.
[24] Niwattanakul, S., et al., “Using of Jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, 2013.
[25] Herrera-Poyatos, A., & Herrera, F., “Genetic and memetic algorithm with diversity equilibrium based on greedy diversification. *arXiv preprint arXiv:1702.03594*, pp 1-27, 2017.
[26] Cui, W., & He, Y., “Biological flower pollination algorithm with orthogonal learning strategy and catfish effect mechanism for global optimization problems,” *Mathematical Problems in Engineering*, 2018.
[27] Mirjalili, S., “Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems,” *Neural Computing and Applications*, vol. 27, pp. 1053-1073, 2016.
[28] Sambandam, R. K., & Jayaraman, S., “Self-adaptive dragonfly based optimal thresholding for multilevel segmentation of digital images,” *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 4, pp. 449-461, 2018.
[29] Raman, G., et al., “Dragonfly algorithm based global maximum power point tracker for photovoltaic systems,” In *International Conference on Swarm Intelligence*, pp. 211-219, Springer, Cham, 2016.