Received: October 29, 2020. Revised: January 13, 2021. 

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

# Performance Evaluation of Various Heuristic Algorithms to Solve Job Shop Scheduling Problem (JSSP)

**Admi Syarif[1]*** **Ade Pamungkas[1]** **Renaldi Kumar[1]** **Mitsuo Gen[2]**

[1]*Department of Computer Science, Faculty of Mathematics and Sciences, Lampung University, Indonesia*
[2]*Deparment of Computer Science, Tokyo University of Science, Tokyo, Japan*
* Corresponding author's Email: admi.syarif@fmipa.unila.ac.id

**Abstract:** Scheduling is a famous optimization problem that seeks the best strategy of allocating resources over time to perform jobs/tasks satisfying specific criteria. It exists everywhere in everyday life, particularly in manufacturing or industrial applications. An essential class of scheduling problems is a job shop scheduling problem (JSSP), an NP-hard optimization problem. Several researchers have reported the use of heuristic methods to solve JSSP. This paper aims to investigate the performance of various heuristic algorithms to solve JSSP. Firstly, we developed a Genetic Algorithm (GA and compared the performance of some heuristic algorithms, including Particle Swarm Optimization (PSO), Upper-level algorithm (UPLA), Differential-based Harmony Search (DHS), Grey Wolf Optimization (GWO), Ant Colony Optimization (ACO), Bacterial Foraging Optimization (BFO), Parallel Bat Optimization (PBA), and Tabu Search (TS). The experimental results of the 28 benchmark test problems validated that the algorithms, except ACO, can provide the optimal solution of JSSP. PBA delivers the most impressive performance that solves 26 cases optimally, with the average error equal to 0.05%. Among those 28 test problems, TS, DHS, and PBA can solve 26 instances optimally, followed by GA that solves 21 cases.

**Keywords:** Combinatorial optimization, Job shop scheduling problem (JSSP), Artificial intelligence, Heuristic algorithms, NP-hard problem.

## 1. Introduction

Scheduling is one of the most essential and commonly encountered classes of optimization problems. Scheduling problems exist everywhere in everyday life, particularly in industrial or manufacturing applications. What makes scheduling problems important is that many manufacturing companies or industries have limited resources and have to satisfy specific criteria. Determining an excellent strategy to schedule tasks will reduce production costs or improve profitability.

There are many variations of scheduling problems for several real-world uses. There are, in general, two classes of fundamental scheduling problems discussed in many pieces of literature. Those are the Job Shop Scheduling Problem (JSSP) and the Flow Shop Scheduling Problem (FSSP). Among them, JSSP has been the most widespread and complex problem. The JSSP model has been vital and practical and challenges many researchers in engineering, computing, and operational [1]. It represents a problem of allocating a set of resources (machine) to perform tasks (job) that consists of $m$ different operations, and the separate device has the other processing time. The main objective is to determine the best machine schedule to do all job with the best objective value, i.e., minimizing makespan ($C_{max}$), mean flow time, mean tardiness, earliness, maximum lateness, etc. [2]. The JSSP with $n$ job and $m$ machine will have $(n!)^m$ possible solutions. Thus, for the relatively large size problem, it will be computationally expensive to solve scheduling problems optimally [3].

Generally, there are two classes of methods for solving JSSP; exact and heuristic methods [4]. The first-class methods include: integer programming [5] [6], Lagrangian relaxation [7], dynamic

programming [8], and Branch and bound [9]. Another class is called heuristic algorithms, first introduced in early 1960. It was initially concerned with increasing the effectiveness of the problem-solving process. Although those methods do not guarantee the finding of an optimal solution, those have been reported useful in solving many challenging optimization problems within a reasonable computational time.

The term heuristic is usually related to the methods or algorithms for solving problems intelligently. Over the last few decades, with the rapid increase of computer technology, we have seen enormous growth in heuristic approaches to various hard and challenging optimization problems, such as Simulated Annealing (SA), Ant Colony Optimization (ACO), Genetic Algorithm (GA), Branch-and-Bound, Tabu Search (TS), and so on.

Among the heuristics algorithms, GA probably has been the most popular approach [10]. Aided by GAs, researchers evolve solutions to complex combinatorial optimization problems easily and rapidly. Our past researches reported the excellent performance of GA in solving various combinatorial optimization problems [11, 12], and [13]. In contrast to other heuristics methods, it utilizes a set population of solutions in its search. It makes GA more robust to solve many real-world problems [10].

For solving JSSP, several researchers have reported the robustness of heuristics methods. These include the TS algorithm by Mauro [14], Ant Colony Optimization (ACO) by Flórez [15], Bacterial Foraging Optimization (BFO) by Zhao [16], Bat Optimization algorithm by Dao [17], and so on. Despite these interests, however, no researcher said the best method to solve JSSP all-time optimally. This fact shows that researches on the performance evaluation of the heuristic algorithms for JSSP are very crucial.

This paper aims to investigate the performance of heuristic algorithms for solving JSSP. First, we developed a GA approach and conducted intensive numerical experiments on a set of Benchmark test problems (3 test problems of Fisher dan Thompson [18], and 25 instances of Lawrence [19]). Further; we compared the results to those of some heuristic algorithms, including Particle Swarm Optimization (PSO), Upper-level algorithm (UPLA), Differential-based Harmony Search (DHS), Grey Wolf Optimization (GWO), Ant Colony Optimization (ACO), Bacterial Foraging Optimization (BFO), Parallel Bat Optimization (PBA), and Tabu Search (TS). The comparison is made based on the solution's quality, the relative error, and the number of instances solved (NIS) optimally for each test problem

The organization for the remainder of the paper is as follows: the next section describes the formulation of JSSP. In the third section, we concern ourselves with some essential discussions of several heuristic algorithms, including GA's working mechanisms. Furthermore, in the fourth section, some comparisons of results from the numerical experiments on Benchmark test problems are presented. We evaluate the algorithms' performance based on the solution quality, the relative error, and the number of instances solved (NIS) optimally. In the end section, we provide the conclusion of this study, showing the approaches; remarkable effectiveness.

