Unified Genetic Algorithm Approach for Solving Flexible Job-Shop Scheduling Problem

Jin-Sung Park 1, Huey-Yuen Ng 2, Tay-Jin Chua 2, Yen-Ting Ng 2 and Jun-Woo Kim 1,*  

Abstract: This paper proposes a novel genetic algorithm (GA) approach that utilizes a multichromosome to solve the flexible job-shop scheduling problem (FJSP), which involves two kinds of decisions: machine selection and operation sequencing. Typically, the former is represented by a string of categorical values, whereas the latter forms a sequence of operations. Consequently, the chromosome of conventional GAs for solving FJSP consists of a categorical part and a sequential part. Since these two parts are different from each other, different kinds of genetic operators are required to solve the FJSP using conventional GAs. In contrast, this paper proposes a unified GA approach that enables the application of an identical crossover strategy in both the categorical and sequential parts. In order to implement the unified approach, the sequential part is evolved by applying a candidate order-based GA (COGA), which can use traditional crossover strategies such as one-point or two-point crossovers. Such crossover strategies can also be used to evolve the categorical part. Thus, we can handle the categorical and sequential parts in an identical manner if identical crossover points are used for both. In this study, the unified approach was used to extend the existing COGA to a unified COGA (u-COGA), which can be used to solve FJSPs. Numerical experiments reveal that the u-COGA is useful for solving FJSPs with complex structures.

Keywords: flexible job-shop scheduling problem; combinatorial optimization; genetic algorithm; candidate order-based genetic algorithm; multichromosome

1. Introduction

Production scheduling is one of the most important decision-making procedures on manufacturing shopfloors, as it helps to utilize resources efficiently and maintain competitiveness in manufacturing companies [1–3]. Most production scheduling problems are NP-hard combinatorial optimization problems such that the optimal schedules are hard to find in polynomial time [4]. In this context, metaheuristic algorithms that can be used to find near optimal schedules in practical time are popular in production scheduling literature [5]. Examples of metaheuristic algorithms for solving production scheduling problems are the genetic algorithm (GA) [6], tabu search (TS) [7], simulated annealing (SA) [8], and particle swarm optimization (PSO) [9].

This paper aims to develop a novel GA for solving the flexible job-shop scheduling problem (FJSP). The classical job-shop scheduling problem (JSP) is one of the most well-known production scheduling problems. It consists of \( m \) machines and \( n \) jobs. A job contains a number of operations to be processed in a fixed order. In the JSP, each operation can be processed by one specific machine [10,11]. The FJSP is an extension of the classical JSP, which allows an operation to be processed by one of two or more machines. In other words, an operation is processed by one of alternative machines in the FJSP [12]. Since a machine for a specific operation is predetermined, the JSP can be solved by specifying
the priorities of given operations such that a high-priority operation precedes others with lower priorities in the queue for given machines. Thus, the JSP can be regarded as a sequencing problem [13,14]. In comparison with the JSP, the FJSP requires an additional decision related to which machine is used to process a specific operation. Consequently, two kinds of decisions, i.e., operation sequencing and machine selection, are required to solve the FJSP [1].

GA has been a popular metaheuristic algorithm for solving the JSP and FJSP in recent decades [10–12,15]. In order to apply GA, a solution of the given problem must be encoded in the form of a string, known as a chromosome [6]. One issue is that a chromosome for the FJSP must consist of two subchromosomes to account for the operation sequence (OS) part and the machine selection (MS) part. In other words, existing GAs for the FJSP are based on a multichromosome. Typically, the MS part is represented by a string of categorical values, whereas the OS part forms a sequence of given operations. Since the MS and OS parts have quite different structures, it is difficult to apply an identical genetic operator to both of them. Consequently, the MS and OS parts are separately evolved by different kinds of genetic operators in existing GAs for the FJSP. This traditional approach has the following limitations: First, it makes the GA hard to implement since different kinds of genetic operators should be applied. In particular, crossover operators for sequencing problems are a lot more complicated than simple one-point or two-point crossovers [16]. Second, there is no correlation between the order and machine in a single operation. In other words, the evolution of the MS part occurs independently from that of the OS part. Assume that an operation has the optimal order and is assigned to the optimal machine in a chromosome. Then, it is desirable that this combination of order and machine should also be maintained in the chromosome’s offspring, which is hard to achieve using the traditional approach. Consequently, the traditional approach can cause a loss of good combinations in chromosomes and unnecessary diversity in the population. To this end, this paper poses two research questions: how can we reduce the number of genetic operators in the GA for the FJSP, which will have a significant impact on the complexity of the GA? Can the GA with a reduced number of genetic operators, designed to maintain good combinations of OS and MS genes, deal with the FJSP effectively?

In order to overcome the limitations of the traditional approach, this paper proposes a novel unified GA approach for the FJSP, which enables the application of an identical crossover operator to both the OS and MS parts. The main idea of the proposed approach is to apply a candidate order-based GA (COGA) based on an identical crossover point to both the OS and MS parts. The COGA is a type of GA developed for solving sequencing problems. The most distinguished feature of the COGA is that simple point-based crossover strategies, including one-point and two-point and uniform crossovers, can be implemented [14]. This enables the application of an identical point-based crossover strategy with an identical crossover point to the OS and MS parts, which may help to maintain the combinations of good order and good machine in parent solutions.

The remainder of this paper is organized as follows: In Section 2, a literature review concerning the GA for the FJSP and COGA is provided. Section 3 outlines the concept of the unified GA approach, and procedures and genetic operators of the unified COGA (u-COGA) are introduced. Section 4 presents the experimental results obtained by applying the u-COGA to various benchmark FJSPs. Finally, Section 5 concludes the paper.

