# A Swarm Intelligence Approach to Flexible Job-Shop Scheduling Problem with No-Wait Constraint in Remanufacturing

Shyam Sundar[1], P.N. Suganthan[1], and T.J. Chua[2]

[1] School of Electrical & Electronic Engineering,
Nanyang Technological University Singapore,
639798, Singapore
{ssundar,epnsugan}@ntu.edu.sg
[2] Singapore Institute of Manufacturing Technology,
71, Nanyang Drive Singapore,
638075, Singapore
tjchua@simtech.a-star.edu.sg

**Abstract.** This paper addresses a flexible job-shop scheduling problem with no-wait constraint (FJSPNW) which combines features of two well-known combinatorial optimization problems – flexible job-shop scheduling problem and no-wait job-shop scheduling problem. To solve FJSPNW with the objective of minimizing the makespan, an artificial bee colony (ABC) algorithm is proposed. This problem finds application in remanufacturing scheduling systems. ABC algorithm is a recently developed swarm intelligence technique based on intelligent foraging behavior of honey bee swarm. Since its inception, it has shown promising performance for the solution of numerous hard optimization problems. Numerical experiments have been performed on a set of standard benchmark instances in order to demonstrate the effectiveness of ABC algorithm.

**Keywords:** Swarm Intelligence, Artificial Bee Colony Algorithm, No-wait, Flexible Job-shop, Scheduling.

## 1 Introduction

In recent years, various scheduling problems have been studied extensively due to their theoretical and practical applications in the planning and manufacturing systems. Job-shop scheduling problem (JSP) is one among them. JSP is a well-known $\mathcal{NP}$-Hard problem [1] combinatorial optimization problem which seeks to schedule $n$ jobs on $m$ machines with some criteria. Each job consists of a predetermined sequence of non-preemptable operations and each operation needs a machine to process. Flexible job-shop scheduling problem (FJSP) [2–5] and no-wait job-shop scheduling problem (NWJSP) [6–10] are also two well-known hard combinatorial optimization problems which are generalization of JSP. FJSP allows an operation to be processed by any machine out of a set of machines, whereas NWJSP doesnot allow waiting time between operations associated with

L. Rutkowski et al. (Eds.): ICAISC 2013, Part II, LNAI 7895, pp. 593–602, 2013.

a job, i.e., all operations associated with a job must be processed continuously one by one without any interruption.

In this paper, we consider a generalized problem called flexible job-shop scheduling problem with no-wait constraint (FJSPNW) which combines both features of FJSP and NWJSP. This problem finds application in remanufacturing scheduling systems. In FJSPNW, a set of $n$ jobs $J_1, J_2, \ldots, J_n$ and a set of $m$ machines $M_1, M_2, \ldots, M_m$ are given. Each job $J_i$ has a predetermined sequence of non-preemptable $N_i$ operations $\{O_{i1}, O_{i2}, \ldots, O_{iN_i}\}$. Each operation $O_{ij}$ can be processed by any machine from a predetermined set of machines. $P_{ijk}$ is the processing time of operation $O_{ij}$ on machine $k$. All jobs and machines are available at time 0. At a given time, each machine can process at most one operation. No waiting time is allowed between operations associated with a job. In other words, all operations associated with a job must be processed continuously one by one without any interruption. Thus, FJSPNW aims to assign each operation to an appropriate machine and to find out a sequence of jobs with no-wait constraint in such a way that some objective is met. This paper addresses FJSPNW with the objective of minimizing the makespan.

To the best of our knowledge, there is no reported work for FJSPNW in the literature. Since FJSPNW is a generalization of FJSP and JSPNW, therefore, FJSPNW is more complex problem than FJSP and JSPNW. In such a condition, metaheuristic techniques can be effective techniques in finding high quality solutions in a reasonable time. It has been seen in recent years that swarm intelligence techniques characterized by the collective behavior and self-organized behavior of swarms have been applied effectively to solve numerous hard optimization problems. Artificial bee colony (ABC) algorithm inspired by intelligent foraging behavior of honey bee swarm [11] is one among them. Since its inception, it has shown promising performance and competitiveness with other state-of-the-art metaheuristic techniques. Initially, it was designed for optimization problems in continuous domain. Later, it was further carried out for discrete optimization problems [12–15]. A detailed survey of the applications of ABC algorithm can be studied in [16]. This paper presents an ABC algorithm for the solution of FJSPNW and the performance of ABC algorithm is tested on a set of standard benchmark instances.

