A ZESTFUL COMBINATION OF ABC WITH GA FOR QUenching of Job Shop Scheduling Problems

R. Rajkumar, A. Muthiah

Department of Mechanical Engineering, Mepco Schlenk Engineering College, Sivakasi, India.

Department of Mechanical Engineering, P.S.R. Engineering College, Sivakasi, India.

ABSTRACT

The job shop scheduling problem (JSSP) has attracted much attention in the field of both information sciences and operations research. This paper considers the permutation Job shop scheduling problem with the objective of minimizing make span. Artificial Bee Colony algorithm (ABC) is one of the search heuristics used to solve global optimization problems in complex search spaces. It is observed that, the efficiency of ABC in solving a Job shop problem can be improved significantly by tailoring another technique Genetic Algorithm (GA) to suit the structure of the problem. In this paper, an effective GA for rectifying Job shop scheduling problems is also proposed. The multi objective job shop scheduling problem can rectify and the performance will improve to an extreme by execution of proposed hybrid ABC-GA approach. The proposed technique will be implemented in the working platform of Matlab and the performance will be analyzed by comparing with the conventional methods for JSSP. Computational results based on some permutation Job shop scheduling show that the GA gives a better solution when compared with the earlier reported results.

KEYWORDS: Heuristic algorithms, make-to-order (MTO), Job-shop scheduling, roulette wheel selection, Total Weighted Tardiness (TWT).

1. INTRODUCTION

Job shop scheduling problem (JSSP) has been widely studied over the last four decades. Many researches involved job shop scheduling has been presented and various approaches are implemented to solve this problem [1]. Scheduling problems may be exist almost everywhere in real industrial world situations. Scheduling involves determination of the order of processing a set of tasks on resources or machines. Job shop scheduling problem (JSSP) involves an assignment of a set of tasks to the workstations (machines) in a predefined sequence in order to optimize one or more objectives considering job performances measures of the system [2]. A job shop environment consists of n job and each job has a given machine route in which some machines can be missed and some can repeat [3]. Various scheduling problems could be solved by various approaches such as Heuristic algorithms [4], Simulation [5], Particle Swarm Optimization (PSO) [6], and Genetic Algorithm [7], are implemented to solve job shop scheduling with sequence dependent setup times.

Generally speaking, scheduling problem is a form of decision-making, where limited resources are allocated to process a set of jobs with the aim to find a most efficient processing order under some given constraining objective functions [8]. As a branch of scheduling problem, a Job Shop Scheduling Problem (JSSP) is quite common in manufacturing and process industry, and the ordering of the jobs can be different for each machine [9]. In job shop scheduling problem (JSSP), the objective is to allocate resources in a way, such that a number of tasks can be completed cost-effectively within a given set of constraints [10]. JSSP is one of the most widely studied problems in computer science, which has great importance to manufacturing industries with the objective to minimize the production cost. The job shop scheduling problem (JSP) is one of most important and difficult problems in the field of production scheduling [11][12]. Production scheduling problems involve the allocation of scarce resources to different tasks in such way as to optimize one or more efficiency-related goals [13]. In most cases, these problems are analyzed as instances of the Job-Shop Scheduling Problem (JSSP), in which given a set of machines and a list of jobs, represented as ordered sequences of operations, to be run on the machines, the goal is to minimize, in particular, the processing time of all jobs, known as make span [14] [15]. Most research on JSSP has focused on the make span criterion (i.e., minimizing the maximum completion time). However, in the make-to-order (MTO) manufacturing environment, due date related performances are apparently more relevant for decision makers, because the in-time delivery of goods is vital for maintaining a high service reputation. Therefore, the research that aims at minimizing lateness/tardiness in JSSP deserves more attention [16] [17].

The flexible job shop scheduling problem (FJSP) is an extension of the classical JSP, in which operations are allowed to be processed by any machine from a given set, rather than one specified machine. Generally, the FJSP is much contiguous to a real production environment and has more practical applicability [18] [19]. However, the FJSP is more complex than the JSP because of its additional decision to assign each operation to the appropriate machine (routing) besides sequencing operations on machines. It has been proved that the FJSP is strongly NP-hard even if each job has at most three operations and there are two machines [20] [21].

Scheduling is the allocation of shared resources over time to competing activities. It has been the subject of a significant amount of literature in the operations research field. Emphasis has been on investigating machine scheduling problems where jobs represent activities and machines represent resources; each machine can process at most one job at a time. Which makes the optimization process more complicated [22].

The rest of this paper is organized as follows. Section 2 provides a brief review on the existing solution methods for JSSP and various scheduling algorithms. Section 3 describes the design of a new artificial bee colony algorithm together with Genetic Algorithm for quenching Job shop scheduling problems. Section 4 presents the computational results that compares the proposed work with various existing techniques to reveal the improved result. Finally, Section 5 concludes the paper.

2. RELATED WORK

Lin Lin et.al [23] proposed a reactive flexible job-shop scheduling problem (rFJSP) under uncertainty...
environment. The most existing reactive scheduling methods were characterized by least commitment strategies such as real-time dispatching that create partial schedules based on local information. In rFJSP, two extensions of these dispatching strategies are to allow the system to select multiple machines assignment, and multiple operation process for each job. So, how to design an effective flexible rescheduling strategy was the key point of this paper. For solving this rFJSP, we propose a hybrid evolutionary algorithm (hEA) with combining genetic algorithm (GA) and particle swarm optimization (PSO). Finally, the experiments verify the effectiveness of proposed hEA, by comparing with different evolutionary approaches for several scale test problems of rFJSP.

Wannaporn Teekeng et.al [24] illustrated a modified version of the genetic algorithm for flexible job-shop scheduling problems (FJSP). The genetic algorithm, a class of stochastic search algorithms, was very effective at finding optimal solutions to a wide variety of problems. The proposed modified GA consists of 1) an effective selection method called “fuzzy roulette wheel selection,” 2) a new crossover operator that used a hierarchical clustering concept to cluster the population in each generation, and 3) a new mutation operator that helped in maintaining population diversity and overcoming premature convergence. The objective of that research was to find a schedule that minimizes the makespan of the FJSP. The experimental results on 10 well-known benchmark instances show that the proposed algorithm was quite efficient in solving flexible job-shop scheduling problems.