2. Research Background

2.1. Genetic Algorithm for Solving the Flexible Job-Shop Problem

FJSP is a well-known extension of the JSP, a classical production scheduling problem. During the past decades, GAs have been widely used to solve production scheduling problems, such as the JSP and FJSP [15]. Typically, a GA chromosome for the JSP contains only OS-type genes, since a solution for the JSP can be generated using a sequence of given operations [10,11,13,17]. In contrast, a GA chromosome for the FJSP typically consists of two types of genes: MS genes and OS genes, which are manipulated by applying different
kinds of genetic operators. Genetic operators adopted by previous research papers on the GA for the FJSP are summarized in Table 1.

Gao et al. [18] used extended order crossover (OX) and uniform crossover for the OS and MS parts, respectively. Furthermore, the authors proposed two kinds of mutation operators: random alternative for the MS part and immigration for the OS part.

In Pezzella et al. [19], the MS part is manipulated by assignment crossover and two mutation operators: random alternative mutation and greedy mutation. Assignment crossover is designed to exchange machine assignment information for certain operations. In other words, only some of the MS genes in a chromosome participate in a single assignment crossover. Random alternative mutation is used to assign an operation to a machine randomly chosen from the set of all alternative machines that process the operation. On the contrary, greedy mutation assigns an operation assigned to the machine with the maximum workload to the machine with the minimum workload. On the contrary, precedence preserving order-based crossover (POX) and precedence preserving shift (PPS) mutation were applied to the OS part. Pazzella et al. [19] used a GA with the aforementioned operators to minimize the makespan of FJSP. Moreover, Defersha and Chen [20] utilized a similar GA for the purpose of minimization of the makespan of the FJSP with sequence-dependent setup times.

Lei [21] applied the GA to solve the FJSP with fuzzy processing times. A two-point crossover operator was adopted for the MS part, whereas two kinds of sequencing crossover operators, i.e., generalized position crossover (GPX) and generalization of the precedence preservative crossover (GPPX), were used to recombine the genes in the OS part. Swap mutation was used to randomly modify the OS part; however, the GA in Lei [21] did not contain the mutation procedure for the MS part.

Wang et al. [22] applied multipoint preservative crossover (MPX) to the MS part and improved precedence operation crossover (IPOX) to the OS part. They used random alternative mutation and greedy mutation to randomly modify the MS part in the chromosome. The authors proposed a greedy mutation operator designed to assign an operation to a machine that provides the shortest processing time (SPT). For the OS part, conventional insertion mutation was adopted.

Zhang et al. [1] used classical two-point crossover and uniform crossover operators to recombine the genes in the MS part of parent solutions, while the genes in the OS part are recombined using preserving order-based crossover (POX). As in Wang et al. [22], the genes in the MS part are mutated by greedy mutation based on an SPT machine. In contrast, the mutation for the OS part is performed by applying swap mutation.

Jiang and Du [23] used two-point crossover for the MS part and POX for the OS part. The mutation operations for the MS part and OS part were random alternative and swap mutation, respectively.

Türkyılmaz and Bulkan [24] adopted three types of crossover operators for solving the FJSP. Uniform crossover was used to perform the crossover operation for the MS part, while two variations of POX, modified-1 POX (MPX1), and modified-2 POX (MPX2), were applied to the OS part. The mutation operators for the MS part and the OS part were random replace and swap mutation, respectively.

Zhang et al. [25] used two-point crossover for the MS part and POX for the OS part. In order to maintain diversity in the population, the authors applied greedy mutation based on an SPT machine to the MS part. Moreover, they introduced a local search-based mutation operator called the adaptive neighbor search method for the mutation of the genes in the OS part.

Table 1 shows that the minimization of the makespan is the most popular objective function in the FJSP literature. This paper also considers minimization of the makespan of the FJSP. In addition, all of the previous research papers utilized a multichromosome that contains the MS and OS parts, which is also the case in this paper.

Most of the relevant research papers applied different crossover operators to the OS and MS parts of the FJSP multichromosome. Various previous research papers applied a
GA with a single crossover operator to the FJSP [26–28]. However, they typically focused on one of the OS and MS parts, and details of the other part are missing. In such papers, the issue of genetic operators for the multichromosome was not discussed appropriately. In contrast, this paper applies a single crossover operator to the OS and MS parts in an evident manner.

**Table 1.** Existing GAs for solving the FJSP.

| Researchers            | Chromosome Structure | Crossover | Mutation | Objective                                      |
|------------------------|----------------------|-----------|----------|------------------------------------------------|
| Gao et al. [18]        | MS + OS              | MS: uniform OS: extended OX | MS: random alternative OS: immigration | Min. makespan, Min. maximal machine workload, Min. total workload |
| Pezzella et al. [19]   | MS + OS              | MS: assignment OS: POX | MS: random alternative, greedy (min. workload) OS: PPS | Min. makespan |
| Defersha and Chen [20] | MS + OS              | MS: assignment OS: POX | MS: random alternative, greedy (min. workload) OS: PPS | Min. makespan |
| Lei [21]               | MS + OS              | MS: two-point OS: GPX, GPPX | MS: N/A OS: swap | Min. Maximum fuzzy completion time |
| Wang et al. [22]       | MS + OS              | MS: MPX OS: IPOX | MS: random alternative, greedy (SPT machine) OS: insertion | Min. makespan, Min. total workload of machines, Min. critical machine workload |
| Zhang et al. [1]       | MS + OS              | MS: two-point, uniform OS: POX | MS: random alternative, greedy (SPT machine) OS: swap | Min. makespan |
| Jiang and Du [23]      | MS + OS              | MS: two-point OS: POX | MS: random alternative OS: swap | Min. makespan |
| Türkyılmaz and Bulkan [24] | MS + OS           | MS: uniform OS: MPX1, MPX2 | MS: random alternative OS: swap | Min. total tardiness |
| Chang et al. [29]      | MS + OS              | MS, OS: two-point, uniform | MS: random alternative OS: neighborhood search | Min. makespan |
| Chen et al. [30]       | MS + OS + TC         | MS, TC: two-point OS: POX | MS: random alternative TC: swap OS: shift | Min. average earliness and tardiness |
| Driss et al. [31]      | MS + OS              | MS: uniform OS: POX | MS, OS: values mutation | Min. makesapn |
| Ishikawa et al. [32]   | MS + OS              | MS: uniform OS: POX | MS: random alternative OS: inversion | Min. makespan |
| Wang et al. [33]       | MS + OS              | MS: two-point OS: POX | MS: shift OS: swap | Min. makespan |
| Zhang et al. [25]      | MS + OS              | MS: two-point OS: POX | MS: greedy (SPT machine) OS: adaptive neighborhood search | Min. makespan, Min. total setup time, Min. total transportation time |
| This paper             | MS + OS              | MS, OS: COGO (one-point, two-point, uniform) | MS: random alternative OS: COGO | Min. makespan |
Certain research papers developed GA hybrid solution methods and other algorithms for solving the FJSP, which are not considered in Table 1. Similar to the GAs in Table 1, almost all of the hybrid methods also adopted different genetic operators to handle the MS and OS parts.

