Hindawi Publishing Corporation
Journal of Optimization
Volume 2016, Article ID 7319036, 13 pages
http://dx.doi.org/10.1155/2016/7319036

Research Article

A Hybrid Genetic Algorithm with a Knowledge-Based Operator for Solving the Job Shop Scheduling Problems

Hamed Piroozfard, Kuan Yew Wong, and Adnan Hassan

Department of Manufacturing and Industrial Engineering, Faculty of Mechanical Engineering, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor, Malaysia

Correspondence should be addressed to Hamed Piroozfard; phamed2@live.utm.my

Received 4 December 2015; Accepted 7 March 2016

Academic Editor: Jein-Shan Chen

Scheduling is considered as an important topic in production management and combinatorial optimization in which it ubiquitously exists in most of the real-world applications. The attempts of finding optimal or near optimal solutions for the job shop scheduling problems are deemed important, because they are characterized as highly complex and NP-hard problems. This paper describes the development of a hybrid genetic algorithm for solving the nonpreemptive job shop scheduling problems with the objective of minimizing makespan. In order to solve the presented problem more effectively, an operation-based representation was used to enable the construction of feasible schedules. In addition, a new knowledge-based operator was designed based on the problem's characteristics in order to use machines' idle times to improve the solution quality, and it was developed in the context of function evaluation. A machine based precedence preserving order-based crossover was proposed to generate the offspring. Furthermore, a simulated annealing based neighborhood search technique was used to improve the local exploitation ability of the algorithm and to increase its population diversity. In order to prove the efficiency and effectiveness of the proposed algorithm, numerous benchmarked instances were collected from the Operations Research Library. Computational results of the proposed hybrid genetic algorithm demonstrate its effectiveness.

1. Introduction

Scheduling is one of the most important topics that ubiquitously exist in many real-world applications. Scheduling is assigning a set of tasks on resources to be performed during a time period, considering the time, capability, and capacity constraints [1]. The main focus is to improve the production efficiency and utilization of resources in order to maximize profit. In manufacturing, many of the scheduling problems are considered to be exceedingly complex in which they are difficult to be solved with exact methods and conventional algorithms. Scheduling problems have been the research interest of many researchers and a great deal of research efforts can be found in different fields of engineering and science, such as operations research, computer science, industrial engineering, mathematics, and management science since 1950.

The job shop scheduling problems (JSSPs) are well-known, important, and complex problems in production management and combinatorial optimization fields which are characterized as NP-hard. Complexity of JSSPs can be calculated by $(n!)^m$ in terms of all possible schedules, in which their complexity is exceedingly increasing as the problem size gets bigger. Garey et al. [2] and Ullman [3] have proved that the JSSPs are amongst the NP-hard problems; thus they cannot be solved with polynomial-time algorithms (unless $P = NP$). The JSSPs have tended to be treated with conventional techniques and exact methods, such as Lagrangian relaxation, branch and bound, heuristic rules, shifting bottleneck (e.g., Carlier and Pinson [4], Adams et al. [5], Vancheeswaran and Townsend [6], Brucker et al. [7], and Lageweg et al. [8]), since job shop instances have been presented by Fisher and Thompson [9]. Exact methods like branch and bound could guarantee global optima; however, computational time could be significantly increased with growing problem size. Over the preceding decade, many different methodologies have been inspired from nature, biology, and physical processing. These techniques have been successfully applied on numerous optimization problems, especially the JSSPs. Among these metaheuristics are genetic algorithm [10, 11], ant colony optimization [12], imperialist competitive algorithm [13, 14], tabu search [15], simulated annealing (SA) [16, 17], particle swarm optimization [18], and immune system [19]. Jain and Meeran [20] and Çaliş and Bulkan [21] have performed comprehensive reviews on JSSP techniques which can be referred to for more detail.

Genetic algorithm (GA) as a powerful search technique imitates the biological evolution and natural selection process. GA was first proposed by Holland [22] and further developed by David Goldberg. This metaheuristic approach is widely employed to find optimal or near optimal solutions for different optimization problems in comparison to other algorithms. GA was first applied on JSSPs by Davis [23] and since then many GA-based algorithms have been presented for solving JSSPs. Croce et al. [11] presented a GA for solving JSSPs with an encoding scheme that was based on preference rules. Yamada and Nakano [24] proposed a GA with a new representation scheme that was based on operation completion time and its crossover was able to generate active schedules. Lee et al. [25] presented a GA with an operation-based representation and a precedence preserving order-based crossover for the JSSPs. Sun et al. [26] developed a modified GA with a clonal selection and a life span strategy for the JSSPs, and the developed algorithm was able to find 21 best known solutions out of 23 benchmarked instances. GA is a powerful search technique in which its global search ability is conspicuous from the literature; however, this metaheuristic algorithm suffers in terms of premature convergence and local search ability.

Hybridization strategy is mainly employed to overcome the drawbacks of GA in terms of local search ability and premature convergence in order to make the algorithm more efficient and powerful. Wang and Zheng [27] proposed a hybrid optimization strategy that was a combination of GA and local search. In this approach, local search algorithm decreased the probability of GA from getting trapped in local optima and this hybrid framework relaxed the parameter dependency of both algorithms. Zhou et al. [28] developed a hybrid heuristic-GA for solving the JSSPs with an adaptive mutation operator for preventing premature convergence. In this framework, GA was applied on the first operations of the machines, and heuristics were used to determine the remaining operations in the restricted solution space. In addition, a neighborhood search technique was applied to further improve the solution quality obtained from the hybrid heuristic-GA. Ombuki and Ventresca [29] presented a dead lock free local search GA with an operation-based representation and UOX crossover that was able to generate feasible solutions. In this algorithm, GA was employed to perform global search and a local search operator was applied for local exploitation. They have also developed another genetic-based hybrid algorithm with tabu search, and based on computational results, the hybrid GA with tabu search outperformed the local search GA. Gonçalves et al. [30] developed a hybrid GA for the JSSPs by combining GA, schedule builder, and a local search operator. In this algorithm, schedule builder was applied to generate schedules using a priority rule, and GA was used to determine priorities. Then, a local search operator further improved the solution quality. Lin and Yugeng [31] developed a hybrid algorithm with a new representation scheme called random keys encoding. In this algorithm, GA was used to obtain an optimal schedule, and then a neighborhood search was introduced to perform local exploitation and increase the solution quality obtained from GA. Zhou et al. [32] proposed a hybrid GA to minimize weighted tardiness in job shop scheduling. In this algorithm, GA and a heuristic were employed for obtaining optimal solutions, in which GA was applied for determining the first operations and the heuristic was used for assigning the remaining operations. Results showed that the hybrid framework performed better than GA and heuristic alone. Asadzadeh and Zamanifar [33] proposed a GA that was implemented in parallel using agents, and agents were also used to create initial populations. Zhang and Wu [17] introduced a hybrid-SA immune system algorithm for minimizing the total weighted tardiness of job shop scheduling. Yusof et al. [34] developed a hybrid micro-GA that was implemented in parallel for the JSSPs. This algorithm was a combination of asynchronous colony GA that consisted of colonies with a small number of populations and autonomous immigration GA with subpopulations.