## 2. Mathematical model of JSSP

Consider the JSSP with *m* machines to perform *n* jobs or tasks. Each job/task consists of *m* operations. The order of operations for the machines is predetermined. The different device is used for a separate action to complete one job. The problem involves designing an effective strategy (called schedule) of assigning some activities to be done by the devices by meeting constraints.

The main objective of JSSP is to determine the best machine schedule to do all jobs with the best objective function, i.e., minimizing makespan($\mathrm{C_{max}}$), mean flow time, mean tardiness, earliness, and maximum lateness. The most common constraint of the JSSP is as follows [20]:

1. A machine can process only a job or task at a time.
2. The machine sequence of the machine to process each job must be the same.
3. The process of a job cannot be interrupted.

Let $t_{ij}$ and $f_{ij}$ are the starting and the finishing time of processing job $j$ at machine $i$. $P_{ij}$ Is the processing time of machine $i$ to perform job $j$. The makespan ($C_{max}$) here represents the finishing time of the last job. The mathematical model of JSSP is as follows [21]:

$$\textbf{min } C_{max} \tag{1}$$

s.t.

$$t_{hj} - t_{ij} \geq P_{ij} \tag{2}$$

$$C_{max} - t_{ij} \geq P_{ij} \tag{3}$$

$$t_{ij} - t_{ik} \geq P_{ik} \quad \text{or} \quad t_{ik} - t_{ij} \geq P_{ij} \tag{4}$$

$$t_{ij} \geq 0 \tag{5}$$

In this model, Eq. (1) is the objective function to minimize the makespan. The constraint (2)

guarantees that the next step of machine $h$ for job $j$ is started after finishing the step at machine $i$ for job $j$. Next, the constraint (3) ensures the makespan is equal to or greater than the finishing time of the last job. Eq. (4) shows that only a machine processed a job at a time. Finally, Eq. (5) is a non-negative constraint.

## 3. Heuristic approaches for JSSP

In this section, we shall describe clearly the drawbacks of previous heuristic approaches used for solving JSSP, including Particle Swarm Optimization (PSO), Upper-level algorithm (UPLA), Differential-based Harmony Search (DHS), Grey Wolf Optimization (GWO), Ant Colony Optimization (ACO), Bacterial Foraging Optimization (BFO), Parallel Bat Optimization (PBA), and Tabu Search (TS). Next, we introduced the design of the GA approach. We emphasize the difference between the methods to clarify the position of this works.

### 3.1 ACO (Ant colony optimization)

Ant Colony Optimization (ACO) is a heuristic algorithm that combines concepts from Artificial Intelligence and Biology, inspired by ants' collective behaviour [15]. Dorigo first introduced ACO for solving the Traveling Salesman Problem. Currently, ACO has solved various fields of our daily life applications. The ACO-based method, called Elitist Ant System (EAS) for JSSP, has been carried out among many by Florez in 2013. Each job consists of a sequence of operations, and each process comes with a determined machine and processing time. They adopt the collective intelligence of many simple agents to determine optimal solutions with minimum makespan.

They presented the obtained results for each of the JSSP instances by Lawrence [22]. They compared the results with those of Tabu Search (TS) and the best-known solution (BKS) taken from [23]. The algorithm implemented, Elitist Ant System, has proven to be competitive by finding the more reliable quality solutions for JSSP [15]. However, it also requires more effort to obtain the best-known solution for all LA instances.

### 3.2 Particle swarm optimization (PSO)

Particle Swarm Optimization (PSO) is a population-based metaheuristic optimization approach, introduced by Eberhart and Kennedy. Animals' behaviour to search for food, such as birds and fishes, inspires the PSO. Each flock of birds or fishes tends to determine its speed based on personal experience and information obtained through interactions with other members. Pongchairerks and Kachitvichyanukul reported the use of PSO to solve JSSP (JSP-PSO) in 2009 [24]. This paper proposed the GLN-PSO algorithm that allows the swarm to explore the other parts of the search spaces simultaneously. To evaluate the algorithm's performance, they had numerical experiments on 33 well-known benchmark test problems from Fisher and Thompson (FT06, FT10, FT20), and the rest from Lawrence. Their computational results show the algorithm can optimally solve the problem 17 times.

### 3.3 Tabu search (TS)

Another popular heuristic method for solving combinatorial optimization problems is Tabu Search. Since Glover originally introduced it in 1986, hundreds of researchers reported the success of Tabu Search (TS) applications to various combinatorial optimization problems. It has been reported among practical algorithms and provides optimal/near-optimal solutions for many cases. TS searches for the best solution based on the local search method's optimization. A TS algorithm's main components are memory structures, a trace of the search's evolution, and strategies to use the memorized information in the best possible way. Dell'Amico first introduced the use of TS for solving the JSSP [14]. Their basic idea is to avoid cycles in the search's evolution by inhibiting the algorithm from reoccurring more recently made moves. They evaluated TS's performance on a set of problem instances, including Lawrence (LA01-40) [22]. Their results show that TS is useful in finding the optimal/near-optimal solutions.

### 3.4 Upper-level algorithm (UPLA)

Nowadays, research on developing the heuristic algorithm for JSSP has become more variegated. In 2019, Pongchairerks proposed a brand new two-level metaheuristic algorithm, consisting of an upper-level algorithm (UPLA) and a lower-level algorithm (LOLA) for the JSSP. The UPLA is a brand new algorithm that begins with a population of the combinations of values from LOLA's input-parameter. At every iteration, UPLA attempts to increase its population by utilizing the feedback returned from LOLA. Thus, LOLA may improve from a local search algorithm to be an iterated local search algorithm.

Furthermore, UPLA and LOLA result in the two-level algorithm, which may adapt to every JSSP instance. Similar to the other population-based algorithms, UPLA examines search space based on the population of the combination from the input

parameter. Real numbers represent all input-parameter values.

