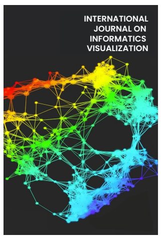
INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv



A Bee Colony Algorithm based Solver for Flow Shop Scheduling Problem

Yosua Halim[a], Cecilia E. Nugraheni[a,*]

[a] *Department. of Informatics, Parahyangan Catholic University, Jl. Ciumbueuit 94, Bandung,40141, Indonesia*
Corresponding author: [*]cheni@unpar.ac.id

***Abstract*—Flow Shop Scheduling (FSS) is scheduled to involve n jobs and m machines in the same process sequence, where each machine processes precisely one job in a certain period. In FSS, when a machine is doing work, other machines cannot do the same job simultaneously. The solution to this problem is the job sequence with minimal total processing time. Many algorithms can be used to determine the order in which the job is performed. In this paper, the algorithm used to solve the flow shop scheduling problem is the bee colony algorithm. The bee colony algorithm is an algorithm that applies the metaheuristic method and performs optimization according to the workings of the bee colony. To measure the performance of this algorithm, we conducted some experiments by using Taillard's Benchmark as problem instances. Based on experiments that have been carried out by changing the existing parameter values, the size of the bee population, the number of iterations, and the limit number of bees can affect the candidate solutions obtained. The limit is a control parameter for a bee when looking for new food sources. The more the number of bees, the more iterations, and the limit used, the better the final time of the sequence of work. The bee colony algorithm can reach the upper limit of the Taillard case in some cases in the number of machines 5 and 20 jobs. The more the number of machines and jobs to optimize, the worse the total processing time.**

***Keywords*— Scheduling; flow shop scheduling; metaheuristics; bee colony algorithm.**

Manuscript received 26 Nov. 2020; revised 29 Jan. 2021; accepted 8 May 2021. Date of publication 30 Jun. 2021.

I. INTRODUCTION

Scheduling is the arrangement of several activities, where these activities have several operations, and each of them is arranged in such a way as to become a schedule. One of the main goals of scheduling is to minimize the total time to complete the entire activity. Scheduling problems are found in the real world, for example, course scheduling, lecturer scheduling, and many more. The scheduling problem is also essential in the industrial world.

In the industrial world, there is a term known as shop scheduling. Shop scheduling is a scheduling problem in which there are several jobs, each of which has several processes or operations carried out by a machine. One type of workshop scheduling is flow shop scheduling (FSS). FSS involves n jobs and m machines, where each job will be processed by all the machines with the same sequence.

The textile industry, an industry that processes raw materials such as cotton into cloth, is included in the manufacturing industry. This industry uses machines in its production process. The production process, which uses a lot of this machine, needs to be arranged in such a way so that it can run optimally. One type of optimality commonly used is "makespan," which is the time needed to process the entire job. The production process of the textile industry usually belongs to FSS.

The production process of the textile industry usually belongs to FSS. Scheduling of jobs in FSS is an NP-hard problem that is generally solved using heuristic and metaheuristic algorithms [1]–[3]. These heuristics include dispatching rules such as FCFS (First Come First Serve) and SPT (Shortest Processing Time), NEH algorithm, Gupta algorithm, Palmer algorithm, and other algorithms [4]–[6]. Another heuristics group is metaheuristics, such as genetic algorithms, simulated annealing, and particle swarm optimization [1]–[3], [8]–[13]. The difference between heuristics and metaheuristics is in the way the machine sequences are generated [14].

In this work, we developed a solver program for solving FSS. This solver is based on a metaheuristic, namely the bee colony algorithm. Karaboga developed the bee colony algorithm in 2005 to solve numerical optimization problems. Nowadays, the bee colony algorithm has become more popular and applied to solve problems in the industry [15]–[21]. The bee colony algorithm is a search algorithm inspired by honeybees' behavior searching for food sources. These algorithm results are locations that have a good number and yield of food for the bee colony.