The most important GA feature proposed in this paper is that crossover operations for both the MS and OS parts are performed by applying an identical operator: the candidate order-based genetic operator (COGO) provided by COGA. This can help to maintain a combination of a good MS gene and a good OS gene in a single operation during the search procedure. Moreover, we can see that the mutation for the OS part is also performed using the COGO. Note that the original version of the COGO is an integrated genetic operator, which can be used for the purposes of both crossover and mutation [14]. Consequently, the u-COGA for the FJSP can be developed by using only two kinds of genetic operators: the COGO and a mutation operator for the MS part. In other words, the structure of the u-COGO is simpler than that of previous GAs for the FJSP.

2.2. Candidate Order-Based Genetic Algorithm

The COGA is a type of GA that is based on the candidate order approach (COA). The main idea of the COA is to generate a feasible sequence of given items by appending one item at a time. Moreover, which item to append is chosen from a set of candidates; an item is a candidate if it is not appended to the sequence under construction and no constraints are violated after its appending [34]. COA can be used to develop metaheuristic algorithms for solving the sequencing problem.

The first example of a COA application is the COGA for solving the classical JSP (Kim, 2014). As a result of its flexibility, the COGA is quite effective for solving sequencing problems with additional constraints. Kim [14] applied the COGA to solve the job sequencing problem and the traveling salesman problem with precedence constraints. Kim and Kim [35] developed COGA for solving a variant of shortest path problem. Another important benefit of COA is that it can be integrated with various metaheuristic algorithms. For instance, Kim [34] developed a candidate order-based tabu search (COTS) for solving the job sequencing problem with precedence constraints. Kim and Kim [36] introduced the concept of the latest order constraint and applied COTS to solve the latest order constrained sequencing problem.

Typically, metaheuristic algorithms for solving sequencing problems have complex structures. However, COA-based metaheuristic algorithms are easier to implement than conventional metaheuristic algorithms. In particular, the COGA provides a distinguishing genetic operator called COGO, which has two important features. First, the COGO enables the application of simple point-based crossover strategies to sequencing problems, including one-point, two-point, and uniform strategies. Thus, both crossover operations for the MS and OS parts in a GA chromosome for the FJSP can be performed using the COGO. Second, the COGO also can be used to mutate a GA chromosome for the sequencing problem. The COGO generates a feasible sequence of given items in a constructive manner, which means a sequence is obtained by iteratively appending one item at a time. The item to be appended is chosen from a set of candidates. The objective of crossover, which is to create offspring similar to the parents, can be achieved by choosing an item with an earlier reference position. Note that the reference position of an item is its order in the parent solution. In contrast, the COGO performs the mutation by choosing an item with a later reference position, since the objective of the mutation is to create offspring dissimilar to the parents [14].

The main contributions of this paper are as follows. Firstly, we can implement the GA for solving the FJSP with only two genetic operators: the COGO and an additional operator for the mutation of the MS part. In other words, the u-COGA is easier to implement than previous GAs for the FJSP. Secondly, various kinds of crossover strategies, such as one-point, two-point, and uniform strategies, can be applied to the u-COGA. Thirdly, numerical
experiments reveal that uniform crossover is the best crossover strategy for the u-COGA for the FJSP.

3. Unified Candidate Order-Based Genetic Algorithm

3.1. Existing GAs for the FJSP

Table 2 provides an example of the FJSP with three jobs (J₁, J₂, and J₃) and three machines (M₁, M₂, and M₃). Each job consists of two operations that should be processed in sequence, and i/ᵗʰ operation of Jᵢ (i = 1, 2, 3) is denoted by oᵢj. An operation is processed by one of its alternative machines. For example, o₁₁ has three alternative machines (M₁, M₂, and M₃). Different alternative machines can have different processing times for a single operation. In Table 2, M₂ is the SPT machine for o₁₁ such that its processing time is shorter than other machines. In contrast, o₁₂ has only two alternative machines (M₂ and M₃), since M₁ is not available for this operation.

Table 2. An example of the FJSP.

| Job | Operation | Processing Time |
|-----|-----------|-----------------|
|     |           | M₁  | M₂  | M₃  |
| J₁  | o₁₁       | 4   | 3   | 5   |
|     | o₁₂       | N/A | 6   | 8   |
| J₂  | o₂₁       | 10  | N/A | 8   |
|     | o₂₂       | 5   | 6   | 4   |
| J₃  | o₃₁       | 7   | 10  | N/A |
|     | o₃₂       | 3   | 4   | 5   |

In order to create a schedule for the FJSP, two kinds of decisions are needed. First, each operation should be assigned to one of its alternative machines. Second, operations should be sequenced. Typically, GAs for the FJSP utilize a multichromosome that consists of the OS part and the MS part for representing the operation sequence and machine assignment, respectively. An example of a multichromosome for the FJSP is shown in the upper part of Figure 1. Let MS(oᵢj) denote the machine to be used to process oᵢj. It is clear that MS(oᵢj) must be an element of a set of alternative machines for oᵢj.