Among JSSP algorithms, the most similar algorithm to the proposed algorithm was the two-level Particle Swarm Optimization (PSO) [24]. The correspondence is that they generate parameterized-active schedules with similar methods on both their lower-level algorithms; furthermore, parameters for both algorithms (lower and upper-level algorithms) control the identical. However, the two-level PSO is different from the suggested two-level metaheuristic algorithm that uses GLN-PSO's framework [25]. The authors assessed the algorithms' performance on 53 well-known benchmark instances, including FT06, FT10, FT20, and LA01-LA40 [26]. Considering the similarity and difference, they also compared their results with those of the two-level PSO [24].

### 3.5 Differential-based harmony search (DHS)

The Differential-based Harmony Search (DHS) to minimize makespan for JSSP was reported by Zhao in 2018 [16]. The DHS improves the variable neighbourhood search (VNS) based on the critical path blocks. The transformed VNS, on the critical path, is embedded into the DHS to seek a more reliable solution based on the blocks. They evaluate DHS's performances on a set of benchmark instances from the OR-library with the objective of minimized makespan [27]. Compared with various HS-based algorithms and other state-of-the-art algorithms, the DHS is superior in solution quality, convergence speed, and stability [16].

### 3.6 GWO (Grey wolf optimization)

The Grey wolf optimization (GWO) algorithm is a new population-oriented heuristic algorithm inspired by grey wolves' social hierarchy and hunting behaviour. Tianhua Jiang introduced GWO, a brand new swarm-based intelligence algorithm, to deal with optimization problems in 2018. The algorithm is based on the crossover operation and adapt the searching operator to minimize the makespan (maximum completion time). They also introduced an adaptive modification method to the algorithm to keep the variety of population. They compared the results with other published algorithms in the two scheduling cases. According to the experimental results, GWO provided better solutions for some instances [28].

### 3.7 Bacterial foraging optimization (BFO)

Kim Passino proposed another intelligent heuristic algorithm, called the Bacterial Foraging Optimization (BFO), in 2002 [29]. The BFO is inspired by the cooperation and competition behaviours of a bacterium named E. coli in seeking food. One of the primary processes in BFO is the evolution process. It begins when the bacterium migrates to a better solution according to the advantaged group's activity. Zhou introduced an algorithm called Chemotaxis-enhanced-BFO (CEBFO) to solve the JSSP [30]. To improve the algorithm's performance, they include a local search operation and chemotaxis with the differential evolution (DE). They conducted some numerical experiments on a set of benchmark problems of JSSP. The results demonstrated a good understanding of the algorithm.

### 3.8 Parallel bat algorithm (PBA)

In 2015, Dao proposed a parallel-based heuristic algorithm version, called the parallel bat algorithm (PBA) [17]. The fundamental structure of the PBA is to divide the distribution of the bat populations into several groups. They offered three schemes, namely a random-key encoding, a makespan, and a communication strategy. To examine the method's accuracy, they had some experiments on 43 (forty-three) benchmark instances (Fisher and Thompson with FT06, FT10, FT20 [18], and Lawrence (LA01–LA40) [31]). They compared their experimental results are to those of the PSO algorithm. These show that the intended approach gives competitive returns.

### 3.9 Genetic algorithm (GA)

Since Holland introduced GA in 1975, it has witnessed many exciting advances in using Genetic Algorithms (GAs) to solve challenging optimization problems in everything from production design to inventory and network design problems. It is a heuristics method, inspired by the process of Darwinian evolution. GA has been a multi-purpose approach for searching the global optimality; adapting GAs to a specific optimization problem is challenging but frustrating. The selection methods, efficient design of the chromosome representation, crossover and mutation process, and GA parameters' value influence GA's success [32]. Therefore, discovering an efficient GA approach system for a particular problem becomes essential in GA research.

#### 3.9.1. The Chromosome representation

When implementing GA for an optimization problem, an important issue is how to generate a chromosome that would bring us to the right solution. For the initial population, we have to create a

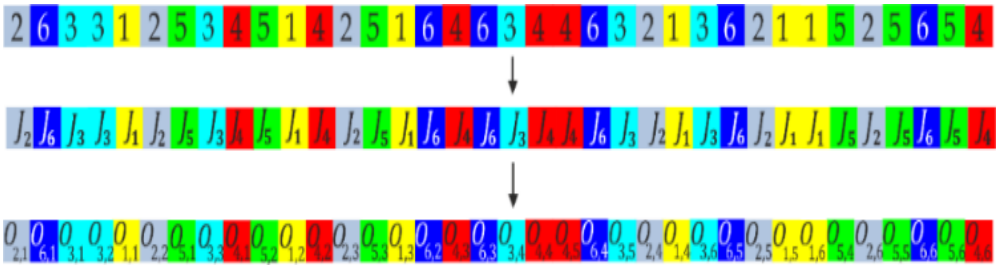

Figure. 1 An example of the chromosome for problem FT06

*pop_size* chromosome. Each chromosome consists of $n \times m$ gen generated randomly, and each *job* will appear $m$ in the chromosome. One illustrates an example of the chromosome for the test problem FT06, having six jobs and six machines.

The chromosome in Figure. 1 indicates that the first activity to be scheduled is job two at machine one, followed by job six at machine one; then, job three at machine one, and so on, according to the order arranged in the chromosome list. $O_{i,j}$ represents the operation for the job *i* at machine *j*.

#### 3.9.2. Genetic operations

**Procedure:** Self Crossover:
**Step 1:** Chose a parent arbitrarily for crossover.
**Step 2:** Determine two crossover points randomly
**Step 3:** Move substring between the above two points

The mutation operation is an essential feature of GA to maintain the chromosome's diversity in the generation. This paper adopts the swap mutation that randomly selects two genes in the chromosome and then exchanges them.

