Artificial Bee Colony Optimization Algorithm for the Field Technician Scheduling Problem

Han Wang
Department of Computer Science
Pennsylvania State University Harrisburg
Middletown, PA 17057
hfw5079@psu.edu

***Abstract*—**Artificial Bee Colony (ABC) optimization algorithm, which is a swarm intelligence technique, has been researched and applied into many difficult problems. This paper addresses an NP-hard real life scheduling problem, the Field Technician Scheduling Problem (FTSP), by applying ABC optimization algorithm. An FTSP's instance contains a set of tasks and a set of technicians, the objective of FTSP is to maximize the total priority of scheduled tasks at these technicians. This paper purposes some heuristics and generate some mutant algorithms of the basic ABC algorithm. By comparing the experimental results of different mutant algorithms, the mutant algorithm that uses all heuristics performs consistently well. We also compare the results of our proposed mutant algorithm with Genetic Algorithm on the same FTSP instances. Our results show that using ABC optimization algorithm can also generate good solutions.

Keywords—Artificial Bee Colony (ABC) optimization algorithm, Field Technician Scheduling Problem (FTSP), Swarm intelligence

I. INTRODUCTION

There are numerous complicated NP-hard problems which cannot be solved by finding the best solution under polynomial time. For example, to solve Travelling Salesman Problem, one of the most famous NP-hard problems, the best exact algorithm found so far requires running time of $O(n^2 2^n)$. In many cases, we cannot afford to spend that much time for the best solution. Instead, an approximate solution, which is optimized, is good enough.

Field Technician Scheduling Problem (FTSP), a real life scheduling problem, is one NP-hard problem. We have only limited technicians but many tasks, sometimes we want to schedule as many tasks as possible at these technicians, sometimes we want to maximize the total benefit of scheduled tasks. Imagine a team leader who manages 10 team members in a development department, he receives hundreds of development tasks from the sales department every day, how can he maximize the total benefit of tasks he can schedule?

In the FTSP, we are given a set of technicians with different work times and skill levels, and a set of tasks with different time windows, priorities, skill requirements and processing times. One technician can schedule a task if only he has the matching skill level. Damm et al. [1] describe the main objective of FTSP, which is "to maximize the sum of priority values associated with the tasks performed". The authors also provide a data set of this problem for the experiments. We will use the same data set to run experiments with ABC optimized algorithm and its mutant algorithms.

The ABC optimization algorithm, proposed by Karaboga and Basturk [2] [3], is a swarm intelligence based optimization technique. The authors introduce a colony of artificial bees that contains three groups of bees: employed bees, onlooker bees and scouts. Each employed bee can be imagined as a solution of the problem, all employed bees share their solutions through the use of a fitness function. Onlooker bees can decide which solution to exploit depending on the fitness function values. Scouts search for finding new solutions. Bees exploit their solutions by neighbor selection, where the best solution is stored. This algorithm has been applied to many problems. Karaboga and Gorkemli [4] applied this algorithm into Travelling Salesman Problem. Karaboga et al. [5] applied this algorithm for training feed-forward neural networks. Gao and Liu [6] improved this algorithm for global optimization.

This paper introduces a new fitness function to judge the quality of each scheduled task. A greedy heuristic using this fitness function during the process of local search and neighbor selection is proposed. Under this heuristic, a technician always wants to drop the worst task and share the best task to neighbors. Different variations of the basic ABC algorithm are also proposed and compared during experiments.

This paper has three main contributions. First, as far as we know, the ABC optimization algorithm has not been applied to FTSP yet. This paper presents some heuristics, uses different combinations of them to generate different mutant algorithms based on ABC optimization and evaluate their effectiveness. The second contribution is that by experimenting the same data set provided by Damm et al. [1], we will compare both results and see which one has a better performance. The third contribution is that, we will find out less time than that used in the referred paper is sufficient for us to achieve similar results, which shows the efficiency of our algorithm.

This paper is organized as follows. Section 2 presents some related work. Section 3 presents the definition and constraints of FTSP. Section 4 presents the basic ABC algorithm and some heuristics we will use. Section 5 introduces mutant algorithms of ABC algorithm by using different heuristics. Section 6 describes the computational experiments and analyzes the results. Section 7 presents the conclusion of our paper and summarizes our work.