![Figure 1. Decoding the multichromosome representation for the FJSP.](image)

In this paper, a sequence of operations indicates the assignment orders. Let OS(oᵢj) denote the order of oᵢj in a sequence of given operations. Note that the encoding scheme for the OS part is the position listing representation, in which each gene specifies an order of the associated item. Thus, the OS part in Figure 1 can be converted into a sequence of given operations, (o₁₁, o₁₂, o₂₁, o₂₂, o₃₁, o₃₂). An operation oᵢⱼ is scheduled prior to oᵢⱼ if OS(oᵢ₁) < OS(oᵢ₂). For instance, o₁₁ is the first operation to be scheduled in the Gantt chart in the lower part of Figure 1, since OS(o₁₁) = 1. Moreover, we can start o₁₁ at time t = 0.
The order of given operations, $OS(a_{ij})$, must satisfy the precedence relationships among operations within a single job. In other words, $OS(a_{ij})$ must be smaller than $OS(a_{ik})$ if $j < k$. In Figure 1, $a_{12}$ is the second operation ($OS(a_{12}) = 2$) of a job and its start time is 4, because we can start $a_{12}$ after its predecessor, $a_{11}$, is completed. Valid $MS(a_{ij})$ and $OS(a_{ij})$ values yield a feasible schedule for the FJSP, which can be represented in the form of a Gantt chart, as shown in Figure 1.

The crossover operation is the “backbone” of the GA, and crossover operators are designed to generate two offspring solutions by recombining the genes of two parent solutions. The offspring solutions inherit the features of their parents. In other words, crossover operators should generate offspring solutions similar to their parents. However, this is difficult to achieve when solving the FJSP by applying a GA based on a multichromosome.

An example of a crossover operation of the existing GA for the FJSP is illustrated in Figure 2. Consider two parent solutions, $P_1$ and $P_2$, in panel (a) in Figure 2. Typically, existing GAs for the FJSP apply different kinds of crossover operators to the OS and MS parts of parent solutions. In panel (b) in Figure 2, POX and classical one-point crossover are applied to the OS and MS part, respectively. POX is a well-known crossover operator for sequencing problems. In panel (b), the OS parts of the solutions are represented in the form of a sequence of operations, in order to apply POX. Job set 1 indicates that an operation $a_{ij}$ has an identical position $OS(a_{ij})$ in both $P_x$ and $C_x$ ($x = 1, 2$) if it belongs to $J_3$, where $C_x$ denotes an offspring solution. In contrast, job set 2 means $C_1$ ($C_2$) inherits precedence relationships among operations of $J_1$ and $J_2$ from $P_2$ ($P_1$).

**Figure 2.** An example of crossover for an existing GA for solving the FJSP.

The one-point crossover operator divides a chromosome into two subparts, i.e., the earlier part and later part, and generates $C_1$ ($C_2$) by combining the earlier part of $P_1$ ($P_2$) and the later part of $P_2$ ($P_1$). Panel (c) in Figure 2 shows a multichromosome representation of $C_1$ and $C_2$, obtained by applying POX and one-point crossover. Similarly, most existing GAs...
for solving the FJSP use different crossover operators to recombine the multichromosome, which consists of the OS and MS parts.

In Figure 2, we can see that the OS part and MS part evolve separately. That is, the crossover procedure of the OS part is not correlated with that of the MS part. In this context, the traditional approach based on two different crossover operators is called the separate evolution strategy in this paper. This strategy has a critical limitation, i.e., a combination of the OS gene and MS gene for an identical operation can be easily broken. For instance, \( o_{31} \) has two associated genes: \( OS(o_{31}) \) and \( MS(o_{31}) \). For \( P_1 \) in panel (a) in Figure 2, \( OS(o_{31}) = 1 \) and \( MS(o_{31}) = M_2 \). If these two genes indicate the optimal assignment order and machine for \( o_{31} \), the combination of \( OS(o_{31}) = 1 \) and \( MS(o_{31}) = M_2 \) should be preserved during the crossover procedure. However, this combination is not found in offspring solutions \( C_1 \) and \( C_2 \) in panel (c) in Figure 2. In more detail, \( OS(o_{31}) \) of \( P_1 \) is inherited to \( C_1 \), whereas \( MS(o_{31}) \) of \( P_1 \) is inherited to \( C_2 \). The u-COGA proposed in this paper is designed to overcome the limitation of the traditional separate evolution strategy for the FJSP.

### 3.2. Crossover of Unified COGA

This section outlines crossover procedure of the u-COGA. The COGO enables the application of simple crossover strategies to sequencing problems, such as one-point, two-point, and uniform crossovers. Thus, a solution of the FJSP is converted into a sequence of operations in order to apply the COGO, as shown in Figure 3. Note that an operation in the FJSP has an additional attribute, \( MS(o_{ij}) \), as shown in panel (b) in Figure 3.

![Figure 3](image)

**Figure 3.** Reference set assignment for parent solutions in the form of a sequence.

The COGO generates a solution in a constructive manner, which means that one operation is appended to a new solution at a time. An operation to be appended to a specific position is chosen with consideration of the reference solution, one of the parent solutions. Thus, the COGO determines the reference solution for each position in a sequence before generating a new solution. An important feature of the COGO is that simple crossover strategies can be used to determine reference solutions. In panel (b) in Figure 3, each sequence is divided into two parts, i.e., the earlier part with the first two operations and later part with the other four operations, by adopting the conventional one-point crossover...
strategy. We can see that $P_1$ ($P_2$) and $P_2$ ($P_2$) are used as reference solutions for the earlier part and later part of $C_1$ ($C_2$), respectively.

An operation to be used as the $p$th position of a new solution is determined after the $p - 1$th operation is appended. An operation to be the $p$th operation is chosen from a set of candidates; an operation is a candidate if and only if it has not been appended to the new solution under construction and no constraints are violated by appending the operation to the $p$th position. The main idea of the COGO is that the objectives of crossover and mutation can be achieved by choosing the appropriate operation from the set of candidates.