#### 3.9.3. Evaluation and selection

In GA implementation, we computed the fitness value to measure how well each chromosome fits the problem's requirements. For JSSP, we can use makespan as the fitness value as follows:

$$\text{Fitness}(x) = \frac{1}{f(x)} \tag{6}$$

where *f(x)* is the objective function (makespan).

The decoding process to compute the makespan ($C_{max}$) of the schedule is as follows:

**Step 1:** Select the chromosome for the decoding process.
**Step 2:** Read gen in the chromosome started from the left.
**Step 3:** Determine the machine number from the machine-order matrix, based on the job operation number's information.
**Step 4:** Determine the processing time from the processing time matrix based on the job operation number's data.
**Step 5:** Determine the maximum time of the last job time.
**Step 6:** Renew the current job finishing time by adding the time to the result of Step 5;
**Step 7:** Repeat Step 2 to Step 6 until the last-gen in the chromosome.

Another essential process of GA is the way to determine the chromosome for the next population. Of course, the selection process should be done based on the fitness value. There have been several selection strategies introduced in GA applications. Here, we adopt the elitist approach by selecting the best *pop_size* chromosome for the next generation.

## 4. Numerical experiments and results

### 4.1 Design of numerical experiments

To evaluate the effectiveness and the efficiency of the algorithm, we first have some numerical experiments for GA on 28 Benchmark test problems: 3 instances (FT06, FT10, dan FT20) of Fisher dan Thompson [18], and 25 instances (LA01-LA25) of Lawrence [19], taken from the OR-library [27]. We implement the algorithm in MATLAB R2015b and run on an Intel Core i5 processor of 2.53 GHz.
The GA parameters are set as: crossover probability ($\boldsymbol{p\_C}$) = 0.4, mutation probability ($\boldsymbol{p\_M}$) = 0.2, population size ($\boldsymbol{pop\_size}$) = 400 and maximum generation (*max_gen*) = 10-2000, for each test problem, the experiments are conducted 10 (ten) times. Table 1 presents the overall results obtained for all test problems, where the best and the average values represent the best and the average fitness value from the 10 (ten) running times. BKS represents the best-known solution in the literature.

Table 1. The experimental results of the GA approach

| Cases | $n \times m$ | ACT* | BKS* | GA | | | Error (%) |
|---|---|---|---|---|---|---|---|
| | | | | Best | Average | Worst | |
| **FT06** | 6 × 6 | 1,00 | 55 | 55 | 55 | 55 | 0,00 |
| **FT10** | 10 × 10 | 2,76 | 930 | 951 | 988.8 | 1030 | 2,26 |
| **FT20** | 20 × 5 | 1,08 | 1165 | 1178 | 1184.3 | 1197 | 1,12 |
| **LA01** | 10 × 5 | 6,00 | 666 | 666 | 666 | 666 | 0,00 |
| **LA02** | 10 × 5 | 3,97 | 655 | 655 | 658.8 | 666 | 0,00 |
| **LA03** | 10 × 5 | 6,44 | 597 | 597 | 611 | 621 | 0,00 |
| **LA04** | 10 × 5 | 3.35 | 590 | 590 | 592 | 601 | 0,00 |
| **LA05** | 10 × 5 | 1,8 | 593 | 593 | 593 | 593 | 0,00 |
| **LA06** | 15× 5 | 2,00 | 926 | 926 | 926 | 926 | 0,00 |
| **LA07** | 15 × 5 | 9,00 | 890 | 890 | 890 | 890 | 0,00 |
| **LA08** | 15 × 5 | 9,00 | 863 | 863 | 863 | 863 | 0,00 |
| **LA09** | 15 × 5 | 1,7,0 | 951 | 951 | 951 | 951 | 0,00 |
| **LA10** | 15 × 5 | 0,80 | 958 | 958 | 958 | 958 | 0,00 |
| **LA11** | 20 × 5 | 2,80 | 1222 | 1222 | 1222 | 1222 | 0,00 |
| **LA12** | 20 × 5 | 2,80 | 1039 | 1039 | 1039 | 1039 | 0,00 |
| **LA13** | 20 × 5 | 2,60 | 1150 | 1150 | 1150 | 1150 | 0,00 |
| **LA14** | 20 × 5 | 1,00 | 1292 | 1292 | 1292 | 1292 | 0,00 |
| **LA15** | 20 × 5 | 20,00 | 1207 | 1207 | 1207 | 1207 | 0,00 |
| **LA16** | 10 × 10 | 767 | 945 | 959 | 977.2 | 997 | 0,00 |
| **LA17** | 10 × 10 | 774 | 784 | 784 | 788.9 | 797 | 0,00 |
| **LA18** | 10 × 10 | 808 | 848 | 848 | 868.5 | 909 | 0,00 |
| **LA19** | 10 × 10 | 1.395 | 842 | 842 | 850 | 874 | 0,00 |
| **LA20** | 10 × 10 | 1.234 | 902 | 907 | 928.4 | 992 | 0,55 |
| **LA21** | 15 × 10 | 1.743 | 1046 | 1061 | 1097 | 1114 | 1,43 |
| **LA22** | 15 × 10 | 1.443 | 927 | 943 | 987.8 | 1046 | 1,08 |
| **LA23** | 15 × 10 | 752 | 1032 | 1032 | 1035.3 | 1054 | 0,00 |
| **LA24** | 15×10 | 1.122 | 935 | 948 | 977 | 994 | 1,39 |
| **LA25** | 15×10 | 2.049 | 977 | 987 | 1015,8 | 1042 | 1,02 |
| **Average** | | | | | | | **0.3162** |

*ACT: Average Computational Time (in second)

### 4.2 Results and discussion

In the above table, the error is computed by using the following formula:

$$Error = \frac{(Best - Optimum) \times 100\%}{Optimum} \quad (7)$$

Here, one can notice the excellent performance of GA to solve JSSP. Despite not reaching the optimal solution all-time, GA presents the optimal solutions (21 instances), with an average error of less than 0.32 percent. The results also show that GA can provide solutions to the problems within reasonable computational time. For some hard/difficult cases, GA can obtain near-optimal solutions with an error from 0.5 to 1.43 percent. More efforts can be made to improve the solutions by possibly hybridizing GA with other local search techniques. The Gantt chart schedule and the convergence of the solution for LA40 are illustrated in Figure. 2 and Figure. 3, respectively.

