A Hybrid Firefly Algorithm with Fuzzy Movement for Solving the Flexible Job Shop Scheduling Problem

Ajchara Phu-ang[1]

ABSTRACT: This paper proposes a hybrid algorithm that applies the firefly algorithm with the new concepts labeled as the self-adaptive roulette wheel selection and the fuzzy movement method for solving the flexible job shop scheduling problem. The steps of the proposed algorithm are similar to the original firefly algorithm, which is based on the concepts of flashing behavior to attract other fireflies. In order to improve the efficiency of the firefly algorithm for the flexible job shop scheduling problem, the proposed algorithm introduces two new ideas. In the first idea, the incorporation of a self-adaptive roulette wheel selection method is embedded with a genetic algorithm to increase the diversity in the machine selection process. In the second idea, a fuzzy movement method is introduced to help enhance the work balancing ability between the high workload machines and the low workload machines. In this study, the benchmark data sets used to test the performance of the proposed algorithm were obtained from the flexible job shop scheduling problem library. The performance of all algorithms is measured in terms of the average percentage deviation from the lower bound. When considering the average over the number of experiments carried out for each data set, the proposed algorithm outperformed the comparison algorithms for the Hurink V and Brandimart data sets.

Keywords: Firefly Algorithm, Fuzzy, Genetic Algorithm, Scheduling Problem

DOI: 10.37936/ecti-cit.202152.239856

Article history:received February 20, 2020; revised May 22, 2020; accepted October 11, 2020; available online April 26, 2021

1. INTRODUCTION

Finding the optimal scheduling solution is an important key to success in the manufacturing industry. Nowadays, every business has a lot of competitors. Each business uses various strategies in order to reach new customers and keep their existing customers. The aforementioned factors depend mainly on customer satisfaction. Any business needs to quickly respond to customers. Hence, the manufacturer manages the schedule to able to deliver products and services to the customer immediately. The flexible job shop scheduling problem (FJSP) is one of the complex scheduling problems in the combinatorial optimization field which appears while a manufacturer is unable to find the balance between their machines and manufacturing activities. Some machines work very hard, but at the same time, some machines are not fully used. When job scheduling is inefficient, the delivery time to the customer is also delayed. Therefore, to achieve a balance between activity and machines, many industries apply optimization algorithms in their scheduling process. In recent years, many researchers have attempted to solve the FJSP by employing a swarm intelligence algorithm which is one class of the biologically inspired metaheuristic algorithms. Some intelligence techniques for solving the scheduling problem are reviewed in the following paragraphs.

Tung-Kuan Liu et al. [1] refined the genetic algorithm (GA) for solving a distributed FJSP (DFJSPs). The DFJSPs correspond to situations where production activities are carried out in manufacturing units located within a single building. In the proposed algorithm, they improved two steps of the original genetic algorithm by utilizing a simple encoding method and an evolutionary combination method. In the refined GA, the initial population generated integrates probability concepts into a real-parameter encoding

[1]The author is with Collage of Innovation, Thammasat University, Thailand., E-mail: ajchara.p@citu.tu.ac.th

method which can substantially reduce the length of chromosomes, thereby saving computation space. Moreover, two types of crossover and two types of mutation are used to enhance the effectiveness of the GA for solving DFJSPs. In the empirical results, two-stage validation is conducted by adopting an algorithm with a classical DFJSP and a real-world problem. The results indicate that their proposed algorithm could effectively overcome the conflicts caused by GA encoding algorithms.

Tianhua Jiang and Chao Zhang [2] employed a swarm intelligence algorithm, the grey wolf optimization (GWO), for solving job shop and FJSP. The grey wolf optimization is a novel swarm-based intelligence algorithm based the social behavior of natural grey wolves. In their paper, a two-point crossover operation is designed to exchange elements of parent individuals which maintain the algorithm working directly in a discrete domain. An adaptive mutation method is used to increase the population diversity. Moreover, they employed a variable neighborhood search to enhance the exploration efficacy. In this process, the optimal solution is identified by changing the neighborhood structures. To evaluate the effectiveness, the discrete GWO algorithm was compared with other published algorithms for two scheduling cases. The experimental results demonstrated that the discrete GWO algorithm has some potential and outperformed some existing published algorithms.

Yuan Yuan and Hua Xu [3] proposed new memetic algorithms for multi-objective FJSP, including the nondominated sorting genetic algorithm II (NSGA-II) adapted in the encoding and decoding process. Then, they applied a selection mechanism to screen the offspring population and transferred it to a novel local search. A novel local search algorithm was embedded into the adapted NSGA-II. Moreover, a hierarchical strategy was incorporated into a novel local search to give varying degrees of consideration to each objective. The novel local search was divided into 2 steps period Neighbor Generation in Local Search is the first step, and Acceptance Rules in Local Search is the second step. Then, the critical operations considered moves. In the experimental studies, they classified the experiment into three types as follows. First, the influence of two alternative acceptance rules was examined. Then, the effectiveness of key components was verified, including genetic search, local search, and the hierarchical. Finally, extensive comparisons were carried out with the state-of-the-art methods on well-known benchmark instances. The results indicated that the proposed MAs outperformed all the other state-of-the-art algorithms.

Hao-Chin Chang et al. [4] used a genetic algorithm for solving the FJSP. In this algorithm, they embedded the Taguchi Optimization Method in the evolution phase to facilitate obtaining high-quality solutions, reducing computational time and increasing convergence speed. In addition, for the mutation methods, random selection and neighborhood search were employed to enhance the search space of the solution. The examination of this paper was divided into two experiments. The first experiment was conducted to state the difference between the convergence speeds obtained using the standard GA and their algorithm. The second experiment was performed to measure the proposed algorithm with the Brandimarte benchmark data set. Moreover, various algorithms were compared. The results show that embedding the Taguchi method in the proposed algorithm can increase the opportunity to discover near-optimal or optimal solutions. Therefore, their algorithm successfully solved the FJSP and produced a feasible solution.

Thitipong Jamrus et al. [5] developed a hybrid approach which combined a particle swarm optimization algorithm with a Cauchy distribution and genetic operators. They solved a FJSP under uncertain processing time for semiconductor manufacturing, which has a complicated of combinatorial complexity. This paper used the particle swarm as a core. In the initial step, the population was created by a randomized population-based optimization method. Then the position of the solution was updated according to the Cauchy distribution for effective particle movement. Thereafter, the particles were exchanged by a genetic, crossover, and mutation operators. In the experiment, the performance of the proposed algorithm was evaluated by using six problems. They compared the proposed algorithm with the conventional approaches. The results showed that some waiting time is eliminated, short make-span was achieved, and resources were used efficiently.