In this paper, an effective hybrid GA is presented for solving the nonpreemptive JSSPs. In order to solve the presented problem more effectively, an operation-based representation is used. In addition, a new knowledge-based operator is designed and adopted within the context of function evaluation. This knowledge-based operator imitates the JSSP's characteristics in order to use the machines' idle times in assigning the operations to the machines. In the reproduction phase of GA, a machine based precedence preserving order-based crossover and two types of mutation operators are developed in order to produce the offspring and mutants. Moreover, SA is employed to increase the solution quality of the schedules obtained from GA and to increase the population diversity in GA to some extent. Having highlighted the important features of the proposed algorithm, it is developed to minimize the makespan of schedules. Finally, computational results of benchmarked problems are used to prove the efficiency of the proposed algorithm.

The rest of this paper is organized as follows. In Section 2, the problem formulation of the JSSP is presented. The proposed hybrid GA is discussed in Section 3. Computational results of benchmarked instances are presented in Section 4, which is followed by a discussion. Finally, conclusions and future work are provided in Section 5.

2. Problem Formulation

The JSSP is a general form of classical scheduling problems which can be defined as follows: given n jobs $N = \{j_1, j_2, \dots, j_n\}$ and each job consists of s operations $S = \{O_{j1}, O_{j2}, \dots, O_{j,s}\}$ that must be processed on m machines in a given technological sequence. The notation $O_{j,s}$ denotes the sth operation of job j with a known processing time, $P_{j,s}$, which has to be processed on one of the machines $M = \{M_1, M_2, M_3, \ldots, M_m\}$. In this environment, each machine can process at most one operation at a time, and an operation of a given job cannot be processed on two machines at the same time. Once operation $O_{j,s}$ starts to be processed on one of the predetermined machines, it must be completed without any preemption. In addition, a job cannot visit the same machine twice, and there are no precedence requirements for different operations of the jobs. It is also assumed that the machines are continuously available, and the traveling times of operations are negligible. Unlike flow shop scheduling, in JSSPs, each job has its unique predetermined route. The sequencing of all operations on all machines is scheduled to minimize $C_{\max}$, that is, the maximum completion time of the jobs ($\max\{C_1, C_2, \ldots, C_j\}$).

The mathematical model of a JSSP with the objective of minimizing makespan is given in (1)–(9) [35, 36]. In this model, B is assumed as a big number, $t_{j,s}$ denotes the start time of operation $O_{j,s}$, $Sm_{i,l}$ is the start time of machine i in the priority of l, and $h_{i,j,s}$ is assigned 1 if operation $O_{j,s}$ is executed on machine i and 0 otherwise:

$$\min \quad Z = \max\left\{C_1, C_2, \ldots, C_j\right\} \quad j = 1, 2, \ldots, n \tag{1}$$

$$\text{S.t.} \quad t_{j,s} + P_{j,s} \le t_{j,s+1} \quad j = 1, 2, \ldots, n;\ s = 1, 2, \ldots, s_j - 1 \tag{2}$$

$$Sm_{i,l} + P_{j,s} X_{i,j,s,l} \le Sm_{i,l+1} \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, l_i - 1;\ s = 1, 2, \ldots, s_j \tag{3}$$

$$Sm_{i,l} \le t_{j,s} + \left(1 - X_{i,j,s,l}\right) B \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, l_i;\ s = 1, 2, \ldots, s_j \tag{4}$$

$$X_{i,j,s,l} \le h_{i,j,s} \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, l_i;\ s = 1, 2, \ldots, s_j \tag{5}$$

$$\sum_j \sum_s X_{i,j,s,l} = 1 \quad i = 1, 2, \ldots, m;\ l = 1, 2, \ldots, l_i \tag{6}$$

$$\sum_i \sum_l X_{i,j,s,l} = 1 \quad j = 1, 2, \ldots, n;\ s = 1, 2, \ldots, s_j \tag{7}$$

$$t_{j,s} \ge 0 \quad j = 1, 2, \ldots, n;\ s = 1, 2, \ldots, s_j \tag{8}$$

$$X_{i,j,s,l} \in \{0, 1\} \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, l_i;\ s = 1, 2, \ldots, s_j. \tag{9}$$

In this model, constraint (2) is concerned with the operations' sequences in which they should follow a specified order. Constraint (3) prevents machines overlapping and enforces each machine to process not more than one operation at the same time. Constraint (4) prevents operations overlapping, so that an operation is assigned to a specified idle machine in a condition such that its previous operation is executed and finished. In addition, for each of the operations, a machine is determined in constraint (5). In constraint (6), operations are assigned to the machines and they are sequenced on the machines. Constraint (7) limits the number of operations to be performed on one machine according to the machine's priority.

3. The Proposed Hybrid Genetic Algorithm

The proposed hybrid algorithm is a combination of two algorithms, namely, GA and SA. The advantages of both algorithms are employed in the proposed framework in order to find the optimum solutions for the JSSPs. In Figure 1, a flowchart of the proposed hybrid genetic algorithm (HGA) is presented. It starts with the initialization of the population at random. The created population is evaluated based on the fitness function, and a new knowledge-based operator is applied in this step to improve the solution quality of the individuals. In addition, this knowledge-based operator is merged with the function evaluation phase, and it works with machines' idle times. This operator is presented in Section 3.2. In the reproduction phase, a selection operator is applied to select the parents for the mating pool, and then a crossover operator is performed to produce the offspring. In addition, a mutation operator is carried out on randomly selected individuals to create the mutants. The created offspring and mutants are evaluated, and then termination conditions are considered in order to guide the algorithm to be on the right path. The termination conditions are described in Section 3.7.

In a situation when the algorithm continues through termination condition 2, that is, termination of GA's generations, SA will be started with the best individual of GA. In SA, a neighborhood search structure is employed with three different operators, namely, swapping, insertion, and reversion as a proposal mechanism. In addition, beta percent of accepted solutions are kept in a pool of individuals as each cooling condition of SA is reached. The SA algorithm is presented in depth in Section 3.6. In this phase of the algorithm, three conditions are set, either to stop the algorithm or to continue with new parameters. If the latest condition or condition 4, that is, termination of SA's outer loop, is reached the SA

