Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 210680, 12 pages
http://dx.doi.org/10.1155/2015/210680

Research Article

Hybrid Genetic Algorithm with Multiparents Crossover for Job Shop Scheduling Problems

Noor Hasnah Moin, Ong Chung Sin, and Mohd Omar

Institute of Mathematical Sciences, Faculty of Science, University of Malaya, 50603 Kuala Lumpur, Malaysia

Correspondence should be addressed to Noor Hasnah Moin; noor_hasnah@um.edu.my

Received 13 June 2014; Revised 8 September 2014; Accepted 8 September 2014

Academic Editor: Shifei Ding

The job shop scheduling problem (JSSP) is one of the well-known hard combinatorial scheduling problems. This paper proposes a hybrid genetic algorithm with multiparents crossover for JSSP. The multiparents crossover operator known as extended precedence preservative crossover (EPPX) is able to recombine more than two parents to generate a single new offspring distinguished from common crossover operators that recombine only two parents. This algorithm also embeds a schedule generation procedure to generate full-active schedule that satisfies precedence constraints in order to reduce the search space. Once a schedule is obtained, a neighborhood search is applied to exploit the search space for better solutions and to enhance the GA. This hybrid genetic algorithm is simulated on a set of benchmarks from the literatures and the results are compared with other approaches to ensure the sustainability of this algorithm in solving JSSP. The results suggest that the implementation of multiparents crossover produces competitive results.

1. Introduction

The job shop scheduling problem (JSSP) is one of the well-known hard combinatorial scheduling problems and it has been studied extensively due its practical importance. During the past three decades, optimization strategies for JSSP ranging from exact algorithms (mathematical programming) to approximation algorithms (metaheuristics) have been proposed [1]. Exact methods which are based on exhaustive enumeration and decomposition method guarantee global convergence and have been successfully applied to small instances. But, for the moderate and large scale instances, they require very high computational time to be either ineffective or inefficient [2]. Therefore, a lot of researchers focused their attention on approximation methods. Metaheuristics is one of the approximation methods that were proposed in the literatures to deal with JSSP which include tabu search (TS) [3], simulated annealing (SA) [4], genetic algorithm (GA) [5], and discrete artificial bee colony (DABC) [6].

In recent years, since the first use of GA based algorithm to solve the JSSP proposed by Davis [7], various GA strategies are introduced to increase the efficiency of GA to find the optimal or near optimal solutions for JSSP [8]. In the GA strategies, hybridization of GA with local search methods provides good results in solving the problems where GA capitalizes on the strength of the local search in locating the optimal or near optimal solutions. For example, Gonçalves et al. [9] and Zhang et al. [10] embedded the local search procedure of Nowicki and Smutnicki [11] into GA due to the effectiveness of the local search that increases the performance of GA. Qing-Dao-Er-Ji and Wang [12] proposed new crossover operator and mutation operator together with local search in improving local search ability of GA.

Additionally, the structure of the GA can be modified and enhanced to reduce the problems often encountered in GA. Yusof et al. [13] implemented a migration operator in GA by using parallelization of GA (PGA) to find the near optimal solutions. Watanabe et al. [14] proposed a GA with search area adaption and a modified crossover operator for adapting to the structure of the solutions space. Ripon et al. [15] embedded heuristic method in the crossover to reduce the tail redundancy of chromosome. Particularly,

recombination operators, especially crossover operators, play important roles in the structure of GA.

Crossover between two parents is traditionally adopted in GA [8] for JSSP but the GA can be modified accordingly to suit the problem at hand including selecting several numbers of parents for the crossover operation which is known as multiparents crossover.

The application of multiparents recombination can be found in different research areas. Mühlenbein and Voigt [16] proposed gene pool recombination (GPR) in solving discrete domain problems. Eiben and van Kemenade [17] introduced the diagonal crossover as the generalization of uniform crossover in GA for numerical optimization problems. Wu et al. [18] proposed multiparents orthogonal recombination to determine the identity of an unknown image contour. The crossover operators that were used in those areas show the good search ability of the operator but are very much problem dependent.

The above literatures indicated the ascendency of multiparents crossover over two parents' crossover. Although multiparents crossover has been used in different fields, to the best of our knowledge, only limited numbers are applied to combinatorial scheduling problems and none has been proposed for JSSP. In particular, Eiben et al. [19] proposed multiparent for the adjacency based crossover (ABC) and Ting et al. [20] developed multiparent extension of partially mapped crossover (MPPMX) for the travelling salesman problems. Although the experimental results point out that ABC of multiparents has no tangible benefit, MPPMX shows significant improvement in the use of multiparents in crossover. In other words, one would expect that, by biasing the recombination operator, the performance of the GA would improve.

In this paper, we propose extended precedence preservative crossover (EPPX) as a multiparent crossover. EPPX is based on the precedence preservative crossover (PPX) proposed by Bierwirth et al. [21]. Because of its ability to preserve the phenotypical properties of the schedules. EPPX as crossover operator will retain this advantage in the GA. EPPX is used in GA in conjunction with neighborhood search to solve JSSP. The rest of the paper is organized as follows. JSSP and the different types of schedules are described in detail in the next section. In Section 3, we present our approach to solve the JSSP: chromosome representation, schedule generation procedure, neighborhood search procedure, and GA with multiparents crossover. Section 4 provides experimental results and analysis. The conclusions are drawn in Section 5.

2. Problem Definition

2.1. Job Shop Scheduling Problem (JSSP). The JSSP can be defined as a set of n jobs that need to be processed on a set of m machines. A job consists of a set of operations J, where O_{ij} represents the jth ($1 \le j \le J$) operation of the ith ($1 \le i \le n$) job. The technological requirement for each operation processing time is denoted as p_{ij} and a set of machines is denoted by M_k ($1 \le k \le m$).

Table 1: Example for 3-job and 3-machine problem.

	Job	Operation routing		
		1	2	3
Processing time	1	3	3	2
	2	1	5	3
	3	3	2	3
Machine sequence	1	M1	M2	M3
	2	M1	M3	M2
	3	M2	M1	M3

Precedence constraint of the JSSP is defined as [22] operation jth must finish before operation jth + 1 in the job. A job can visit a machine once only. Only one operation at a time for one time is allowed to be processed in a machine. It is assumed that the delay time for the job transfer machine will be neglected and operation allocation for machine will be predefined. Preemption of operations is not allowed. There are no precedence constraints among the operations of different jobs.