Rui Zhang [25] proposed an artificial bee colony algorithm based on criticality information for solving job shop scheduling problems was proposed. The defined criticality values reflect the properties of both the objective function and the most crucial jobs at different stages of the optimization process. The criticality information was extracted and used as a local search optimizer to increase the convergence speed of the optimization process. Computational results show that the proposed algorithm was effective.

Rui Zhang et.al [26] illustrate to minimize the total weighted tardiness in JSSP. Considering the high complexity, a novel artificial bee colony (ABC) algorithm was proposed for solving the problem. A neighborhood property of the problem was discovered, and then a tree search algorithm was devised to enhance the exploitation capability of ABC. According to extensive computational tests, the proposed approach was efficient in solving the job shop scheduling problem with total weighted tardiness criterion.

Li-Ning Xing et.al [27] proposed a Knowledge-Based Ant Colony Optimization (KBACO) algorithm was proposed in this paper for the Flexible Job Shop Scheduling Problem (FJSP). KBACO algorithm provides an effective integration between Ant Colony Optimization (ACO) model and knowledge model. In the KBACO algorithm, knowledge model learned some available knowledge from the optimization of ACO, and then applied the existing knowledge to guide the current heuristic searching. The performance of KBACO was evaluated by a large range of benchmark instances taken from literature and some generated by ourselves. Final experimental results indicated that the proposed KBACO algorithm outperforms some current approaches in the quality of schedules.

Przemysław Korytkowski et.al [28] proposed a heuristic method based on ant colony optimization to determine the suboptimal allocation of dynamic multi-attribute dispatching rules to maximize job shop system performance (four measures were analyzed: mean flow time, max flow time, mean tardiness, and max tardiness). In order to assure high adequacy of the job shop system representation, modeling was carried out using discrete-event simulation. The proposed methodology constitutes a framework of integration of simulation and heuristic optimization. Simulation was used for evaluation of the local fitness function for ants. A case study was used in this paper to illustrate how performance of a job shop production system could be affected by dynamic multi attribute dispatching rule assignment.

3. A ZESTFUL COMBINATION OF ABC WITH GA FOR QUENCHING OF JOB SHOP SCHEDULING PROBLEMS

The job shop scheduling problem (JSSP) has been known as a very challenging combinatorial optimization problem. The proposed part of this work starts from here. In this paper a hybrid artificial bee colony algorithm along with Genetic algorithm is employed to meet the challenges in multi objective Job Scheduling Problems. Various multi objective problems need to be rectified by the proposed work are Minimization of maximum completion time (Total completion Time of a Job), Minimization of total workload (Total work load of all Machine), Minimization of critical machine workload (Work Loads of Busiest Machines), Operation precedence (Prioritized Job) and optimal machine selection (Choose best machine for Job allocation).

In general the artificial bee colony algorithm is executed in three phases, namely employed bee phase, onlooker bee phase and scouts bee phase. Employed bee phase is to perform exploitation of food source. Onlooker bees are waiting in the hive for making decision. Scout bees balance the exploration and exploitation capability of the algorithm. For this the scout bees are divided into two parts. The scout bees in one part perform randomly search in the predefined region while each scout bee in another part randomly select one non-dominated solution. In this proposed work the random searches of both the parts are employed by GA encapsulated in scout bee phase. Hence our proposed work will comprises of three phases they are employed bee phase, onlooker bee phase and Scout bee phase composed of GA. The multi objective job shop scheduling problem can rectify and the performance will improve to an extreme by execution of proposed hybrid ABC-GA approach. This proposed approach is shown in figure .1.
The JSSP is usually characterized as one in which a set of jobs is to be processed over a period of time, each job consisting of one or more operations to be performed in a specified sequence on specified machines and requiring some processing time. Also, there is a necessity to overcome the scheduling problems that describes the loss between the actual structure and the representation in a job shop scheduling. So, the above said reasons intended to do the proposed research. The proposed hybrid ABC-GA approach is executed as follows:

3.1 Artificial bee colony algorithm

The artificial bee colony (ABC) algorithm is a new swarm intelligence based optimizer. It mimics the cooperative foraging behavior of a swarm of honey bees. ABC is proposed here for optimizing multi-variable and multi-modal continuous functions. Especially, the number of control parameters in ABC is fewer than other population-based algorithms, which makes it easier to be implement. Meanwhile, the optimization performance of ABC is comparable and sometimes superior to the state-of-the-art metaheuristics. Therefore, ABC has aroused much interest and has been successfully applied to solve different kinds of optimization problems.

This ABC algorithm systematically incorporates exploration and exploitation mechanisms, so it is suitable for solving complex scheduling problems. However, due to its continuous feature, the traditional ABC algorithm cannot be directly applied to scheduling problems with inherent discrete nature. Indeed, in canonical ABC, each solution is represented by a vector of floating-point numbers. But for scheduling problems, each solution is naturally a permutation of integers. To address this issue, two kinds of approaches can be identified in the literature.

1. A transformation scheme is established to convert permutations into real numbers and vice versa. In this way, we only need to add a few lines to the encoding and decoding procedures, and it is not necessary to change the implementation of ABC itself.

2. The search operators in ABC are modified to suit the permutation representation. The redesign of these operators should be problem-dependent and thus requires a specific analysis of the optimization problem.

In this paper, we use ABC as the basic optimization framework for solving JSSP, and meanwhile we will combine the above two treatments in the hope of devising a more effective ABC. To our knowledge, this is the first attempt that ABC is applied to JSSP.