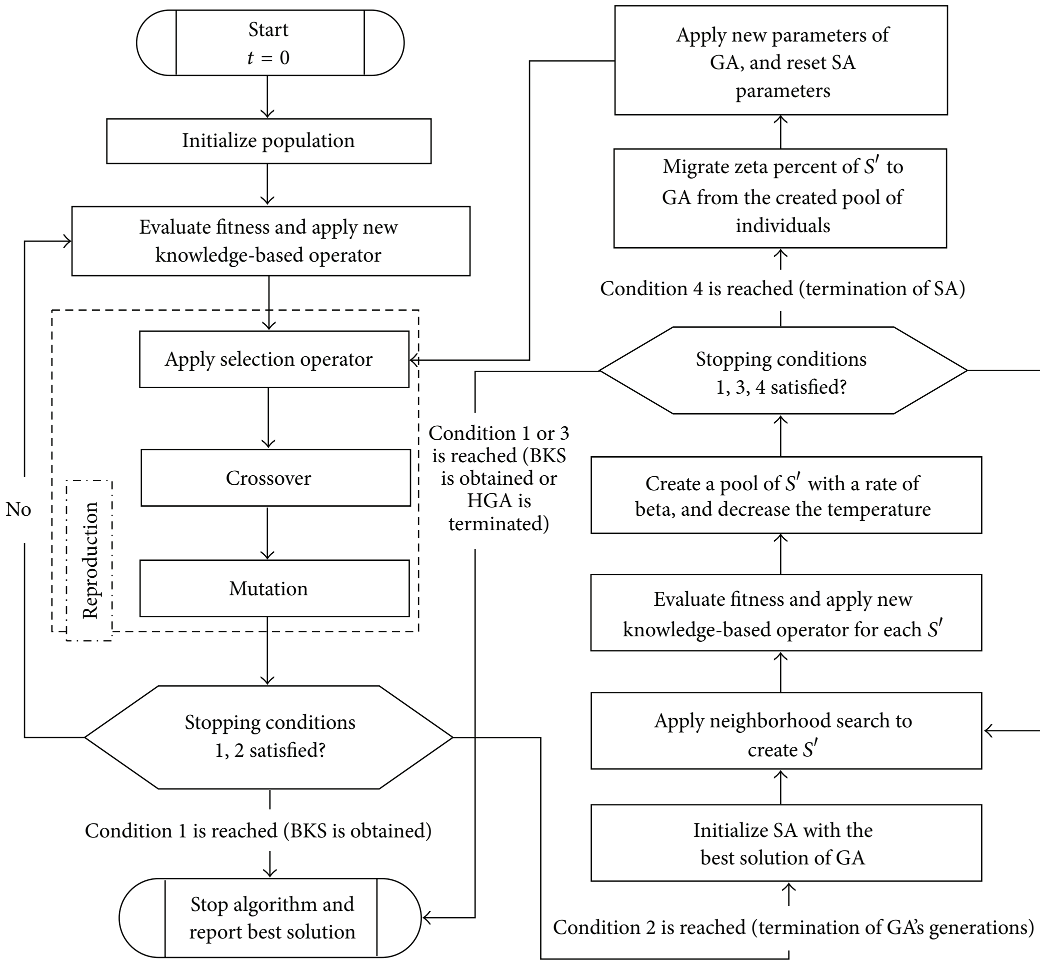
Start
$t = 0$
Initialize population
Evaluate fitness and apply new knowledge-based operator
Apply selection operator
Crossover
Mutation
Reproduction
No
Stopping conditions 1, 2 satisfied?
Condition 1 is reached (BKS is obtained)
Stop algorithm and report best solution
Condition 1 or 3 is reached (BKS is obtained or HGA is terminated)
Condition 2 is reached (termination of GA's generations)
Initialize SA with the best solution of GA
Apply neighborhood search to create S'
Evaluate fitness and apply new knowledge-based operator for each S'
Create a pool of S' with a rate of beta, and decrease the temperature
Stopping conditions 1, 3, 4 satisfied?
No
Condition 4 is reached (termination of SA)
Migrate zeta percent of S' to GA from the created pool of individuals
Apply new parameters of GA, and reset SA parameters

Figure 1: Flowchart of the proposed hybrid genetic algorithm.

parameters are reset and new parameters of GA are applied. Moreover, zeta percent of unique individuals are migrated to GA from the migration pool of SA.

3.1. Encoding and Decoding. In any algorithm, the first and the most important step is to find appropriate encoding and decoding procedures in order to represent the problem. In this paper, an operation-based representation is adopted to represent the permutation of operations of different jobs [34, 37]. Based on this representation approach, the schedule could be constructed if the technological constraints are satisfied. In this approach, a chromosome consists of $n \times m$ genes in which each of the genes represents the sequence of the operations that should be executed on the machines. Each of the operations is denoted with a positive integer value that is starting from 1 to n. The number of occurrences for each of the integer values is equal to the number of operations. In other words, the kth occurrence of an integer value in the chromosome represents the kth operation of the job with respect to the technological sequences. Consider a JSSP that is given in Table 1. In this table, operation routing, machine, and processing time of a small problem with 4 jobs and 3 machines are tabulated. Suppose a randomly generated

Table 1: An example of a job shop problem with 4 jobs and 3 machines.

Job	Processing time, machine number		
j_1	1, 1	3, 2	2, 3
j_2	8, 2	5, 1	10, 3
j_3	5, 1	4, 3	8, 2
j_4	4, 3	10, 1	6, 2

chromosome is given as $\{3\ \ 2\ \ 4\ \ 1\ \ 3\ \ 1\ \ 2\ \ 3\ \ 2\ \ 4\ \ 1\ \ 4\}$. In this chromosome, each job consists of three operations, and due to this reason each job occurs three times within the length of the chromosome. For instance, the sixth and ninth genes of this chromosome represent the second operation of job one and third operation of job two, respectively. In addition, each chromosome is composed of additional information such as machine number, processing time, start time, and finish time that are attached to the chromosome.

In order to decode the chromosome and construct the schedule, we start from the most left to the most right of the chromosome; that is, the first gene in the left side of the chromosome should be scheduled first, followed by

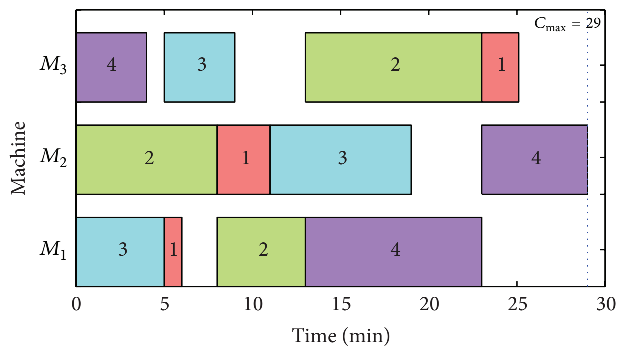

Figure 2: The schedule of 4 × 3 job shop scheduling problem.

the second gene, until the last gene of the chromosome. Based on Table 1, the first machine should process $O_{11}, O_{22}, O_{31}, O_{42}$, the second machine should process O_{12}, O_{21}, O_{33}, O_{43}, and the last machine should process $O_{13}, O_{23}, O_{32}, O_{41}$. According to the chromosome {3 2 4 1 3 1 2 3 2 4 1 4} and considering the process constraints to be met, the operation sequences of the jobs to be performed on machines 1, 2, and 3 are as follows. These sequences for the first, second, and third machines are $[O_{31}, O_{11}, O_{22}, O_{42}]$, $[O_{21}, O_{12}, O_{33}, O_{43}]$, and $[O_{41}, O_{32}, O_{23}, O_{13}]$, respectively. According to these sequences, the first operation of each machine should be scheduled by considering the process and time constraints. Therefore, operations O_{31}, O_{21}, and O_{41} must be scheduled on machines 1, 2, and 3 at the permissible time one after another, respectively. Then, the second operations $[O_{11}, O_{12}, O_{32}]$, third operations $[O_{22}, O_{33}, O_{23}]$, and fourth operations $[O_{42}, O_{43}, O_{13}]$ of each machine must be scheduled by considering the process and time constraints. In addition, each set of them must be scheduled separately one after another on machines 1, 2, and 3, respectively, at the permissible time. The procedures that are applied in this encoding and decoding can guarantee feasible schedules. In Figure 2, the schedule of the considered chromosome is shown.

3.2. Fitness Function and the Knowledge-Based Operator.

The fitness function in an optimization problem usually determines the probability of a solution that can be passed to the next generation. In other words, the solution quality is determined by applying this operator and the chromosomes with higher quality will have a higher chance of surviving; however, the less fitted chromosomes must be discarded from the population. In the JSSPs, many different performance evaluators exist for defining the fitness function. In this research, we use makespan or $C_{\max}$ as the fitness function in order to evaluate each of the chromosomes.

In this algorithm and in the context of the fitness function, a new knowledge-based operator is designed based on the problem characteristics. This operator is designed based on the machines' idle times that exist in the job shop environments. In addition, this knowledge-based operator is applied during the function evaluation phase in order to decrease the computational time of the algorithm and to cover all the chromosomes that need to be evaluated. To design this operator more efficiently, the following steps are applied.