The main objective of JSSP is to find the minimum makespan for the scheduling. The finish time of job i with last operation, J, is represented by F_{iJ}. The time for the whole schedule to complete or the makespan is also the maximum finish time of a set of the jobs i. Therefore, the makespan is expressed as follows:

$$\text{Makespan} = \max\left(F_{iJ}\right). \tag{1}$$

Let $G(k)$ be the set of operations being processed in machine k and let

$$X_{O_{ij},k} = \begin{cases} 1 & \text{if } O_{ij} \text{ has been assigned to machine } k \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The conceptual model of the JSSP is shown as below:

$$\text{Minimize}\left\{\max\left(F_{iJ}\right)\right\} \tag{3}$$

$$F_{ij} \le F_{ij+1} - p_{ij+1}, \quad j = 1, 2, \ldots J, \quad \forall i \tag{4}$$

$$\sum_{O_{ij} \epsilon G(k)} X_{O_{ij},k} \le 1, \quad \forall k. \tag{5}$$

The objective function represented by (3) minimizes the maximum finish time in the set of the jobs i and therefore minimizes the makespan. Equation (4) satisfies precedence relationships between operations and (5) imposes that an operation can only be assigned to a machine at a time.

Table 1 shows an example of 3 jobs and 3 machines and their sequences for JSSP.

2.2. Type of Schedules. Three types of feasible schedule are considered: semiactive, active, and nondelay schedule [22]. In a semiactive schedule, there are no operations that can be started earlier without altering the sequences of the operations. In a semiactive schedule the makespan may often be reduced by shifting an operation to the left without delaying

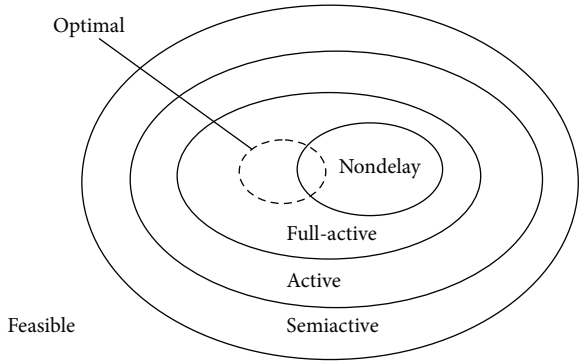


Figure 1: Relationship of the schedules.

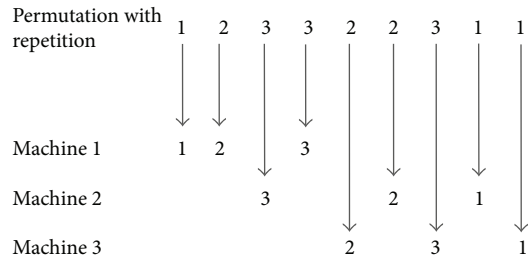


Figure 2: Permutation with repetition representation for 3 jobs and 3 machines.

other jobs. Reassignment of operations is called a permissible left shift. An active schedule is a schedule with no more permissible left shifts. Therefore, the set of active schedules is a subset of semiactive schedules. In a nondelay schedule, no machine is kept idle at a time when it could begin processing other operations and hence the set of nondelay schedules is a subset of active schedules. In addition the search space can be further reduced by implementing a full-active schedule introduced by Zhang et al. [10]. A full-active schedule is defined as a schedule where there is no more permissible left or right shift. Optimal solution of the scheduling always lies in the full-active schedule. Figure 1 illustrates the interaction of the schedules.

In order to generate the full-active schedule, we employ a scheduling approach called iterative forward-backward pass [23] in Section 3.1.2 which performs a kind of local search that can be used to introduce heuristic improvement into genetic search.

3. Hybrid Genetic Algorithm

GA is a stochastic search optimization technique that mimics the evolutionary processes in biological systems. This approach begins with a population, which represents a set of potential solutions in the search space. Each individual in the population is assigned a value by GA according to a problem specific objective function. The individuals will attempt to combine the good features in each individual in the population using a reproduction operator step such as crossover or mutation in order to construct individuals which are better suited than previous individuals. Through this evolution process, individuals that are less fit tend to be replaced by fitter individuals to generate a new population which eventually the desired optimal solutions will be found.

3.1. Schedule Generator Procedure

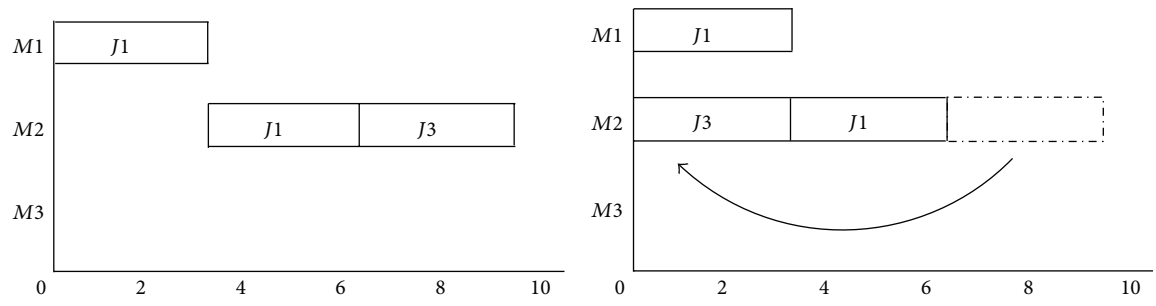
3.1.1. Chromosome Representation and Decoding. Representation of JSSP in chromosome is classified by Cheng et al. [22] into two approaches: direct and indirect. The direct approach directly encodes and decodes the representation into schedule while the indirect approach needs a schedule builder to encode and decode the chromosome into the schedule. The indirect approach is adopted in this study to represent the individual.

The chromosome in this paper is represented as operation based representation. In this representation, i represents the number of jobs $i = 1, 2, 3 \ldots, n$ and is repeated according the total number of operations in the job. Figure 2 illustrates the representation of 3 jobs and 3 machines. The chromosome is represented as [1 2 3 3 2 2 3 1 1], where jobs 1, 2, and 3 are represented as numbers 1, 2, and 3 in the chromosome, respectively. The number of occurrences in each chromosome depends on the number of operations required. The chromosome is scanned from left to right and at the jth occurrence of a job number refers to the jth operation in the technological sequence of this job. The chromosome created is always feasible and legal.

A scheduling can be built by constructing a schedule builder that performs a simple local search to decode the genes of the chromosome from left to right to a list of ordered operations. The first operation in the list is scheduled first, then the second operation, and so on. The operation will always be shifted to the left until time is equal to zero or inserted into a blank time interval between operations to find the earliest completion time. The process is repeated until all operations are scheduled. A schedule generated by the procedure can be guaranteed to be an active schedule [22]. Figure 3(a) illustrates the scheduling encoded by following operation sequences in the chromosome [1 1 3 $\cdots$]. Applying the process will enable the job to find possible earlier start time before being appended as last operation in the machine (Figure 3(b)) and the chromosome encoded is transformed to [1 3 1 $\cdots$]. In this representation, two or more chromosomes decoded may be translated into an identical schedule.