3.1.1 The ABC optimization framework

In the ABC algorithm, the artificial bees are classified into three groups: the employed bees, the onlookers and the scouts. A bee that is exploiting a food source is classified as employed. The employed bees share information with the onlooker bees, which are waiting in the hive and watching the dances of the employed bees. The onlooker bees will then choose a food source with probability proportional to the quality of that food source. Therefore, good food sources attract more bees than the bad ones. Scout bees search for new food sources randomly in the vicinity of the hive. When a scout or onlooker bee finds a food source, it becomes employed. The employed bees share information with the onlooker bees, which are waiting in the hive and watching the dances of the employed bees. The onlooker bees will then choose a food source with probability proportional to the quality of that food source. Therefore, good food sources attract more bees than the bad ones. Scout bees search for new food sources randomly in the vicinity of the hive. When a scout or onlooker bee finds a food source, it becomes employed. When a food source has been fully exploited, all the employed bees associated with it will abandon the position, and may become scouts again. Therefore, scout bees perform the job of “exploration”, whereas employed and onlooker bees perform the job...
of “exploitation”. The operations in scout bee are performed by utilizing Genetic Algorithm. Which facilitate the work of the scout bee phase more robust. In the proposed algorithm, a food source corresponds to a possible solution to the optimization problem, and the nectar amount of a food source corresponds to the fitness of the associated solution. In ABC, the first half of the colony consist of employed bees and the other half are onlookers. The number of employed bees is equal to the number of food sources (SN) because it is assumed that there is only one employed bee for each food source. Thus, the number of onlooker bees is also equal to the number of solutions under consideration. The ABC algorithm starts with a group of randomly generated food sources. The main procedure of ABC can be described as follows.

**Initialization phase**
Step 1: Initialize parameters, including the number of food sources or the number of employed bees, number of onlooker bees and number of scout bees.
Step 2: Initialize population of food sources with random solutions.

**Employed bee phase**
Step 3: Calculate the objective value of each food source and then determine the best food resource.
Step 4: For every employed bee, generate a new food source.
Step 5: Calculate the objective value for every new food source and compute the best food source.

**Onlooker bee phase**
Step 6: Calculate the probability of selecting food source
Step 7: Calculate the number of onlooker bees to be sent to the food source.
Step 8: Every onlooker bee, generate the new food sources.

**Scout bee phase**
Step 9: Initialize scout bees with random solutions and update the best food sources.
Step 10: Determine the worst employed bees and replace them with the scout bees if the scout bees are better.
Step 11: If a stopping criterion is met, then the best food source is obtained with its objective value.

**Flow Chart Representation of Working of ABC algorithm**
(1) **The initialization phase:** This is the starting phase of the ABC algorithm. The SN initial solutions are randomly generated D-dimensional real vectors.

\[ F_i = \{F_{i,1}, F_{i,2}, ..., F_{i,D} \} \]  

(1)

\( F_i \) represents the i-th food source, which is obtained by

\[ F_{i,d} = F_{d}^{\text{min}} + r \times (F_{d}^{\text{max}} - F_{d}^{\text{min}}) \]  

(2)

Where \( r \) is a uniform random number in the range \([0,1]\) and \( F_{d}^{\text{min}} \) and \( F_{d}^{\text{max}} \) are the lower and upper bounds for dimension \( d \) respectively \( d = 1, ..., D \).

(2) **The employed bee phase:** In this phase, each employed bee is associated with a solution. She exerts a random modification on the solution (original food source) for finding a new solution (new food source). This implements the function of neighborhood search. The new solution \( v_i \) is generated from \( F_i \) using a differential expression:

\[ S_{i,d} = F_{i,d} + r' \times (F_{k,d} - F_{i,d}) \]  

(3)

Where \( d \) is randomly selected from \( \{1, ..., D\} \) \( k \) is randomly selected from \( \{1, ..., SN\} \) such that, \( k \neq i \), and \( r' \) is a uniform random number in the range [-1, 1]. Once \( s_i \) is obtained, it will be evaluated and compared to. If the fitness of \( s_i \) is better than that of \( x_i \) (i.e., the nectar amount of the new food source is higher than the old one), the bee will forget the old solution and memorize the new one. Otherwise, she will keep working on \( x_i \).

(3) **The onlooker bee phase:** In this phase, when all employed bees have finished their local search, they share the nectar information of their food source with the onlookers, each of whom will then select a food source in a probabilistic manner. The probability \( P_{bi} \) by which an onlooker bee chooses food source \( x_i \) is calculated as follows:

\[ P_{bi} = \frac{f_i}{\sum_{i=1}^{SN} f_i} \]  

(4)

Where \( f_i \) is the fitness value of \( x_i \). Obviously, the onlooker bees tend to choose the food sources with higher nectar amount. Once the onlooker has selected a food source \( x_i \), she will also conduct a local search on \( x_i \) according to Equ. (3). As in the previous case, if the modified solution has a better fitness, the new solution will replace \( x_i \).

(4) **The scout bee phase:** In the scout bee phase of ABC, if the quality of a solution cannot be improved after a predetermined number (limit) of trials, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout will then produce a food source randomly by using Equ. (2).

3.1.2 **Framework of the Genetic Algorithm (GA) for Job shop scheduling**

The random production of food sources in scout bee is performed by GA. To enhance the performance of genetic searches and to avoid premature convergence, an GA is proposed with the following changes. In the GA, a single type of crossover operator is applied to the whole population from start to finish, which is not good for retaining useful information and maintaining diversity if the evolution tends to be premature. In GA, a set of crossover operators are used each with the given probabilities. Multiple crossover operators ensure that the diversity can be enhanced and the search region can be extended.

Similarly a set of mutation operators are applied in GA. This will enrich the search template so that the exploration and exploitation abilities can be simultaneously enhanced on the basis of the advantage of combining several different search mechanisms. Computation shows that when the best solution remains unimproved for a certain number of generations in the GA process, the solution quality will be difficult to be improved further even if the generation continues. Therefore, in this GA the following changes are incorporated to avoid premature convergence:

(a) **Elitist strategy:** Remove the worst chromosome from the current population and add the best chromosome into that population.

(b) **Hyper mutation:** When the number of generations without improving the best solution is greater than a pre-specified constant (Gm), premature convergence can be assumed then increase the probability of mutation (hyper mutation), and continue the search. The purpose of increasing the probability is to diversify the population of GA.

(c) **Re-Assign(Insert):** New randomly generated population if make span is not converging for the generation Gp.

Thus, we provide a framework for the GA. GA preserves the generality of GA and can be easily implemented and applied to any kind of optimization problems.

**Genetic algorithm for Job scheduling**

Step 1. Initialize the population as input number of processors, number of jobs.

Step2. A process started

Step2.1 Evaluate the fitness function (makespan).

Step2.2 Perform selection to select the best individuals from the current population.

Step2.3 Perform two-point crossover. Choose pairs of chromosomes (task). Choose a random point exchange machine assignment from that point until the end of the chromosome.