We are interested in developing a computer program, a solver, which can be used to solve FSS by applying the bee colony algorithm. The research questions that arise are how to model FSS and how the performance of the solver. Thus, this work aims to develop a bee colony algorithm-based solver for FSSP and measure its performance. The rest of the paper is organized as follows. Section II presents the materials and methods, including Flow Shop Scheduling and the Bee Colony Algorithm. Section III discusses the development of the solver and the performance measurement as well. The conclusion is given in Section IV.

II. Material and Method

This section presents the Flow Shop Scheduling Problem and describes the algorithm used to solve the problem, namely the Bee Colony Algorithm.

A. *Flow Shop Scheduling*

The flow shop scheduling problem is the problem of arranging the sequence of a set of jobs with a certain number of processes on several series of machines. The work will be done on existing machines in the selected order. Each process will be carried out systematically and sequentially. In the flow shop scheduling, the order of the work will be arranged so that the total time of all work can be minimized.

Here are some terms used in scheduling: [2]

- *Processing time*: the time needed for a machine to do an operation.
- *Delay time*: the time lag from a machine complete doing a job to starting a new job.
- *Total time*: the amount of time it takes for a machine to complete all operations in one job.
- *"makespan"*: the amount of time it takes for all machines to complete all jobs.

The input data required in running a flow shop scheduling are *number of work*, *number of machines*, and *detailed processing time* for each process for each job. Every job has a different processing time on each machine. Variation different processing sequences usually result in different "makespan."

Let us use a small FSS as an illustration. Assume there are three jobs (J1, J2, J3) and five machines (M1, M2, M3, M4, M5) for processing those jobs. Table I shows all the time needed for each machine for processing each job.

TABLE I
Processing Time

Jobs	**Machines**				
	M1	**M2**	**M3**	**M4**	**M5**
J1	6	5	3	9	5
J2	8	1	8	5	6
J3	2	1	3	8	6

This is an FSSP, aiming to find the ordering with the shortest total processing time or "makespan". If we use the First Come First Serve principle, then the ordering is J1-J2-J3, and the total time needed to complete (or "makespan") is 42-time units. Fig.1 depicts the Gantt Chart for this solution.

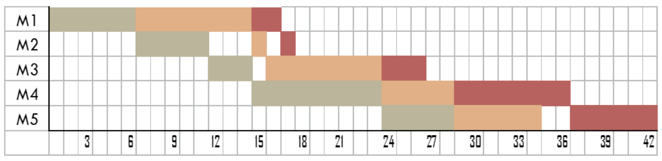


Fig. 1 Gantt Chart for First Come First Serve principle.

It can be seen on the Gantt Chart in Fig. 1 that after M1 finishes working on J1, M2 immediately works on J1. When M2 finishes working on J1, M2 does not immediately work on J2; this is because M1 is still working on J2, causing a delay on M2. M2 has to wait for M1 to finish working on J2 before M2 can start working on J2.

Let take another job order. If we use the Shortest Processing Time principle, then the job ordering becomes J3-J2-J1, and the "makespan" is 38-time units. Fig.2 depicts the Gantt Chart for this solution.

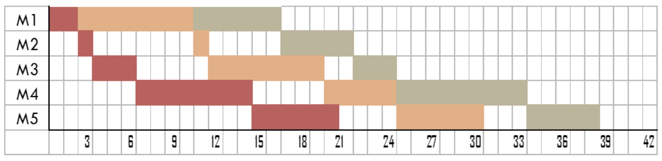


Fig. 2 Gantt Chart for Shortest Processing Time principle.

B. *Bee Colony Algorithm*

1) *Analogy*: According to Karaboga [5], the bee colony algorithm was created based on bees' behavior while foraging. Bee communicates with one another through the dance they make. The bees that collect food share information about the flowers' direction and distance and the amount of nectar in flower with their hive partners by performing this dance. Bee colonies can also quickly adjust their search patterns. This process can be seen as an optimization process.

There are three essential components of foraging behavior. The first component is a food source. A food source's value depends on many factors, such as proximity hive, food concentration, and how to extract food efficiently. The second component is employed bee. Employed bees share information on food sources and the fertility of these food sources with other worker bees. The last component is the unemployed foragers. There are two types of unemployed bees, namely, onlooker bees and scout bees. Onlooker bees wait in the hive and select a food source after receiving information from the worker bees. Onlooker bee decides to choose a food source. Food sources that have good fertility have a greater chance of being chosen by onlooker bees. The bee scout's responsibility is to find new food sources randomly.