The rest of this paper is organized as follows: Section 2 outlines a brief introduction to ABC algorithm. Section 3 presents ABC algorithm for FJSPNW which will be referred to as ABC_FJSPNW. Section 4 demonstrates computational results, whereas Section 5 concludes the proposed work.

## 2 ABC Algorithm

ABC algorithm developed by Karaboga [11] is a swarm intelligence technique based on intelligent foraging behavior of honey bee swarm. The colony of real bees is classified into three groups – Scout, employed and onlooker bees. Scout bees are those bees that are searching new food sources in the vicinity of the hive. When the search of discovering a food source is successful, scout bee becomes

employed bee. Employed bees are those bees that are currently exploiting food sources. After exploiting, they carry loads of nectar from the food sources to the hive and participate in communicating information about their food sources to the onlooker bees by means of dancing. The nature and duration of the dance performed by the dancing bee depend on the richness of the food source. Onlooker bees are those bees that are waiting for the employed bees. After arrival of employed bees, onlooker bees watch numerous dances of employed bees and then, they select a food source with a probability which is directly proportional to the nectar content of that food source. It helps in attracting more and more number of onlooker bees towards a good quality of a food source. As soon as, an onlooker bee selects a food source, it becomes an employed bee. If a food source is exploited completely, then this food source is abandoned by those employed bees that associate with this food source and all these employed bees become either scouts or onlookers.

In ABC algorithm [11], artificial bees are also classified into three groups – scout, employed and onlooker bees. Each food source represents a feasible solution to the problem under consideration, whereas its nectar content represents the fitness of the solution. Since in ABC algorithm, each food source is associated with an employed bee. So, the number of food sources is equal to the number of employed bees. ABC algorithm begins with initializing a fixed number of solutions randomly. Then, a search process that includes two following phases is carried out repeatedly until the termination criterion is satisfied.

1. *Employed Bee Phase*: Each employed bee determines a new food source (solution) in the neighborhood of its associated food source. If the new food source is better than its currently associated solution based on fitness, then it moves to the new neighboring food source, otherwise it continues with the old one. If a food source doesnot improve for *limit* number of trials (iterations), then it is presumed that this solution is exploited completely. In such a situation, employed bee that associates with this food source abandons its food source and becomes a scout bee. As soon as, this scout bee generates a new solution (food source) randomly, it again becomes an employed bee.
2. *Onlooker Bee Phase*: All employed bees participate in communicating information about their fitnesses with onlooker bees. With the help of probability based selection method which biases towards the good quality of solutions (food sources) over the bad ones, each onlooker bee selects a solution. A good solution will be selected by more and more number of onlooker bees. All onlooker bees, similar to employed bee phase, also determine new food sources in the neighborhood of its selected food source. After that, among all new neighboring food sources determined by onlookers that associate with a particular food source $k$ and food source $k$ itself, the best solution will be assigned to the position of $k^{th}$ food source. This phase is completed when the new positions of all food sources are determined.

# 3 ABC_FJSPNW

Main characteristics of ABC_FJSPNW are described as follows:

## 3.1 Solution Encoding

Since FJSPNW is characterized by two things – sequence of jobs with no-wait constraint and assignment of operations to machines, therefore, in order to encode (represent) a solution, a list called *job_list* is used to maintain the sequence of jobs. In addition, for each job $i$ in *job_list*, another list called *machine_list* of all selected machines used for processing their operations $\in i$ is used. It is to be noted that each index of *machine_list* represents the operation number of the corresponding job.