Figure 4 illustrates a procedure for generating the earlier part of $C_1$, an offspring of $P_1$ and $P_2$ in Figure 3. Let $RP(o_{ij}, X)$ denote the reference position of an operation $o_{ij}$ in a solution $X$. In other words, $o_{ij}$ is the $RP(o_{ij}, X)$th operation in $X$. The main idea of the COGO is that the objectives of crossover and mutation can be achieved by choosing candidates with the smallest and the largest reference position values, respectively. For the first position in $C_1$, all operations are candidates, since no operation is appended to $C_1$, yet. Among the candidates, an operation with the smallest reference position regarding $P_1, o_{31}$ is chosen as the first operation. Similarly, $o_{11}$ is appended to the second position in $C_1$. Note that $MS(o_{ij})$s of the offspring are inherited from the associated reference solution. That is, $MS(o_{31})$ and $MS(o_{11})$ of $C_1$ are the same as those of $P_1$.

![Figure 4. Generating the earlier part of the offspring solution.](image)

Figure 5 shows the crossover procedure for the later part of $C_1$, where $P_2$ is used as the reference solution. We can see that the positions of $o_{12}, o_{21}, o_{22},$ and $o_{32}$ in the later part of $C_1$ are different from those of $P_2$. Nevertheless, the precedence relationships among them in $P_2$ are also maintained in $C_1$. Moreover, $MS(o_{ij})$ of the operations in the later part are inherited from $P_2$. Consequently, $C_1$ inherits from both $P_1$ and $P_2$. Since the $MS(o_{ij})$ value of the reference solution is maintained in the offspring solution, the crossover procedure described in Figures 3–5 helps to preserve a combination of the OS and MS genes.

In contrast, the mutation operation for the $p$th position of a new solution is performed by choosing a candidate with the largest reference position in the COGO procedure. That is, crossover and mutation are integrated in the COGO. Assume that $o_{ij}$ is the candidate with the largest reference position for the $p$th position of a new solution chosen by the COGO in mutation mode. Then, $OS(o_{ij})$ of the new solution is likely to be dissimilar to that of the reference solution. However, $MS(o_{ij})$ is not affected by the mutation procedure provided by the COGO. Hence, we need an additional mutation operator for $MS(o_{ij})$, and
this paper applies random alternative mutation to the MS part of the multichromosome for the FJSP.

Figure 4. Generating the earlier part of the offspring solution.

Figure 5. Generating the later part of the offspring solution.

3.3. Initialization, Fitness Function, Selection, and Elitism

Population initialization is the first step in the GA. Population can be defined as a set of chromosomes. There are several ways to initialize the population in the GA; however, the solutions in the initial population are randomly generated in this paper.

The fitness function of the GA is an objective function that needs to be maximized. In order to minimize the makespan of the FJSP, we use the reciprocal of the makespan as the fitness function of the u-COGA.

Selection is the procedure of creating a mating pool by copying the solutions in the current generation. The goal of the selection procedure is to choose the solutions with higher fitness values with larger probabilities. In this paper, we use conventional roulette wheel selection, in which a solution in the current population is chosen with a probability proportional to its fitness value.

Furthermore, the elitism strategy is applied to the u-COGA proposed in this paper. This strategy ensures that the best solutions in the current generation are also included in the mating pool. In other words, the elitism strategy helps to prevent desirable features of solutions being lost during the search procedure. Consequently, some solutions in the mating pool are determined by the elitism strategy, while the others are chosen by the roulette wheel selection procedure.

4. Numerical Experiments

In this paper, the u-COGA was written in Java language and was tested in a Windows 10 (64-bit) environment with an AMD Ryzen 7 2700X eight-Core processor (3.7 GHz) and 32 GB memory. The performance of the u-COGA was evaluated using the benchmark problems for the FJSP provided by the well-known Hurink library [37]. Depending on the average number of alternative machines for each operation, benchmark problems in the Hurink library are divided into three groups: eData, rData, and yData, as listed in Table 3.
Table 3. Groups of benchmark problems in the Hurink library.

| Data Group | Description                                         |
|------------|-----------------------------------------------------|
| eData      | Few operations can be assigned to more than one machine (Low Complexity) |
| rData      | Most of the work can be assigned to some machines (Moderate Complexity) |
| yData      | Each operation can be assigned to several machines (High Complexity) |

Benchmark problems from each data group are summarized in Table 4, where \( n \) is the number of operations, \( m \) is the number of machines, and \( M_{ij} \) is a set of alternative machines for an operation \( o_{ij} \). Moreover, \( \text{Avg} |M_{ij}| \) and \( \text{Max} |M_{ij}| \) indicate the average and the maximum size of \( M_{ij} \), respectively. Each data group contains three instances, and the instances of the yData group have the most complex structures, in that they generally have \( \text{Avg} |M_{ij}| \) and \( \text{Max} |M_{ij}| \) larger than the other groups. On the contrary, eData contains the simplest instances.

Table 4. Benchmark problem instances.

| Data Group | Instance | \( n \) | \( m \) | \( \text{Avg} |M_{ij}| \) | \( \text{Max} |M_{ij}| \) |
|------------|----------|--------|--------|----------------|----------------|
| eData      | mt06     | 6      | 6      | 1.15           | 2 \((m \leq 6)\) |
|            | mt10     | 10     | 10     |                 | 3 \((m \geq 10)\) |
|            | mt20     | 20     | 5      |                 |                |
| rData      | mt06     | 6      | 6      | 2               | 3               |
| vData      | mt10     | 10     | 10     | \(\frac{1}{2}m\) | \(\frac{3}{5}m\) |
|            | mt20     | 20     | 5      |                 |                |

As shown in Table 5, we applied three types of u-COGA, SS, TT, and UU to the benchmark problems in Table 4. The symbols ‘S’, ‘T’, and ‘U’ indicate one-point, two-point, and uniform crossover, respectively. In addition, the first and second symbols in ‘type’ indicate the crossover strategies for the OS part and MS part, respectively. For example, the first type of u-COGA, i.e., SS, applies the one-point crossover strategy to both the OS and MS parts.