### 4.3 Comparison of some heuristic methods

In this research, we evaluate the merit and the limitation of the approaches by comparing the results of some heuristic algorithms, including Ant Colony Optimization (**ACO)** [15], Particle Swarm Optimization (PSO) [33], Tabu Search (TS) [14], Upper-level algorithm (UPLA) [26], Differential-based Harmony Search (DHS) [16], Grey Wolf Optimization (GWO) [28], Bacterial Foraging Optimization (BFO) [30], Parallel Bat Optimization (PBA) [17]**,** and the proposed Genetic Algorithm (GA). The performances are measured based on the solution quality, the number of instances solved (NIS) optimally, and the relative error. We made a

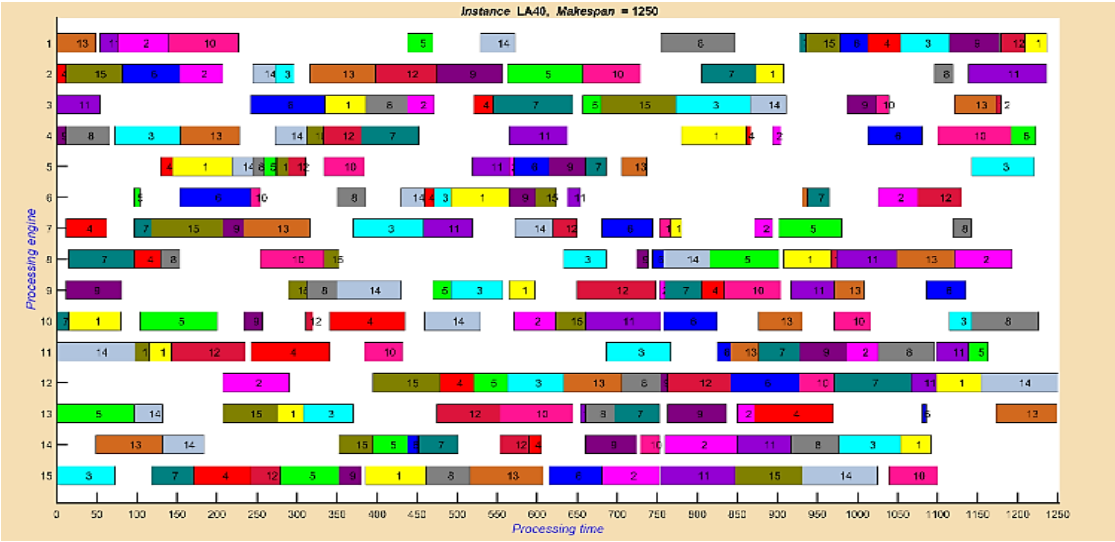


Figure. 2 Gantt chart schedule for LA40

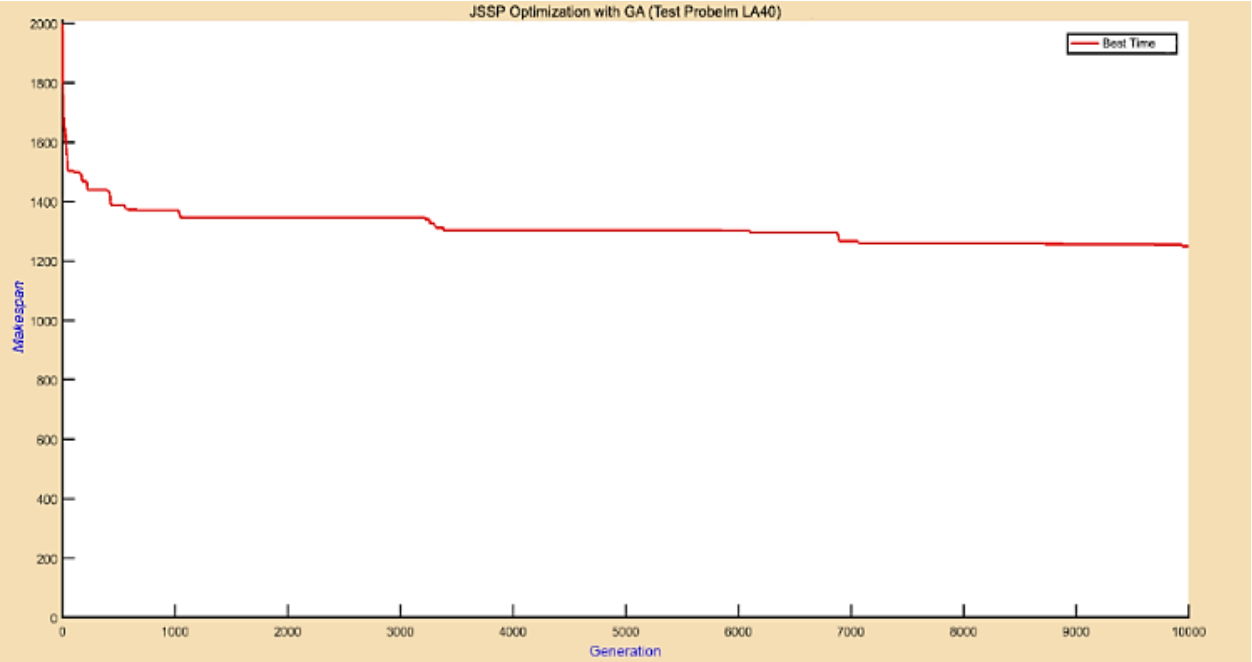


Figure. 3 The convergence of the objective function in the generation for LA40

Table 2. Performance of the heuristic approaches on all instances

| **Test Problem** | **Size ($n \times m$)** | **BKS*** | **ACO** [15] | **PSO** [33] | **TS** [14] | **UPLA** [26] | **DHS** [16] | **GWO** [28] | **BFO** [30] | **PBA** [17] | **GA (Proposed)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FT06 | 6 × 6 | 55 | - | 55 | - | 55 | 55 | 55 | 55 | 55 | 55 |
| FT10 | 10 × 10 | 930 | - | 951 | - | 930 | 930 | 940 | 937 | 930 | 951 |
| FT20 | 20 × 5 | 1165 | - | 1191 | - | 1165 | 1165 | 1178 | 1171 | 1165 | 1178 |
| LA01 | 10 × 5 | 666 | 666 | 666 | 666 | 666 | 666 | 666 | 666 | 666 | 666 |
| LA02 | 10 × 5 | 655 | 669 | 663 | 655 | 655 | 655 | 655 | 655 | 655 | 655 |
| LA03 | 10 × 5 | 597 | 623 | 603 | 597 | 597 | 597 | 597 | 597 | 597 | 597 |
| LA04 | 10 × 5 | 590 | 611 | 611 | 590 | 590 | 590 | 590 | 590 | 590 | 590 |
| LA05 | 10 × 5 | 593 | 593 | 593 | 593 | 593 | 593 | 593 | 593 | 593 | 593 |
| LA06 | 15× 5 | 926 | 926 | 926 | 926 | 926 | 926 | 926 | 926 | 926 | 926 |
| LA07 | 15 × 5 | 890 | 890 | 890 | 890 | 8 q90 | 890 | 890 | 890 | 890 | 890 |
| LA08 | 15 × 5 | 863 | 863 | 863 | 863 | 863 | 863 | 863 | 863 | 863 | 863 |
| LA09 | 15 × 5 | 951 | 951 | 951 | 951 | 951 | 951 | 951 | 951 | 951 | 951 |
| LA10 | 15 × 5 | 958 | 958 | 958 | 958 | 958 | 958 | 958 | 958 | 958 | 958 |
| LA11 | 20 × 5 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 |
| LA12 | 20 × 5 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 |
| LA13 | 20 × 5 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 |
| LA14 | 20 × 5 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 |
| LA15 | 20 × 5 | 1207 | 1212 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 |
| LA16 | 10 × 10 | 945 | 1005 | 959 | 945 | 945 | 945 | 956 | 945 | 945 | 945 |
| LA17 | 10 × 10 | 784 | 812 | 784 | 784 | 784 | 784 | 790 | 785 | 784 | 784 |
| LA18 | 10 × 10 | 848 | 885 | 848 | 848 | 848 | 848 | 859 | 848 | 848 | 848 |