Yi Lu et al. [6] solved an energy-conscious flexible job shop scheduling by using a new population-based intelligence algorithm named the discrete water wave optimization (DWWO) algorithm. At the beginning, the population was randomly created according to the encoding method. In their paper, a perturbation procedure was presented based on three mutation operations: Machine Assignment, Speed-level Selection, and Operation Permutation. The perturbation procedure was randomly performed to obtain a new wave. Because a random perturbation may result in a poor solution, the seeking memory pool (SMP) mechanism was adopted as neighbor search method for each current wave. When a new wave X was not improved after several propagation operations, a crossover operation was conducted to a new wave X. In the performance tests, they evaluated the performance of the proposed algorithm with the three comparison algorithms. The results showed that their algorithm has advantages in solving the energy conscious FJSP.

Lu Sun et al [7] proposed an effective hybrid cooperative co-evolution algorithm (hCEA) for solving fuzzy flexible job shop scheduling. The framework of this algorithm is based on the structure of the cooperative co-evolution algorithm. They hybrid the particle swarm optimization with the genetic algorithm into the cooperative co-evolution algorithm. For solution representation, they applied the random key-based representation and the multi-stage representation to ensure that the operation sequence string and the machine assignment string are feasible. For the genetic operators, precedence operation crossover (POX) and job-based order crossover (JOX) were applied to the operation sequence string to pass a good feature to the offspring. Moreover, swapping mutation and one-bit mutation were utilized for the operation sequence string and the machine assignment string respectively. Thereafter, they adopted both ranking selection and tournament selection to select survival of the fittest solution. For the PSO operator, the set-based random grouping strategy was used to enhance the performance of the CEA. The update component and a critical path based local search strategy help this algorithm find better solutions. This paper proposed the self-adaptive parameter mechanism for adjusting the parameters automatically instead of using fixed parameters. The results showed that the hCEA outperformed the comparison algorithms.

Rylan H. Caldeira and A. Gnanavelbabu [8] solved the FJSP by using an improved Jaya algorithm (JA). They intended to improve the algorithm to prevent getting trapped in the local optimum and needing specific tuning parameters to obtain an optimal solution. They utilized the JA with a Neighborhood-based local search technique to perform deep exploration and exploitation of the search space. In the initial step, the multiple initialization rules were applied to create an effective initial solution. Then, they employed the JA to move towards a better solution and away from a worse solution. Finally, they used the critical path concept for reducing the neighborhood size and improving the rate of convergence. The performance of the improved Jaya algorithm demonstrated that the IJA was effective for solving the FJSP.

Zhenwei Zhu and Xionghui Zhou [9] proposed the grey wolf optimizer for solving the multi-objective FJSP. The grey wolf optimizer algorithm is inspired from the social hierarchy and the hunting mechanism of grey wolves in nature. This algorithm started by presenting two different encoding schemes named discrete encoding and continuous encoding. Next, they generated the population by using a random method and divided the population into different sets. The binary tournament selection method was used to select the leading wolf. An operator of searching for prey, called the crossover operator, was applied to the job sequencing part, operation sequencing part, and machine assignment part. Moreover, they employed the swap mutation operator to exchange a pair of the different elements at various positions. The mutation contributed to prevent the premature convergence to the local optimum area. The elitist preservation was presented to select the population for the next iteration. The experiments demonstrated that this algorithm overwhelms other competing algorithms on the majority of test instances.

Alejandro Vital-Soto et al. [10] presented the Mathematical modeling and the hybridized bacterial foraging optimization algorithm (HBFOA) for solving the FJSP with sequencing flexibility. The steps of the bacterial foraging optimization algorithm are based on the food foraging behavior of E. coli bacteria. They enhanced local search capability by embedding the simulated annealing (SA) algorithm. At the first step, a new dispatching rule called the minimum number of operations is presented to create the initial swarm population. Then, the modified swarming process designed to explore new other population by guiding the bacterium to a most promising area was applied. Then, they used the reproduction technique to sort the bacteria based on their objective value. The comparative results show that the HBFOA outperformed all classical instances presented in this paper.

Jie Gao et al [11] developed a hybrid genetic algorithm (hGA) for the multi objective FJSP. In their algorithm, a variable neighborhood descent (VND) works under the framework of the GA. The VND consists of two local search procedures, local search of moving one operation and local search of moving two operations. The VND is present to up speed the convergence rate of the simple GA. In addition, the VND is employed to identify and break the existent critical paths one by one in order to get a new schedule with a smaller makespan. Moreover, in the crossover stage, they applied an order crossover method to generate the offspring solution. The results showed that, their study found 38 new better solutions.

Yuan Yuan and Hua Xu [12] presented the discrete differential evolution (HDE) algorithms for solving the FJSP. In this algorithm, they enhanced the local searching ability by incorporating a critical path local search algorithm in the DE framework to balance the exploration and exploitation. In their study, two neighborhood structures are presented, HDE-N1 and HDE-N2. The experimental results demonstrated that HDE- N2 obtained higher quality solutions than HDE- N1. A performance comparison demonstrated that their algorithm outperformed several recently proposed algorithms.

A.E. Eiben et al. [13] presents a new approach to selection in Genetic Algorithms (GAs) called the Self-adaptation of tournament size. This is used as a mechanism for regulating EA parameters. These parameters are updated during the search. The details of the result shown that their work can increase performance in terms of speed.

This paper aims to develop a hybrid meta-heuristic algorithm based on the concept of the discrete firefly algorithm for solving the complex scheduling problem named FJSP. Within our framework, a genetic algorithm (GA) is as the initial process to keep a good quality of the initial population. The new ideas are called the self-adaptive roulette wheel and the fuzzy movement mechanism. They are used to enhance machine selection diversity and to create activity balance, respectively.

The remainder of this paper is organized as follows. Section 2 presents the background theories. Section 3 presents the steps of the proposed algorithm. The experimental results are given in Section 4. Finally, Section 5 is the conclusion.

2. RELATED WORK

The framework of the proposed algorithm involves several theories. Details of each theory are explained next.

2.1 The flexible job shop scheduling problem

The flexible job shop scheduling problem (FJSP) involves a problem where it is hard to find a polynomial algorithm to solve it in polynomial time. The problem is well known as being non-deterministic polynomial-time hard (NP-Hard). Details of the FJSP are described next.