Step 1. The idle points of each machine must be found. Then, for each of these idle points, the idle start time, idle finish time, and length of idle time must be calculated.

Step 2. Based on the position of the idle point on the machine sequence list, a candidate operation is chosen from the right side of the machine sequence list in order to be shifted to the idle position by considering the duration of idle time and the processing time of the candidate operation.

Step 3. If the length of idle time is bigger than or equal to the processing time of the candidate operation, it will be accepted conditionally. Otherwise, it will be rejected.

Step 4. If the candidate operation is rejected for transfer, the operator goes back to the second step and chooses the subsequent operation.

Step 5. For the conditionally accepted candidate operation, all of the processing constraints must be considered in order to reject the shift or accept it. For instance, the previous operation of the candidate operation must be completed.

Step 6. If all of the constraints are satisfied, the candidate operation will be transferred to its new position. Otherwise, the candidate operation will remain in its own position.

Step 7. For each of the machines, Steps 2–6 should be continued until the last operation on the machine sequence list.

Consider the 4 jobs 3 machines job shop problem that is given in Table 1. Suppose the chromosome is given as {3 2 4 1 3 1 2 3 2 4 1 4}, in which its schedule is depicted in Figure 2. The new knowledge-based operator with the mentioned procedures, 1 to 7, is applied to the third machine of this chromosome as follows. In the third machine, there are two idle points in which the first one starts at 4 and finishes at 5, and the second one starts at 9 and finishes at 13. Based on the machine sequence list $[O_{41}, O_{32}, O_{23}, O_{13}]$ and Figure 2, the candidate operations for the first and second idle points can be $[O_{32}, O_{23}, O_{13}]$ and $[O_{23}, O_{13}]$, respectively. Consider the candidate operations for the first idle point with the duration of 1 minute, the processing time for any of the given candidate operations $[O_{32} = 4, O_{23} = 10, O_{13} = 2]$ is more than 1 minute. Therefore, the first idle point will remain untouched. However, the candidate operations for the second idle point with the processing time of $[O_{23} = 10, O_{13} = 2]$ and its idle duration of 4 minutes lead to a possibility of a shift. The second candidate operation in the list, O_{13}, has a processing time less than the length of the second idle point. Then, for this operation, the fifth step (i.e., considering all of the constraints) must be executed in order to have a feasible schedule. It is clear that if operation O_{13} is transferred to the second idle position, none of the constraints is violated. Therefore, operation O_{13} is transferred to its new position and the finish time of the third machine is decreased to 23 minutes as depicted in Figure 3.

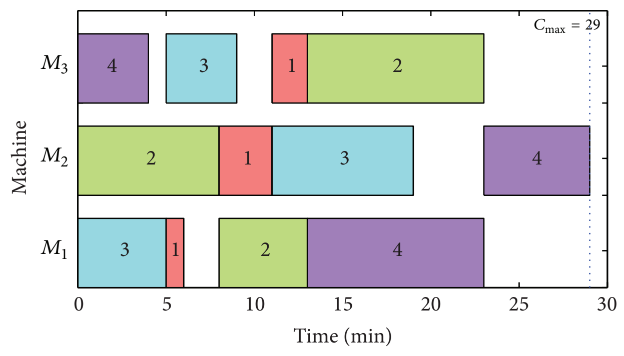

FIGURE 3: The schedule of a 4 × 3 job shop scheduling problem after applying the new knowledge-based operator on machine 3.

3.3. Selection Operator. A good selection technique can increase GA's performance in terms of reaching faster to the optimal solutions. In this paper, the Roulette Wheel selection technique which is the most commonly used operator is employed for the selection of parents [34]. In addition, the elitism approach is applied in this selection technique in order to retain the fittest chromosomes for the next generation and to prevent the solution from getting worse from one generation to another. In the Roulette Wheel selection, we used the Boltzmann Probability $P(x) = \exp(-\beta' \times f(x))/f'(x)$ to calculate the probability of each chromosome. In this equation, $P(x)$ is the probability of each chromosome, β' is the selection pressure, $f(x)$ is the fitness of each individual, and $f'(x)$ is the fitness of the worst individual in the generation. It should be noted that we added $f'(x)$ to the original equation of Boltzmann Probability in order to make the selection pressure independent of the problem scale. In addition, the normalized probability of each selected individual is given as $P(x)_{\text{norm}} = P(x)/\sum_{x=1}^{\text{max.pop}} P(x)$.

3.4. Crossover. In GA, crossover is the most important operator in comparison to other operators and, in fact, it is the backbone of the algorithm. The crossover operator is performed by combining the information of the first and second parents and the produced offspring with the features of both parents can be either better or worse than their parents. In addition, the main goal of this operator is to produce better and feasible offspring from the parental information. In this paper, a machine based precedence preserving order-based crossover (POX) is proposed for generating the feasible offspring [25]. The following detailed steps are taken in order to implement the POX operator.

Step 1. First, two individuals are selected as parents (P_1, P_2) by applying the Roulette Wheel selection.

Step 2. Then, two sets of subjobs are selected and called sj_1 and sj_2. One of the subjobs is selected from the bottleneck machines and the other one is selected randomly amongst the n remaining jobs.

Step 3. In this step, the elements of the first subjob (sj_1) are copied from the first parent (P_1) to the exact positions in the first child (O_1), and the same goes for the second subjob; that is, the elements of the second subjob (sj_2) are copied from the second parent (P_2) to the exact alleles in the second child (O_2).

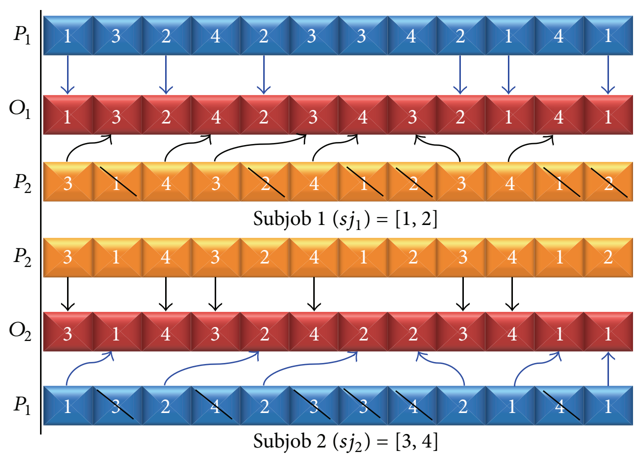

FIGURE 4: The procedures of a POX crossover in generating the offspring.

Step 4. All of the alleles in the first subjob (sj_1) are deleted in the second parent (P_2), and the same goes for the second subjob; that is, all of the alleles in the second subjob (sj_2) are deleted in the first parent (P_1).

Step 5. The remaining alleles in the first and second parents (P_1, P_2) are transferred to the second and first offspring (O_2, O_1) from the most left to the most right, respectively.

The procedures that are used in implementing the proposed POX operator lead to the feasible solutions in which it does not need a repairing mechanism. Figure 4 shows the procedures of producing offspring from the parental information by applying the POX operator on a 4 × 3 JSSP.

3.5. Mutation. In the reproduction phase, mutation is the second way of exploring the solution space. Mutation operator can prevent the algorithm from being trapped in the local optima, and it makes the algorithm faster in achieving better solutions. In addition, it could make perturbation in the chromosome in order to increase the population diversity. In this algorithm, two types of mutation operators, namely, swapping and insertion, are employed. Not only can these mutation operators increase the diversity of the population, but the insertion operator could carry out intensive search. It should be noted that one of these mutation operators should be selected randomly in order to create an offspring, and they are described as follows.