In this model, a bee colony has three groups of bees, as listed above. Half of the bee colonies are employed bees, and the other half are onlooker bees. For every food source, there is one employed bee. In other words, the number of employed bees is the same as the number of food sources around the hive. Employed bees whose food sources have run out will become scout bees.

2) Algorithm: The working principle of the bee colony algorithm is given in Fig 3. Each search cycle consists of three steps:

- The employed bee moves to the food source and calculates the amount of its nectar.
- Onlooker bee movement to select a food source.
- Assign the bee scouts to possible food sources.

In this model, a bee colony has three groups of bees, as listed above. Half of the bee colonies are employed bees, and the other half are onlooker bees. For every food source, there is one employed bee. In other words, the number of employed bees is the same as the number of food sources around the hive. Employed bees whose food sources have run out will become scout bees.

```
1.  Initial food source positions.
2.  Calculate the nectar amounts.
3.  Determined the new food positions for the employed bees.
4.  Calculate nectar amounts.
5.  If all onlookers are distributed, go to 9; otherwise, go to 6.
6.  Select a food source for the onlooker.
7.  Determine a neighbor food source position for the
    onlooker.
8.  Go to 4.
9.  Memorize the position of best food source.
10. Find the abandoned food source.
11. Produce new position for the exhausted food source.
12. If the termination criterion is satisfied, go to 13; otherwise,
    go to 3.
13. Final food position.
```

Fig. 3 The bee colony algorithm.

The position of the food source represents the solution to the problem to be optimized. The amount of nectar from a food source corresponds to the quality of the food source's solution. Bee's onlookers are placed on the food source using a probability-based selection process. As the number of nectar increases, the probability value where the onlooker bee prefers the food source will also increase. Each bee colony has a scout bee, which is an explorer of that colony. Scout bee does not have any guidance when it comes to finding food. They look for all kinds of food sources. As a result of this behavior, bee scouts are characterized by low search costs and the average low in the food source's quality. Now and then, a scout bee can accidentally find a good food source [21].

In the bee colony algorithm, scout bees can find candidate solutions quickly. In this task, one of the employed bees is selected and defined as the scout bee. This selection is controlled by a control parameter called limit. If a predetermined number of trials cannot improve the solution representing food sources, then the food source is abandoned by him, and the employed bee will become a scout bee. The number of attempts to remove food sources is equal to the limit value, which is the algorithm's control parameter. In an intense search process, the exploration and exploitation processes must be carried out together. In the bee colony algorithm, when onlooker bee and employed bee perform the search space's exploitation process, scout bee regulates the exploration process.

Employed bees are assigned to a source. The number of food sources equals the number of working bees. Bees have calculated a new solution by flying to food sources closest and maintain the best solution when it reaches the source. Onlooker bees wait in the hive and make the decision to choose a food source based on the information of the employed bees. The number of onlooker bees is the same as employed bees, and they allocate food sources based on their probability.

Scout bees are responsible for randomly looking for new food sources. Scout bees randomly search for new solutions when food sources do not improve after several iterations. The source found by scout bees will replace the existing solution if the nectar found is better than the previous one.

3) Fitness Value: The fitness value of each food source is calculated by the formula below [22]:

$$fit = \frac{1}{1+F_i} \tag{1}$$

where, F_i is the "makespan" value for solution i. This fitness value will be used to determine the probability value of a solution.

4) Employed Bee Stage: The second stage is when the employed bee exploits food sources. Before an employed bee sends information to the onlooker bee, an employed bee will look for new food sources around the food sources obtained from previous scout bees. This stage means generating a candidate for a new solution or solution. Operations that can be used to insert and swap operations. Later on, this new solution will be chosen again by onlooker bee to compare the fertility value of these food sources.