Step2.4. Mutation: Randomly select a task. Randomly, reassign it to the new machine.

Step 3. The process is repeated until the stopping criterion is met. (Best fitness, minimum completion time)

Step 4. Stop
3.1.3 Steps of GA in Job shop scheduling

Step 0: Given the parameters required, such as population size $P_S$, crossover probability $P_c$, mutation probability $P_m$, etc., set $k=1$ and randomly generate an initial population of size $P_{c-1}$. 

$$P(k)=\{F_1(k), F_2(k), \ldots, F_{P_c}(k)\} \quad (5)$$

Step 1: Evaluate Make span for all the chromosomes in $P(k)$, and set the best two chromosomes $'F'$ and $'F''$, respectively.

Step 2: Set current best make span as $M_k$

Step 3: If $M_k = M_{k-1}$, then set countmut=countmut+1 and.

$$cntrndpop=cntrndpop+1$$

Else set countmut=0 and 

$$cntrndpop=0$$

Step 4: Set $q=0$

Step 5: Generate a random number $r$. If $r \leq p$ to, select two parents $F_1$ and $F_2$ from $P(k)$ by tournament selection, else select the parents by roulette wheel selection.

Step 6: Generate $r$. If $r \leq P_c$, go to step 6.1 else go to step 6.2.

6.1. Perform any one of the following crossover operations.

6.1.1. Generate $r$. If $r \leq P_t$, perform two-point crossover.

6.1.2. If $P_t < r (P_t + P_m)$ perform Partially Mapped crossover (PMX).

6.1.3. If $P_t < r (P_t + P_m)$ perform Similar Job Order crossover (SJOX).

6.1.4. Else perform Linearly Ordered Crossover (LOX).

Let $F_1'$ and $F_2'$ be the offsprings of crossoversing parents $F_1$ and $F_2$. 

Go to step 7.

6.2. Let $F_1'$ and $F_2'=F_2$. 

DisplayOptimalShedule
Step 7: Generate \( r \). If \( r \leq P_m \), go to step 7.1 else go to step 7.2.

7.1. Perform any one of the following mutation operations.

7.1.1. Generate \( r \). If \( r \leq P_j \), perform arbitrary three-job change mutation.

7.1.2. If \( P_j < r \leq (P_j + P_j) \), perform arbitrary two-job change mutation.

7.1.3. Else perform Shift change mutation.

Mutate for \( F'_1 \), to generate chromosome \( F'_1 \) and \( F'_2 \) to generate chromosome \( F'_2 \). Then, put \( F'_1 \) and \( F'_2 \) into \( P(k+1) \) and let \( q = q+1 \). Go to step 8.

7.2. Let \( X'_i = X_i \) and \( X'_2 = X'_2 \). Then, put \( X'_1 \) and \( X'_2 \) into \( P(k+1) \) and let \( q = q+1 \).

Step 8: If \( q < P_j \), then go to step 5; otherwise, go to step 9.

Step 9: If \( \text{countmut} > G_m \), then change the mutation probability \( P_b = \Psi * P_b \), else no change in \( P_m \) value.

Step 10: If \( \text{cntrndpop} \geq G_n \), then generate new population of size \( 0.75 \times P_b \times 0.75 \times P_s \) randomly and replace the current population.

Step 11: Update \( F''_1 \), \( F''_2 \) and \( M_i \) in \( P(k) \).

Step 12: Adopt Elitist Strategy. Insert the two best chromosomes \( F''_1 \) and \( F''_2 \) into the current population by removing two worst chromosomes (having max. make span).

Step 13: If \( k > N_s \), go to step 14. Else set \( k = k + 1 \) and go to step 3.

Step 14: Output make span value and the corresponding sequence as the result.

Stop.

3.1.4 The Adaptive ABC-GA algorithm for Job scheduling

Initialization phase

Step 1: Initialize input parameters, including the number of food sources or the number of employed bees, number of onlooker bees and number of scout bees.

Step 2: Initialize populations by generating random food sources for each food source.

Step 3: Calculate the fitness value \( F(\text{food source}) \) of each food source and then determine the best food source.

Employed bee phase

Step 4: For every employed bee, generate a new food source \( N_f(\text{food source}) \) by using the crossover and mutation operations.

Step 5: Calculate fitness value \( F(\text{food source}) \) for every newly generated food source and compute the best food source.

Onlooker bee phase

Step 6: For every onlooker bee, generate a new food source \( N_f(\text{food source}) \) by using the crossover and mutation operations.

Step 7: Calculate fitness value \( F(\text{food source}) \) for every newly generated food source and compute the best food source.

Scout bee phase

Step 9: Initialize scout bees with random solutions and compute fitness value \( F(\text{food source}) \) for these random solutions.

Step 10: Find the best scout bee among the randomly generated food sources using the fitness value \( F(\text{food source}) \).

Step 11: The scout bee’s best food source \( B(\text{food source}) \), the employed bee’s best food source \( B(\text{food source}) \) and the onlooker bee’s best food source \( B(\text{food source}) \) are compared based on their fitness values using GA.

Step 12: Among these food sources, the best food source is stored in the scout bee’s phase and remaining food sources are given to the next iteration.

Step 13: The process is repeated until the stopping criterion is met. Then, the best food source is obtained with its objective value from the scout bee’s phase.

Detailed implementation of the ABC-GA Algorithm

Step 1: Initialize the population of solutions \( F_i \).

Step 2: Evaluate the population.

Step 3: \( \text{cycle} = 1 \).

Step 4: repeat

Step 5: Produce new solutions (food source positions) \( S_i,j \) in the neighborhood of \( F_i \) for the employed bees using the formula \( S_i,j = F_i,j + \Phi_i j (F_i,j - F_k,j) \) \( k \) is a solution in the neighborhood of \( i \), \( \Phi \) is a random number in the range \([-1,1]\) and evaluate them.

Step 6: Apply the greedy selection process between \( F_i \) and \( S_i \).