## 3.2 Initialization

In order to maintain a balance between diversity and quality in the population, an iterative procedure which consists of random and greedy strategies is applied to generate each initial solution $S$ of the population. Initially, $S$ is an empty solution and all jobs are added to a set called $U$. A job $i$ is selected uniformly at random from $U$. This selected job $i$ is added to the first empty position of *job_list* of $S$. Hereafter, in order to process each operation $j$ of this selected job $i$, i.e., $O_{ij}$ without violating no-wait constraint, iteratively a machine $k$ is selected for $O_{ij}$ from its predefined set. Selection of a machine $k$ for $O_{ij}$ from its predefined set is done either randomly or greedily. With probability $P_r$, a machine is selected uniformly at random from its predefined set, otherwise a machine having minimum processing time is selected with probability 1 - $P_r$. $P_r$ is an empirical parameter. This machine $k$ for $O_{ij}$ is added to $j^{th}$ index of *machine_list* of the selected job $i$. This whole iterative procedure continues until assignment of all operations $\in i$ to their machines is completed. Selected job $i$ is deleted from $U$. This whole iterative procedure continues until $U$ becomes empty.

To ensure the feasibility of a solution after the generation of a solution, a solution decoding procedure is applied to decode the solution which is explained in the next subsection.

## 3.3 Solution Decoding

Since FJSPNW is a generalization of NWJSP, therefore, solution decoding procedure for NWJSP can also be applied for FJSPNW. The procedure for solution decoding [18] is described as follows:

**Step 1.** Initially, each machine is allotted with zero to infinity idle time interval.

**Step 2.** Select a job in the sequence of the schedule (solution), then find all matching among idle time intervals of machines and the processing times of all operations of the current job without violating no-wait constraint.

**Step 3.** Update idle time interval of each machine.
**Step 4.** If all jobs are chosen, then the procedure stops. Otherwise goto Step 2.

It is to be noted according to [18] that if the processing time of any operation of the current job doesnot succeed to match with its corresponding machine's idle time interval, then that operation is delayed until it finds a corresponding idle time interval, and simultaneously every other operation of the same job is also delayed by the same amount and also a re-check is done so that the processing time of every operation of the current job matches with its corresponding machine's idle time interval without violating no-wait constraint. Basically, this decoding procedure delays the start time of each job of the schedule so that the processing times of its all operations fall in their corresponding machines' idle time interval and it also doesnot violate no-wait constraint.

Each candidate solution is uniquely associated with an employed bee. The fitness (makespan) of each solution is calculated.

### 3.4 Probability of a Selecting a Food Source

Each onlooker bee selects a food source (solution) with the help of binary tournament selection method which is a probability based selection method. In this method, two different food sources are selected uniformly at random from the population. With probability $P_{bt}$, best of them is selected, otherwise worse one is selected. $P_{bt}$ is an empirical parameter.

### 3.5 Determination of a Neighboring Food Source

In order to find a high quality solution (food source), the problem structure must be exploited as much as possible. Since the problem structure of FJSPNW is based on the sequence of jobs with no-wait constraint and assignment of operations to machines, therefore, two different methods – multi-point insert method [17] and perturbation method – are used in a mutually exclusive way for determining a solution $Y$ in the neighborhood of a solution $X$. Multi-point insert method uses the concept of utilizing solution components from another solution [12][17]. It is also based on this assumption that if a component is positioned at right place in a good solution (schedule), then there is high possibility that this component should be exactly at the same position or vicinity to the same position in many other good solutions. Whereas, perturbation method perturbs the sequence of jobs and assignment of operations to machines in order to avoid the solution to be trapped in a local optima. With probability $P_{mpi}$, multi-point insert method is applied, otherwise perturbation method is applied with probability 1-$P_{mpi}$. $P_{mpi}$ is an empirical parameter.

1. *Multi-point Insert Method*: In this method, an another solution $Z$ which is different from $X$ is selected randomly from the population. Initially, $Y$ is an empty solution. After that, *nrp* different positions are selected randomly