The FJSP is one of the problems in the combinatorial optimization field. The objective of this problem is to arrange a sequence of operations according to a precedence constraint, including assigning each operation to a limited number of machines. In the FJSP, there is a set of R jobs: $\{J_1, J_2, \ldots, J_R\}$. Each job may have a different number of operations N ($N \in R$): $\{O_{Ji}, O_{Ji}, \ldots, O_{RN}\}$. Each operation can be processed by any of the feasible machines: $\{M_1, M_2, \ldots, M_K\}$. Processing time of one operation may vary when running on different machines. However, there are several rules that must be considered:

- Each job R is autonomous from the others.
- The precedence relation between the operations of job R must also be taken into account.
- Only one machine can perform an operation in one unit of processing time.
- The objective function of the FJSP is to minimize the completion time (C) stated as equation 1.
- The T_{ijk} denotes the processing time of the O_{ij} on the machine K.

$$\min\ C_{max} = \left(\sum\nolimits_{i=1}^{R} \sum\nolimits_{j=1}^{N} T_{ijk}\right) \qquad (1)$$

Table 1: *An example of FJSP.*

Job/Operation	Machines		
	M1	M2	M3
Job 1/ Operation 1: O_{11}	3	2	2
Job 1/ Operation 2: O_{12}	-	-	4
Job 1/ Operation 3: O_{13}	2	1	3
Job 2/ Operation 1: O_{21}	5	5	5
Job 2/ Operation 2: O_{22}	2	3	4
Job 3/ Operation 1: O_{31}	1	1	3
Job 3/ Operation 2: O_{32}	-	2	2

Table 1 shows an example of FJSP. Job 1 consists of three operations. The first operation is represented by O_{11}, the second and the third operations are O_{12} and O_{13} respectively. O_{11} can be processed by three machines as follows. When running O_{11} on machine M_1, the processing time (T) is 3 units. In the same way, both machine M_2 and machine M_3 complete operation O11 in 2 units of processing time.

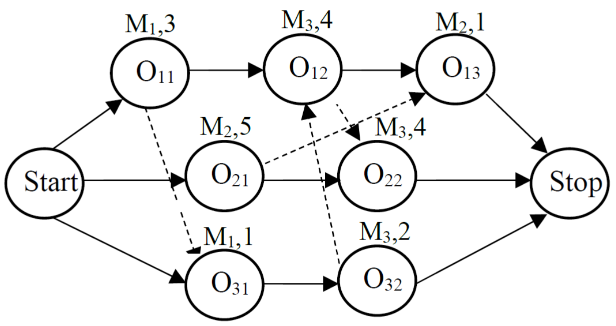

Fig.1: *An example of the schedule node.*

In figure 1, each node represents an operation. A node represents a chosen machine and corresponding processing time: (Machine, Processing time). A set of Disjunctive arcs (− − − − −) represent a sequence of operations which show processing on similar machines. A set of Conjunctive arcs (—) indicate a sequence of operations on each job.

We can explain ordering each of arc in terms of machine constraints. Node O_{11} and O_{31} can start anytime when machine M_1 is available. Node O_{21} and O_{13} are able to start anytime when machine M_2 is free. However, in this example, Node O_{13} could begin after node O_{21} finished, because the waiting time of node O_{13} is more than node O_{21}. According to the waiting time constraints, Node O_{31} could start first and then followed by node O_{12} and O_{22} respectively.

Because of a precedence relation, node O_{12} is able to start when node O_{11} is completed and Operation O_{13} can begin after node O_{12} is finished. Node O_{21} must be processed before node O_{22}. Finally, node O32 can start after node O_{31} is processed.

In addition, the critical path is the longest path from the starting node to the ending node. In figure 1, there exist two critical paths. First: Start—O_{31}—O_{32}—O_{12}—O_{22}—Stop. Second: Start—O_{11}—O_{12}—O_{22}—Stop.

2.2 The Firefly Algorithm

The Firefly algorithm (FA) belongs to a class of biologically inspired algorithms proposed by Xin-She Yang [14]. The aim of the FA is to solve modern numerical optimization and NP-hard problems such as the travelling salesman problem. The FA attempts to imitate the flashing behavior and the light intensity of fireflies. Fundamental functions of such flashes exist to attract other fireflies, and to allure potential prey. In addition, attractiveness is proportional to their brightness. For any two flashing fireflies, the firefly whose flash is less bright will move towards to the brighter one. Moreover, light intensity decreases while the distance increases. In the FA, each firefly represents one solution. The light intensity and the attractiveness are the objective function. The following are the steps of the FA algorithm:

- Initial population of fireflies X_i is randomly generated.
- The light intensity and attractiveness values are calculated.
- Firefly X_1 moves towards firefly X_2 when the fitness value of firefly X_2 is better. This step is reached when all fireflies are chosen.
- Light intensity and the attractiveness value are updated, then the fireflies are ranked according to their fitness value. The firefly whose best fitness value is answer.

2.3 The Genetic Algorithm

The genetic algorithm (GA) is an evolutionary algorithm presented by John Holland. The idea behind the GA is to imitate the genetic process of life. The GA is commonly used to generate high-quality solutions to optimization. The GA consists of four important processes as follows:

1) GA starts with a randomly created initial population which contains a number of chromosomes. Each chromosome represents a solution of the problem [15]. Thereafter, the fitness value of each chromosome is evaluated.

2) Parent selection is the process that pairs the initial chromosome. The chromosomes which have high fitness values have a high likelihood of being chosen to be parent chromosomes.

3) The crossover operation is the operator used to exchange some portion of two parent chromosomes. In the crossover process, when the random number is lower than the crossover probability (Pc), two new chromosomes called offspring are created by swapping a genome of two parent chromosomes.

4) The mutation operation is the sequence alteration process of the genome used to alter the offspring chromosome. This operator protects the algorithm from becoming stuck in a local optimum area.

5) In the selection process, the chromosome which has a high fitness value will survive to be included in the next generation.

2.4 Fuzzy Theory

Fuzzy theory was initiated by Lotfi A. Zadeh in 1965. The idea of fuzzy logic allows flexible thinking by a computer, by applying a more human-like way of thinking in the programming of computers [16]. The difference between classical and fuzzy sets is established by introducing a membership function [17]. The membership function is used to determine which element should be a member of which subset. Figure 2 shows an example of classical sets. It consists of three sets: Light, Medium and Heavy. A degree of membership is only a binary value: True (1) or False (0). If the v element is lower than 50, it is defined as Light. If the v element is more than 50 and lower than 100, it is assigned as Medium. When the v element is more than 100, it is set as Heavy. However, the membership degree of fuzzy set varies from 1 to 0. Figure 3 shows an example of a fuzzy set. When the element v is lower than 50, it belongs to Light. Nevertheless, if the v element is 75, it can be a member of Light or Medium depending on the membership function.

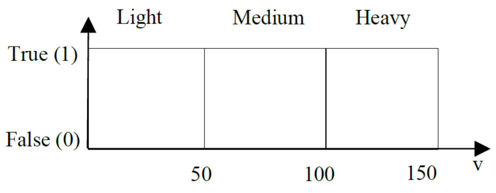

Fig.2: *An example of a classical set.*

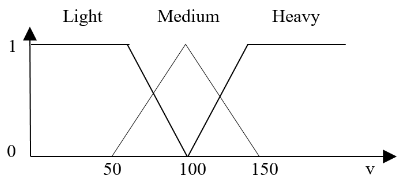

Fig.3: *An example of a fuzzy set.*

3. HYBRID FIREFLY ALGORITHM WITH FUZZY MOVEMENT

This paper proposes a new algorithm based on the firefly algorithm for solving FJSP. The steps of the proposed algorithm are explained in the following subsections.