3.1.2. Schedule Generation Procedure. The procedure used to construct full-active schedule is based on a scheduling scheme called iterative forward-backward pass. This approach was proposed by Lova et al. [23] which had been shown to produce significant improvement in reducing makespan in the other field of scheduling problems.

The algorithm described in Section 3.1.1 uses a forward pass approach to generate an active schedule with the makespan $\max(F_{iJ})$ in the schedule in which the operations are able to shift left until time is equal to zero. Backward pass is a reverse process of the forward pass where the operations



(a) Before left shifting (b) After left shifting

Figure 3: Permissible left shift for semiactive schedule.

in a schedule start from end of the schedule and end at the beginning of the schedule $\min(F_{i1})$, in which the operations are able to shift right until time is equal to $\max(F_{iJ})$.

The iterative forward-backward pass approach can be described with the following steps.

Step 1. An active schedule chromosome is generated by forward pass with maximum makespan $\max(F_{iJ})$.

Step 2. Apply backward pass on the chromosome from Step 1. Chromosome is scanned starting from right to left to generate a schedule with start time $\max(F_{iJ})$ (Figure 4). Through the right shifting in the schedule, we can obtain the beginning of the schedule $\min(F_{i1})$ and the makespan of this schedule is given as

$$BC_{\max} = \max\left(F_{iJ}\right) - \min\left(F_{i1}\right). \quad (6)$$

$BC_{\max}$ and $\min(F_{i1})$ denote makespan and minimum time, respectively, in this new schedule. The new schedule is encoded into a new chromosome with makespan $BC_{\max}$.

Step 3. If $BC_{\max} < \max(F_{iJ})$, the makespan obtained from the schedule in Step 2 is less than the makespan in the schedule of Step 1, there is improvement of the schedule makespan, and then the new chromosome is used in Step 1; otherwise the active schedule chromosome and the makespan generated by forward pass are maintained. Steps 1 and 2 are repeated until there is no further improvement on the schedule (Figure 5).

In this iterative function, the makespan of both processes is mutually restricted and hence the makespan of new solution generated either is lesser or remains unchanged. The last schedule generated by forward pass is equivalent to the last schedule generated by backward pass with the same makespan and they are sharing the same critical path.

Figure 6 shows the steps where each chromosome generated by the GA employed the schedule generator procedure. The procedure starts from generating the full-active schedule by schedule generation procedure and then applying the neighborhood search to improve the schedule obtained. After improvement of the schedule ends, the corresponding quality of the makespan is obtained. The neighborhood search is presented in the next section.

3.1.3. Neighborhood Search Procedure. Reduction of the search space does not guarantee the optimal solution will be found. Therefore we use a neighborhood search as an exploitation mechanism to decrease the makespan. This mechanism is restricted to searching the possible solutions in a critical path that consists of longest sequences of operation in a schedule. Swapping the operations on the path by using neighborhood search significantly reduces the total length of the makespan [11].

Instead of the swap operation which is determined deterministically as in [11], we modify the operation such that we chose the operations to be swapped randomly in a critical block. The neighborhood search starts with the identification of the critical path in the schedule generated by the scheduling process. Operations on the critical path are called critical operations. A critical block consists of a maximal sequence of adjacent critical operations that are processed on the same machine [11]. Our neighborhood is defined as the random swap between two jobs in a critical block that contains two or more operations. No swap is made if the critical block contains only one operation.

All possible moves of the operations are predetermined (Figure 7). A swap of the operations is accepted if it improves the makespan; otherwise the operations remain unchanged. Once the swap is accepted, the new critical path must be identified. The procedure is repeated and stops if there is no swap that can improve the makespan. Pseudocode for the neighborhood search is presented in Algorithm 1.

3.2. Hybrid GA. In our proposed hybrid GA, the search methods will be based on intensification and diversification mechanisms. The neighborhood search acts as an intensification mechanism that exploits the better solution in an individual and GA functions as diversification mechanism that explores the search space to provide different individuals for the local search. Structure of the hybrid GA that is based on a standard GA may be represented by the pseudocode in Algorithm 2.

Population is initialized randomly with a set of parameters and a particular chromosome which is often referred to as an individual. Each individual is evaluated by a fitness function and the parents selected for the recombination, which favors fitter individuals. The selected chromosomes

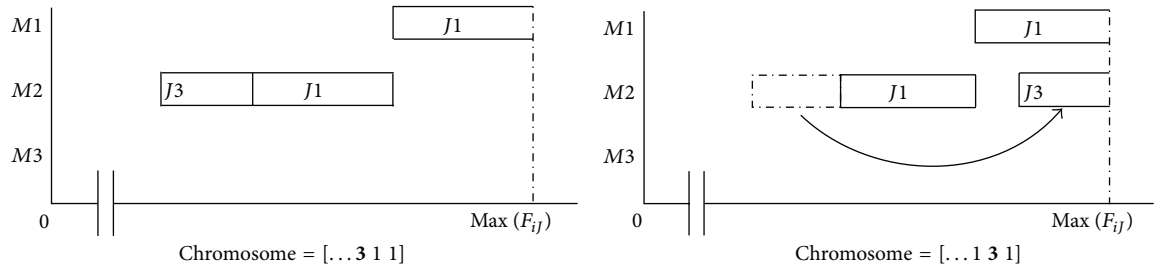


Figure 4: Backward pass.

```
while New solution accepted = true
New solutions accepted = false
Determine the critical path, critical block in New schedule
List out the possible swaps of the operations
    for        k = 1 to total of possible swaps do
               Swap a pair of operations
               New schedule generated and makespan recalculated (New makespan)
               if        New makespan < Current makespan
                                 Current makespan = New Makespan
                                 New solution accepted = true
               end if
    end for
end while
```

Algorithm 1: Pseudocode for neighborhood search.

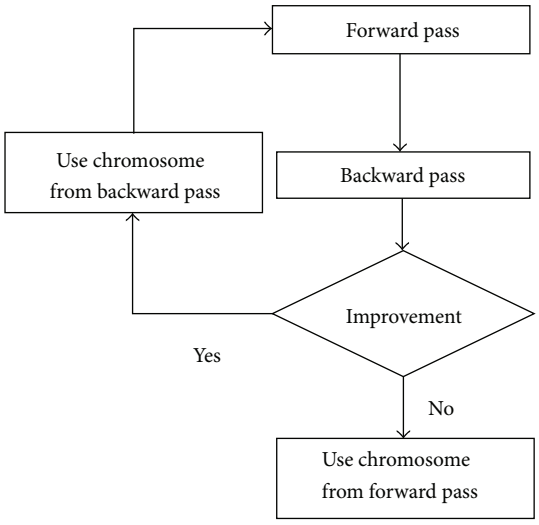


Figure 5: Iterative forward-backward pass.

are recombined using mechanisms of crossover and mutation to produce offspring. These offspring are then evaluated and then elitism strategy is applied to the new offspring with best fitness to replace the worst individuals in the previous population to generate a new population. Termination criterions of the GA in JSSP are set to terminate once the GA had achieved the optimal solution (if have) or maximum number of generations is reached.