Table 5. Types of crossover operators applied to the existing GA and u-COGA.

| Type | u-COGA | Existing GA |
|------|--------|-------------|
|      | OS     | MS          | Type | OS | MS |
| SS   | One-point | One-point | PS   | POX | One-point |
| TT   | Two-point | Two-point | PT   | POX | Two-point |
| UU   | Uniform | Uniform | PU   | POX | Uniform |

For comparison, the existing GA was also applied to the benchmark problems. The existing GA uses POX to recombine the OS parts of the chromosomes, while crossover for the MS part is performed in a one-point, two-point, and uniform manner. Thus, we have three types of existing GAs: PS, PT, and PU, as shown in Table 5.

The experiment results obtained by applying the u-COGA and the existing GA are summarized in Tables 6–8, where 20 experimental repetitions were performed for each case. \( C_{\text{max}} \) and \( \text{SD}(C_{\text{max}}) \) are the average makespan and standard deviation of the makespan, respectively. The six types of GAs applied to a single benchmark problem instance are ranked in ascending order of \( C_{\text{max}} \). For instance, TT and PS are the best and worst algorithms for the mt06 instance of eData in Table 6, respectively. All experimental results were obtained with a population size = 50, a crossover rate = 0.8, and a mutation rate = 0.01. In
addition, the elitism policy was used to preserve the five best solutions in the previous generation. A single experiment was terminated when its iteration number reached the maximum iteration limit = 300.

Table 6. Experiment results for eData.

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| eData 6 × 6 (mt06) | C\text{max} | 49.95 | 50.3 | 50.85 | 49.05 | 48.9 | 47.9 |
| | SD(C\text{max}) | 1.43 | 1.72 | 1.93 | 1.67 | 1.48 | 1.33 |
| | Rank | 4 | 5 | 6 | 3 | 2 | 1 |

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| eData 10 × 10 (mt10) | C\text{max} | 1094.7 | 1090.1 | 1093.9 | 1078.8 | 1085 | 1050.85 |
| | SD(C\text{max}) | 24.67 | 27.93 | 22.59 | 27.67 | 28.19 | 22.64 |
| | Rank | 6 | 4 | 5 | 2 | 3 | 1 |

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| eData 20 × 5 (mt20) | C\text{max} | 1134.15 | 1135.15 | 1134.75 | 1074.2 | 1101.9 | 1097.4 |
| | SD(C\text{max}) | 42.50 | 33.31 | 44.20 | 62.89 | 33.86 | 41.03 |
| | Rank | 4 | 6 | 5 | 1 | 3 | 2 |

Table 7. Experiment results for rData.

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| rData 6 × 6 (mt06) | C\text{max} | 49.45 | 48 | 47.9 | 48.25 | 47.85 | 47.25 |
| | SD(C\text{max}) | 1.79 | 1.56 | 1.51 | 1.74 | 1.53 | 1.74 |
| | Rank | 6 | 4 | 3 | 5 | 2 | 1 |

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| rData 10 × 10 (mt10) | C\text{max} | 997.95 | 1008.15 | 1002.05 | 996.35 | 997.8 | 962.2 |
| | SD(C\text{max}) | 28.14 | 23.04 | 19.33 | 29.50 | 25.35 | 26.02 |
| | Rank | 4 | 6 | 5 | 2 | 3 | 1 |

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|--------|
|           | PS | PT | PU | SS | TT | UU |
| rData 20 × 5 (mt20) | C\text{max} | 1090.1 | 1091.1 | 1083.35 | 1090.3 | 1074.9 | 1056.55 |
| | SD(C\text{max}) | 41.07 | 41.74 | 37.27 | 34.45 | 36.89 | 31.46 |
| | Rank | 4 | 6 | 3 | 5 | 2 | 1 |
Tables 6–8 provide the following observations. First, u-COGAs generally produce shorter makespan values than existing GAs, which indicates that the u-COGA is a promising approach for solving the FJSP. Second, for seven out of nine instances in Tables 6–8, two instances (mt06 and mt10) of eData, three instances (mt06, mt10, and mt20) of rData, and two instances (mt06 and mt10) of vData, the shortest makespan values were found by the UU-type u-COGA. In other words, uniform crossover was the best crossover strategy for minimizing the makespan of the FJSP. The performances of SS- and TT-type u-COGAs were not as good as that of the UU-type. Third, excepting one benchmark problem instance, i.e., mt06 of vData in Table 8, the worst makespan value for each instance was obtained by one of the existing GAs, which reveals the limitation of the separate evolution strategy for solving the FJSP. In particular, PS- and PU-type GAs produced the worst makespan values for four instances and three instances, respectively. Finally, the u-COGA and existing GA did not demonstrate significant differences in SD($C_{\text{max}}$) values.

Table 9 shows the average rank of the algorithms listed in Table 5. Again, we can see that the UU-type u-COGA has the smallest average rank, which means that it generally produces a shorter makespan than the other algorithms. In contrast, the average ranks of the individual existing GAs are larger than those of the u-COGAs. Consequently, the average rank of all u-COGAs (2.33) is noticeably smaller than that of all existing GAs (4.63), and we can conclude that the unified evolution strategy of the u-COGA helps to find better solutions for the FJSP.