| LA19 | 10 × 10 | 842 | 875 | 857 | 842 | 842 | 842 | 845 | 844 | 842 | 842 |
| LA20 | 10 × 10 | 902 | 912 | 910 | 902 | 902 | 902 | 937 | 907 | 902 | 907 |
| LA21 | 15 × 10 | 1046 | 1107 | 1074 | 1048 | 1052 | 1046 | 1090 | - | 1046 | 1061 |
| LA22 | 15 × 10 | 927 | 1018 | 944 | 933 | 927 | 927 | 970 | - | 933 | 937 |
| LA23 | 15 × 10 | 1032 | 1051 | 1032 | 1032 | 1032 | 1032 | 1032 | - | 1032 | 1032 |
| LA24 | 15×10 | 935 | 1011 | 971 | 941 | 941 | 979 | 982 | - | 941 | 948 |
| LA25 | 15×10 | 977 | 1062 | 987 | 979 | 982 | 1016 | 1008 | - | 977 | 987 |

Table 3. The error comparison of the heuristic approaches in all instances

| Test Problem | Dimensi ($n \times m$) | ACO | PSO | TS | UPLA | DHS | GWO | BFO | PBA | GA |
|---|---|---|---|---|---|---|---|---|---|---|
| FT06 | 6 × 6 | - | 0.00 | - | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| FT10 | 10 × 10 | - | 2.26 | - | 0.00 | 0.00 | 1.08 | 0.75 | 0.00 | 2.26 |
| FT20 | 20 × 5 | - | 2.23 | - | 0.00 | 0.00 | 1.12 | 0.52 | 0.00 | 1.12 |
| LA01 | 10 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA02 | 10 × 5 | 2.14 | 1.22 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA03 | 10 × 5 | 4.36 | 1.01 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA04 | 10 × 5 | 3.56 | 3.56 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA05 | 10 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA06 | 15× 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA07 | 15 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA08 | 15 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA09 | 15 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA10 | 15 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA11 | 20 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA12 | 20 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA13 | 20 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA14 | 20 × 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA15 | 20 × 5 | 0.41 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| LA16 | 10 × 10 | 6.35 | 1.48 | 0.00 | 0.00 | 0.00 | 1.16 | 0.00 | 0.00 | 0.00 |
| LA17 | 10 × 10 | 3.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.77 | 0.13 | 0.00 | 0.00 |
| LA18 | 10 × 10 | 4.36 | 0.00 | 0.00 | 0.00 | 0.00 | 1.30 | 0.00 | 0.00 | 0.00 |
| LA19 | 10 × 10 | 3.92 | 1.78 | 0.00 | 0.00 | 0.00 | 0.36 | 0.24 | 0.00 | 0.00 |
| LA20 | 10 × 10 | 1.11 | 0.89 | 0.00 | 0.00 | 0.00 | 3.88 | 0.55 | 0.00 | 0.55 |
| LA21 | 15 × 10 | 5.83 | 2.68 | 0.19 | 0.57 | 0.00 | 4.21 | - | 0.00 | 1.43 |
| LA22 | 15 × 10 | 9.82 | 1.83 | 0.65 | 0.00 | 0.00 | 4.64 | - | 0.65 | 1.08 |
| LA23 | 15 × 10 | 1.84 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | - | 0.00 | 0.00 |
| LA24 | 15×10 | 8.13 | 3.85 | 0.64 | 0.64 | 4.71 | 5.03 | - | 0.64 | 1.39 |
| LA25 | 15×10 | 8.70 | 1.02 | 0.20 | 0.51 | 3.99 | 3.17 | - | 0.00 | 1.02 |