3.1 Encoding and Decoding Method

According to Section 2.1, the FJSP consists of two sub-problems that need to be figured out. Hence, the solution of this article is divided into two parts. First part represents the sequence of the operation and the second part shows the machines that are allocated to each operation in the first part. In the first part of figure 4, the J_{1a} represents the first operation of job_1 (O_{11}). At the same time, the J_{2a} represents the first operation of job_2 (O_{21}). J_{1b} displays the second operation of job_1 (O_{12}). J_{1c} displays the third operation of job_1 (O_{13}). J_{3a} represents the first operation of job_3 (O_{31}). J_{2b} represents the second operation of job_2 (O_{22}) and J_{3b} displays the second operation of job_3 (O_{32}). For the second part of figure 4, consider from left to right, M_2 represents machine which is assigned to run the J_{1a}, M_3 is the machine allocated to run J_{1b}, M_2 displays machine used to run J_{1c} and is similar to the others.

For the decoding method, the computation time of the solution was first computed according figure 5. Then, the quality of each solution was evaluated by the fitness value which is stated in expression 2.

$$fitness\ value = \frac{1}{C_{max}} \tag{2}$$

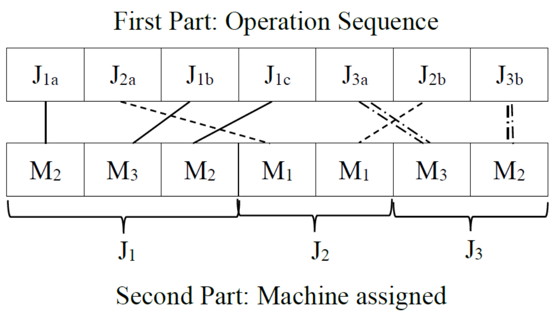

Fig.4: *An example of solution representation.*

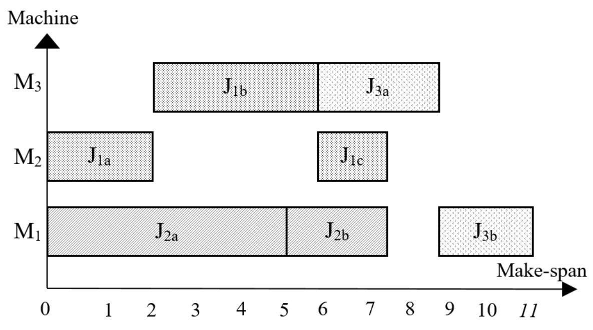

Fig.5: *An example of computation time calculation.*

3.2 Initialization

The GA is widely applied to generate high-quality solutions to optimization problems. To achieve a good quality for the initial population, this article utilized the GA to generate an initial population. At the beginning, N solutions which consist of the operation sequence part and the machine assigned part are randomly generated. They are called the random solutions.

Now a new idea named The Self-Adaptive Roulette Wheel Selection will be presented. The main concept of the self-adaptive roulette wheel is to select the pairs of chromosomes for the crossover operation. This stage uses the fitness value and the standard deviation (S.D.) to determine the distribution of the machines. This means that when any solution has high the fitness value and the S.D. value is low, we assume that the workload of all machines in the above solution is well balanced. As a result, the slot boundary of the above solution in a roulette wheel is big. In contrast, when the fitness value is low and the S.D. value is high, the slot size of the above solution in the roulette wheel slot is small. This gives a solution in which good balancing has a higher chance of being chosen.

To complete the adaptive roulette selection, the following steps are performed:

First Step: Calculate the Prob.(S) value as shown in equation 3. Then a roulette wheel is created, where the boundary of the roulette wheel will be chosen according the Prob.(S) value. So that Prob.(s) is the probability of selection, n is the total frequency of selection of all machines and X is the frequency of selection of machine number j.

$$Prob.(S)_i = \frac{fitness\ value_i^{S.D._i}}{\sum_{i=1}^{N} fitness\ value_i^{S.D._i}} \tag{3}$$

$$S.D._i = \sqrt{\frac{n\sum X_j^2 - (\sum X_j^2)}{n(n-1)}} \tag{4}$$

Second step: Pick the random number between zero and one. Thereafter, determine which slot of the roulette wheel the random value falls within.

Third step: Choose a solution that is representative of the above slot, then call it the father solution.

Fourth step: This article uses a Sampling Without Replacement Method (SWRM). For the SWRM, once the solution is sampled, that solution is not placed back for re-sampling. Hence, the slot which is a representative of the father solution is removed from a roulette wheel.

Fifth step: To choose the mother solution, repeat steps one to step three.

Sixth step: After the father and mother solutions are selected, put a slot which is a representation of the father solution back on the roulette wheel.

Step Seven: Repeat steps one through six until the desired number of parent solutions has been obtained.

When the number of the parent solutions is equal to the defined number, the next process is the crossover operation.

For the crossover operation process, we picked a random number between zero and one. If the random number is less than the crossover probability (P_c), the one-point crossover and the uniform crossover are employed to create the operation sequence and machine assignment parts of the offspring solution as shown in figures 6 and 7 respectively. In figure 6, we randomly select a crossover point on both parent solutions. Then, an element in the first half of the offspring$_1$ solution is inherited from the father solution. At the same time, the second part of the offspring$_1$ solution is carried on with elements of the mother solution if those elements are not shown in the first half of the offspring1 solution. In the opposite direction, the offspring$_2$ solution is created by copying the first part of mother solution to the first half of the offspring$_2$ solution, then scanning the father solution from left to right and filling the second half of the offspring$_2$ solution with elements of father solution which do not appear in the first half of offspring$_2$ solution. In Figure 7, the uniform solution is randomly generated. To create the machine assignment part of the offspring1 solution, when the element of the uniform solution is number 1, the gene of the father solution is copied. Meanwhile, if the element of the uniform solution is number 0, the gene of the mother solution is inherited. On the other hand, when the element of the uniform solution is number 1, the gene in the mother solution is copied to procreate the machine assignment part of the offspring2 solution. Simultaneously, the gene of the father is inherited when the element of the uniform solution is number 0.

When all of the crossover process has been completed, the mutation process is started. For each of the offspring solutions, a number between zero and one is randomly generated. If the random number is less than the mutation probability (Pm), the swap method is implemented in the operation sequence part and the random selection method is employed in the machine assignment part.

After that, the begin solution and the offspring solution are combined. In this paper, the GA algorithm is terminated after 50 iterations. The best N number of the combined solutions which are arranged in descending order according to their fitness value is selected and defined as the initial population.