**Algorithm 1**. Pseudo-code of ABC_FJSPNW

```
Initialize EP solutions e_1, e_2, ..., e_EP;
BEST ← Best solution ∈ {e_1, e_2, ..., e_EP};
while Termination criteria is not satisfied do
    for l ← 1 to EP do
        e_nbr ← Nbring_Sol(e_l);
        if e_nbr is better than e_l then
            e_l ← e_nbr;
        else if e_l has not improved for a predetermined number of iterations,
        i.e., limit iterations then
            Scout bee;
        if e_l is better than BEST then
            BEST ← e_l;
    for l ← 1 to OP do
        I_l ← Selection(e_1, e_2, ..., e_EP);
        o_l ← Nbring_Sol(e_{I_l});
        if o_l is better than BEST then
            BEST ← o_l;
    for l ← 1 to OP do
        if o_l is better than e_{I_l} then
            e_{I_l} ← o_l;
```

from *job_list* of $Z$. All jobs corresponding to selected positions as well as their associated machines in their *machine_list* are inserted into the same positions of *job_list* and their *machine_list* of $Y$, respectively. The remaining empty positions in *job_list* of $Y$ are inserted with those jobs of $X$ which are not in $Y$ according to the order in which they appear in $X$. At the same time, all machines in *machine_list* associated with those jobs in $X$ that participate in $Y$ for insertion are also inserted into the same *machine_list* of their corresponding jobs in $Y$. It is to be noted that *nrp* is an empirical parameter.

2. *Perturbation Method*: This method itself involves two strategies. First a copy, say $Y$, of the solution $X$ is created. Then both strategies are applied one by one on the solution $Y$. In the first strategy, two different jobs are selected randomly from *job_list*. After that these two selected jobs as well as their associated machines in their *machine_list* are swapped. This swap strategy is applied for *nswp* times, where *nswp* is the empirical parameter. Hereafter, second strategy is applied. In this strategy, a job $i$ is selected randomly from *job_list*, then an index $j$ of *machine_list* corresponding to the job $i$ is selected randomly. It should be noted that an index of *machine_list* represents the operation number of the job $i$, i.e. $O_{ij}$. After that the machine $k$ assigned for $O_{ij}$ is replaced with an another machine $k'$ which has the minimum processing time in its machine set of $O_{ij}$. It is to be noted that

if the machine set of $O_{ij}$ has single machine, then this replacement strategy starts afresh. In addition to this one, if the selected machine $k'$ is found to be machine $k$, then a different machine is randomly selected from its machine set of $O_{ij}$ which must be different from $k$. This strategy is applied for at most $nmc$ times, where $nmc$ is an empirical parameter.

### 3.6 Other Features

An employed bee that associates with a food source (solution) becomes a scout bee iff this solution doesnot improve for a predetermined number of trials called *limit* parameter. When it takes place, this scout bee will generate a new solution which is similar to initialization procedure (see subsection 3.2) for generating an initial solution. As soon as a solution is generated, this scout bee again becomes employed bee. The parameter *limit* plays a vital role in ABC algorithm as it provides a balance between exploration and exploitation.

Algorithm 1 explains the pseudo-code of ABC_FJSPNW, where $EP$ and $OP$ denotes the number of employed and onlooker bees respectively. *Nbring_ Sol(e)* is a function that determines a solution in the neighborhood of the solution $e$ and returns this neighboring solution. *Selection*($e_1$, $e_2$, ..., $e_{EP}$) is an another function used by an onlooker bee for selecting a solution from solutions $e_1$, $e_2$, ..., $e_{EP}$. This function returns the index of selected solution.

## 4 Computational Results

ABC_FJSPNW has been implemented in C and executed on a Linux based 3.2 GHz × 4 i5 with 3.7 GB RAM system. Since to the best of our knowledge, there is no reported work for FJSPNW in the literature. So, in order to test ABC_FJSPNW we have used a set of standard benchmark instances – BRdata set [3] – which are treated as FJSPNW instances. BRdata set contains 10 instances. In this benchmark set, the number of jobs, i.e., $n$ varies from 10 to 20, while the number of machines, i.e., $m$ varies from 6 to 15.