5) Insert and Swap: According to this algorithm, employee bees generate food sources around their current position. Operators commonly used to produce this solution are insert and swap processes. The insert operator generates a food source by removing a job from its original position and inserting it into another position. In contrast, the swap operator generates a food source by swapping two random jobs. An example of using the swap and insert operators can be seen in Fig. 4. There are two food sources represented as the order of the job carried out.

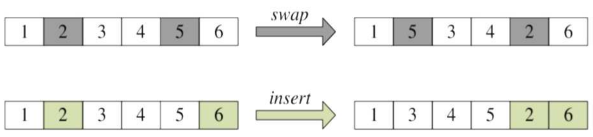


Fig. 4 The bee colony algorithm.

Six jobs are done sequentially. The first row of Fig. 4 shows the use of the swap operator. Job 2 is exchanged with job 5. When this operation is executed, the position of the other jobs does not change. In the second row of Fig. 2 is the use of the insert operator. In the insert operation, job 2 will be inserted between job 6 and job 5. After this operation is executed, the position of the other job also changes.

6) Determining Probability Value: The probability value will be calculated by taking one food source that the employed bee has worked on. The food source's fitness value will be divided by the total fitness value of all food sources. Onlooker bee chooses a food source depending on the probability value of food source p_i. Below is a formula to calculate the probability value for a food source [22].

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} F_n} \tag{2}$$

where fit_i is the value of the fitness solution i, which is evaluated by the employed bee, and SN is the number of food sources, which is the same as the number of solutions.

7) Onlooker Bee Stage: After a solution has been selected, the onlooker bee will go to the food source, and then its performance is compared to the old food source. If the new food source is the same or better than the old food source, then the food source is replaced with a new food source; otherwise, it is not replaced.

8) Scout Bee Stage: At this stage, if the solution cannot be further increased through a predetermined number of limits, the solution will be abandoned, and the employed bee will become a scout bee. Scout bees will search for food sources randomly in the same way as initial population initialization. After bee's scouts work, bee's scouts are back to being employed bee. The probability value will be calculated by taking one food source that the employed bee has worked on. The food source's fitness value will be divided by the total fitness value.

III. Results and Discussion

This section describes the modeling of the solver and the experiments conducted for measuring the solver's performance.

A. FSS Modelling

The bee colony algorithm will receive input data from a flow shop case, and then it will look for the optimal order of jobs or the solution from that case. The bee colony algorithm will find a sequence of processes that produce a minimum "makespan."

This algorithm represents a selection of existing solutions as the best food source for bees with the help of the fitness value as a guide to the fertility value. With these guidelines, the bees are assumed to be able to choose the most optimal path. The optimization process using the bee colony algorithm needs to pay attention to several things. The bee colony algorithm needs to know how to represent a solution as a food source, get a neighbor's solution, get the chance to choose a solution, and many other things. This procedure can influence the optimization results of the bee colony algorithm.

During the optimization process, the bee colony algorithm performs a search process repeatedly. In each search process, the bee colony algorithm distributes some bees that will work on the flow shop case according to the solution chosen by each bee. The best solution that a bee has chosen will be saved and compared with the previous search process's best solution. The algorithm is given in Fig. 5.

Fig 5 presents a bee colony algorithm that has been adapted for FSSP problems. We add this input parameter setting for giving the algorithm all the parameters needed, namely number of machines, number of jobs, processing times, number of bees, maximum iteration, limit. The number of bees entered can be modeled as a candidate solution because each bee will carry one food source in which the food source is modeled as a job sequence. The operation time of each job is needed to find the "makespan" of a given sequence of jobs. In this scheduling problem, we want to look for the order of the best work and its "makespan". Each work sequence, which can be made from the data processing time, is a candidate solution.

```
1. Input parameters setting.
    a. Number of machines, number of jobs, processing
       times
    b. Number of bees, maximum iteration, limit
2. Initial solution.
    a. Initial solution generation.
    b. Schedule generation
3. Employee bee stage.
    a. Neighbor solution finding
    b. Schedule generation
4. Onlooker bee stage.
    a. Probability calculation
    b. Neighbor solution finding.
    c. Schedule generation and "makespan" calculation.
5. Scout bee stage.
    a. New solution finding
    b. Schedule generation
6. If new solution has better "makespan" then go to 7;
   otherwise go to 8.
7. Replace current solution with new solution.
8. If maximum iteration is reached, then go to 11; otherwise
   go to 9.
9. Iteration incrementation.
10. Go to 3.
11. Finish.
```

Fig. 5 The bee colony algorithm for FSS.