Step 7: Calculate the probability selection process for the solutions \( x_i \) by means of their fitness values using the equation (1)

\[
P_i = \frac{f_{i}}{\sum f_{i}}
\]

In order to calculate fitness values of \( t \) solutions we employed the following equation (eq. 2):

\[
f_{i}^p = \begin{cases} 1 & \text{if } f_i \geq 0 \\ \frac{1}{1 + \text{abs}(f_i)} & \text{if } f_i < 0 \end{cases}
\]

Normalize \( P_i \) values into \([0,1]\)

Step 8: Produce the new solutions (new positions) \( S_i \) for the onlookers from the solutions \( F_i \), selected depending on \( P_i \), and evaluate them.

Step 9: Apply the greedy selection process for the onlookers between \( F_i \) and \( S_i \).

Step 10: Determine the abandoned solution (source), if exists, and replace it with a new randomly produced solution \( F_i \) for the scout using the equation (3)

\[
F_{i,j} = \text{min} + \text{rand}(0,1) \times (\text{max} - \text{min})
\]

Step 11: Memorize the best food source position (solution) achieved by GA.

Step 12: \( \text{cycle} = \text{cycle} + 1 \).

Step 13: until \( \text{cycle} = \) Maximum Cycle Number (MCN)
Example
The 5*5 Job instance, made up of 25 operations. The number of possible schedules S (solutions) can be calculated with Eq. (1).

\[ S = (n!)^m \]  

Where, \( n \) = Total number of jobs  
\( m \) = Total number of machines

3.2 Implementation of the GA for Job shop scheduling
In the Genetic algorithm it executes the fitness evaluation of various jobs from the ABC algorithm are further evaluated for Fitness determination. The Evaluation of fitness of various jobs are as follows:

3.2.1 Solutions encoding and population initialization. A job-permutation-based encoding scheme has been widely used by many authors for permutation job shop scheduling. Hence similar encoding scheme is adopted in this work. For example, in a seven-job problem a chromosome could be \([4751632]\). This represents the sequence where job 4 gets processed first on all the machines followed by job 7 and so on. The permutation has to be feasible, i.e. there cannot be neither missing nor repetition of jobs.

Traditionally, in GA the initial population is generated at random. However random initialization of the population may not give good results. Remaining chromosomes of the initial population are generated at random.

3.2.2 Fitness evaluation function. In order to mimic the natural process of the survival of the fittest, the fitness evaluation function assigns to each member of the population a value reflecting their relative superiority (or inferiority). Each chromosome has an evaluation criterion based on the objective function.

Since GA is used for maximization problems, a minimization problem can be suitably converted into a maximization problem using a fitness function. The fitness function is:

\[ \text{Fit} = \frac{1}{C_{\text{max}}} \]  

Where \( C_{\text{max}} \) is the makespan, which has to be minimized.

3.2.3 Selection scheme. Selection scheme specifies the methodology employed to select the chromosome from the current population for regeneration. In GA, two classical selection schemes, namely roulette wheel selection and tournament selection are employed. In roulette wheel selection, parents are selected according to their fitness value. The better the fitness, the more chances to be selected. The following procedure is used for selection.

Chromosome \( F_i \) is selected if

\[ \sum_{j=1}^{i-1} f(F_j) < r \leq \sum_{j=1}^{i} f(F_j) \]  

Where \( f(F_j) \) is the fitness value of chromosome and \( r \) is the random number between 0 and 1. In Tournament Selection, predetermined numbers of chromosomes are randomly selected from the population and the chromosome with the best fitness value is considered to be regenerated. Here selection is based on a competition within a subset of the population.

3.2.4 Crossover. Many different general and specific crossover operators have been proposed in the literature. Since the sequence is a permutation of elements there should be neither repeated nor missing elements to maintain feasibility. In this work, the
following types of crossover operators are used, which are described as follows. A probability is attached to each type of crossover and each time any one type is selected using Monte-Carlo simulation.

1. **Two point crossover**
   - **Step 1:** Two points are randomly selected for dividing the parents (fig. 2).

   ![Figure 2](image)
   **Figure 2 Two point crossover**
   - **Step 2:** The jobs outside the selected two points are directly inherited from the parent to the child.
   - **Step 3:** The remaining elements in the child are filled by scanning the other parent from left to right and entering the elements not already present.

3.2.5 **Mutation**. Mutation generates an offspring solution by randomly modifying the parent’s feature. It helps to preserve a reasonable level of population diversity, and provides a mechanism to escape from local optima. For each child obtained from crossover, the mutation operator is applied independently with a probability pm. In this work, three types of mutation operators are used, which are described as follows. A probability is attached to each type of mutation and each time any one type is selected using Monte-Carlo simulation.

1. Arbitrary three-job change
   - The three jobs are randomly selected, and replaced at random locations amongst the selected jobs (figure 3).

2. Arbitrary two-job change
   - Two jobs are selected at random and their positions are interchanged as shown below (figure 4).

3. Shift change
   - In this type of mutation, a job at a random position is removed and inserted at another random position as shown below (figure 5).

3.2.6 **Restart scheme**. Finally, to avoid premature convergence and to escape local optimum, two counters called `countmut` and `cntrndpop` are introduced as explained below.

(I) (a) Let $M_K =$ current generation’s best make span and $M_{K-1}$ previous generation best make span.

If $M_K = M_{K-1}$, then, set `countmut = countmut + 1` and `cntrndpop = cntrndpop + 1`. Else set `countmut = 0` and `cntrndpop = 0`.

**TABLE 1: OVERALL PROCESS SEQUENCING OUTPUT FOR 3*3 OBTAINED FROM GA WITH MATLAB SIMULATION METHOD**

<table>
<thead>
<tr>
<th>Processing sequence for possible schedule</th>
<th>Job processing with respect to their machines and processing times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(J3 M1 74)</td>
</tr>
<tr>
<td>2</td>
<td>(J1 M2 71)</td>
</tr>
<tr>
<td>3</td>
<td>(J1 M1 32)</td>
</tr>
<tr>
<td>4</td>
<td>(J2 M3 85)</td>
</tr>
<tr>
<td>5</td>
<td>(J1 M1 158)</td>
</tr>
<tr>
<td>6</td>
<td>(J3 M2 159)</td>
</tr>
<tr>
<td>7</td>
<td>(J3 M3 130)</td>
</tr>
<tr>
<td>8</td>
<td>(J2 M2 193)</td>
</tr>
<tr>
<td>9</td>
<td>(J1 M3 218)</td>
</tr>
</tbody>
</table>

If there is no improvement in the best solution so far for the pre-specified generations $G_m$, then increase the probability of mutation and continue the search. The purpose of increasing the probability is to diversify the population of GA i.e. dynamically change the mutation rate if makespan not converging for the generation $G_m$ i.e. if `countmut > G_m`, then, $P_m = \Psi \ast P_m$.