3.2.1. Fitness and Selection Method. In this paper, we use nonlinear ranking to rank the evaluated chromosomes and each chromosome competes with the others and the selected chromosome survives to the next generation based on the fitness value (objective function). Chromosome with greater fitness indicates the greater probability to survive. The highest ranking chromosome in a population is considered the best solution. It is noted that the lowest makespan is given the highest ranking.

Stochastic universal sampling (SUS) is applied to select the parents for recombination in our GA. SUS is one of the selection methods that are often used in practice because they have less stochastic noise and have a constant selection pressure [24]. This fitness based proportionate selection will choose the chromosomes process with minimum spread and zero bias. SUS uses N equally spaced pins on wheel, where N is the number of selections required. The population is shuffled randomly and a single random number in the range $[0 - \sum f_i/N]$ is generated in which $\sum f_i$ represents the sum of all the fitness values of the individuals in the population.

3.2.2. Proposed Extended Precedence Preservative Crossover (EPPX). Generally in natural biological system the reproduction takes place between two parents (bisexual) or in a single

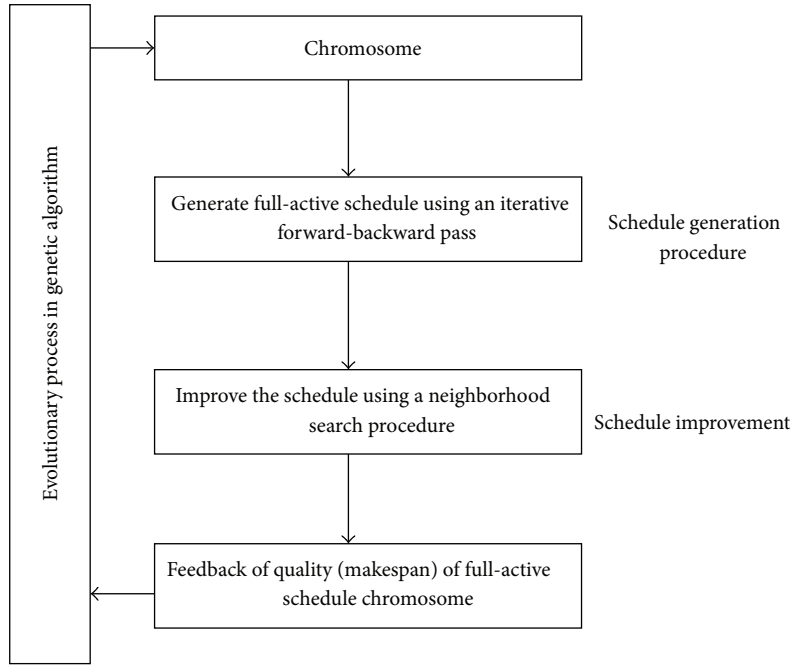


Figure 6: Architecture of schedule generation procedure.

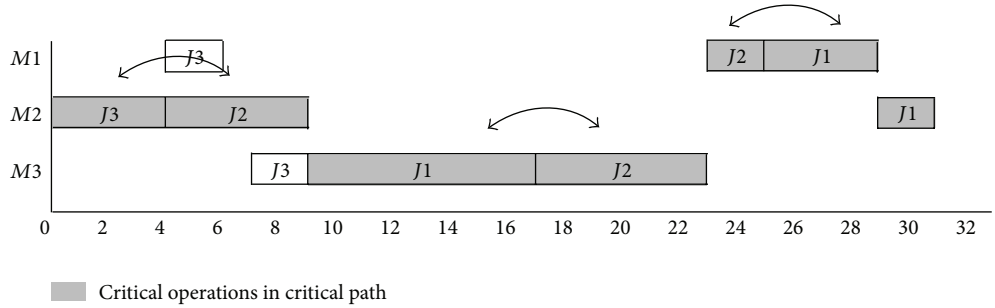


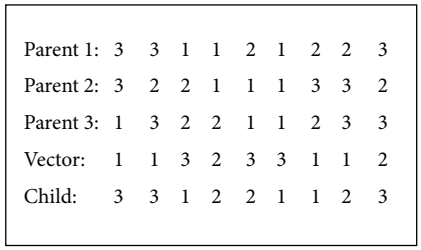
Figure 7: Critical path, critical operations, and possible operations swaps.

```
Initialize population
Evaluation
while termination criterion = false
        Selection
        Crossover
        Mutation
        Evaluation
        Reinsertion
End while
```

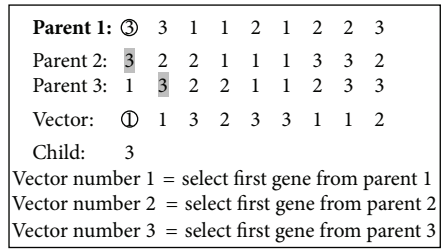
Algorithm 2: Pseudocode for a standard GA.

parent (asexual). However from the computational perspective, there is no restriction on the number of parents to use. Therefore some of the multiparents crossover operators are extended from the two-parent crossover operators [17–20] for recombination. As mentioned before, our proposed EPPX is an extension of PPX. A crossover mask in the form of a vector is generated randomly to determine in which parent the genes, specified in the mask, are to be selected for recombination. The multiparents will then recombine into a single offspring (Figure 8(a)). Starting from the first element on the mask vector, number 1 in the first element of the mask vector indicates that the first gene in that parent 1 is selected. In general, the mask vector indicates from which parent the element is to be selected. In this example, the selected job (job 3) is selected and eliminated in the other parents (Figures 8(b) and 8(c)). The second element in the mask indicates that the first element (after deletion) is to be selected also from parent 1 (Figure 8(c)). The third element in the mask shows that the first element in parent 3 is selected (Figure 8(d)). The process continues until all the elements in the mask have been examined. Algorithm 3 outlines the algorithm for multiparents crossover for JSSP.