The bee colony algorithm will help determine the optimal job-taking order. Therefore, this algorithm will represent the preferred order of taking that is available as a food source. The bee colony algorithm does not have a guide when selecting initial food sources; therefore, the bee colony algorithm randomly selects the initial food source.

The food source is a collection of food sources that the bees will select in a colony. As an example, it can be seen in Fig. 6, suppose there are three jobs and three bees, then a random number search process will be carried out for each bee, and a sequence of workings and their "makespan" will be obtained, for example (2,1,3), (3,1, 2), (1,2,3). The order of work obtained can be represented as an initial source of food or the initial position, knowing that there is a food source in the job sequence. Each sequence of this process will have a different "makespan" depending on the order in which it is done. It should also be noted that this initial search for food sources can be in the same order as other bees.

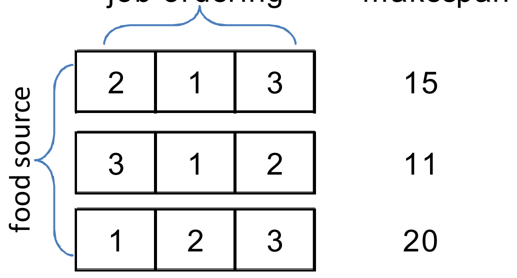


Fig. 6 Food source and job ordering mapping.

The bee colony algorithm generates new food sources around their current position. In this way, the operations that can be used are the swap and insert operations as described in the previous chapter. In this process, each worker bee already has an initial position of the food source, carried out by the scout bees. The worker bee will search for new food sources around the food source position it knows, meaning that the worker bee will get a new sequence of jobs obtained from the swap or insert operation.

In Fig. 7, bee one will perform a neighbor solution search and perform a swap operation on job 2 and job 3. In this way, the bee gets a new food source position or a new work sequence. As already said previously, processing one bee may have the same order of processing as another. This new food source will be compared to its "makespan" with the previous food source. If the "makespan" of the new food source is smaller than the previous one, then the food source will become a new position for the bee.

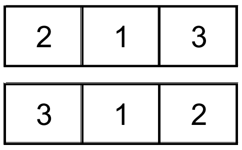


Fig. 7 The result of neighbor solution finding.

After the neighbor solution search process is completed, the odds of each solution will be calculated. Keep in mind that the bee colony algorithm will always provide a chance for a solution to be chosen. Even though the solution can be considered a less than optimal solution, it can still be selected. The way to get the chance for a solution is obtained by dividing the solution's fitness value by the total fitness value obtained for all food sources described in Section II.

During the optimization process, the bee colony algorithm will continuously perform a search phase. At each search phase, a solution will be formed in the form of a food source based on its fitness value. After a search phase is complete, changes to the fitness value will be made based on each bee's food source. The optimization process will be terminated if a condition is met. Various parameters can determine the time to stop a search process. The search process can be considered complete if it has passed a predetermined number of iterations. The more iterations that are done will produce more optimal results, but it will take more processing time.

B. *Experimental Results*

This experimental test aims to look for factors that can affect the results of the solver. This test is done by conducting experiments using solver with different input parameters. First experiment is to know the effect of the number of bees on the "makespan". We run two experiments, first experiment with 5 bees and the second experiments with 50 bees. Second experiment is to know the effect of the maximum iteration on the "makespan". We run two experiments, first experiment with 250 iteration and the second experiments with 2500 iteration. Third experiment is to know the effect of the limit number on the "makespan". We run two experiments, first experiment with 10 and the second experiments with 100.