(1) Swapping operator: To apply the swapping operator, first, two random numbers are generated which are the positions of two alleles in the chromosome (e.g., $R = \{2, 5\}$). Then, all of the parental information is copied to the exact positions in the offspring, except the randomly selected alleles which must be

exchanged or swapped in the offspring. For instance, consider the parent as $\{2, 1, 1, 3, 2, 3\}$ which is randomly selected from the population. The offspring of this chromosome, by applying the swapping operator, would be $\{2, 2, 1, 3, 1, 3\}$.

(2) Insertion operator: To apply the insertion operator, first, two random numbers are generated to be the positions of two alleles in the chromosome (e.g., $R = \{6, 4\}$). Then, all of the parental information is copied to the exact positions in the offspring, except the randomly selected alleles. The randomly selected allele with a smaller job-value is positioned on the left of the other random allele with a bigger job-value. For instance, given the randomly selected parent as $\{3, 1, 2, 3, 1, 2\}$, the offspring of this chromosome, by applying the insertion operator, would be $\{3, 1, 2, 2, 3, 1\}$.

3.6. Simulated Annealing. The SA approach was inspired by the physical annealing process of solid materials, and it is characterized as a stochastic local search approach [38]. The search operator in SA is occasionally permitted to go through any unfavorable direction, and it enables the algorithm to escape from the local solutions and get towards the global solutions. In SA, this characteristic could be attained through probabilistically accepting the worse solutions.

In the proposed HGA, SA starts with the best solution of GA. Then, a proposal mechanism that consists of three operators, namely, swapping, insertion, and reversion, is applied in order to generate a new neighborhood solution (S') based on the current solution (S). The new knowledge-based operator is applied on the newly generated solution (S'), and then it is evaluated based on the objective function. If the newly evaluated neighborhood solution is equal to or better than the current solution, the new neighborhood solution will be accepted ($f(S') \leq f(S)$). Otherwise, the algorithm will continue the search process with a solution (S' or S) that is decided through a probabilistic acceptance function. In addition, acceptance of a solution is based on the individual's objective function value and the current temperature (T) of the algorithm ($e^{[(f(S)-f(S'))/T]}$). In each inner loop of SA, an accepted solution is kept in a new pool of individuals. As the inner loops of the algorithm are terminated, the initial value of the temperature should be modified based on the annealing schedule. In addition, beta percent of unique individuals are kept and the remaining individuals are discarded from the new pool of individuals in each outer loop of SA. Moreover, with the termination of the outer loop, zeta percent of unique individuals are kept for the migration purpose and the remainders are discarded. It should be noted that the migration rate must be low, and it is used to increase the population diversity of GA to some extent.

In the proposal mechanism of SA, three operators, namely, swapping, insertion, and reversion, are used, in which one of them is randomly selected and applied in order to create the neighborhood solution. Among these operators, swapping and insertion were explained in Section 3.5, and the reversion operator is described as follows. First, two random positions are selected within the length of the individual (e.g., $R = \{1, 4\}$), and then the substrings between them are inverted. For instance, given the current solution as $\{2, 1, 3, 1, 2, 3\}$, by applying the reversion operator, the new neighborhood solution would be $\{1, 3, 1, 2, 2, 3\}$.

3.7. Ending Conditions. In the proposed HGA, four different conditions are provided in order to terminate the algorithm entirely or partially. The first termination condition is the achievement of the best known solution and the third termination condition is set as the maximum number of generations in the main loop of HGA. If the algorithm reaches either the first or third condition, the whole algorithm will be terminated at once. The second and fourth partial conditions are defined as the maximum number of generations in GA and the maximum number of outer loops in SA, respectively.

4. Computational Experiments and Discussion

4.1. Basic Data. The main aim of this section is to evaluate the performance and validate the efficiency of the proposed HGA based on the well-studied job shop instances. For this reason, two classes of the job shop instances were used in which the first class was presented by Lawrence [39], LA01 to LA40, and the second class was introduced by Fisher and Thompson [9], FT06 and FT20. Different dimensions of instances in terms of jobs and machines were used, including 6×6, 10×5, 15×5, 20×5, 10×10, 15×10, 20×10, 30×10, and 15×15 which were collected from the Operations Research Library. In addition, some of the reported algorithms in the literature [16, 19, 27, 29–31, 33, 40–45] were used in order to compare with the proposed HGA.

4.2. Computational Results. To develop the proposed HGA, MATLAB R2010a was used, and the algorithm was run on a computer which was functioning with Windows XP, Intel® Core™ Duo CPU T2450 at 2 GHz, and 2.49 GB of RAM. Each of the benchmarked instances was tested 10 times independently with the following tuned parameters: population size $N_{\text{pop}} = 200$, crossover probability on a population $P_c = 0.5$, mutation probability on a population $P_{\text{mu}} = 0.8$, initial temperature of SA, $T_0 = 30$, cooling rate of temperature in SA, $\alpha = 0.9$, keeping rate of individuals $\beta = 0.05$, migration rate of individuals $\zeta = 0.002$, and updated value of $P_{\text{mu}} = 0.9$. In addition, the selection pressure of the Roulette Wheel operator is set at $\beta' = 7$ for all benchmarked instances.

The computational experiments of the 42 well-studied job shop instances with the above tuned parameters and 10 replications per each benchmarked instance have been carried out, and the results are depicted in Table 2. Table 2 consists of the instance name, dimension of the problem (jobs × machines), best known solution (BKS), and results of our proposed HGA which consist of four columns including the best solution of the replications, relative deviation (RD), and average solution and worst solution of the algorithm in 10 runs. These are the most important indexes in the comparison process of the algorithm in order to check its effectiveness and consistency. The remaining columns of Table 2 are the obtained results of the compared algorithms, including

Table 2: Experimental results.