II. RELATED WORK

The FTSP is a scheduling problem that is similar to Vehicle Routing Problem with Time Windows (VRPTW), which has been researched in many papers. Solomon [9] proposed tour-building algorithms that brought good performance. Desrochers et al. [10] proposed an optimization algorithm based on column generation approach that was proved successful. Gambardella et al. [11] proposed multiple ant colony systems for this problem. Tsang and Voudouris [12] introduced a fast local search and a guided local search algorithm that "help local search to escape local optima and distribute search effort" for scheduling problems.

FTSP is recognized as an extension to the Vehicle Routing Problem with Time Windows (VRPTW). Major differences between them are:

- Each schedule (task) in FTSP is assigned with a priority, which is the most important vector to judge how good a solution is.
- Each schedule in FTSP is assigned with a skill level, only technicians with the matching skill level can work on it.
- Each schedule in VRPTW is assigned with a demand while each vehicle has assigned max capacity, any vehicle cannot be overloaded.

Xu J and Chiu SY [8] formally defined the FTSP, and also introduced a greedy heuristic, a local search algorithm, and a greedy randomized adaptive search procedure for this problem. The authors presented four local search operations, addition, exchange, change, and swap, some of which will also be used in our paper.

Damm et al. [1] not only provided data set for this problem, but also introduced three heuristics: shortest travel time, nearest technician, and cluster. "A biased random key genetic algorithm (BRKGA)" was also proposed and applied. Experiments were made and results were presented. These results will be compared with those generated by ABC optimization algorithm in our paper.

III. PROBLEM DEFENITION

In the FTSP, each instance has the following data:

- number of tasks
- number of technicians
- for each task: priority, time window, process time and location
- for each technician: work time and the skill capability to all the tasks

And we want to maximize the total priority of all scheduled tasks, with following constraints:

1) Each task has a specific time window {start_Time, end_Time}. It cannot be executed before start_Time, and it cannot be finished after the end_Time. Any task scheduled by a technician can only be executed in this time window.

2) Each technician has a specific work time window {start_Time, end_Time}. He cannot start work before start_Time, and cannot work overtime.

3) Technicians have to take some travel time from the location of one task to that of next task, and the travel time is calculated by Euclidean distance. If the two consecutive tasks that one technician schedules are at the same location, then travel time is zero.

4) For each task, only technicians who have the matching skill level can schedule it. In each instance we are given the binary arrays of skill level map which indicates if each technician has the matching skill level for each task. Technicians with low skill level cannot schedule a task that requires high skill level.

5) Each task can only be scheduled at most once. No two technicians can schedule the same task, and no technician can schedule one task two or more times. This is very important in this project implementation. We need to prevent this error from happening.

We are using the same data set as Damm et al. [1] and we want to compare both experimental results. At the beginning of running each instance we will generate some initial solutions by either greedy heuristics or random method. We will record the best initial solution, the best solution by ABC optimization algorithm and the best solution by the biased random key genetic algorithm [1]. We want to show how many instances we achieve the same good solution as the referred paper, and how much we improved from the best initial solution for the rest of instances.

IV. ABC AND HEURISTICS

As we introduced before, the key idea of ABC algorithm is to do neighbor selection by a probability selection and generate a mutant of the solution, and then see if this mutant is better than original solution. If yes, the original solution is replaced by this mutant. The following pseudo codes describe the basic ABC optimization algorithm.

Algorithm 1: basic ABC

1. Generate initial solutions for each worker bee
2. While stopping criterion is not satisfied do
   a) Get fitness function values of each bee
   b) Each onlooker bee decides which worker bee to follow
   c) Roulette wheel selects one bee
   d) Select one scheduled task of selected bee
   e) Other bees try to schedule selected task
      i. If successful, set count to 0
      ii. Otherwise, add count
   f) Check if any bee reaches maximum count. If yes, this bee's solution is abandoned so that it needs a new solution to exploit
   g) Store the best solution so far
3. End while

Since FTSP is an NP-hard problem and we want to optimize our solution, we will try more different heuristics.