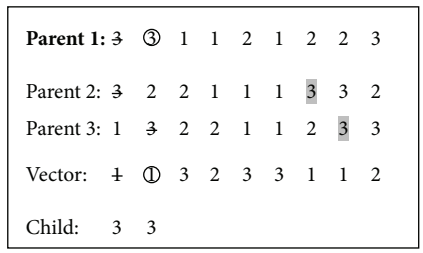
3.2.3. Mutation. Mutation is a genetic operator, analogous to the biological mutation, which is used to maintain genetic



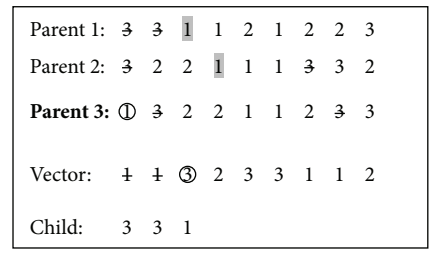
(a) Example of EPPX



(b) First step—vector number 1; the first gene in parent 1 is selected and the same job from the other parents is removed



(c) Second step—previous selected gene will be deleted, first gene (after deletion) in parent 1 is selected, and the job from the other parents are removed



(d) Repeat—the process will continue until at the end of the vector

Figure 8: EPPX crossover.

Crossover vector generated randomly
Select three parents: → *S1*, *S2*, and *S3*
for $k = 1$ **to** length of the chromosome **do**
Select vector number by position kth starting from the left element in the vector
case
Vector number 1:
Choose first gene at left most *S1*
Search same job number at left most in *S2* and *S3*
Remove the first gene in *S1* and gene searched in *S2* and *S3*
Vector number 2:
Choose first gene at left most *S2*
Search same job number at left most in *S1* and *S3*
Remove the first gene in *S2* and gene searched in *S1* and *S3*
Vector number 3:
Choose first gene at left most *S3*
Search same job number at left most in *S1* and *S2*
Remove the first gene in S3 and gene searched in *S1* and *S2*
end case
Selected gene insert to new chromosome by sequence from left to right
end for

Algorithm 3: Pseudocode for EPPX (3 jobs and 3 machines).

diversity from one generation of a population of chromosomes to the next. In this study, the mutation is applied by selecting two genes in different positions and different jobs inside the same chromosome to be swapped. The process will be repeated if two genes selected are at the same position or the same job. Figure 9 illustrates the swapping of the two genes in the chromosome.

4. Experiment Setup and Results

4.1. Experimental Setup. To test the performance of our hybrid GA, we consider the benchmark from four classes of different JSSP test problems: instances FT06, FT10, and FT20 from Fisher and Thompson [25], instances ORB01 to ORB10 from Applegate and Cook [26], instances ABZ5 to

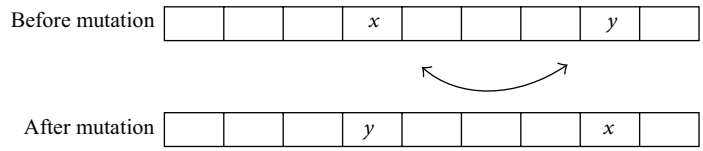


Figure 9: Mutation by swapping two genes in the chromosome.

Table 2: Output for different crossover rates and mutation rates.

	Mut_01	Mut_02	Mut_03	Mut_04	Mut_05	Mut_06	Mut_07	Mut_08	Mut_09	Mut_10	Average	Relative error (%)
Crs_10	971.96	978.20	972.64	968.06	969.28	972.34	971.58	965.46	963.24	968.12	970.09	4.31
Crs_9	967.38	975.94	965.82	968.24	970.30	965.36	963.44	961.14	961.00	960.32	965.89	3.86
Crs_8	971.54	968.00	967.24	965.92	966.10	964.28	964.58	958.92	958.84	959.12	964.45	3.70
Crs_7	973.94	968.32	969.26	963.72	963.48	961.58	958.46	957.84	959.02	958.94	963.46	3.60
Crs_6	972.96	968.32	969.30	965.26	964.02	965.10	964.08	960.16	959.90	959.58	964.87	3.75
Crs_5	971.88	975.96	975.60	969.12	962.58	967.36	960.92	961.60	961.04	957.10	966.32	3.90

Crs_10 represents crossover rate at 1.0, Crs_9 represents crossover rate at 0.9, and so on.
Mut_01 represents mutation rate at 0.1, Mut_02 represents mutation rate at 0.2, and so on.

ABZ9 from Adams et al. [27], and instances YN1 to YN4 from Yamada and Nakano [28]. The instances that we use were downloaded from http://people.brunel.ac.uk, OR-Library of Brunel University.

MATLAB 7.11 R2010b is used to develop the GA and the simulations are run using workstation Intel Xeon 12 GB RAM with 2.4 GHz processor. The population size comprises 100 individuals for problems FT06, FT10, ABZ5, ABZ6, and ORB01–ORB10. In large sized problems (FT20, ABZ7–ABZ9, and YN1–YN4), their population sizes are increased to 150. The number of parents for EPPX ranges from 3 to 10. Crossover rate and mutation rate are set to 0.7 and 1.0, respectively. After the recombination process, elitism strategy is applied; 10% of the fittest new offspring replaces 10% of the worst individuals in the previous population to generate a new population. Each instance is run for 50 times for different numbers of parents to find out their best solution available.

4.1.1. Maximum Number of Generations. In the multiparent crossover, the parents will be recombined to generate one child. Therefore, the relationship of different numbers of parents with different total solutions (offspring) for a population exists. Thus the total solutions are defined as

$$\text{total solutions} = \frac{\text{MAXGen} \times \text{population size}}{\text{number of parents}}. \quad (7)$$

The maximum number of generations is denoted as MAXGen. To be fair, the total number of generations is adjusted to make sure that different numbers of parents for recombination generate approximately the same number of total solutions by referring to (8). Total solutions for instances FT06, FT10, ABZ5 and ABZ6, and ORB01–ORB10 after recombination process are set at around 5000 schedules. In addition, problems FT20, ABZ7–ABZ9, and YN1–YN4 are set at around 10,000 schedules. Among the multiparents crossover, three-parent crossover operator is used as reference because we consider them as the starting point of multiparents recombination (more than two parents). Consider

$$\text{MAXGen} = \frac{\text{total solutions} \times \text{number of parents}}{\text{populations size}}. \quad (8)$$

4.1.2. Crossover Rate. In our cases, due to the different problems sizes and different numbers of parents for recombination, we need to fix crossover rate and mutation rate to use them for all numbers of parents. Instance FT10 is selected for the parameters testing because it is considered as a difficult problem. Among the multiparents crossovers, three-parent crossovers are used because we consider them as the starting point of multiparents recombination (more than two parents).

The dependencies between the crossover and mutation rates are tested by the GA. The crossover rates are set from 1.0 to 0.5 with varied mutation rates from 0.1 to 1.0. Each case (example: crossover rate = 1.0, mutation rate = 0.1) will be run for 100 times and the average of each case will be figured out. The relative errors are calculated by computing the difference between the average solutions for each crossover rate and the optimal solution of FT10 (930).