(II) Re-Assign (i.e. replace) new randomly generated chromosomes into population if make span not converging for the generation $G_p$ i.e. if `cntrndpop > G_p`, then, replace 75% of current population by randomly generated new chromosomes.

4. **GENERATION OF TEST INSTANCES**

4.1. The test problems and parameter setting

Here the comparison for the performance of the proposed ABC-GA with the existing ABC and genetic algorithm for TWT-JSSP. Here, GA is chosen based on two reasons. First, GA is the one of the most powerful meta-heuristics for solving TWT-JSSP up till now. Second, the structure of GA is very similar and thus comparable to our ABC-GA, that is, both of them rely on a population-based exploration integrated with a local search-based exploitation. The algorithms have been implemented and tested in the Matlab platform.

4.2. The results and discussions

With respect to Group (I), here a computational time limit of 30s is imposed on both algorithms in order to make the comparisons meaningful. The average results obtained by ABC-GA, ABC and GA from 10 independent runs, respectively, for each instance are listed in Table 1.
According to the table, ABC-GA out performs ABC and GA in a statistically significant manner for 19 out of the 66 instances. For 28 instances, ABC obtains exactly the same results as GA (both algorithms are able to reach the optimum in all the 10 runs). For another 13 instances, ABC obtains better average results than GA but the difference is statistically insignificant. For only six instances, ABC is out performed by GA in terms of the average solution quality, but the difference is ABC-GA out performs a significant result in all the cases.

Table. 2 set a computational time limit for both algorithms: \((0.4 \times n \times m)\) for solving an instance of size \(n \times m\). Considering the large amount of data, the computational results are processed in the following way before listed in the tables. For each instance \(i\), ABC-GA, ABC and GA are, respectively, run for 10 independent times. The best objective value obtained in the 10 runs by algorithm \(d_i(A \in \{ABC, GLS\})\) is denoted as \(\text{TWT}_i(a)\), the worst denoted as \(\text{TWT}_w(a)\) and the average value denoted as \(\text{TWT}_{av}(a)\).

TWT-JSSP instances, which are randomly generated in the following way. For a specific problem size, the processing route of each job is a random permutation of the \(m\) machines, and the required processing time of each operation follows a uniform distribution \(U(1,99)\) and takes only integer values. The due date of each job is set based on the total processing time of the job as

\[
d_j = u \times f \times \sum_{i=1}^{m} P_i\]

(8)

In this expression, \(u \sim U[1,1.1\times \max(1,n/m)]\) is a random number uniformly distributed in the interval \([1,1.1 \times \max(1,n/m)]\), and \(O_j\) denotes the set of operations that constitute job \(j\).

Next, we can calculate the relative objective values (\(\text{RTWT}\)) by taking \(\text{TWT}_i(ABC)\) as reference:

\[
\text{RTWT}_i^j (a) = \frac{\text{TWT}_w(a)}{\text{TWT}_i(a)} (ABC) \quad (9)
\]

\[
\text{RTWT}_{av}^j (a) = \frac{\text{TWT}_{av}(a)}{\text{TWT}_i(a)} (ABC) \quad (10)
\]

Finally, when the above steps have been performed for each instance in the considered instance set, we calculate the average relative values (\(\overline{\text{RTWT}}\)) over this set as

\[
\overline{\text{RTWT}}_i(a) = \frac{1}{5} \sum_{i=1}^{5} \text{RTWT}_i(a) \quad (12)
\]

\[
\overline{\text{RTWT}}_{w}(a) = \frac{1}{5} \sum_{i=1}^{5} \text{RTWT}_w(a) \quad (13)
\]

\[
\overline{\text{RTWT}}_{av}(a) = \frac{1}{5} \sum_{i=1}^{5} \text{RTWT}_{av}(a)\]

(14)

where \(\text{RTWT}_i^j (a)\) - relative objective values of total weighted tardiness in best case

\(\text{RTWT}_{av}(a)\) - average relative values of total weighted tardiness

\(\text{RTWT}_i^j (a)\) - relative objective values of total weighted tardiness in worst case

<table>
<thead>
<tr>
<th>Size (n \times m)</th>
<th>ABC</th>
<th>GA</th>
<th>ABC-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10 \times 10)</td>
<td>1.000</td>
<td>1.250</td>
<td>1.104</td>
</tr>
<tr>
<td>(20 \times 5)</td>
<td>1.000</td>
<td>1.212</td>
<td>1.113</td>
</tr>
<tr>
<td>(10 \times 20)</td>
<td>1.000</td>
<td>1.273</td>
<td>1.128</td>
</tr>
<tr>
<td>(20 \times 10)</td>
<td>1.000</td>
<td>1.255</td>
<td>1.124</td>
</tr>
<tr>
<td>(20 \times 15)</td>
<td>1.000</td>
<td>1.243</td>
<td>1.161</td>
</tr>
<tr>
<td>(50 \times 6)</td>
<td>1.000</td>
<td>1.209</td>
<td>1.141</td>
</tr>
<tr>
<td>(20 \times 20)</td>
<td>1.000</td>
<td>1.213</td>
<td>1.127</td>
</tr>
<tr>
<td>(40 \times 10)</td>
<td>1.000</td>
<td>1.173</td>
<td>1.117</td>
</tr>
<tr>
<td>(50 \times 10)</td>
<td>1.000</td>
<td>1.212</td>
<td>1.151</td>
</tr>
<tr>
<td>(100 \times 5)</td>
<td>1.000</td>
<td>1.257</td>
<td>1.184</td>
</tr>
</tbody>
</table>

Table 2: The Computational Results for TWT-JSSPInstance.

Figure 6: The relative performance of the three algorithms on each problem size

In this way, the computational results are summarized in Tables 6–8 with respect to each instance set. For most instances, the gap between the best and the worst solutions obtained by ABC-GA is smaller than the gap obtained by ABC and GA, which implies that ABC-GA performs more robustly in different executions and for different scheduling instances. Additionally, in terms of the best performance in the 10 runs, ABC-GA almost always finds a better solution than ABC and GA, which validates the effectiveness of the proposed approach.