A. Shortest Travel Time (STT) introduced by Damm et al. [1]

Firstly, we sort all tasks in a specific order. There are three options:

- Descending order of priority
- Ascending order of process time
- Descending order of priority / process time

Secondly, for each task in order, we decide the list of technicians that can schedule this task.

Then we calculate the travel time of each technician from location of previous task to that of this task. Choose the technician with shortest travel time. This satisfies our logical thought, since shorter travel time increases the probability for technicians to schedule more tasks. This paper uses this heuristic to generate some initial solutions.

B. Random Solution (RS)

By using this method, we don't use swap – one of the local search methods presented by Xu J and Chiu SY [8] any more. Random solutions bring more tasks to be selected and chosen. They are also improved during the process of running the experiments, like better candidate pools that make selecting good candidate tasks more efficient. This method is used to generate some new solutions.

C. Add one task WithOut Drop (AWOD)

Given a task to one technician, he can decide if he can schedule this task without dropping any scheduled tasks. This method works similar to add – one of the local search methods, and will definitely increase the total priority of solution.

D. Add one task With Drop (AWD)

Given a task to one technician, if he cannot simply add this task without drop, he can consider if he can schedule this task at the cost of dropping one scheduled task. This method will generate a mutant of the current solution, and then we can determine if this mutant is better than the original solution. Also, sometimes the current solution is stuck in local optima; in other words, it hasn't been improved after some specific time. This method can increase the probability of jumping out of stuck optima.

E. EXchange Two tasks among technicians (EXT)

Given two technicians, Tech1 and Tech2, suppose they have scheduled Task1 and Task2. We can try to exchange the two tasks, Tech1 to schedule Task2 and Tech2 to schedule Task1, if they are able to. This method will not increase the total priority of solution, but it can help them try to schedule more tasks and also increase the probability of jumping out of stuck local optima.

In our experiments, we also consider the possibility that one technician can schedule the other technician's task while the other cannot. This makes the solution more active, though it hurts the temporary solution's quality.

F. EXchange Whole technicians' tasks among solutions (EXW)

Given two solutions of the same instance, since they have same technicians and tasks, they are able to exchange all tasks scheduled by each other. We need to ensure that no task is scheduled more than once. This will bring quicker improvement than EXT.

G. Shrink Scheduled Tasks (SST)

For one task to be scheduled by one technician, the execute time can be a range. In other words, the task can be scheduled earlier or later. This method is to move the previous task earlier and current task later, by which the gap between these two tasks gets larger. Therefore, it increases the probability of the technician scheduling more tasks in this gap.

In our implementation, we always try to schedule one task at the very earliest slot, and we don't consider the idle time that technicians go back to origin before leaving work. So the most important thing under this method is to move the last task as late as possible.

H. Select Worst Task for one technician (SWT)

One technician may be able to schedule several tasks. From the implementation we observe that the number is about 5 to 10. This method is to use greedy heuristic to select the worst task to be dropped or exchanged. This satisfies our logical thought since we want to drop the worst task so that we have more opportunities to schedule better tasks if we are technicians.

Our paper uses this greedy heuristic. For one task, we consider the time spent related on it. The time when the previous task ends and the time when the next task begins are the two boundaries, and the gap between them is the whole time related to this task. This whole time contains the travel time from the previous task to current task and from current task to the next task, waiting time for current task and next task, and the process time of current task. We divide the priority of this task by this whole time, and this is our objective function value for this method. We sort objective function values for all scheduled tasks in ascending order and use roulette wheel probability selection to choose one task to be dropped when operating exchange or drop.

I. Select Best Task from one technician (SBT)

Using the same method as SWT we can get the objective function values for all scheduled tasks. We sort all scheduled tasks' objective function values in descending order and use roulette wheel probability selection to choose one task as the best task.

To select the best task among scheduled tasks, our paper also uses another greedy heuristic where we sort the tasks only by priority / process time since sometimes we consider the context of one task in one solution different from that in another solution. We can use this method for neighbor selection; that is, we want to share the best task in one solution with other solutions.