### Table 8. Experiment results for vData.

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|---------|
| $C_{\text{max}}$ | 48 | 46.9 |
| SD($C_{\text{max}}$) | 2.10 | 1.62 |
| Rank | 4 | 1 |

### vData 6 × 6 (mt06)

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|---------|
| $C_{\text{max}}$ | 48 | 47.8 |
| SD($C_{\text{max}}$) | 1.40 | 1.67 |
| Rank | 3 | 2 |

### vData 10 × 10 (mt10)

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|---------|
| $C_{\text{max}}$ | 945.35 | 920.25 |
| SD($C_{\text{max}}$) | 21.61 | 22.73 |
| Rank | 6 | 1 |

### vData 20 × 5 (mt20)

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|---------|
| $C_{\text{max}}$ | 1074.8 | 1054.55 |
| SD($C_{\text{max}}$) | 28.81 | 27.59 |
| Rank | 6 | 2 |

### Table 9. Average ranks of the algorithms.

| Algorithm | Existing GA | u-COGA |
|-----------|-------------|---------|
| $Rank$ | 4.89 | 4.63 |

| $Rank$ | 4.67 | 2.33 |
Table 10 compares the best makespan of the existing GAs and u-COGAs. For instance, the best makespan for the existing GAs for mt20 of vData is 1068.85, while the best makespan for a u-COGA for the same instance is 1053.20, as shown in Table 8. Moreover, the reduction ratio in Table 10 was calculated as follows: (the best makespan of the existing GAs—the best makespan of u-COGAs)/(the best makespan of the existing GAs). This index was used to measure how much improvement was achieved by the u-COGA.

Table 10. Comparison of the best makespan obtained by the existing GAs and u-COGAs.

| ALGORITHM | eData | u-COGA | ReductionRatio (%) | rData | u-COGA | ReductionRatio (%) | vData | u-COGA | ReductionRatio (%) |
|-----------|-------|--------|-------------------|-------|--------|-------------------|-------|--------|-------------------|
| MT06      | 49.95 | 47.9   | 4.10              | 47.9  | 47.25  | 1.36              | 47.8  | 46.9   | 1.88              |
| MT10      | 1030.1| 1074.2 | 5.29              | 1083.35| 1056.55| 2.47              | 1068.85| 1053.2 | 1.46              |

In Table 10, we can see that the u-COGA always produces better solutions for the FJSP than the existing GA. The reduction ratio values range from 1.36% to 5.29% (average reduction ratio = 2.79%). That is, the existing GA sometimes demonstrates a similar performance to the u-COGA; however, the u-COGA generally produces superior results for the FJSP.

5. Conclusions and Further Remarks

In order to apply the GA to solve a combinatorial optimization problem, a solution of the problem must be represented in the form of a chromosome comprising a number of genes. A solution for the classical JSP can be encoded using only OS-type genes. In contrast, a solution for the FJSP is typically represented in the form of a multichromosome, which consists of the OS and MS parts. Since the structures of the OS and MS parts are quite different, existing GAs generally apply different kinds of genetic operators to them. Consequently, the evolution procedures of the two parts are independent from each other, and a combination of OS and MS genes for an identical operation is easily broken during the search procedure. Moreover, the application of different kinds of crossover operators can make the GA complex and hard to implement and maintain. In order to overcome these problems, this paper proposes the u-COGA, which can be used to solve the FJSP.

The u-COGA utilizes an integrated genetic operator called the COGO to handle the multichromosome for the FJSP. The COGO enables the application of conventional crossover strategies, such as one-point, two-point, and uniform crossovers to the OS part. The u-COGA is designed to apply an identical crossover strategy to both the OS and MS parts. Moreover, the COGO generates a new sequence of given operations by choosing one operation at a time, which is suitable for the purposes of crossover and mutation. In order to perform the crossover operation, the COGO chooses a candidate with the smallest reference position. On the contrary, the mutation operation is performed by choosing a candidate with the largest reference position. In other words, the COGO of the u-COGA is used to perform three operations: crossover for the OS part, crossover for the MS part, and mutation for the OS part. Mutation for the MS part should be performed in a different manner, and random alternative mutation is used in this paper. Consequently, crossover and mutation are performed using two genetic operators: the COGO and mutation operator for the MS part in the u-COGA. Note that four different kinds of operators, i.e., crossover for the OS part, crossover for the MS part, mutation for the OS part, and mutation for the MS part, are required to implement the existing GAs for the FJSP. In other words, the COGO enables one to solve the FJSP using a GA with a simpler structure.

In order to evaluate performance of the u-COGA, numerical experiments were performed using an FJSP benchmark problem library. The experiment results reveal that the performance of the u-COGA is generally superior to that of the existing GAs in terms of the makespan. In more detail, the u-COGA improves the makespan values for the FJSP by about 2.79%. Notably, the best solution for a benchmark problem instance was generally found using the u-COGA based on the uniform crossover strategy. In contrast, the worst
solution was generally found by one of the existing GAs. This means that the u-COGA can explore the search space of the FJSP more effectively, and we can conclude that the GA for the FJSP should contain procedures for maintaining good combinations of good OS and MS genes; thus, the u-COGA is a promising approach for solving complex combinatorial optimization problems.

The authors propose three directions for future research on the u-COGA. First, this paper only considered one objective function, which is the minimization of the makespan. Thus, u-COGA performance regarding other objective functions should be investigated. Second, the primary advantage of the original version of the COGA is that it can flexibly handle additional constraints, such as precedence or position-related constraints. In this regard, in future research, we plan to apply the u-COGA to the FJSP with additional constraints. Third, it is expected that the u-COGA can help to solve other combinatorial optimization problems such that solutions have to be represented in the form of a multi-chromosome. Therefore, the authors will attempt to apply the u-COGA to combinatorial optimization problems in various other fields.