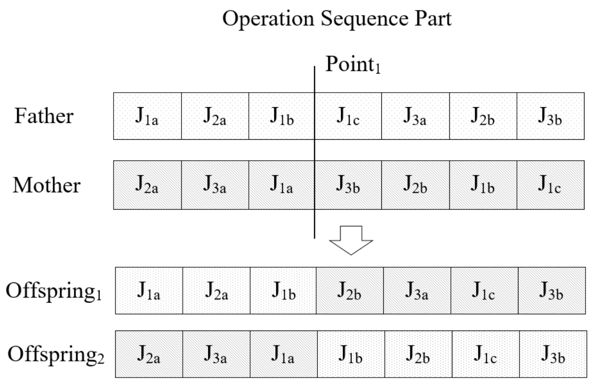
Operation Sequence Part
Point$_1$
Father
J$_{1a}$ J$_{2a}$ J$_{1b}$ J$_{1c}$ J$_{3a}$ J$_{2b}$ J$_{3b}$
Mother
J$_{2a}$ J$_{3a}$ J$_{1a}$ J$_{3b}$ J$_{2b}$ J$_{1b}$ J$_{1c}$
Offspring$_1$
J$_{1a}$ J$_{2a}$ J$_{1b}$ J$_{2b}$ J$_{3a}$ J$_{1c}$ J$_{3b}$
Offspring$_2$
J$_{2a}$ J$_{3a}$ J$_{1a}$ J$_{1b}$ J$_{2b}$ J$_{1c}$ J$_{3b}$

Fig.6: *An example of the one-point crossover operation..*

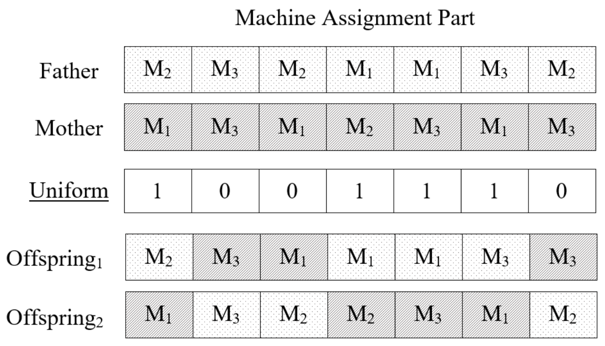
Machine Assignment Part
Father
M$_2$ M$_3$ M$_2$ M$_1$ M$_1$ M$_3$ M$_2$
Mother
M$_1$ M$_3$ M$_1$ M$_2$ M$_3$ M$_1$ M$_3$
Uniform
1 0 0 1 1 1 0
Offspring$_1$
M$_2$ M$_3$ M$_1$ M$_1$ M$_1$ M$_3$ M$_3$
Offspring$_2$
M$_1$ M$_3$ M$_2$ M$_2$ M$_3$ M$_1$ M$_2$

Fig.7: *An example of the uniform crossover operation..*

3.3 Attractiveness Values Calculation

After the initial population has been completely created, the attractiveness value of each machine is calculated. In this stage, the attractiveness value (A) is calculated by using the machine load statistics. The first statistic is the number of times that each is machine chosen (Freq.), with the purpose of knowing the selected frequency of each machine. The second statistic is the summation workload of each machine (SLoad), in order to know which machine is overloaded. The attractiveness value is computed with equation (5), where M is the number of machines.

$$A_M = Freq_M * SLoad_M \tag{5}$$

When any machine has a high frequency of selection including a high workload, as a result, the above machine has a high attractiveness value. Therefore, the likelihood that the above machines being moved will also increase. In this paper, the attractiveness value is used to manage the workload balance of the machines. Figure 8 shows the computation of Avg. Freq and Avg. SLoad.

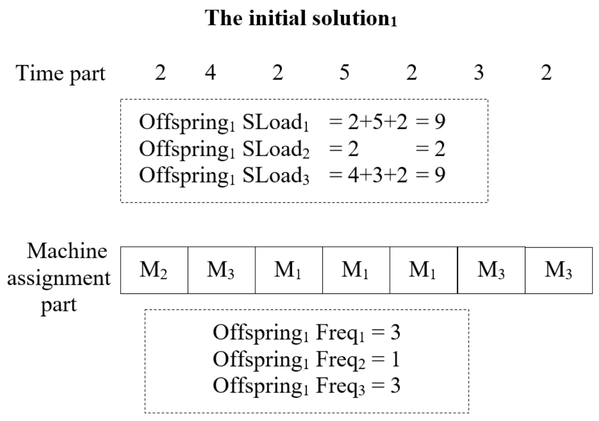
The initial solution$_1$
Time part
2 4 2 5 2 3 2
Offspring$_1$ SLoad$_1$ = 2+5+2 = 9
Offspring$_1$ SLoad$_2$ = 2 = 2
Offspring$_1$ SLoad$_3$ = 4+3+2 = 9
Machine assignment part
M$_2$ M$_3$ M$_1$ M$_1$ M$_1$ M$_3$ M$_3$
Offspring$_1$ Freq$_1$ = 3
Offspring$_1$ Freq$_2$ = 1
Offspring$_1$ Freq$_3$ = 3

Fig.8: *An example of machine load calculation.*

3.4 Fuzzy Movement Method

After generating the initial population and defining the attractiveness value of each machine completely, the fuzzy movement method is employed for each member of the initial population. The fuzzy movement method is used here to enhance the work balancing ability between the high and low workload machines. The procedure of the fuzzy movement method is as follows:

- Create fuzzy sets: In the proposed algorithm, each machine is represented by a trapezoidal and triangular fuzzy set. The support size of the fuzzy set depends on the attractiveness value of each machine. If the attractiveness value of the machine is high, the support size of the fuzzy set is wide. Whereas, if the attractiveness value of the machine is low, the support size of the fuzzy set is small. In addition, the overlapping size of the fuzzy set is adjusted by the value. Tables 2 and 3 show the instances involved in the fuzzy set. An example of a fuzzy set is illustrated in figure 9.
- Pick a random number between zero and the right corner position of the rightmost fuzzy set areas. Then, the membership degree of each machine is computed. Table 4 shows an example of membership degree calculation. When the random number is equal to 0.62, the membership degree of machine number 3 is higher than the others. Therefore machine number 3 is selected.
- Next, the operation sequence part of the initial solution$_1$ is explored. Thereafter, determine the operation that executed on machine number 3. Then randomly choose 40% of the specified operation and call it the chosen operation. For each of the chosen operations, we replace the machine number 3 by randomly choosing another machine from the candidate set. In detail, the candidate machine which has the lowest workload is first priority to be selected. An example of this step is shown in figure 10.
- At the same time, to enhance the diversity capability, the element in the operation sequence part is moved by applying the insertion technique. The procedure of the insertion technique is as follows: (a) Randomly select the swap points on the solution, then (b) Move the element on the operation sequence part as shown in Figure 11.