The parameters tested in this experimental test were the number of bees, the number of iterations carried out, and the limits on bees. If there are differences in the experimental results, differences in input parameters will undoubtedly affect the results. The results obtained will be compared with the upper limit of Eric Taillard's website, and the order of work that is not changed is the order of the work, which is named with a simple "makespan" [7], [20]. Experimental testing was carried out in each case. Other parameters that are not tested will use the Karaboga program's parameters: The number of bees is 10, the number of iterations is 2500, and the limit is 100.

Experimental testing to be carried out will use the Taillard case in some jobs, namely: 20 jobs 20 machines, 50 jobs on 20 machines, 100 jobs on 20 machines, 200 jobs on 20 machines, and 500 jobs on 20 machines.

1) Fitness Value: Food sources are candidate solutions to be selected. In the bee colony algorithm, each bee will have precisely one food source, which means the bee itself is a candidate solution. The number of bees used in each experiment is different. It aims to see a (relatively) significant difference in the results of the software. In this experiment, the number of bees will be worth 10% of the number of bees in the 2nd experiment. The number of iterations is 2500, which is obtained from the Karaboga parameter and the limit is 100. The result is given in Fig. 8.

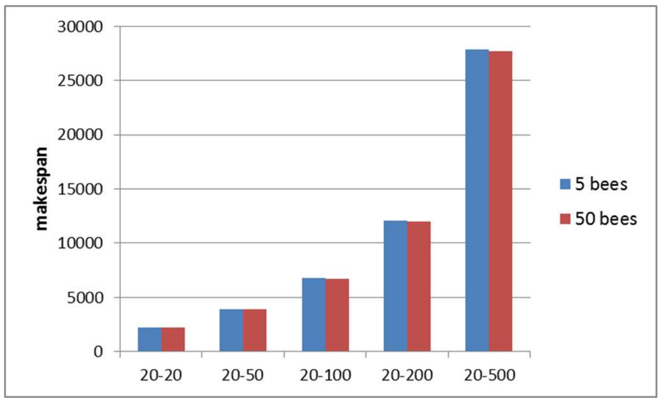


Fig. 8 The correlation of the number of bees with "makespan"..

2) Maximum iteration: The number of iterations used in experiments is varied. It aims to see differences in software results. In this experiment, the number of iterations will be 10% of the number of iterations of the 2nd experiment. The number of bees is 10, which is obtained from the Karaboga parameter and the limit is 100. The result is given in Fig. 9.

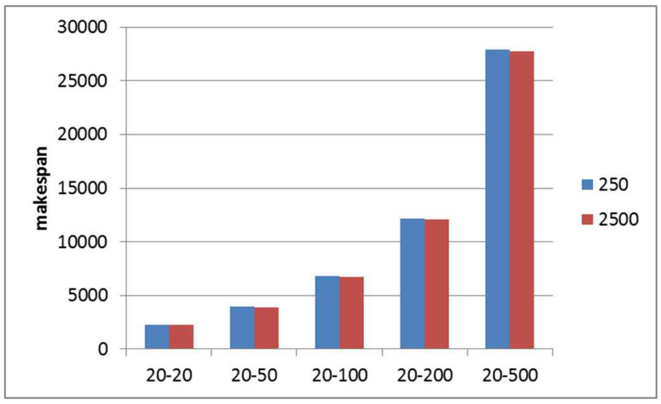


Fig. 9 The correlation of maximum iteration with "makespan".

3) Limit: The number of limits used in the experiment is different. It aims to see differences in software results. In this experiment, the limit amount will be 10% of the many limits of the second experiment. The number of bees is 10, which is obtained from the Karaboga parameter, and the number of iterations is 2500.

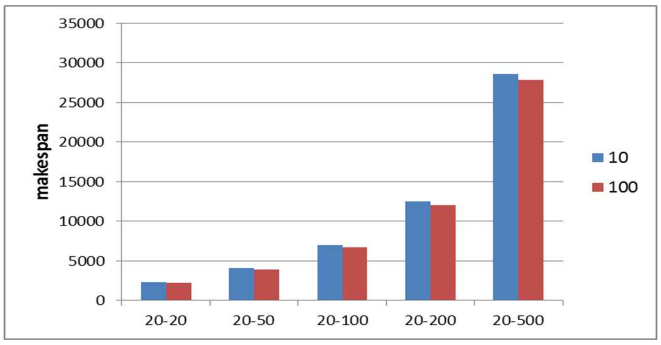


Fig. 10 The correlation of limit with "makespan".