In Table 2, the relative error for crossover rate at 0.7 appears as the lowest value compared to the other crossover rates. The frequencies of optimal solutions for FT10 at the crossover rate 0.7 that obtain from the simulation had shown the highest occurrences as indicated in Figure 10. Thus, it is reasonable for us to use the crossover rate at 0.7 to compare to other values.

4.1.3. Mutation Rate. In the multiparents crossover application, especially in the JSSP, there is a lack of information about the mutation rate values. Hence, we try to find the suitable mutation rate for our GA. Due to the inconsistencies of the results between the crossover rates and mutation rates we obtained from the simulation, Figure 11 plotted the best fit line for the problems. All lines for the different crossover

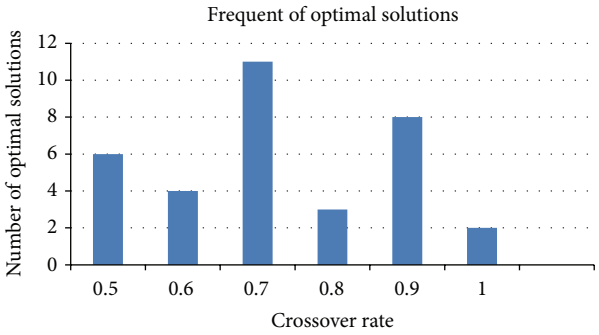


Figure 10: Frequency of optimal solutions appears (FT10) at different crossover rates.

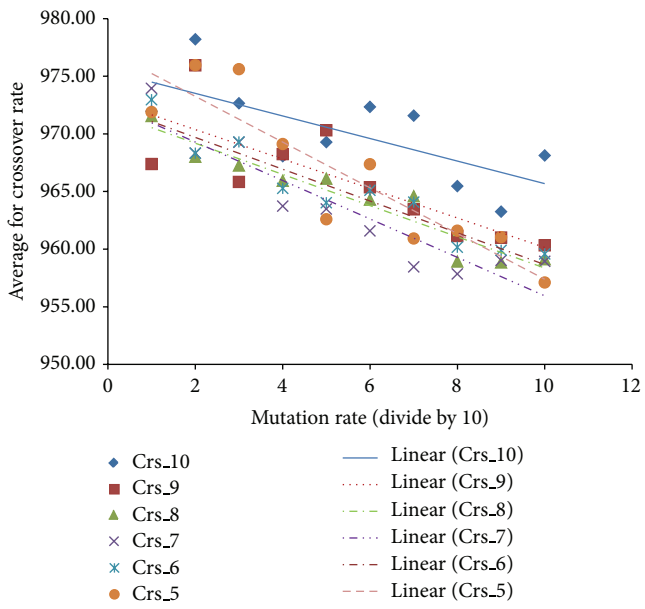


Figure 11: Best fit line for crossover with different mutation rates.

rates are decreasing from left to right which means that the average solutions will get better when the mutation rates increase. Thus, we conclude that the last mutation rate (1.0) performs the best when applied in this GA and it will be used as parameter for other instances and problems.

4.2. Results and Discussions

4.2.1. Comparison with the Other Permutation Crossover Operator That uses Two Parents. Table 3 lists the GAs using permutation crossover operator with various hybridization or modification that had been applied on the FT problem. The GAs in the list (except EPPX) use only two parents for crossover operation and the results vary because of different strategies of hybridization. Initially the earlier papers [21, 29] was unable to obtain the optimal solutions for FT10 using the proposed crossover operators. Then later by hybridization or modification using this method, the optimal results of FT10 can be reached, including our EPPX. EPPX is able to perform well in the FT20 as compared to the GP-GA and IPPX with the deviation of 1.12%. The acceptable ranges of comparable GAs are up to 7% [21]. Tested results show that EPPX is able to get the solution within these values and is considered applicable for solving the JSSP. Instance FT06 can be solved easily with the number of parents used for crossover ranging from 3 to 10 parents. For difficult problems such as FT10, EPPX also performs well because it is able to get the optimum solution with the number of parents of 3 and 5 in crossover operation. As the variable of JSSP increases (instance FT20), a deviation between optimal solution and EPPX which obtains the best solution of 1178 by using 6 parents in EPPX's crossover operation occurs.

4.2.2. Comparison with Different Algorithms. ORB problem contains ten 10 × 10 instances. To illustrate the effectiveness, the proposed multiparent crossover is compared with the other algorithms such as synergistically combining small and large neighborhood schemes (SLENP) [31], parallel genetic algorithms (PGA) [13] with two-parent crossover, and greedy randomized adaptive search procedure (GRASP) [32]. The relative errors (REs) are calculated based on deviation between BKS and the best solution found (Best). In EPPX, different numbers of parents are used for recombination. The numbers of parents (MP) that achieve the best solutions are recorded in the table.

Table 4 states the name, size (number of jobs x number of machines), the best known solutions (BKS), the best solutions found (Best), multiparents that obtain the best solutions (MP), and relative error (RE) for each test instance. The RE is calculated from the gap between Best and BKS in percentage. The effectiveness of the proposed algorithm is analyzed by referring to the total relative error, total RE, in the last lane. In the ORB problem, EPPX found the BKS in 5 instances. The best solutions found are located in the different numbers of parents for different instances. Total RE provided by EPPX is lower than GRASP but higher than SLENP and PGA. This shows that EPPX performs averagely among these methods.

ABZ problem contains five instances with the sizes 10×10 and 20 × 15. ABZ8 and ABZ9 and no optimal solutions have been known. Table 5 shows the detailed results of comparison for ORB instances. EPPX is able to perform better as compared to PGA and GRASP with lower total RE.

YN problem consists of four instances with size of 20 × 20 that are still open problems. EPPX is compared with TSSA and PGA in Table 6 because algorithm GRASP did not provide the results of each instance.

4.2.3. The Performance Evaluation of EPPX. FT, ORB, ABZ, and YN problems are used to evaluate the performance of our hybrid GA which incorporates EPPX. Tables 3–6 show that, in 9 out of 22 cases, the EPPX has been able to find the best known solution for the corresponding benchmark instance. Table 4 shows that our results are competitive when compared to PGA achieving the same solutions in 5 out of 10 instances. However when the problem size increases (Tables 5 and 6) the performances of our algorithm are better with smaller relative error compared to PGA and GRASP. The computational results indicated that EPPX is capable of

Table 3: Results for FT06, FT10, and FT20 with n jobs $\times$ m machines.

Author(s)	Year	Crossover operator	FT06 (6 × 6)	FT10 (10 × 10)	FT20 (20 × 5)
Optimum			55	930	1165
Gen et al. [29]	1994	Partial schedule exchange crossover	55	962	1175
Bierwirth et al. [21]	1996	Generalized permutation GP-GA	55	936	1181
Park et al. [30]	2003	Parallel genetic algorithm PGA	55	930	1173
Ripon et al. [15]	2011	Improved precedence preservation crossover IPPX	55	930	1180
	2014	Multiparents crossover EPPX	55	930	1178

Table 4: Results for ORB problems.