3.5 End Stage

In the final step, the termination criterion is checked. If the predefined number of iterations has not been reached, the best N number of the resultant solutions is selected as the new population for the next generation. Then, the algorithm branches back to stage 3.2

Table 2: *Fuzzy set support size calculation.*

Machine	Attractiveness value (A_M)	Support size
M1	3*9 = 21	21/44 = 0.477
M3	3*9 = 21	21/44 = 0.477
M2	1*2 = 2	2/44 = 0.046
	= 44	= 1

Table 3: *The support corners of fuzzy set.*

Machine	Left	Center	Right
M1	0	0.248	0.477
M3	0.273	0.512	0.750
M2	0.643	0.666	0.689

Table 4: *Membership degree calculation.*

Random value; X = 0.62			
Machine	Condition	Membership degree	
M_1	X > 0.47	0	0
M_3	0.51 < X < 0.75	(0.75-0.62)/ (0.75-0.51)	<u>0.54</u>
M_2	X < 0.64	0	0

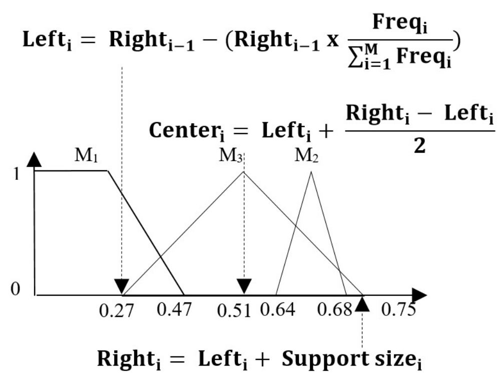

Fig.9: *An example of a Fuzzy Set.*

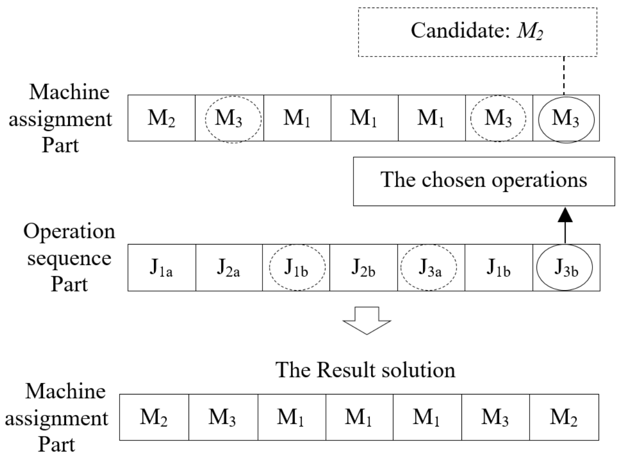

Fig.10: *An example of the machine moving method.*

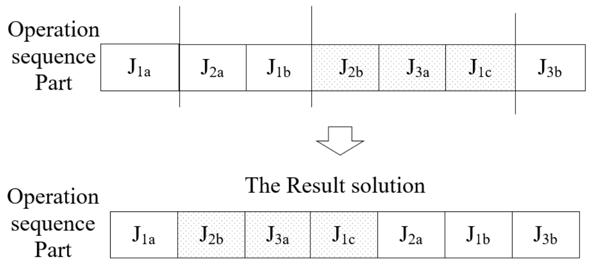

Fig.11: *An example of the operation moving method.*

4. EXPERIMENTAL RESULTS

4.1 Test sets

This article tests the performance of the proposed algorithm by using the three benchmark data sets obtained from the FJSP library [18]. The descriptions of the data sets are given below:

- The Hurink V data set consists of 40 test problems where all operations may be assigned to several machines.
- The Chambers and Barnes data set consists of 21 test problems. Each problem can be assigned to many different machines. The number of machines ranges from 11 to 18.
- The Brandimarte data set consists of 10 test problems. Each problem has a different number of jobs which range from 10 to 20. It has a number of machines ranging from 6 to 15.

4.2 Results and discussion

The performance of the proposed algorithm is compared to those of the current state-of-the-art metaheuristic algorithms in terms of the number of best solutions achieved and the average percentage deviation (Avg. PD) from the lower bound (Avg. PD. LB). The comparison is carried out in terms of the ability to obtain the optimal solution.

Table 5 shows the performance of the proposed algorithm. This test set is categorized into 8 groups according to a number of jobs and machines. For this data set, the model is run multiple times. Therefore, the average value (Avg) is the optimal choice to use to evaluate the performance of all algorithms.

All four algorithms found the lower bound solution for the La06-La20 and La 36-La40 test sets. The proposed algorithm, the hGA, and the IJA can obtain the best result for the La01-La05 test sets. For the La31-La35 test sets, only the hGA reached the optimal solution. For the La21-La25 and the La26-La30, the proposed algorithm came in first place and was followed by the hGA. When considering all test sets of the Hurink V data set, the proposed algorithm successfully found the solutions that deviated the least from their corresponding lower bounds, at 0.250 percentage.

Table 6 shows the comparative results of the proposed algorithm with other algorithms on the Chambers and Barnes data set. The table demonstrates that all of the compared algorithms perform almost equally well for the mt10x and mt10xxx test sets. The proposed algorithm is in line with the hGA and the IJA for the mt10xx, mt10xy and setb4xxx test sets. For the setb4c9, the proposed algorithm and the hGA obtained the best results. The proposed algorithm and the IJA are winning with the setb4x and setb4xx test sets. The IJA obtained the best result in the mt10c1, mt10xyz, setb4xy, setb4xyz, seti5c12, seti5cc, seti5x, seti5xx, seti5xxx, seti5xy, and seti5xyz test sets. Then second best was our proposed algorithm. In addition, the proposed algorithm found the best result for the setb4cc and mt10cc test sets. To be more specific, the proposed algorithm can obtain the answer which is not achieved by the other algorithms in 2 out of 21 problems, and the IJA is the best performer in 11 out of 21 problems. The IJA has the smallest mean percentage deviation from the lower bound at 22.37. The proposed algorithm came in second place by obtaining solutions that deviated the least from their corresponding lower bounds, at 22.46 percentage mean relative error. Regarding the computational time, the IJA takes the least time, followed by the proposed algorithm and then the hGA.

Table 7 illustrates the comparative results of the proposed algorithm and four of the metaheuristic algorithms on the Brandimarte data set. By the way, the Avg. value is not present in the hCEA. Therefore, the Best value is used to evaluate the performance of all compared algorithm instead. This table shows that all algorithms successfully found the lower bound of the MK03 and MK08. All four methods tie for first place for the MK01 MK04 and MK09. For the MK02, the proposed algorithm, hGA, HDE-N2, and hCEA obtained the best results. The proposed algorithm is in line with the hGA, HDE-N2, and IJA for the MK05. Likewise, with the MK06, the proposed algorithm, HDE-N2, and IJA obtain better results than the hGA and hCEA. Moreover, the proposed algorithm can obtain the answer which has not been achieved by other algorithms with the MK07. For the MK10, the proposed algorithm, hGA, and IJA are the closest to the lower bound. When considering all the test sets together, the average percentage deviation from the lower bound of the proposed algorithm is better than those of other algorithms at 18.88, followed by the HDE-N2 at 19.10. As for the computational time, the hGA is the winner. The proposed algorithm is slightly slower than that of the IJA, but much quicker than that of the HDE-N2.