Since parameters used in ABC_FJSPNW play vital roles in the performance of ABC_FJSPNW, therefore, their values are chosen carefully after a large number of experiments. Different-2 values for which parameters have been experimented are described as follows: [50, 100, 150, 200] for the total number of employed bees $EP$, [100, 150, 200, 250] for the total number of onlooker bees $OP$, [25, 50, 100] for $limit$, [0.80, 0.85, 0.90] for $P_{bt}$, [0.40, 0.50, 0.60] for $P_r$, [0.85, 0.90, 0.95] for $P_{mpi}$, $[0.1 \times n, 0.2 \times n, 0.3 \times n]$ for $nrp$, $[0.1 \times n, 0.2 \times n, 0.3 \times n]$ for $nswp$, $[0.2 \times nop, 0.3 \times nop, 0.4 \times nop]$ for $nmc$. $nop$ is the total number of operations for each instance. In experimentation, the value of a single parameter is tested while keeping the values of other parameters fixed. After a large number of experimentation, it is observed that ABC_FJSPNW performs better in most of the cases when $EP = 100$, $OP = 200$, $limit = 50$, $P_{bt} = 0.85$, $P_r = 0.5$, $P_{mpi} = 0.9$, $nrp = 0.2 \times n$, $nswp = 0.2 \times n$, $nmc = 0.3 \times nop$. For each instance, ABC_FJSPNW has been executed 10 times (runs) with a different random seed. Total number of generations for each instance is $n \times m \times 30$.

**Table 1.** Results of ABC_FJSPNW on BRdata instances

| **Instance** | $n \times m$ | *nop* | **ABC_FJSPNW** | | | |
|---|---|---|---|---|---|---|
| | | | Value | Avrg. | SD | ATET |
| Mk01 | $10 \times 6$ | 55 | 45 | 46.90 | 0.83 | 2.26 |
| Mk02 | $10 \times 6$ | 58 | 36 | 37.50 | 0.67 | 2.37 |
| Mk03 | $15 \times 8$ | 150 | 238 | 250.30 | 6.07 | 22.45 |
| Mk04 | $15 \times 8$ | 90 | 78 | 80.40 | 1.56 | 8.06 |
| Mk05 | $15 \times 4$ | 106 | 247 | 251.20 | 2.93 | 7.83 |
| Mk06 | $10 \times 15$ | 150 | 110 | 120.20 | 4.02 | 26.01 |
| Mk07 | $20 \times 5$ | 100 | 186 | 189.40 | 2.69 | 10.96 |
| Mk08 | $20 \times 10$ | 225 | 795 | 837.50 | 22.92 | 108.25 |
| Mk09 | $20 \times 10$ | 240 | 620 | 640.00 | 11.63 | 100.67 |
| Mk10 | $20 \times 15$ | 240 | 430 | 445.90 | 16.80 | 135.09 |

Experimental results of each instance obtained by ABC_FJSPNW is reported in Table 1. In Table 1, Instance denotes the name of the instance, $(n, m)$ denotes the size of the instance, i.e., the number of jobs $n$ and the number of machines $m$, *nop* presents the total number of operations associated with the instance, Value denotes the best value obtained, Avrg. denotes the average value over 10 runs, SD denotes the standard deviation, ATET represents the average total execution time in second over 10 runs. The value of SD denotes the robustness of ABC_FJSPNW for most of the instances.

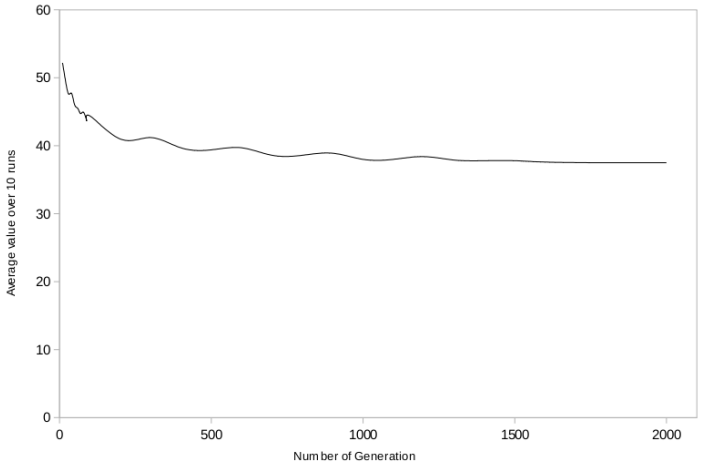


**Fig. 1.** Convergence behavior of ABC_FJSPNW for Mk02 instance

Fig. 1 depicts the convergence behavior of ABC_FJSPNW for the Mk02 instance. In this figure, average value over 10 runs according to the number of generations is considered. This figure shows the improvement of average solution quality of the Mk02 instance over the number of generations.