When the problem size expands, the improvement of ABC-GA over ABC and GA tends to be more remarkable. To observe the trend more clearly, here define the performance ratio as the ratio between the average objective value of ABC and GA over the average objective value of ABC-GA for each problem size. The result is shown as a bar graph in Fig. 6 (with a trend line). This phenomenon also verifies that the local search module in ABC-GA is more efficient than the one in ABC and GA when faced with a large search space.

4.3 Application of ABC-GA for minimizing make span
In this section, the proposed ABC-GA is used to solve 13 instances of JSSP with the make span criterion (i.e., \( \| \sum_{i=1}^{n} x_{ij} \|^2 \)). The aim is to demonstrate that ABC-GA can yield high-quality solutions under the make span criterion although it has not been tailored to this objective. For this purpose, compare ABC-GA with a recent approach specifically developed for \( \| \sum_{i=1}^{n} x_{ij} \|^2 \), i.e., the Genetic Algorithm.

The parameters of GA are all set according to the original paper (reference set size: 20×2; cross over probability: 0.9; local search stopping criterion: 500 solution evaluations). The average computational results are given in Table 9, where “n” indicates the cases where ABC has achieved the optimal solution of the problem in all the 10 runs.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size n×m</th>
<th>ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT06</td>
<td>6x6</td>
<td>56</td>
</tr>
<tr>
<td>FT10</td>
<td>10x10</td>
<td>933</td>
</tr>
<tr>
<td>FT20</td>
<td>20x5</td>
<td>1167</td>
</tr>
<tr>
<td>LA01</td>
<td>10x5</td>
<td>672</td>
</tr>
<tr>
<td>LA02</td>
<td>10x5</td>
<td>655</td>
</tr>
<tr>
<td>LA03</td>
<td>10x5</td>
<td>610</td>
</tr>
<tr>
<td>LA09</td>
<td>15x5</td>
<td>955</td>
</tr>
<tr>
<td>LA10</td>
<td>15x5</td>
<td>960</td>
</tr>
<tr>
<td>LA15</td>
<td>20x5</td>
<td>1210</td>
</tr>
<tr>
<td>LA19</td>
<td>10x10</td>
<td>842</td>
</tr>
<tr>
<td>LA21</td>
<td>15x10</td>
<td>1049</td>
</tr>
<tr>
<td>LA24</td>
<td>15x10</td>
<td>940</td>
</tr>
<tr>
<td>LA25</td>
<td>15x10</td>
<td>982</td>
</tr>
<tr>
<td>LA27</td>
<td>20x10</td>
<td>1243</td>
</tr>
<tr>
<td>LA29</td>
<td>20x10</td>
<td>1180</td>
</tr>
<tr>
<td>LA36</td>
<td>15x15</td>
<td>1274</td>
</tr>
<tr>
<td>LA37</td>
<td>15x15</td>
<td>1408</td>
</tr>
<tr>
<td>LA38</td>
<td>15x15</td>
<td>1196</td>
</tr>
<tr>
<td>LA39</td>
<td>15x15</td>
<td>1238</td>
</tr>
<tr>
<td>LA40</td>
<td>15x15</td>
<td>1233</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size n×m</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT06</td>
<td>6x6</td>
<td>58</td>
</tr>
<tr>
<td>FT10</td>
<td>10x10</td>
<td>935</td>
</tr>
<tr>
<td>FT20</td>
<td>20x5</td>
<td>1170</td>
</tr>
<tr>
<td>LA01</td>
<td>10x5</td>
<td>685</td>
</tr>
<tr>
<td>LA02</td>
<td>10x5</td>
<td>655</td>
</tr>
<tr>
<td>LA03</td>
<td>10x5</td>
<td>621</td>
</tr>
<tr>
<td>LA09</td>
<td>15x5</td>
<td>960</td>
</tr>
<tr>
<td>LA10</td>
<td>15x5</td>
<td>980</td>
</tr>
<tr>
<td>LA15</td>
<td>20x5</td>
<td>1229</td>
</tr>
<tr>
<td>LA19</td>
<td>10x10</td>
<td>842</td>
</tr>
<tr>
<td>LA21</td>
<td>15x10</td>
<td>1050</td>
</tr>
<tr>
<td>LA24</td>
<td>15x10</td>
<td>944</td>
</tr>
<tr>
<td>LA25</td>
<td>15x10</td>
<td>984</td>
</tr>
<tr>
<td>LA27</td>
<td>20x10</td>
<td>1241</td>
</tr>
<tr>
<td>LA29</td>
<td>20x10</td>
<td>1193</td>
</tr>
<tr>
<td>LA36</td>
<td>15x15</td>
<td>1292</td>
</tr>
<tr>
<td>LA37</td>
<td>15x15</td>
<td>1424</td>
</tr>
<tr>
<td>LA38</td>
<td>15x15</td>
<td>1220</td>
</tr>
</tbody>
</table>
Table 5: The computational results of make span in Artificial Bee Colony algorithm Genetic Algorithm (ABC-GA)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size n × m</th>
<th>ABC-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT06</td>
<td>6 × 6</td>
<td>53</td>
</tr>
<tr>
<td>FT10</td>
<td>10 × 10</td>
<td>927</td>
</tr>
<tr>
<td>FT20</td>
<td>20 × 5</td>
<td>1163</td>
</tr>
<tr>
<td>LA01</td>
<td>10 × 5</td>
<td>663</td>
</tr>
<tr>
<td>LA02</td>
<td>10 × 5</td>
<td>653</td>
</tr>
<tr>
<td>LA03</td>
<td>10 × 5</td>
<td>592</td>
</tr>
<tr>
<td>LA09</td>
<td>15 × 5</td>
<td>950</td>
</tr>
<tr>
<td>LA10</td>
<td>15 × 5</td>
<td>953</td>
</tr>
<tr>
<td>LA15</td>
<td>20 × 5</td>
<td>1201</td>
</tr>
<tr>
<td>LA19</td>
<td>10 × 10</td>
<td>840</td>
</tr>
<tr>
<td>LA21</td>
<td>15 × 10</td>
<td>1045</td>
</tr>
<tr>
<td>LA24</td>
<td>15 × 10</td>
<td>932</td>
</tr>
<tr>
<td>LA25</td>
<td>15 × 10</td>
<td>976</td>
</tr>
<tr>
<td>LA27</td>
<td>20 × 10</td>
<td>1233</td>
</tr>
<tr>
<td>LA29</td>
<td>20 × 10</td>
<td>1149</td>
</tr>
<tr>
<td>LA36</td>
<td>15 × 15</td>
<td>1265</td>
</tr>
<tr>
<td>LA37</td>
<td>15 × 15</td>
<td>1391</td>
</tr>
<tr>
<td>LA38</td>
<td>15 × 15</td>
<td>1182</td>
</tr>
<tr>
<td>LA39</td>
<td>15 × 15</td>
<td>1226</td>
</tr>
<tr>
<td>LA40</td>
<td>15 × 15</td>
<td>1220</td>
</tr>
</tbody>
</table>