Table 8 illustrates the performance between using the fuzzy movement method and not using the fuzzy movement method. The results show that the fuzzy movement method helps enhance the performance of the proposed algorithm and causes the algorithm to

Table 5: *The comparative performance of the proposed algorithm with other algorithms on the Hurink V data set.*

Test Set	Proposed Best (Avg)	hGA [11] Best (Avg)	HDE-N2 [12] Best (Avg)	IJA [8] Best (Avg)
La01-La05	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.01)**	0.00 **(0.00)**
La06-La10	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**
La11-La15	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**
La16-La20	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**
La21-La25	0.59 **(0.64)**	0.60 **(0.68)**	0.57 **(0.96)**	0.77 **(0.84)**
La26-La30	0.10 **(0.11)**	0.11 **(0.13)**	0.10 **(0.19)**	0.13 **(0.15)**
La31-La35	0.01 **(0.03)**	0.00 **(0.00)**	0.03 **(0.03)**	0.01 **(0.01)**
La36-La40	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**	0.00 **(0.00)**
% Avg. PD. LB	*0.233 (0.250)*	*0.237 (0.270)*	*0.233 (0.393)*	*0.303 (0.333)*

Table 6: *The results of the proposed algorithm with other algorithms on the Chambers and Barnes data set.*

Test Set	Job/ Machine	Lower Bound	Proposed			hGA [11]		HDE-N2 [12]		IJA [8]	
			Best	Avg.	CPU time (s)	Avg.	CPU time (s)	Avg.	CPU time (s)	Avg.	CPU time (s)
mt10c1	10/11	655	927	**927.20**	26.21	**927.20**	12.87	**927.72**	174.19	**927.10**	11.80
mt10cc	10/12	655	908	**908.00**	19.57	**910.00**	12.24	**910.60**	165.61	**908.03**	12.39
mt10x	10/11	655	918	**918.00**	18.60	**918.00**	12.69	**918.58**	179.22	**918.00**	13.80
mt10xx	10/12	655	918	**918.00**	20.27	**918.00**	11.70	**918.38**	179.84	**918.00**	10.70
mt10xxx	10/13	655	918	**918.00**	22.05	**918.00**	11.52	**918.00**	179.39	**918.00**	9.11
mt10xy	10/12	655	905	**905.00**	21.48	**905.00**	12.24	**905.56**	169.77	**905.00**	11.39
mt10xyz	10/13	655	847	**847.40**	20.53	**849.00**	10.71	**851.14**	160.24	**847.03**	20.70
setb4c9	15/11	857	914	**914.00**	23.59	**914.00**	45.99	**917.12**	313.02	**914.06**	17.21
setb4cc	15/12	857	907	**907.00**	21.91	**914.00**	38.79	**909.58**	316.89	**907.20**	15.89
setb4x	15/13	846	925	**925.00**	20.02	**931.00**	43.20	**925.82**	338.30	**925.00**	10.63
setb4xx	15/12	846	925	**925.00**	20.88	**925.00**	42.57	**925.64**	336.24	**925.00**	16.12
setb4xxx	15/13	846	925	**925.00**	20.51	**925.00**	36.09	**925.48**	353.55	**925.00**	18.84
setb4xy	15/12	845	910	**911.00**	31.03	**916.00**	41.49	**914.00**	330.18	**910.00**	12.45
setb4xyz	15/13	838	903	**903.80**	21.34	**905.00**	39.15	**905.28**	314.64	**903.30**	22.90
seti5c12	15/16	1027	1173	**1173.80**	24.90	**1175.00**	72.27	**1175.42**	1141.43	**1170.90**	26.70
seti5cc	15/17	955	1137	**1137.40**	22.31	**1138.00**	63.00	**1137.76**	1222.53	**1136.50**	24.20
seti5x	15/16	955	1204	**1204.00**	41.09	**1204.00**	66.78	**1205.64**	1112.77	**1198.20**	32.80
seti5xx	15/17	955	1199	**1199.40**	39.11	**1203.00**	63.72	**1202.68**	1078.60	**1197.10**	44.50
seti5xxx	15/18	955	1197	**1198.20**	33.56	**1204.00**	63.36	**1202.26**	1087.12	**1194.80**	20.78
seti5xy	15/17	955	1136	**1137.00**	38.22	**1136.50**	63.18	**1137.98**	1250.62	**1135.70**	19.80
seti5xyz	15/18	955	1125	**1125.80**	35.00	**1126.00**	57.96	**1129.76**	1244.22	**1125.10**	39.10
% Avg. PD. LB			*22.46*			*22.65*		*22.67*		*22.377*	

Table 7: *The comparative performance of the proposed algorithm with other algorithms on the Brandimarte data set.*

Test Set	Job/ Machine	Lower Bound	Proposed			hGA [11]		HDE-N2 [12]		IJA [8]		hCEA [7]	
			Best	Avg.	CPU time (s)	Best	CPU time (s)	Best	CPU time (s)	Best	CPU time (s)	Best	CPU time (s)
MK01	10/6	36	**40**	40.0	2.14	**40**	1.47	**40**	4.01	**40**	0.15	**40**	-
MK02	10/6	24	**26**	26.0	3.35	**26**	3.44	**26**	6.09	**27**	2.41	**26**	-
MK03	15/8	204	**204**	204.0	2.02	**204**	17.47	**204**	30.70	**204**	1.19	**204**	-
MK04	15/8	48	**60**	60.1	2.76	**60**	3.42	**60**	12.58	**60**	3.82	**60**	-
MK05	15/4	168	**172**	172.0	19.11	**172**	6.82	**172**	37.89	**172**	3.08	**173**	-
MK06	10/15	33	**57**	57.2	40.22	**58**	5.30	**57**	98.32	**57**	37.13	**59**	-
MK07	20/5	133	**137**	137.8	38.03	**139**	6.23	**139**	26.38	**139**	25.56	**140**	-
MK08	20/10	523	**523**	523.0	30.62	**523**	8.44	**523**	189.41	**523**	0.51	**523**	-
MK09	20/10	209	**307**	307.2	42.00	**307**	18.71	**307**	122.87	**307**	19.83	**307**	-
MK10	20/10	165	**197**	197.0	110.01	**197**	19.36	**198**	265.80	**197**	60.62	**200**	-
% Avg. PD. LB			*18.88*			*19.34*		*19.10*		*19.45*		*19.96*	