Author Contributions: Conceptualization, H.-Y.N. and J.-W.K.; methodology, J.-S.P. and J.-W.K.; software, J.-S.P.; validation, J.-S.P. and J.-W.K.; formal analysis, J.-S.P. and J.-W.K.; investigation, J.-S.P. and J.-W.K.; resource, H.-Y.N., T.-J.C. and Y.-T.N.; data curation, J.-S.P.; writing—original draft preparation, J.-S.P. and J.-W.K.; writing—review and editing, J.-S.P., H.-Y.N. and J.-W.K.; supervision, J.-W.K.; project administration, H.-Y.N., Y.-T.N. and J.-W.K.; funding acquisition, H.-Y.N., Y.-T.N. and J.-W.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2020R1F1A1049251) and the Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (MOTIE: Ministry of Trade Industry and Energy; Grant No. N0002429).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References
1. Zhang, G.; Gao, L.; Shi, Y. An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst. Appl. 2011, 38, 3563–3573. [CrossRef]
2. Harjunkoski, I.; Maravelias, C.T.; Bongers, P.; Castro, P.M.; Engell, S.; Grossmann, I.E.; Hooker, J.; Mendez, C.; Sand, G.; Wassick, J. Scope for industrial applications of production scheduling models and solution methods. Comput. Chem. Eng. 2014, 62, 161–193. [CrossRef]
3. Pinedo, M.L. Scheduling: Theory, Algorithms, and Systems, 5th ed.; Springer: Berlin/Heidelberg, Germany, 2016.
4. Parente, M.; Figueira, G.; Amorim, P.; Marques, A. Production scheduling in the context of Industry 4.0: Review and trends. Int. J. Prod. Res. 2020, 58, 5401–5431. [CrossRef]
5. Hussain, K.; Salleh, M.N.M.; Cheng, S.; Shi, Y. Metaheuristic research: A comprehensive survey. Artif. Intell. Rev. 2019, 52, 2191–2233. [CrossRef]
6. Holland, J.H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence; University of Michigan Press: Ann Arbor, MI, USA, 1975.
7. Glover, F. Tabu Search: A Tutorial. Interfaces 1990, 20, 74–94. [CrossRef]
8. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. Science 1983, 220, 671–680. [CrossRef] [PubMed]
9. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN’95 International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
10. Cheng, R.; Gen, M.; Tsujimura, Y. A Tutorial Survey of Job-Shop Scheduling Problems using Genetic Algorithms-I. represent. Comput. Ind. Eng. 1996, 30, 983–997. [CrossRef]
11. Cheng, R.; Gen, M.; Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: Hybrid genetic search strategies. Comput. Ind. Eng. 1999, 36, 343–364. [CrossRef]
12. Chaudhry, I.A.; Khan, A.A. A research survey: Review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. 2016, 23, 551–591. [CrossRef]
13. Kim, J.W. Developing a job shop scheduling system through integration of graphic user interface and genetic algorithm. Mul-timed. Tools Appl. 2015, 74, 3329–3343. [CrossRef]
14. Kim, J.W. Candidate Order based Genetic Algorithm (COGA) for Constrained Sequencing Problems. *Int. J. Ind. Eng. Theory Appl. Pract.* 2016, 23, 1–12.

15. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* 2021, 80, 8091–8126. [CrossRef]

16. Poon, P.; Carter, J. Genetic algorithm crossover operators for ordering applications. *Comput. Oper. Res.* 1995, 22, 135–147. [CrossRef]

17. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* 2021, 80, 8091–8126. [CrossRef]

18. Kim, J.W. A Job Shop Scheduling Game with GA-based Evaluation. *Appl. Math. Inf. Sci.* 2014, 8, 2627–2634.

19. Gao, J.; Sun, L.; Gen, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Comput. Oper. Res.* 2008, 35, 2892–2907. [CrossRef]

20. Pezzella, F.; Morganti, G.; Ciaschetti, G. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Comput. Oper. Res.* 2008, 35, 3202–3212. [CrossRef]

21. Lei, D. A Genetic Algorithm for Flexible Job Shop Scheduling with Fuzzy Processing Time. *Int. J. Prod. Res.* 2010, 48, 2995–3013. [CrossRef]

22. Wang, X.; Gao, L.; Zhang, C.; Shao, X. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *J. Adv. Manuf. Technol.* 2015, 53, 1832–1848. [CrossRef]

23. Parjapati, S.K.; Jain, A. Optimization of Flexible Job Shop Scheduling Problem with Sequence Dependent Setup Times Using Genetic Algorithm Approach. *Int. J. Math. Comput. Nat. Phys. Eng.* 2015, 9, 41–47.

24. Elgendy, A.; Mohammed, H.; Elhakeem, A. Optimizing Dynamic Flexible Job Shop Scheduling Problem Based on Genetic Algorithm. *Int. J. Curr. Eng. Technol.* 2017, 7, 368–373.

25. Chang, H.-C.; Chen, Y.-P.; Liu, T.-K.; Chou, J.-H. Solving the Flexible Job Shop Scheduling Problem with Makespan Optimization by Using a Hybrid Taguchi-Genetic Algorithm. *IEEE Access* 2015, 3, 1740–1754. [CrossRef]

26. Chen, J.; Weng, W.; Rong, G.; Fujimura, S. Integrating Genetic Algorithm with Time Control for Just-In-Time Scheduling Problems. *IFAC-Paper* 2015, 48, 893–897. [CrossRef]

27. Driss, I.; Mouss, K.N.; Laggoun, A. A New Genetic Algorithm for Flexible Job-Shop Scheduling Problems. *J. Mech. Sci. Technol.* 2015, 29, 1273–1281. [CrossRef]

28. Ishikawa, S.; Kubota, R.; Horio, K. Effective Hierarchical Optimization by a Hierarchical Multi-Space Competitive Genetic Algorithm for the Flexible Job-Shop Scheduling Problem. *Expert Syst. Appl.* 2015, 42, 9434–9440. [CrossRef]

29. Wang, L.; Luo, C.; Cai, J. A Variable Interval Rescheduling Strategy for Dynamic Flexible Job Shop Scheduling Problem by Improved Genetic Algorithm. *J. Adv. Transp.* 2017, 2017, 1–12. [CrossRef]

30. Kim, J.W. Performance Comparison of Neighborhood Structures of Tabu Search Algorithms for Sequencing Problems. *Adv. Sci. Lett.* 2017, 23, 10423–10426.

31. Hurink, J.; Jurisch, B. and Thole, M. Tabu Search for the Job-shop Scheduling Problem with Multi-purpose Machines. *Oper. Res. Spektrum* 1994, 15, 205–215. [CrossRef]