Instances	Size	BKS	EPPX			SLENP		PGA		GRASP	
			Best	RE	MP	Best	RE	Best	RE	Best	RE
ORB01	10 × 10	1059	1077	1.70	9	1059	0.00	1060	0.09	1070	1.04
ORB02	10 × 10	888	889	0.11	3, 4, 6, 7, 9	888	0.00	889	0.11	889	0.11
ORB03	10 × 10	1005	1022	1.69	7	1005	0.00	1020	1.49	1021	1.59
ORB04	10 × 10	1005	1005	0.00	5, 9	1005	0.00	1005	0.00	1031	2.59
ORB05	10 × 10	887	890	0.34	5	887	0.00	889	0.23	891	0.45
ORB06	10 × 10	1010	1021	1.09	4	1010	0.00	1013	0.30	1013	0.30
ORB07	10 × 10	397	397	0.00	10	397	0.00	397	0.00	397	0.00
ORB08	10 × 10	899	899	0.00	4	899	0.00	899	0.00	909	1.11
ORB09	10 × 10	934	934	0.00	10	934	0.00	934	0.00	945	1.18
ORB10	10 × 10	944	944	0.00	3–7, 9	944	0.00	944	0.00	953	0.95
	Total RE			4.93			0.00		2.22		9.32

Table 5: Results for ABZ problems.

Instances	Size	BKS	EPPX			SLENP		PGA		GRASP	
			Best	RE	MP	Best	RE	Best	RE	Best	RE
ABZ5	10 × 10	1234	1234	0.00	3	1234	0.00	1236	0.16	1238	0.32
ABZ6	10 × 10	943	943	0.00	7, 8	943	0.00	943	0.00	947	0.42
ABZ7	20 × 15	656	683	4.12	4	662	0.91	685	4.42	723	10.21
ABZ8	20 × 15	665	701	5.41	3	668	0.45	704	5.86	729	9.62
ABZ9	20 × 15	678	712	5.01	4	688	1.47	723	6.64	758	11.80
	Total RE			14.54			2.83		17.08		32.37

Table 6: Results for YN problems.

Instances	Size	BKS	EPPX			SLENP		PGA	
			Best	RE	MP	Best	RE	Best	RE
YN1	20 × 20	884	911	3.05	7	892	0.90	933	5.54
YN2	20 × 20	907	941	3.75	4	911	0.44	944	4.08
YN3	20 × 20	892	928	4.04	4	900	0.90	928	4.04
YN4	20 × 20	968	1011	4.44	6	982	1.45	1018	5.17
	Total RE			15.28			3.69		18.82

adapting to more difficult and larger size problems in comparison with PGA that only uses two parents for crossover. The best solutions found are located in the different numbers of parents for different instances indicated in column MP in all tables.

Table 7 displays the average computational times for different numbers of parents for each instance. Optimal solutions for easy problem (FT06) can be found in shorter time while, as for harder problem, the algorithm needs more time to find out the near optimal solutions. It can be seen that our algorithms require on average less than 15 seconds CPU time. The CPU times of EPPX, PGA, and GRASP are calculated per iteration meanwhile SLENP is based on the time per run.

The computational time for EPPX and PGA which are based on genetic algorithm is large for harder problems.

Table 7: Average computational time for different number of parents.

Instances	Size	Computational time (in second)										
		EPPX (number of parents)								SLENP	PGA	GRASP
		3	4	5	6	7	8	9	10			
FT 06	6 × 6	**0.02**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.01	0.76	0.70
FT 10	10 × 10	**0.93**	0.72	**0.57**	0.46	0.41	0.37	0.33	0.30	9.22	1.54	2.90
FT 20	20 × 5	**1.19**	0.89	0.73	0.61	0.52	0.45	0.40	0.38	2.73	1.52	4.30
ORB01	10 × 10	1.02	0.78	0.64	0.52	0.45	0.38	**0.37**	0.33	5.96	1.56	2.90
ORB02	10 × 10	**0.92**	**0.70**	0.57	**0.46**	**0.41**	0.35	**0.32**	0.29	0.48	1.56	3.80
ORB03	10 × 10	1.07	0.82	0.66	0.55	**0.48**	0.40	0.38	0.34	7.42	1.56	3.10
ORB04	10 × 10	0.93	0.73	**0.59**	0.47	0.41	0.36	**0.33**	0.30	7.58	1.56	3.10
ORB05	10 × 10	0.98	0.81	**0.60**	0.49	0.43	0.37	0.34	0.30	12.09	1.56	2.80
ORB06	10 × 10	1.11	**0.80**	0.64	0.53	0.46	0.40	0.37	0.33	9.17	1.56	3.10
ORB07	10 × 10	0.90	0.70	0.55	0.46	0.40	0.35	0.32	**0.29**	0.28	1.56	3.20
ORB08	10 × 10	1.01	**0.75**	0.63	0.51	0.46	0.39	0.35	0.33	6.02	1.56	3.10
ORB09	10 × 10	0.92	0.72	0.57	0.47	0.41	0.36	0.33	**0.30**	0.51	1.56	3.10
ORB10	10 × 10	**0.98**	**0.74**	**0.59**	**0.48**	**0.42**	0.36	**0.33**	0.30	0.18	1.56	2.90
ABZ5	10 × 10	**0.84**	0.64	0.51	0.41	0.36	0.31	0.29	0.26	3.55	1.78	0.30
ABZ6	10 × 10	0.79	0.61	0.53	0.43	**0.38**	**0.34**	0.31	0.28	0.15	1.56	3.10
ABZ7	20 × 15	6.17	**4.77**	3.78	3.29	2.79	2.42	2.09	1.97	64.82	7.36	17.40
ABZ8	20 × 15	**6.45**	4.76	3.85	3.21	2.77	2.52	2.05	1.93	55.94	7.28	18.20
ABZ9	20 × 15	5.83	**4.38**	3.53	2.97	2.49	2.19	1.96	1.79	35.82	7.32	17.10
YN1	20 × 20	12.19	9.32	7.56	6.46	**5.39**	4.66	4.13	3.79	40.04	7.60	—
YN2	20 × 20	11.72	**8.97**	7.32	6.00	5.18	4.48	3.95	3.80	62.31	7.60	—
YN3	20 × 20	14.09	**10.43**	8.56	7.13	6.14	5.28	4.71	4.45	42.18	7.60	—
YN4	20 × 20	10.52	7.75	6.38	**5.30**	4.51	3.89	3.45	3.26	56.05	7.60	—

However the large computational time displayed by SLENP and GRASP which are based on single solution is larger when compared to EPPX and PGA. PGA is expected to perform better in terms of computational time because the algorithm is run in parallel. Based on the calculation per iteration, EPPX is more efficient compared to PGA and GRASP because it requires less time to do the calculation.