Instance name	Size	BKS	Proposed HGA				HGAPSA	PGA	MMIA	MA	HGA	HGA	Param	LSGA	HGA	AIS	Grasp	HGA	BS	RCS
			Best	RD (%)	Ave.	Worst	2012	2010	2009	2008	2007	2006	2005	2004		2003	2000	2001	1999	1996
FT06	6 × 6	55	55*	0.00	55	55	—	55	55	55	—	55	55	55	55	—	55	55	—	55
FT20	20 × 5	1165	1165*	0.00	1174	1178	—	1196	—	1165	—	1175	1165	1209	1192	—	1169	1165	—	1165
LA01	10 × 5	666	666*	0.00	666	666	—	666	666	666	666	666	666	—	—	666	666	666	666	666
LA02	10 × 5	655	655*	0.00	657.5	666	—	655	655	655	655	655	655	—	—	655	655	—	704	655
LA03	10 × 5	597	597*	0.00	597.7	604	—	617	597	597	617	603	597	—	—	597	604	—	650	597
LA04	10 × 5	590	590*	0.00	590	590	—	607	590	590	606	590	590	—	—	590	590	—	620	590
LA05	10 × 5	593	593*	0.00	593	593	—	593	593	593	593	593	593	—	—	593	593	—	593	593
LA06	15 × 5	926	926*	0.00	926	926	—	926	926	926	926	926	926	—	—	926	926	926	926	926
LA07	15 × 5	890	890*	0.00	890	890	—	890	890	890	897	890	890	—	—	890	890	—	890	890
LA08	15 × 5	863	863*	0.00	863	863	—	863	863	863	863	863	863	—	—	863	863	—	863	863
LA09	15 × 5	951	951*	0.00	951	951	—	951	951	951	951	951	951	—	—	951	951	—	951	951
LA10	15 × 5	958	958*	0.00	958	958	—	958	958	958	958	958	958	—	—	958	958	—	958	958
LA11	20 × 5	1222	1222*	0.00	1222	1222	—	1222	1222	1222	1222	1222	1222	—	—	—	1222	1222	1222	1222
LA12	20 × 5	1039	1039*	0.00	1039	1039	—	1039	—	1039	1039	1039	1039	—	—	—	1039	—	1039	1039
LA13	20 × 5	1150	1150*	0.00	1150	1150	—	1150	1150	1150	1150	1150	1150	—	—	—	1150	—	1150	1150
LA14	20 × 5	1292	1292*	0.00	1292	1292	—	1292	—	1292	1292	1292	1292	—	—	—	1292	—	1292	1292
LA15	20 × 5	1207	1207*	0.00	1207	1207	—	1207	1207	1207	—	1207	1207	—	—	—	1207	—	1207	1207
LA16	10 × 10	945	946	0.11	946.2	947	—	994	—	945	—	945	945	959	959	945	946	945	988	945
LA17	10 × 10	784	784*	0.00	786.4	787	—	793	784	784	—	784	784	792	792	785	784	—	827	784
LA18	10 × 10	848	848*	0.00	848.8	852	—	860	—	848	—	848	848	857	857	848	848	—	881	848
LA19	10 × 10	842	842*	0.00	844.4	850	—	873	857	844	—	851	842	860	860	848	842	—	882	848
LA20	10 × 10	902	907	0.55	907	907	—	912	—	907	—	907	907	907	907	907	907	—	948	907
LA21	15 × 10	1046	1050	0.38	1055	1061	—	1146	1088	—	—	1067	1046	1114	1097	—	1091	1058	1154	1069
LA22	15 × 10	927	927*	0.00	937.1	943	—	1007	—	—	—	941	935	989	980	—	960	—	985	937
LA23	15 × 10	1032	1032*	0.00	1032	1032	—	1033	—	—	—	1032	1032	1035	1032	—	1032	—	1051	1032
LA24	15 × 10	935	943	0.86	950.5	956	—	1012	—	—	—	957	953	1032	1001	—	978	—	992	942
LA25	15 × 10	977	984	0.72	992.3	1000	—	1067	—	—	—	993	986	1047	1031	1022	1028	—	1073	981
LA26	20 × 10	1218	1218*	0.00	1218	1218	—	1323	—	—	—	1218	1218	1307	1295	—	1271	—	1269	1218
LA27	20 × 10	1235	1255	1.62	1257	1261	—	1359	—	—	—	1265	1256	1350	1306	—	1320	—	1316	1285
LA28	20 × 10	1216	1217	0.08	1221	1225	—	1369	—	—	—	1233	1232	1312	1302	1277	1293	—	1373	1216
LA29	20 × 10	1152	1179	2.34	1189	1204	—	1322	—	—	—	1182	1196	1311	1280	1248	1293	—	1252	1208
LA30	20 × 10	1355	1355*	0.00	1355	1355	1355	1437	—	—	—	1355	1355	1451	1406	—	1368	—	1435	1355
LA31	30 × 10	1784	1784*	0.00	1784	1784	1790	1844	1784	—	—	1784	1784	1784	1784	—	1784	1784	1784	1784
LA32	30 × 10	1850	1850*	0.00	1850	1850	1860	1907	—	—	—	1850	1850	1850	1850	—	1850	—	1850	1850
LA33	30 × 10	1719	1719*	0.00	1719	1719	1719	—	—	—	—	1719	1719	1745	1719	—	1719	—	1719	1719
LA34	30 × 10	1721	1721*	0.00	1721	1721	1725	—	—	—	—	1721	1721	1784	1758	—	1753	—	1780	1721
LA35	30 × 10	1888	1888*	0.00	1888	1888	1895	—	—	—	—	1888	1888	1958	1888	1903	1888	—	1888	1888
LA36	15 × 15	1268	1294	2.05	1300	1307	1279	—	—	—	—	1292	1279	1358	1357	1323	1334	1292	1401	1292
LA37	15 × 15	1397	1418	1.50	1431	1436	1408	—	—	—	—	1418	1408	1517	1494	—	1457	—	1503	1411

Table 2: Continued.

Instance name	Size	BKS	Proposed HGA				HGAPSA	PGA	MMIA	MA	HGA	HGA	Param	LSGA	HGA	AIS	Grasp	HGA	BS	RCS
			Best	RD (%)	Ave.	Worst	2012	2010	2009	2008	2007	2006	2005	2004		2003	2000	2001	1999	1996
LA38	15 × 15	1196	1222	2.17	1230	1238	1219	—	—	—	—	1231	1219	1362	1338	1274	1267	—	1297	1278
LA39	15 × 15	1233	1249	1.30	1250	1251	1246	—	—	—	—	1251	1246	1391	1343	1270	1290	—	1369	1233
LA40	15 × 15	1222	1233	0.90	1238	1243	1241	—	—	—	—	1246	1241	1323	1311	1258	1259	—	1347	1247
Average relative deviation (%)				**0.35**			**0.7**	**3.4**	**0.32**	**0.04**	**0.49**	**0.65**	**0.41**	**5.41**	**4.07**	**1.61**	**1.8**	**0.38**	**4.23**	**0.63**

BKS: best known solution.
RD: relative deviation in percentage.
Ave.: average of 10 independent runs.
*Optimal solution achieved by our HGA.

Table 3: Comparison of the results.

Reference	CA	NBPS	NBKSO		ARD (%)		Improvement in the proposed HGA
			CA	Proposed HGA	CA	Proposed HGA	(%)
Rakkiannan and Palanisamy [16]	HGAPSA	11	2	6	0.7	0.7	0
Asadzadeh and Zamanifar [33]	PGA	34	14	26	3.4	0.2	3.2
Luh and Chueh [40]	MMIA	18	16	17	0.32	0.02	0.3
Yang et al. [41]	MA	22	20	20	0.04	0.03	0.01
Hasan et al. [42]	HGA	14	11	14	0.49	0.00	0.49
Lin and Yugeng [31]	HGA	42	26	29	0.65	0.35	0.3
Gonçalves et al. [30]	Param	42	30	29	0.41	0.35	0.06
Ombuki and Ventresca [29]	LSGA	27	3	14	5.41	0.54	4.87
	HGA	27	6	14	4.07	0.54	3.53
Coello et al. [19]	AIS	23	12	14	1.61	0.44	1.17
Binato et al. [43]	Grasp	42	23	29	1.8	0.35	1.45
Wang and Zheng [27]	HGA	9	7	6	0.38	0.28	0.1
Sabuncuoglu and Bayiz [44]	BS	40	16	27	4.23	0.36	3.87
Nuijten and Aarts [45]	RCS	42	30	29	0.63	0.35	0.28

NBPS: number of benchmarked problems solved.
NBKSO: number of best known solutions obtained.
ARD: average relative deviation in percentage.
CA: compared algorithms.