From the results of three experiments, namely the number of bees, the number of iterations, and the number of limits, we can see that the greater the parameters used, the better the results. However, the more the number of bees and iterations, the time it takes to get a solution is also getting longer. The number of known limits does not affect the time to find a solution.

IV. Conclusion

We have successfully built a solver program to solve flow shop scheduling problems using the Bee Colony algorithm. Functional testing has been carried out for this solver to guarantee its functionality. Moreover, we conducted some experiments to measure the performance of the Bee Colony algorithm by using this program. From the experimental results, the parameter has significant effects on the solver's performance. The number of bees and the maximum iteration are directly proportional to the "makespan" produced.

In general, the "makespan" produced by the solver is no better than benchmarks. However, in the case of 20 job 5 machines, the resulting "makespan" is close to the benchmark's upper limit. We are now working on combining two metaheuristics, namely genetics algorithm and firefly algorithm, for solving flow shop scheduling problems. We are also considering applying this approach for solving another variant of FSS, namely Flexible FSS [22].

Acknowledgment

The Indonesian Ministry of Research supported this work, Technology and Higher Education (RistekDikti), under the research scheme of Higher Education Excellence Applied Research 2019-2021 with contract number: III/LPPM/2020-04/104-PE-S.

References

[1] Mumtaz, J., Zailin, G., Mirza, J., Rauf, M., Sarfraz, S., & Shehab, E. (2018). "makespan" minimization for flow shop scheduling problems using modified operators in genetic algorithm. In K. Case, & P. Thorvald (Eds.), Advances in Manufacturing Technology XXXII - Proceedings of the 16th International Conference on Manufacturing Research, ICMR 2018, incorporating the 33rd National Conference on Manufacturing Research (Vol. 8, pp. 435-440). IOS Press BV. https://doi.org/10.3233/978-1-61499-902-7-435

[2] Laxmi A. Bewoor, V. Chandra Prakash, Sagar U. Sapkal. (2017). Comparative Analysis of Metaheuristic Approaches for "makespan" Minimization for No Wait Flow Shop Scheduling Problem. International Journal of Electrical and Computer Engineering (IJECE) Vol.7, No.1, February2017, pp. 417~423. ISSN: 2088-8708, DOI: 10.11591/ijece.v7i1.pp417-423.

[3] Bultmann, M., Knust, S., & Waldherr, S. (2018). Flow shop scheduling with flexible processing times. OR Spectrum, 40(3), 809-829. https://doi.org/10.1007/s00291-018-0520-8Modrak, V. dan Pandian, R. S. (2010) Flow shop scheduling algorithm to minimize completion time for -jobs -machines problem. *Tehni ki vjesnik*, **17,3**, 273–278.

[4] Nugraheni, C. E., Abednego, L. (2016) A comparison of heuristics for scheduling problems in textile industry. *Jurnal Teknologi*, **78:6-6**, 99–104.

[5] Kurniawati, D.A., Nugroho, Y. I. (2017). Computational Study of N-Job M-Machine Flow Shop Scheduling Problems: SPT, EDD, NEH, NEH-EDD, and Modified-NEH Algorithms. Journal of Advanced Manufacturing SystemsVol. 16, No. 04, pp. 375-384 (2017).

[6] Nugraheni, C. E., Abednego, L. (2021). A Combination of Palmer Algorithm and Gupta Algorithm for Scheduling Problem in Apparel Industry. International Journal of Fuzzy Logic Systems Vol. 11, No. 1, January 2021.

[7] Sauvey, C., Sauer, N. (2020). Two NEH Heuristic Improvements for Flowshop Scheduling Problem with "makespan" Criterion. Algorithms 2020, 13, 112; doi:10.3390/a13050112.

[8] Dr. S. Sridhar, S. Sabareesanand Dr. R. Kannan, Particle Swarm Optimization Approach for Flow Shop Scheduling Problem – a Case Study, International Journal of Mechanical Engineering and Technology, 9(11), 2018, pp. 72–80. http://www.iaeme.com/IJMET/issues.asp?JType=IJMET&VType=9&IType=11.