## 5 Conclusions

In this paper, an artificial bee colony (ABC) algorithm is proposed for a generalized problem called flexible job-shop scheduling problem with no-wait constraint (FJSPNW) combining the features of flexible job-shop scheduling problem and no-wait job-shop scheduling problem. The proposed approach has been tested on a set of standard benchmark instances for FJSPNW. Experimental results as well as convergence behavior of the proposed approach demonstrate its effectiveness.

To the best of our knowledge, there is no reported work for FJSPNW in the literature. So, as a future work, we intend to propose other metaheuristic techniques for this problem. The concepts used in this paper can also be applied to other scheduling problems.

**Acknowledgements.** This research is partially supported by Remanufacturing Scheduling Systems Program of A*Star in Singapore under Grant 112 290 4021.

## References

1. Garey, M.R., Johnson, D.S., Sethi, R.: The Complexity of Flow Shop and Job-shop Scheduling. Mathematics of Operations Research 1, 117–129 (1976)
2. Brucker, P., Schlie, R.: Job-shop Scheduling with Multi-purpose Machines. Computing 45, 369–375 (1990)
3. Brandimarte, P.: Routing and Scheduling in a Flexible Job Shop by Tabu Search. Annals of Operations Research 41, 157–183 (1993)
4. Mastrolilli, M., Gambardella, L.M.: Effective Neighborhood Functions for the Flexible Job Shop Problem. Journal of Scheduling 3, 3–20 (2000)
5. Gao, J., Sun, L., Gen, M.: A Hybrid Genetic and Variable Neighborhood Descent Algorithm for Flexible Job Shop Scheduling Problems. Computers & Operations Research 35, 2892–2907 (2008)
6. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of Machine Scheduling Problems. Annals of Discrete Mathematics 1, 343–362 (1977)
7. Sahni, S., Cho, Y.: Complexity of Scheduling Shops with No Wait in Process. Mathematics of Operations Research 4, 448–457 (1979)
8. Kamoun, H., Sriskandarajah, C.: The Complexity of Scheduling Jobs in Repetitive Manufacturing Systems. European Journal of Operational Research 70, 350–364 (1993)
9. Hall, N.J., Sriskandarajah, C.: A Survey on Machine Scheduling Problems with Blocking and No-wait in Process. Operations Research 44, 510–525 (1996)
10. Zhu, J., Li, X.: An Effective Meta-heuristic for No-wait Job Shops to Minimize Makespan. IEEE Transactions on Automation Science and Engineering 9, 189–198 (2012)

11. Karaboga, D.: An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report - TR06, Computer Engineering Department, Erciyes University, Turkey (2005)
12. Singh, A.: An Artificial Bee Colony Algorithm for the Leaf-Constrained Minimum Spanning Tree Problem. Applied Soft Computing 9, 625–631 (2009)
13. Pan, Q.-K., Tasgetiren, M.F., Suganthan, P.N., Chua, T.J.: A Discrete Artificial Bee Colony Algorithm for the Lot-Streaming Flow Shop Scheduling Problem. Information Sciences 181, 2455–2468 (2011)
14. Sundar, S., Singh, A.: A Swarm Intelligence Approach to the Quadratic Minimum Spanning Tree Problem. Information Sciences 180, 3182–3191 (2010)
15. Sundar, S., Singh, A.: A Swarm Intelligence Approach to the Quadratic Multiple Knapsack Problem. In: Wong, K.W., Mendis, B.S.U., Bouzerdoum, A. (eds.) ICONIP 2010, Part I. LNCS, vol. 6443, pp. 626–633. Springer, Heidelberg (2010)
16. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A Comprehensive Survey: Artificial Bee Colony (ABC) Algorithm and Applications. Artificial Intelligence Review (2012), doi:10.1007/s10462-012-9328-0
17. Sundar, S., Singh, A.: A Swarm Intelligence Approach to the Early/Tardy Scheduling Problem. Swarm and Evolutionary Computation 4, 25–32 (2012)
18. Brizuela, C.A., Zhao, Y., Sannomiya, N.: No-wait and Blocking Job-shops: Challenging Problems for GA's. In: Proceeding of IEEE International Conference on Systems, Man, and Cybernetics, pp. 2349–2354 (2001)