HGAPSA, PGA, MMIA, MA, HGA, HGA, Param, LSGA, HGA, AIS, Grasp, HGA, BS, and RCS, which were presented by Rakkiannan and Palanisamy [16], Asadzadeh and Zamanifar [33], Luh and Chueh [40], Yang et al. [41], Hasan et al. [42], Lin and Yugeng [31], Gonçalves et al. [30], Ombuki and Ventresca [29], Coello et al. [19], Binato et al. [43], Wang and Zheng [27], Sabuncuoglu and Bayiz [44], and Nuijten and Aarts [45], respectively. In order to calculate the relative deviation for each of the benchmarked instances, the formula $\mathrm{RD} = 100 \times (\mathrm{BFM} - \mathrm{BKS})/\mathrm{BKS}$ was used. In this formula, BFM is the best found makespan and BKS is the best known solution. In addition, the average execution times of 10 replications for the biggest benchmarked instances LA31 to LA35 were recorded as 152.2, 252.1, 143.6, 373.8, and 248.4 seconds, respectively. The average execution times for the other smaller benchmarked instances were shorter than these. Overall, it can be said that the computational time of our proposed HGA was reasonable.

4.3. Discussion. It is obvious that our proposed HGA was able to find optimal or near optimal solutions for the benchmarked problems. Table 2 presents that our HGA has obtained the best known solutions for 69.05 percent of the instances; that is, 29 out of 42 instances have reached the best known solutions. In the small sized benchmarked instances such as LA01 to LA15, FT06, and FT20, most of the reported algorithms and the proposed HGA were able to achieve the best known solutions. However, in the bigger sized benchmarked instances, LA16–LA40, the proposed algorithm was able to achieve equal or better solutions in comparison to most of the reported algorithms. Moreover, the results of the proposed HGA which were obtained without applying the new knowledge-based operator on the big sized instances (LA36–LA40) were 1300, 1428, 1232, 1251, and 1242, respectively. It is clear that the obtained results using the new knowledge-based operator in the proposed HGA are better in comparison to those without the knowledge-based operator. This implies that the new knowledge-based operator can increase the solution quality of the proposed HGA.

Table 3 lists the compared algorithms (CA), number of benchmarked problems solved (NBPS), number of best known solutions obtained (NBKSO), and average relative deviation (ARD) for the compared algorithms and the proposed HGA. In addition, the last column shows the improvement made in our HGA with respect to the other algorithms, in which it is the subtraction between the average relative deviation of the compared algorithms and the proposed HGA. Based on Table 3, the average relative deviation of the proposed HGA is only 0.35 percent for the 42 well-studied job shop instances (average deviation of the best found solutions by the proposed HGA from the best known solutions). It is clear that the proposed HGA has made a considerable improvement in the solution quality of the benchmarked instances in comparison to those of the other algorithms. For illustration purposes, the optimum schedule of LA22 and near optimal schedule of LA40 that were obtained by the proposed HGA are presented in Figures 5 and 6, respectively.

5. Conclusions

This paper has proposed a hybrid genetic algorithm that is a combination of GA and SA for solving the nonpreemptive JSSPs in order to minimize the makespan of schedules. In the proposed algorithm, GA was applied for global exploration among the chromosomes of a population, and SA was used to carry out local exploitation around the individuals. An

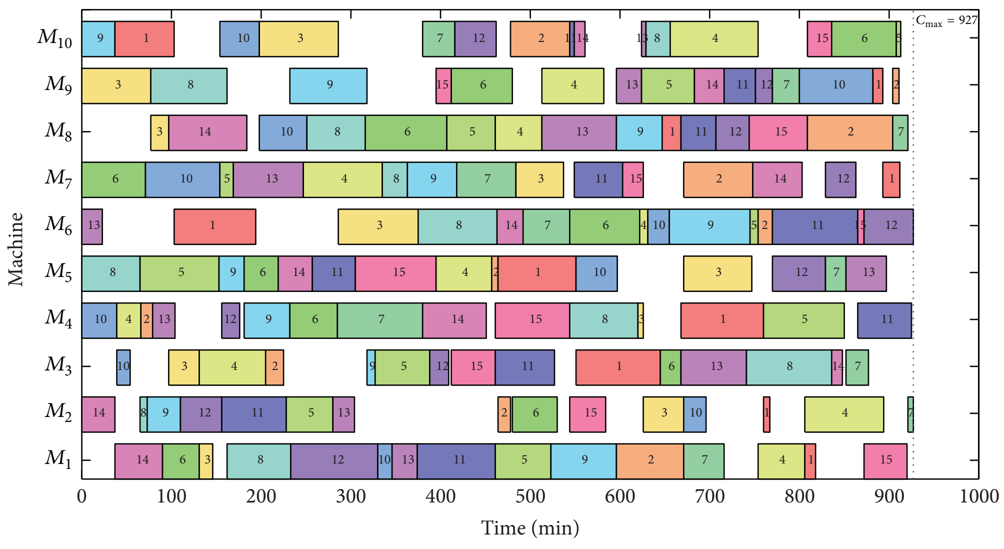

FIGURE 5: The optimal schedule of LA22 obtained by our HGA.

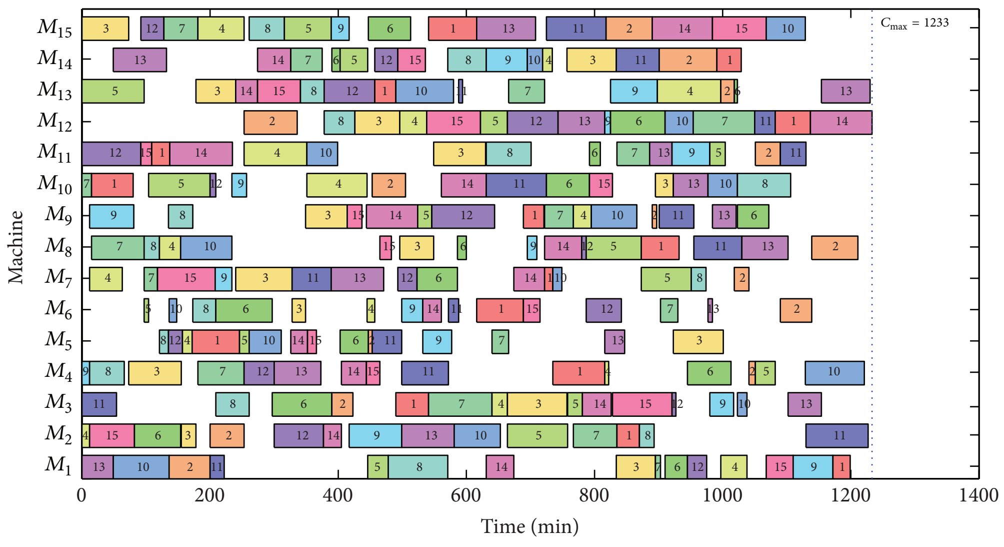

FIGURE 6: The near optimal schedule of LA40 obtained by our HGA.

operation-based representation was used for the solution encoding of the algorithm. In addition, a new knowledge-based operator based on the problem characteristics was designed, and it was able to increase the solution quality of the schedules. To produce the offspring and mutants, a machine based precedence preserving order-based crossover and two types of mutation operators, namely, swapping and insertion, were used in order to increase the population diversity and intensify the search. Moreover, the SA approach with its neighborhood search ability was applied to further improve the solution quality which was obtained from GA. The proposed HGA was tested on some of the well-studied benchmarked instances which were collected from the Operations Research Library, and the results were compared with other algorithms. Computational results show that generally the proposed algorithm has an average relative deviation less than those of the compared algorithms, and this proves the effectiveness and efficiency of the proposed approach.

For future work, we suggest taking into account greenness issue in scheduling problems as it is a new frontier that is being extended in the manufacturing areas. In addition, new developed algorithms like imperialist competitive algorithm can be implemented in the proposed framework with the suggested knowledge-based operator to see its performance. Moreover, one could consider developing new operators to further increase the population diversity of the algorithm and even developing an operator to measure the population diversity.