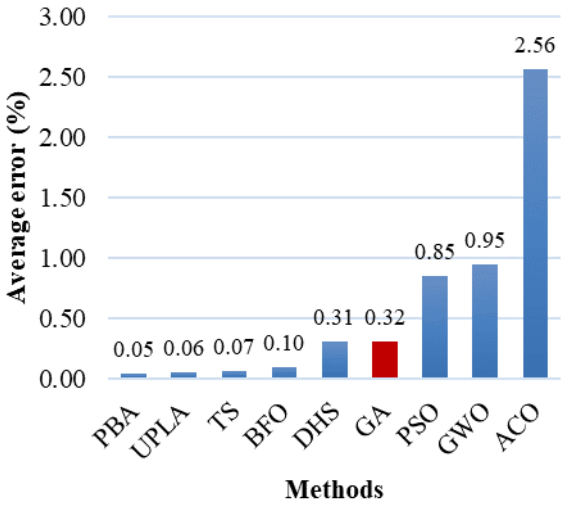


Figure. 4 The comparison of average error

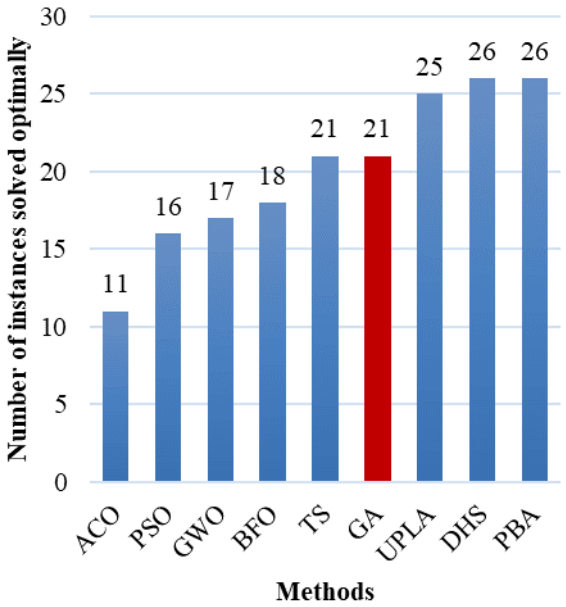


Figure. 5 The comparison of NIS optimally by the heuristic methods

comparison of the results for 28 benchmark test problems (FT06, FT10, F20) and 25 instances (LA01-LA25) of Lawrence [19]. Table 2 summarizes the comparative results. We also computed the percentage relative error concerning BKS, as shown in the following Table 3.

We also analyze the algorithm based on the average errors and the number of instances solved

(NIS) optimally. We illustrated the comparison of the average errors and obtained NIS by the algorithms in Figure. 4 and Figure. 5, respectively. These results indicate that, though no algorithm can give the optimal solution, the algorithms effectively find the optimal/near-optimal solutions to the problems. Almost all algorithms, except ACO, can provide an error of less than 1 percent. PBA presents the most impressive performance that can solve 26 cases optimally, with the average error equal to 0.05%. Among those 28 test problems, TS, DHS, and PBA can solve 26 instances optimally, followed by GA that solves 21 cases.

## 5. Conclusion

This paper analyzed the performance of some heuristic algorithms for solving JSSP. First, we developed the GA approach and conducted some intensive numerical experiments on a set of Benchmark instances from the literature. We investigated some heuristic methods' performance, based on the solution quality, the relative error, and the number of instances solved (NIS) optimally. The results validate that, though no method presents optimal solutions at all times, the heuristics are robust in searching for the optimal solutions of JSSP. Among them, the PBA is the most effective algorithm that solves 26 instances optimally with an average error of 0.05%, followed by DHS, UPLA, and GA. The computational results show that the proposed GA can obtain competitive results in both NIS (21 cases with an average error of 0.32%) and computational time. These findings add to a growing body of literature on the applications of heuristics.

## Conflicts of Interest

The authors declare no conflict of interest.

## Author Contributions

Admi Syarif supervised the study, analyzed the results, verified the study's findings, and wrote the paper. Mitsuo Gen gave the idea of the algorithm; Renaldi Kumar and Ade Pamungkas designed and performed the experiments;

## Acknowledgments

This research was supported by the Scientific "Riset Unggulan" Program, Ministry of Education and Culture, Lampung University, the Grant-in-Aid 2226/UN26.21/PN/2019, Indonesia, 2019. The authors thank the anonymous reviewers for their valuable comments and suggestions on this paper.

## References

[1] J. K. Lenstra and A. H. G. Rinnooy Kan, "Computational Complexity of Discrete Optimization Problems", *Annals of Discrete Mathematics*, Vol. 4, pp. 121–140, 1979.

[2] J. F. Gonçalves, J. J. De Magalhães Mendes, and M. G. C. Resende, "A hybrid genetic algorithm for the job shop scheduling problem", *European Journal of Operational Research*, Vol. 167, No. 1, pp. 77–95, 2005.

[3] E. A. C. Uzorh and N. Innocent, "Solving Machine Shops Scheduling Problems using Priority Sequencing Rules Techniques", *The International Journal of Engineering and Science*, Vol. 3, No. 6, pp. 1813–2319, 2014.

[4] K. Akram, K. Kamal, and A. Zeb, "Fast simulated annealing hybridized with quenching for solving job-shop scheduling problem", *Applied Soft Computing*, Vol. 49, pp. 510–523, 2016.

[5] C. Özgüven, Y. Yavuz, and L. Özbakir, "Mixed-integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence-dependent setup times", *Applied Mathematical Modelling*, Vol. 36, No. 2, pp. 846–858, 2012.