J. *Combination of Constructive method and Local search (CCL)*

Evangeline [13] proposed this combination and we can run this algorithm by making some empty initial solutions. Empty solutions will be constructed using chosen local search methods and neighbor selection. This method is also more controllable for each technician to try to do his best in each step.

V. ALGORITHM MUTANTS

Algorithm 1 is the basic ABC algorithm. It uses STT, RS, AWOD, and AWD heuristics listed above. No other heuristics are applied in this algorithm. To compare the experiment results and show which one or which combination is better, we try many different mutants of ABC algorithm with one or more heuristics listed above.

A. *Mutant Algorithm with Local search (MAL)*

This is basic ABC plus EXT, EXW, and SST heuristics. In each round when running the algorithm, these three local search methods will be tried to help improve the solutions.

Algorithm 2. MAL

1. Generate initial solutions for each worker bee
2. While stopping criterion is not satisfied do
   a-g) Steps a-g same as algorithm 1
   h) Try exchange tasks among technicians
   i) Try exchange whole technicians' tasks among solutions
   j) Try shrink scheduled tasks
3. End while

B. *Mutant Algorithm of Combination of Constructive and Local search (MACCL)*

This is a mutant of algorithm MAL and we also try constructive method. We make some of the initial solutions empty and see how well this constructive method performs. Later on we will compare this mutant algorithm with others.

C. *MAL with Select Worst (MASW)*

This is a mutant algorithm of MAL. When performing local search, especially during the process of exchange and drop, we apply the heuristic SWT to choose the worst task to be exchanged or dropped. We don't want to keep good tasks.

D. *MASW with Select Best by Priority / process time (MASWSBP)*

MASW only requires selecting the worst task to drop or exchange, but it doesn't ask which to select if we want to share one good task with neighbors. In this algorithm, we try another greedy heuristic that sorts scheduled tasks on priority / process time, to select the best one. Since we use heuristic SBT, we think this should work better than MASW.

E. *MASW with Select Best by Objective function (MASWSBO)*

This is very similar to MASWSBP. In this approach, we also use the same objective function proposed in heuristic SWT to select the best task for neighbor selection. This mutant algorithm should work better than MASW; however, it's hard to predict whether it's better or worse than MASWSBP until experiments are run. Later on in the experiments section we will see the comparison.

F. *MASWSBO with Combination of Constructive and Local search (MASWSBOCCL)*

This is a hybrid mutant of MACCL and MASWSBO. We want to include all heuristics we have introduced so far and would like to see if this is the best mutant algorithm in our paper.

VI. EXPERIMENTS

In the data set provided by Damm et al. [1], there are 13 cases and each case contains 80 instances. Table 1 shows how many tasks and technicians that one instance has corresponding to the case number. The running time for each instance in the same case are also shown in the table 1. Since the data set is too large and our experiment time is limited, we run only 40 instances for each case number.

TABLE 1: instances and running time generated by [1]

Case #	Tasks #	Technicians #	Running time (seconds)
1	16	2	3
2	26	2	5
3	30	3	10
4	39	3	50
5	45	7	75
6	64	5	80
7	80	13	180
8	100	10	250
9	120	20	480
10	150	25	600

In our experiments, for each instance we will record the best initial solution (INIT), the best solution generated by our algorithm (ABC), and the best solution (GA) generated by Damm et al. [1]. For the instances that ABC is equal or higher than GA, we will record the count. For the rest instances, we will calculate the average percentage of how much our algorithm can improve from the best initial solution in same case.

Suppose we have N instances in one case. Let C be the number of instances that we achieve same same or higher objective values. Then there are (N–C) instances that we get lower objective value. Let IP be the improved percentage of one such instance, let AIP be the average improved percentage of all such instances in the same case.

$$IP = 100\% * (ABC - INIT) / (GA - INIT)$$

$$AIP = (\sum_{i=1}^{(N-C)} IPi) / (N - C)$$

Obviously AIP will be a double value between 0 and 1. In following tables, the number followed by the double value indicates C, the number of instances has equal or higher objective values than the results in the referred paper. We will

run experiments multiple times and show average values. Some experimental results under our algorithms are available at https://github.com/gillmylady/Master_Project/tree/master/exp_data.