Table 8: *The comparative performance between using and not using the fuzzy movement method.*

Test Set	Job/ Machine	Lower Bound	Proposed with the fuzzy movement method			Proposed without the fuzzy movement method		
			Best	Avg.	CPU time (s)	Best	Avg.	CPU time (s)
MK01	10/6	36	**40**	40.0	2.14	**43**	43.0	2.74
MK02	10/6	24	**26**	26.0	3.35	**27**	27.5	3.30
MK03	15/8	204	**204**	204.0	2.02	**204**	204.0	3.01
MK04	15/8	48	**60**	60.1	2.76	**60**	60.5	3.45
MK05	15/4	168	**172**	172.0	19.11	**174**	175.8	22.31
MK06	10/15	33	**57**	57.2	40.22	**60**	61.5	40.01
MK07	20/5	133	**137**	137.8	38.03	**139**	139.0	40.23
MK08	20/10	523	**523**	523.0	30.62	**523**	523.0	29.55
MK09	20/10	209	**307**	307.2	42.00	**309**	310.5	48.92
MK10	20/10	165	**197**	197.0	110.01	**202**	203.3	152.22
% Avg. PD. LB			*18.88*			*21.71*		

converge on the optimal results rapidly.

In summary, our proposed algorithm is the clear winner. It outperformed the comparison algorithms in most cases on small to medium-sized problems. Although the proposed algorithm shows the greatest advantage over the comparison algorithms, it is not guaranteed that the proposed algorithm will be better on large and highly complex data sets. Hence, in further study, other methods such as local search algorithms should be tried to enhance the deep search capability.

5. CONCLUSION

This paper presents a hybrid firefly algorithm with fuzzy movement for solving the FJSP. To improve the quality of the initial population at the starting point, the GA is embedded in the initial stage. In improved GA, a new method called adaptive roulette wheel selection is proposed to enhance the diversity capability. In addition, one-point and uniform crossover are employed in the crossover stage. In the proposed algorithm framework, the introduction of attractiveness values calculation and a new selection operator called the fuzzy movement technique are used to help improve balancing between the activities and the machines. This algorithm was tested on 3 sets of standard benchmarks, the Hurink V data set, the Chambers and Barnes data set, and the Brandimarte data set. The results demonstrate that the proposed algorithm outperforms the compared algorithms in terms of the ability to achieve the optimal solution for all benchmark data sets except the Chambers and Barnes data set. In future work, we plan to develop multi-objective proposed algorithms for the multi-objective FJSP and employ the proposed algorithm to solve other kinds of discrete optimization problems.

References

[1] T. Liu, Y. Chen and J. Chou, "Solving Distributed and Flexible Job-Shop Scheduling Problems for a Real-World Fastener Manufacturer," in *IEEE Access*, vol. 2, pp.1598-1606, 2014.

[2] T. Jiang and C. Zhang, "Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases," in *IEEE Access*, vol. 6, pp. 26231 - 26240, 2018.

[3] Y. Yuan and H. Xu, "Multi objective Flexible Job Shop Scheduling Using Memetic Algorithms," in*IEEE Transactions on Automation Science and Engineering*, vol. 12, no.1, pp. 336 - 353, 2015.

[4] H. Chang, Y. Chen, T. Liu and J. Chou, "Solving the Flexible Job Shop Scheduling Problem with Makespan Optimization by Using a Hybrid Taguchi-Genetic Algorithm," in*IEEE Access*, vol. 3, pp. 1740 - 1454, 2015.

[5] T. Jamrus, C. Chien, M. Gen and K. Sethanan, "Hybrid Particle Swarm Optimization Combined with Genetic Operators for Flexible Job-Shop Scheduling Under Uncertain Processing Time for Semiconductor Manufacturing," in*IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no.1, pp. 32 - 41, 2018.

[6] Y. Lu, J. Lu and T. Jiang, "Energy-Conscious Scheduling Problem in a Flexible Job Shop Using a Discrete Water Wave Optimization Algorithm," in*IEEE Access*, vol. 7, pp. 101561- 101574, 2019.

[7] L. Sun, L. Lin, M. Gen and H. Li, "A Hybrid Cooperative Co-evolution Algorithm for Fuzzy Flexible Job Shop Scheduling," *IEEE TRANSACTION ON FUZZY SYSTEM*, vol. 27, no.5, pp. 1008 - 1022, May 2019.

[8] R. Caldeiran and A. Gnanavelbabu, "Solving the flexible job shop scheduling problem using an im-

proved Jaya algorithm," *Computers & Industrial Engineering*, vol. 137, pp. 1 - 16, 2019.

[9] Z. Zhu and X. Zhou, "An efficient evolutionary grey wolf optimizer for multi-objective flexible job shop scheduling problem with hierarchical job precedence constraints," *Computers & Industrial Engineering*, vol. 140, 2020.

[10] A. Vital-Soto, A. Azab and M. F. Baki, "Mathematical modeling and a hybridized bacterial foraging optimization algorithm for the flexible job-shop scheduling problem with sequencing flexibility," *Journal of Manufacturing Systems*, vol. 54, pp.74-93, 2020.

[11] J. Gaoa, L. Sun and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers & Operations Research*, vol. 35, pp. 2892 – 2907, 2008.

[12] Y. Yuan and H. Xu, "Flexible job shop scheduling using hybrid differential evolution algorithms," *Computers & Industrial Engineering*, vol. 65, pp.246-260, 2013.

[13] A.E. Eiben, M.C. Schut and A.R. de Wilde, "Boosting Genetic Algorithms with Self-Adaptive Selection," *2006 IEEE Congress on Evolutionary Computation*, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, pp. 1584-1589.

[14] X.S. Yang, "Firefly Algorithms for Multimodal Optimization," *International Symposium on Stochastic Algorithms: SAGA 2009: Stochastic Algorithms: Foundations and Applications*, vol. 5792., pp 169-178, 2019.

[15] P. Sinha, A. Chandwani and T. Sinha, "Algorithm of construction of Optimum Portfolio of stocks using Genetic Algorithm," *Munich Personal RePEc Archive*, pp. 1-34, 2013.

[16] Martin Hellmann, *Fuzzy Logic Introduction*, TU-Berlin, 2001.

[17] Raul Rojas, *Fuzzy Logic, Neural Networks*, Springer-Verlag, Berlin, 1996, ch. 11.

[18] `http://people.idsia.ch/~monaldo/fjsp.html`

Ajchara Phu-ang graduated Ph.D. in Information Technology from King Mongkut's Institute of Technology Ladkrabang and now she is working at College of Innovation, Thammasat University, where she is now an instructor in Innovation and Digital Transformation Department. Her research area is about Optimization Problem, Machine Learning and Bio-inspired algorithms.