[9] Arık, O. A. (2020). Population-based Tabu search with evolutionary strategies for permutation flow shop scheduling problems under effects of position-dependent learning and linear deterioration. Soft computing. https://doi.org/10.1007/s00500-020-05234-7.

[10] Deb S.,Tian Z.,Fong S.,Tang R.,Wong R.,&Dey N..(2018).Solving permutation flow-shop scheduling problem by rhinoceros search algorithm.Soft Computing,22(18),6025-6034.

[11] Mohammadzadeh, H., Sahebjamnia, N., Fathollahi-Fard, A., Hajiaghaei-Keshteli, M. (2018). New Approaches in Metaheuristics to Solve the Truck Scheduling Problem in a Cross-docking Center. International Journal of Engineering, 31(8), 1258-1266.

[12] Ilyass Mzili, Mohammed Essaid Riffi, Fatiha Benzakri (2020). Discrete penguins search optimization algorithm to solve flow shop scheduling problem. International Journal of Electrical and Computer Engineering (IJECE) Vol.10, No.4, August 2020, pp. 4426~4435ISSN: 2088-8708, DOI: 10.11591/ijece.v10i4.pp4426-4435.

[13] Sanjeev Kumar, R., Padmanaban, K., & Rajkumar, M. (2018). Minimizing "makespan" and total flow time in permutation flow shop scheduling problems using modified gravitational emulation local search algorithm. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 232(3), 534–545. https://doi.org/10.1177/0954405416645775.

[14] Nugraheni, C. E., Abednego, L. (2016). On the Development of Hyper Heuristics Based Framework for Scheduling Problems in Textile Industry. International Journal of Modeling and Optimization, Vol. 6, No. 5, October 2016.

[15] Weixing Su, Hanning Chen, Fang Liu, Na Lin, Shikai Jing, Xiaodan Liang, Wei Liu (2017). A novel comprehensive learning artificial bee colony optimizer for dynamic optimization biological problems, Saudi Journal of Biological Sciences, Volume 24, Issue 3, 2017, Pages 695-702, ISSN 1319-562X.

[16] Hagh, M.T., Orandi, O.B. (2018). A Modified Artificial Bee Colony Algorithm Application for Economic Environmental Dispatch. IOP Conf. Series: Materials Science and Engineering 339 (2018) 012008 doi:10.1088/1757-899X/339/1/012008.

[17] Chong, Chin & Sivakumar, Appa Iyer & Low, Malcolm Y. H. & Gay, Kheng. (2006). A Bee Colony Optimization Algorithm to Job Shop Scheduling. Simulation Conference, WSC. 06. 1954-1961. 10.1145/1218112.1218469.

[18] X. Li and S. Ma, (2017). "Multiobjective Discrete Artificial Bee Colony Algorithm for Multiobjective Permutation Flow Shop Scheduling Problem with Sequence Dependent Setup Times," in IEEE

Transactions on Engineering Management, vol. 64, no. 2, pp. 149-165, May 2017, doi: 10.1109/TEM.2016.2645790.

[19] Oğuzhan Ahmet Arık. (2021). Artificial bee colony algorithm including some components of iterated greedy algorithm for permutation flow shop scheduling problems. Neural Computing and Applications. Issue 8/2021

[20] Taillard, E. (1993) Benchmark for basic scheduling problems. *European Journal of Operational Research*, 64, 278–285.

[21] Karaboga, D. (2005) An idea based on honeybee swarm for numerical optimization. Technical Report TR06. Erciyes University, Kayseri, Turkiye.

[22] Liao, T., Aydın, D, Stützle, T. (2013) Artificial bee colonies for continuous optimization: Experimental analysis and improvements. Springer, 13, 1935–3820.

[23] Nugraheni, C.E., Abednego, L., Widyarini, M. (2020). A Tabu Search Based Hyper-heuristic for Flexible Flowshop Scheduling Problems. International Journal of Advanced Science and Technology, 29(2), 301 - 310.