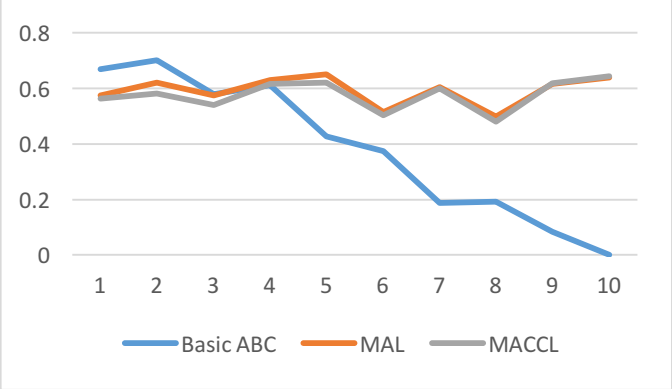
A. Comparison of basic ABC, MAL and MACCL

MAL is basic ABC plus local search, and MACCL is basic ABC plus combination of constructive method and local search. Basic ABC algorithm without local search methods doesn't sound like a good idea at the beginning, since we thought some local search methods should help improve the solutions. However, the experimental data as below shows unexpected results.

TABLE 2: comparison of ABC, MAL and MACCL

Case #	Basic ABC	MAL	MACCL
1	0.669(9.7)	0.574(7.7)	0.563(5)
2	0.701(5)	0.621(2.3)	0.581(0.7)
3	0.579(0.3)	0.575(2.7)	0.540
4	0.613(0.7)	0.631(1)	0.616
5	0.427	0.650	0.620
6	0.375	0.515	0.503
7	0.187	0.605	0.601
8	0.192	0.499	0.481
9	0.084	0.616	0.619
10	N/A	0.640	0.644

Fig. 1. Comparison of basic ABC, MAL, and MACCL



From the data we can see that MAL and MACCL have similar performance while basic ABC without local search surprised us and did especially well in small instances, case 1 and case 2. Starting from case 5, MAL and MACCL work better among all case numbers. We can also see the larger the case (size of instance), the worse basic ABC works, while MAL and MACCL can still produce results with around 60% improvement from the best initial solution.

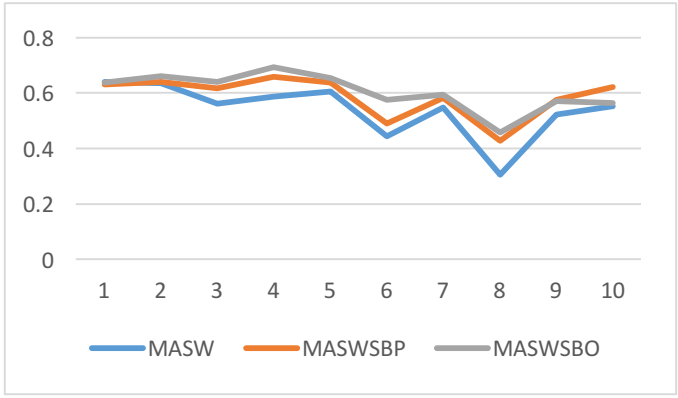
B. Comparison of MASW, MASWSBP, MASWSBO.

MASW is mutant of MAL plus heuristic SWT. MASWSBP is the mutant of MASW with additional heuristic SBT using greedy priority / process time. MASWSBO is the mutant of MASW with additional heuristic SBT using the objective function.

TABLE 3: comparison of MASW, MASWSBP and MASWSBO

Case #	MASW	MASWSBP	MASWSBO
1	0.641(12)	0.630(9)	0.637(10)
2	0.635(5)	0.639(3)	0.661(7)
3	0.561(1)	0.618(1)	0.641(3)
4	0.587	0.659(1)	0.694(2)
5	0.606(1)	0.638(0.7)	0.654(1)
6	0.444	0.490	0.576
7	0.548	0.582	0.594
8	0.305	0.429	0.458
9	0.522	0.576	0.571
10	0.553	0.622	0.564

Fig. 2. Comparison of MASW, MASWSBP, and MASWSBO



From above figure we can see algorithm MASWSBO shows slightly better performance than MASWSBP and much better than MASW. This makes sense because all these three mutant algorithms use the same objective function to select the worst task, and if we also use some greedy heuristic to select the best, it helps improvement for the neighbor selection process.