The highlighted time in the EPPX shows the computational time of the result achieved by different number of parents. Multiparents are recombined to produce a single child resulting in a decreasing rate of computational time with the increasing number of parents. It is proven that increasing the number of parents in the EPPX may reduce the calculation time to obtain the optimal result in some instances.

5. Conclusion

This paper presents a hybrid genetic algorithm with multiparent crossover, EPPX, for the job shops scheduling problem. The hybrid GA combines genetic algorithm with neighborhood search in which GA explores the population while the neighborhood search exploits the individual solution to find better solution. The chromosome represented by operation based representation is used to generate a full-active schedule through iterative forward-backward pass which can further reduce the search space.

In the experimental results, EPPX using multiparent is able to get the solutions within the acceptable range of GA values. Results show that the best solutions are obtained from different numbers of parents for crossover; thus it is proven that GA is not restricted to two-parent crossover in order to find the best solution. The number of parents used in EPPX and GA is dependent on the problem and it may be observed that the best solutions for different instances are produced by different numbers of parents.

In future works, suitable ranges for number of parents for crossover are needed to determine and solve JSSP. It is also anticipated that the efficiency of GA can be increased by embedding other local searches into GA. Further research is necessary to reduce the computational time as a way to improve the efficiency of hybrid genetic algorithm with multiparent crossover for JSSP.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work has been supported by University of Malaya Research Grant Scheme UMRG-RG116-10AFR.

References

[1] A. Jones, L. C. Rabelo, and A. T. Sharawi, "Survey of job shop scheduling techniques," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, New York, NY, USA, 1999.

[2] S. Ólafsson, "Metaheuristics," in *Handbooks in Operations Research and Management Science*, vol. 13, pp. 633–654, 2006.

[3] C. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007.

[4] S. Z. Song, J. J. Ren, and J. X. Fan, "Improved simulated annealing algorithm used for job shop scheduling problems," in *Advances in Electrical Engineering and Automation*, vol. 139, pp. 17–25, Springer, Berlin, Germany, 2012.

[5] Z. Yaqin, L. Beizhi, and W. Lv, "Study on job-shop scheduling with multi-objectives based on genetic algorithms," in *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM '10)*, pp. v10294–v10298, IEEE, October 2010.

[6] M. Yin, X. Li, and J. Zhou, "An efficient job shop scheduling algorithm based on artificial bee colony," *Scientific Research and Essays*, vol. 6, no. 12, pp. 2578–2596, 2011.

[7] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, vol. 140, 1985.

[8] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies," *Computers & Industrial Engineering*, vol. 36, no. 2, pp. 343–364, 1999.

[9] J. F. Gonçalves, J. J. 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.

[10] C. Zhang, Y. Rao, and P. Li, "An effective hybrid genetic algorithm for the job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 39, no. 9-10, pp. 965–974, 2008.

[11] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

[12] R. Qing-Dao-Er-Ji and Y. Wang, "A new hybrid genetic algorithm for job shop scheduling problem," *Computers and Operations Research*, vol. 39, no. 10, pp. 2291–2299, 2012.

[13] 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.

[14] M. Watanabe, K. Ida, and M. Gen, "A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 48, no. 4, pp. 743–752, 2005.

[15] K. S. N. Ripon, N. H. Siddique, and J. Torresen, "Improved precedence preservation crossover for multi-objective job shop scheduling problem," *Evolving Systems*, vol. 2, no. 2, pp. 119–129, 2011.

[16] H. Mühlenbein and H. M. Voigt, "Gene pool recombination in genetic algorithms," in *Proceedings of the Metaheuristics Conference*, Kluwer Academic Publishers, Norwell, Mass, USA, 1995.

[17] A. E. Eiben and C. H. van Kemenade, "Diagonal crossover in genetic algorithms for numerical optimization," *Control and Cybernetics*, vol. 26, no. 3, pp. 447–466, 1997.

[18] A. Wu, P. W. M. Tsang, T. Y. F. Yuen, and L. F. Yeung, "Affine invariant object shape matching using genetic algorithm with multi-parent orthogonal recombination and migrant principle," *Applied Soft Computing Journal*, vol. 9, no. 1, pp. 282–289, 2009.

[19] A. E. Eiben, P.-E. Raué, and Z. Ruttkay, "Genetic algorithms with multi-parent recombination," in *Parallel Problem Solving from Nature—PPSN III*, vol. 866 of *Lecture Notes in Computer Science*, pp. 78–87, 1994.

[20] C.-K. Ting, C.-H. Su, and C.-N. Lee, "Multi-parent extension of partially mapped crossover for combinatorial optimization problems," *Expert Systems with Applications*, vol. 37, no. 3, pp. 1879–1886, 2010.

[21] C. Bierwirth, D. C. Mattfeld, and H. Kopfer, "On permutation representations for scheduling problems," in *Parallel Problem Solving from Nature—PPSN IV*, vol. 1141 of *Lecture Notes in Computer Science*, pp. 310–318, Springer, Berlin, Germany, 1996.

[22] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation," *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 983–997, 1996.

[23] A. Lova, C. Maroto, and P. Tormos, "Multicriteria heuristic method to improve resource allocation in multiproject scheduling," *European Journal of Operational Research*, vol. 127, no. 2, pp. 408–424, 2000.

[24] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[25] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local jobshop scheduling rules," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds., pp. 225–251, Prentice Hall, Englewood Cliffs , NJ, USA, 1963.

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

[27] 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.

[28] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," in *Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature (PPSN '92)*, R. Manner and B. Manderick, Eds., pp. 281–290, Brussels, Belgium, 1992.

[29] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problems by genetic algorithm," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Humans, Information and Technology*, vol. 2, pp. 1577–1582, October 1994.

[30] B. J. Park, H. R. Choi, and H. S. Kim, "A hybrid genetic algorithm for the job shop scheduling problems," *Computers & Industrial Engineering*, vol. 45, no. 4, pp. 597–613, 2003.

[31] M. Amirghasemi and R. Zamani, "A synergetic combination of small and large neighborhood schemes in developing an effective procedure for solving the job shop problem," *SpringerPlus*, vol. 3, no. 1, 2014.

[32] S. Binato, W. J. Hery, D. M. Loewenstern, and M. G. C. Resende, "A GRASP for job shop scheduling," in *Essays and Surveys in Metaheuristics*, pp. 59–79, Springer, 2001.