[6] D. Catanzaro, L. Gouveia, and M. Labbé, "Improved integer linear programming formulations for the job Sequencing and tool Switching Problem", *European Journal of Operational Research*, Vol. 244, No. 3, pp. 766–777, 2015.

[7] P. Baptiste, M. Flamini, and F. Sourd, "Lagrangian Bounds for Just-In-Time Job-Shop Scheduling", *Computers & Operations Research*, Vol. 35, No. 3, pp. 906–915, 2008.

[8] J. A. S. Gromicho, J. J. Van Hoorn, F. Saldanha-da-Gama, and G. T. Timmer, "Solving the job-shop scheduling problem optimally by dynamic programming", *Computers & Operations Research*, Vol. 39, No. 12, pp. 2968–2977, 2012.

[9] P. Brucker, E. K. Burke, and S. Groenemeyer, "A branch and bound algorithm for the cyclic job-shop problem with transportation", *Computers & Operations Research*, Vol. 39, No. 12, pp. 3200–3214, 2012.

[10] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, New York, 2000.

[11] A. Syarif, Y. S. Yun, and M. Gen, "Study on multi-stage logistic chain network: A spanning tree-based genetic algorithm approach", *Computers and Industrial Engineering*, Vol. 43, No. 1–2, pp. 299–314, 2002.

[12] M. Gen and A. Syarif, "Double Spanning Tree-Based Genetic Algorithm for Two-Stage Transportation Problem", *International Journal of Knowledge-Based and Intelligent Engineering Systems*, Vol. 7, No. 4, pp. 214–221, 2003.

[13] A. Syarif, D. Anggraini, K. Muludi, Wamiliana, and M. Gen, "Comparing Various Genetic Algorithm Approaches for Multiple-Choice Multi-Dimensional Knapsack Problem (mm-KP)", *International Journal of Intelligent Engineering and Systems*, Vol. 13, No. 5, pp. 455–462, 2020.

[14] M. Dell'Amico and M. Trubian, "Applying tabu search to the job-shop scheduling problem", *Annals of Operations Research*, Vol. 41, No. 3, pp. 231–252, 1993.

[15] E. Flórez, W. Gómez, L. Bautista, E. Florez, W. Gomez, and Lola Bautista, "An Ant Colony Optimization Algorithm for Job Shop Scheduling Problem", *International Journal of Artificial Intelligence & Applications (IJAIA)*, Vol. 4, No. 4, pp. 53–66, 2013.

[16] F. Zhao, S. Qin, G. Yang, W. Ma, C. Zhang, and H. Song, "A Differential-Based Harmony Search Algorithm with Variable Neighborhood Search for Job Shop Scheduling Problem and Its Runtime Analysis", *IEEE Access*, Vol. 6, pp. 76313–76330, 2018.

[17] T. K. Dao, T. S. Pan, T. T. Nguyen, and J. S. Pan, "Parallel bat algorithm for optimizing makespan in job shop scheduling problems", *Journal of Intelligent Manufacturing*, Vol. 29, No. 2, pp. 451–462, 2018.

[18] H. Fisher and G. L. Thompson, "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules", *Industrial Scheduling*, Vol. 3, No. 2, pp. 225–251, 1963.

[19] S. Lawrence, "Supplement to Resource-Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques", *Energy Procedia*, Vol. 4, No. 7, pp. 4411–4417, 1984.

[20] H. Zaher, N. Ragaa, and H. Sayed, "A novel Improved Bat Algorithm for Job Shop Scheduling Problem", *International Journal of Computer Applications*, Vol. 164, No. 5, pp. 24–30, 2017.

[21] K. Ploydanai and A. Mungwattana, "Algorithm for Solving Job Shop Scheduling Problem Based on machine availability constraint", *International Journal on Computer Science and Engineering (IJCSE)*, Vol. 02, No. 05, pp. 1919–1925, 2010.

[22] L. Davis, *Job Shop Scheduling with Genetic Algorithms*, Psychology Press, East Sussex, United Kingdom, 1985.

[23] D. Applegate and W. Cook, "Computational study of the job-shop scheduling problem", *ORSA Journal on Computing*, Vol. 3, No. 2, pp. 149–156, 1991.

[24] P. Pongchairerks and V. Kachitvichyanukul, "A two-level Particle Swarm Optimization algorithm on Job-Shop Scheduling Problems", *International Journal of Operational Research*, Vol. 4, No. 4, pp. 390–411, 2009.

[25] P. Pongchairerks and V. Kachitvichyanukul, "A Non-Homogenous Particle Swarm Optimization with Multiple Social Structures", In: *Proc. of the 2005 International Conf. on Simulation and Modeling*, Nakornpathom, Thailand, pp. 132–136, 2005.

[26] P. Pongchairerks, "A two-level metaheuristic algorithm for the job-shop scheduling problem", *Complexity*, Vol. 2019, pp. 1–11, 2019.

[27] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt (accessed Feb. 07, 2020).

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

[29] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control", *IEEE Control Systems Magazine*, Vol. 22, No. 3, pp. 52–67, 2002.

[30] F. Zhao, X. Jiang, C. Zhang, and J. Wang, "A chemotaxis-enhanced bacterial foraging algorithm and its application in job shop scheduling problem", *International Journal of Computer Integrated Manufacturing*, Vol. 28, No. 10, pp. 1106–1121, 2015.

[31] J. Dossey, A. Otto, L. Spence, and C. Eynden, *Discrete Mathematics*, 2nd Edition, Harpercollins College Div, New York City, New York, 1993.

[32] A. Syarif, W. Wamiliana, P. Lumbanraja, and M. Gen, "Study on genetic algorithm (GA) approaches for solving Flow Shop Scheduling Problem (FSSP)", In: *Proc. of the 5th International Conf. on Science, Technology and Interdisciplinary Research (IC-STAR 2019)*, Bandar Lampung, Indonesia, 2020.

[33] P. Pongchairerks, "Particle swarm optimization algorithm applied to scheduling problems", *Science Asia*, Vol. 35, No. 1, pp. 89–94, 2009.