C. Comparison of MACCL, MASWSBO and MASWSBOCCL

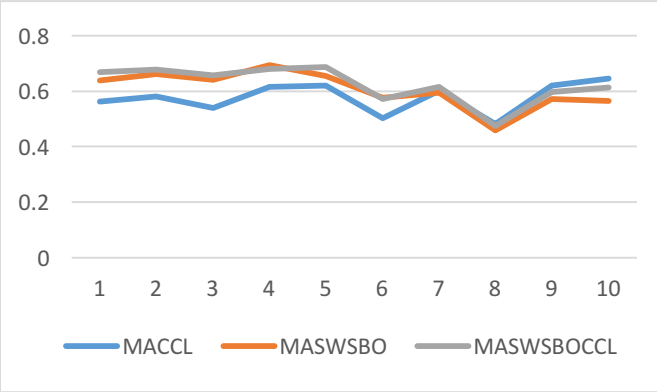
Mutant algorithm MACCL shows good results in the first comparison, mutant algorithm MASWSBO has comparatively better performance in the second comparison. We want to compare these two algorithms to the hybrid of them, the combination method of constructive heuristic and local search with greedy selecting by using the objective function. We hope we can discover which one works best overall in our experiments.

TABLE 4: comparison of MACCL, MASWSBO, and MASWSBOCCL

Case #	MACCL	MASWSBO	MASWSBOCCL

1	0.563(5)	0.637(10)	0.667(10)
2	0.581(0.7)	0.661(7)	0.677(4)
3	0.540	0.641(3)	0.656(1)
4	0.616	0.694(2)	0.679(1.5)
5	0.620	0.654(1)	0.686(1)
6	0.503	0.576	0.572(0.5)
7	0.601	0.594	0.615
8	0.481	0.458	0.474
9	0.619	0.571	0.597
10	0.644	0.564	0.612

Fig. 3. Comparison of MACCL, MASWSBO, and MASWSBOCCL



In the first comparison we see that MACCL doesn't work well in small instances. From above table, if we use SWT and SBT heuristics, a large improvement in small instances is shown. For larger size of instances, MASWSBOCCL works slightly better than MASWSBO and slightly worse than MACCL. Overall, we would suggest using MASWSBOCCL that contains all proposed heuristics for the FTSP.

D. Experiments on the stopping criterion

As we introduced before, to compare our experiment data with the results from referred paper, we were using the same stopping criterion which is the specific running time for each instance, as shown in table 1. During our experiments, we find that sometimes we can stop the criterion earlier than the specific time; in other words, we can spend less time to achieve similar results that we present above.

In our implemention, if we detect one solution that is not improved after a hard-coded count (say it HC), this solution is abandoned. We are going to see how good the abandoned solution is. If we find that after some consecutive count (say it CC), the abandoned solution is not improved, we will stop the loop. Let LC be the limit count when we stop the loop. We make the following mutant algorithm of MAL (name it MALS) for this approach.

Algorithm 3: MALS

1. Generate initial solutions for each worker bee
2. Let BAV be the best abandoned value, set BAV = 0
   Let CC be the consecutive count, set CC = 0
3. While stopping criterion is not satisfied do
   a-e) steps a-e same as Algorithm 1
   f) Check if any bee reaches HC. Let BOV be the best objective value of bee that reaches HC. If BOV > BAV, set CC = 0, set BAV = BOV; otherwise CC = CC + 1. Abandon this solution.
   g) If CC > LC, break this while loop
   h) Try exchange tasks among technicians
   i) Try exchange whole technicians' tasks among solutions
   j) try shrink scheduled tasks
   k) Store the best solution so far
4. End while

We run 40 instances in case 5 and try different LC values as shown in the following table.