Competing Interests

The authors declare that they have no competing interests.

References

[1] M. L. Pinedo, "Introduction," in *Scheduling: Theory, Algorithms, and Systems*, pp. 1–10, Springer, New York, NY, USA, 2012.

[2] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[3] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, pp. 384–393, 1975.

[4] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989.

[5] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988.

[6] R. Vancheeswaran and M. A. Townsend, "Two-stage heuristic procedure for scheduling job shops," *Journal of Manufacturing Systems*, vol. 12, no. 4, pp. 315–325, 1993.

[7] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics*, vol. 49, no. 1–3, pp. 107–127, 1994.

[8] B. J. Lageweg, J. K. Lenstra, and A. H. Rinnooy Kan, "Job-shop scheduling by implicit enumeration," *Management Science*, vol. 24, no. 4, pp. 441–450, 1977/78.

[9] H. Fisher and G. L. Thompson, *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1963.

[10] R. Qing-Dao-Er-Ji, Y. Wang, and X. Wang, "Inventory based two-objective job shop scheduling model and its hybrid genetic algorithm," *Applied Soft Computing*, vol. 13, no. 3, pp. 1400–1406, 2013.

[11] F. D. Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 15–24, 1995.

[12] J. Zhang, X. Hu, X. Tan, J. H. Zhong, and Q. Huang, "Implementation of an ant colony optimization technique for job shop scheduling problem," *Transactions of the Institute of Measurement and Control*, vol. 28, no. 1, pp. 93–108, 2006.

[13] J. Zhang, P. Zhang, J. Yang, and Y. Huang, "Solving the Job Shop Scheduling Problem using the imperialist competitive algorithm," *Advanced Materials Research*, vol. 845, pp. 737–740, 2012.

[14] H. Piroozfard and K. Y. Wong, "An imperialist competitive algorithm for the job shop scheduling problems," in *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM '14)*, pp. 69–73, IEEE, Selangor, Malaysia, December 2014.

[15] V. A. Armentano and C. R. Scrich, "Tabu search for minimizing total tardiness in a job shop," *International Journal of Production Economics*, vol. 63, no. 2, pp. 131–140, 2000.

[16] T. Rakkiannan and B. Palanisamy, "Hybridization of genetic algorithm with parallel implementation of simulated annealing for job shop scheduling," *American Journal of Applied Sciences*, vol. 9, no. 10, pp. 1694–1705, 2012.

[17] R. Zhang and C. Wu, "A hybrid immune simulated annealing algorithm for the job shop scheduling problem," *Applied Soft Computing Journal*, vol. 10, no. 1, pp. 79–89, 2010.

[18] D. Lei, "A Pareto archive particle swarm optimization for multi-objective job shop scheduling," *Computers and Industrial Engineering*, vol. 54, no. 4, pp. 960–971, 2008.

[19] C. A. C. Coello, D. C. Rivera, and N. Cortés, "Use of an artificial immune system for job shop scheduling," in *Artificial Immune Systems*, J. Timmis, P. Bentley, and E. Hart, Eds., pp. 1–10, Springer, Berlin, Germany, 2003.

[20] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[21] B. Çaliş and S. Bulkan, "A research survey: review of AI solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 961–973, 2015.

[22] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–72, 1992.

[23] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 136–140, Hillsdale, NJ, USA, 1985.

[24] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," in *Proceedings of the Parallel Problem Solving from Nature (PPSN-II '92)*, pp. 281–290, Elsevier Science, Brussels, Belgium, 1992.

[25] K.-M. Lee, T. Yamakawa, and K.-M. Lee, "Genetic algorithm for general machine scheduling problems," in *Proceedings of the 2nd International Conference on knowledge-Based Intelligent Electronic Systems*, pp. 60–66, IEEE, Adelaide, Australia, April 1998.

[26] L. Sun, X. Cheng, and Y. Liang, "Solving job shop scheduling problem using genetic algorithm with penalty function," *International Journal of Intelligent Information Processing*, vol. 1, no. 2, pp. 65–77, 2010.

[27] L. Wang and D.-Z. Zheng, "An effective hybrid optimization strategy for job-shop scheduling problems," *Computers & Operations Research*, vol. 28, no. 6, pp. 585–596, 2001.

[28] H. Zhou, Y. Feng, and L. Han, "The hybrid heuristic genetic algorithm for job shop scheduling," *Computers & Industrial Engineering*, vol. 40, no. 3, pp. 191–200, 2001.

[29] B. M. Ombuki and M. Ventresca, "Local search genetic algorithms for the job shop scheduling problem," *Applied Intelligence*, vol. 21, no. 1, pp. 99–109, 2004.

[30] J. F. Gonçalves, J. J. D. M. 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.

[31] L. Lin and X. Yugeng, "A hybrid genetic algorithm for job shop scheduling problem to minimize makespan," in *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA '06)*, pp. 3709–3713, IEEE, Dalian, China, June 2006.

[32] H. Zhou, W. Cheung, and L. C. Leung, "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm," *European Journal of Operational Research*, vol. 194, no. 3, pp. 637–649, 2009.

[33] L. Asadzadeh and K. Zamanifar, "An agent-based parallel approach for the job shop scheduling problem with genetic algorithms," *Mathematical and Computer Modelling*, vol. 52, no. 11-12, pp. 1957–1965, 2010.

[34] R. Yusof, M. Khalid, G. T. Hui, S. Md Yusof, and M. F. Othman, "Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm," *Applied Soft Computing Journal*, vol. 11, no. 8, pp. 5782–5792, 2011.

[35] M. Šeda, "Mathematical models of flow shop and job shop scheduling problems," *International Journal of Applied Mathematics & Computer Sciences*, vol. 4, no. 4, pp. 122–127, 2008.

[36] K.-H. Kim and P. J. Egbelu, "A mathematical model for job shop scheduling with multiple process plan consideration per job," *Production Planning and Control*, vol. 9, no. 3, pp. 250–259, 1998.

[37] W. Cheung and H. Zhou, "Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times," *Annals of Operations Research*, vol. 107, no. 1–4, pp. 65–81, 2001.

[38] S. Kirkpatrick, C . D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[39] S. Lawrence, "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques," Tech. Rep., GSIA, Carnegie Mellon University, 1984.

[40] G.-C. Luh and C.-H. Chueh, "A multi-modal immune algorithm for the job-shop scheduling problem," *Information Sciences*, vol. 179, no. 10, pp. 1516–1532, 2009.

[41] J.-H. Yang, L. Sun, H. P. Lee, Y. Qian, and Y.-C. Liang, "Clonal selection based memetic algorithm for job shop scheduling problems," *Journal of Bionic Engineering*, vol. 5, no. 2, pp. 111–119, 2008.

[42] S. M. K. Hasan, R. Sarker, and D. Cornforth, "Hybrid genetic algorithm for solving job-shop scheduling problem," in *Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS '07)*, pp. 519–524, IEEE, Melbourne, Australia, July 2007.

[43] S. Binato, W. J. Hery, D. M. Loewenstern, and M. G. Resende, "A GRASP for job shop scheduling," in *Essays and Surveys on Metaheuristics*, pp. 59–79, 2002.

[44] I. Sabuncuoglu and M. Bayiz, "Job shop scheduling with beam search," *European Journal of Operational Research*, vol. 118, no. 2, pp. 390–412, 1999.

[45] W. P. M. Nuijten and E. H. L. Aarts, "A computational study of constraint satisfaction for multiple capacitated job shop scheduling," *European Journal of Operational Research*, vol. 90, no. 2, pp. 269–284, 1996.