Table 6: The computational results for minimizing Cmak with Best known solution.

<table>
<thead>
<tr>
<th>SL No</th>
<th>Problem</th>
<th>Size n × m</th>
<th>BKS</th>
<th>ABC</th>
<th>GA</th>
<th>ABC-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FT06</td>
<td>6 × 6</td>
<td>55</td>
<td>56</td>
<td>58</td>
<td>53</td>
</tr>
<tr>
<td>2</td>
<td>FT10</td>
<td>10 × 10</td>
<td>930</td>
<td>933</td>
<td>935</td>
<td>927</td>
</tr>
<tr>
<td>3</td>
<td>FT20</td>
<td>20 × 5</td>
<td>1165</td>
<td>1167</td>
<td>1170</td>
<td>1163</td>
</tr>
<tr>
<td>4</td>
<td>LA01</td>
<td>10 × 5</td>
<td>666</td>
<td>672</td>
<td>685</td>
<td>663</td>
</tr>
<tr>
<td>5</td>
<td>LA02</td>
<td>10 × 5</td>
<td>655</td>
<td>655</td>
<td>655</td>
<td>655</td>
</tr>
<tr>
<td>6</td>
<td>LA03</td>
<td>10 × 5</td>
<td>597</td>
<td>610</td>
<td>621</td>
<td>592</td>
</tr>
<tr>
<td>7</td>
<td>LA09</td>
<td>15 × 5</td>
<td>951</td>
<td>955</td>
<td>960</td>
<td>950</td>
</tr>
<tr>
<td>8</td>
<td>LA10</td>
<td>15 × 5</td>
<td>958</td>
<td>960</td>
<td>980</td>
<td>953</td>
</tr>
<tr>
<td>9</td>
<td>LA15</td>
<td>20 × 5</td>
<td>1207</td>
<td>1210</td>
<td>1229</td>
<td>1201</td>
</tr>
<tr>
<td>10</td>
<td>LA19</td>
<td>10 × 10</td>
<td>842</td>
<td>842</td>
<td>842</td>
<td>840</td>
</tr>
<tr>
<td>11</td>
<td>LA21</td>
<td>15 × 10</td>
<td>1046</td>
<td>1049</td>
<td>1050</td>
<td>1045</td>
</tr>
<tr>
<td>12</td>
<td>LA24</td>
<td>15 × 10</td>
<td>935</td>
<td>940</td>
<td>944</td>
<td>932</td>
</tr>
<tr>
<td>13</td>
<td>LA25</td>
<td>15 × 10</td>
<td>977</td>
<td>982</td>
<td>984</td>
<td>976</td>
</tr>
<tr>
<td>14</td>
<td>LA27</td>
<td>20 × 10</td>
<td>1235</td>
<td>1243</td>
<td>1241</td>
<td>1233</td>
</tr>
<tr>
<td>15</td>
<td>LA29</td>
<td>20 × 10</td>
<td>1153</td>
<td>1180</td>
<td>1193</td>
<td>1149</td>
</tr>
<tr>
<td>16</td>
<td>LA36</td>
<td>15 × 15</td>
<td>1268</td>
<td>1274</td>
<td>1292</td>
<td>1265</td>
</tr>
<tr>
<td>17</td>
<td>LA37</td>
<td>15 × 15</td>
<td>1397</td>
<td>1408</td>
<td>1424</td>
<td>1391</td>
</tr>
<tr>
<td>18</td>
<td>LA38</td>
<td>15 × 15</td>
<td>1196</td>
<td>1196</td>
<td>1220</td>
<td>1182</td>
</tr>
<tr>
<td>19</td>
<td>LA39</td>
<td>15 × 15</td>
<td>1233</td>
<td>1238</td>
<td>1250</td>
<td>1226</td>
</tr>
<tr>
<td>20</td>
<td>LA40</td>
<td>15 × 15</td>
<td>1222</td>
<td>1233</td>
<td>1250</td>
<td>1220</td>
</tr>
</tbody>
</table>

Figure 7: The computational results for minimizing Cmak with Best known solution.

According to the table, ABC-GA out performs ABC and GA in a statistically significant manner for seven out of the 13 instances. For another two instances, ABC-GA, ABC and GA obtain exactly the same...
results. For three instances, the difference between the three algorithms is observable but not significant. For only one instance (MT10), ABC-GA is out performed by ABC and GA in a statistically significant manner. The overall results show that ABC-GA is still competitive with the dedicated algorithm for $jm|\text{mak}|\text{tard}$ although it is not specifically designed for make span minimization in JSSP. But also it shows better results for minimizing make span by tackle all the JSS problems.

5. CONCLUSION

In this paper, a hybrid ABC-GA approach is proposed to allocate the available jobs to the exact resources. The proposed method has overcome the problems achieved by Job shop scheduling. The drawbacks of existing techniques were solved by considering some efficient factors in job scheduling process. Thus, the proposed technique has achieved high performance in allocating the available jobs to the precise resources and also attained a high efficiency. The performance of the proposed job scheduling technique was analyzed with two hybrid techniques namely ABC and GA with experimental results that proves, that the proposed job scheduling technique has attained high accuracy and efficiency than the existing techniques. Hence, the proposed hybrid ABC-GA approach for job scheduling technique is capable of finding the optimal jobs to the resources and also achieving the minimum completion time.

REFERENCE