TABLE 5: MALS experiments

Algorithm, LC	AIP
MAL	0.650
MALS, LC = 3	0.489
MALS, LC = 10	0.509
MALS, LC = 50	0.589
MALS, LC = 200	0.647

For each instance in case 5, previously we were using as long as 75 seconds, while the average time of running one instance using MALS where LC = 200 is only 38.5 seconds. This experiment shows that for some instances we don't need as much time as the referred paper; therefore, our algorithm can produce results more efficiently. More experiments are required to determine the optimal parameter of LC.

E. Incorrect results in the referred paper

As we run the experiments, we also find some incorrect results from referred paper. We consider them incorrect, because for some instances the best objective value in the referred paper is even smaller than that of the best initial solution we generated. And we did conflict check in our implementation so that the best initial solutions were correct solutions. We ran half of the whole instances in case 1 and found incorrect results as shown in the following table.

TABLE 6: incorrect instances in case 1

Instance ID	$INIT_i$	ABC_i	GA_i
R_1_1	40	44	22
C_1_3	49	55	27
C_1_13	65	76	36
C_1_19	73	79	39

RC_1_13	61	71	35
RAD_1_7	58	60	31
RAD_1_17	72	89	44
RAD_1_19	58	66	22

Out of 400 instances we ran, the total number of incorrect instances we found is 15.

VII. CONCLUSIONS

We've applied the basic ABC optimization and some mutants of it by using some heuristics to the instances of FTSP. We also did some experiments to compare the performances of proposed mutant algorithms.

Though our algorithm doesn't work as well as the referred paper [1], in most cases no matter how large the instance is, we were able to achieve around 60% improvement from the best initial solution.

Among all mutants of basic ABC algorithms, MASWSBOCCL, the mutant with all presented heuristics works best and performs consistently from small instances to large instances. This makes sense since we've seen how powerful the combination of constructive and local search is and we know we should drop the worst task and show the best task to neighbors during the process, just like how a real technician wants to schedule more tasks.

We also find that we can stop the running loop earlier than the specific time. In future work, we will do more experiments to find the best parameter for this approach. In addition, we will try some more heuristics and try to optimize the process of our algorithm to seek better solutions when running the same data set.

ACKNOWLEDGEMENTS

The author would like to thank his master project advisor Dr. Omar EI Ariss who provided invaluable advice and assistance.

REFERENCES

[1] Damm R B, Resende M G C, Ronconi D P. A biased random key genetic algorithm for the field technician scheduling problem[J]. Computers & Operations Research, 2016, 75: 49-63.

[2] Karaboga D, Basturk B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems[C]//International Fuzzy Systems Association World Congress. Springer Berlin Heidelberg, 2007: 789-798.

[3] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm[J]. Journal of global optimization, 2007, 39(3): 459-471.

[4] Karaboga D, Gorkemli B. A combinatorial artificial bee colony algorithm for traveling salesman problem[C]//Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on. IEEE, 2011: 50-53.

[5] Karaboga D, Akay B, Ozturk C. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks[C]//International Conference on Modeling Decisions for Artificial Intelligence. Springer Berlin Heidelberg, 2007: 318-329.

[6] Gao W, Liu S. Improved artificial bee colony algorithm for global optimization[J]. Information Processing Letters, 2011, 111(17): 871-882.

[7] Narasimhan H. Parallel Artificial Bee Colony (PABC) Algorithm[C]//NaBIC. 2009: 306-311.

[8] Xu J, Chiu SY. Effective heuristic procedures for a field technician scheduling problem. J Heuristics 2001;7(5):495–509.

[9] Solomon M M. Algorithms for the vehicle routing and scheduling problems with time window constraints[J]. Operations research, 1987, 35(2): 254-265.

[10] Desrochers M, Desrosiers J, Solomon M. A new optimization algorithm for the vehicle routing problem with time windows[J]. Operations research, 1992, 40(2): 342-354.

[11] Gambardella L M, Taillard É, Agazzi G. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows[J]. 1999.

[12] Tsang E, Voudouris C. Fast local search and guided local search and their application to British telecom's workforce scheduling problem. Oper Res Lett 1997; 20(3): 119–27.

[13] Evangeline R C. A modified bee colony optimization algorithm for nurse rostering problem[J]. Int J Innov Res Adv Eng (IJIRAE), 2014, 1(2): 31-35