Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 906305, 12 pages
http://dx.doi.org/10.1155/2015/906305

Research Article

A Two-Dimensional Genetic Algorithm and Its Application to Aircraft Scheduling Problem

Ming-Wen Tsai,[1] Tzung-Pei Hong,[2,3] and Woo-Tsong Lin[1]

[1] *Department of Management Information Systems, National Chengchi University, No. 64, Section 2, ZhiNan Road, Wenshan District, Taipei City 11605, Taiwan*
[2] *Department of Computer Science and Information Engineering, National University of Kaohsiung, 700 Kaohsiung University Road, Kaohsiung 81148, Taiwan*
[3] *Department of Computer Science and Engineering, National Sun Yat-Sen University, 70 Lienhai Road, Kaohsiung 80424, Taiwan*

Correspondence should be addressed to Ming-Wen Tsai; 94356503@nccu.edu.tw

Received 16 November 2014; Revised 9 February 2015; Accepted 24 February 2015

Academic Editor: Bin Jiang

Genetic algorithms have become increasingly important for researchers in resolving difficult problems because they can provide feasible solutions in limited time. Using genetic algorithms to solve a problem involves first defining a representation that describes the problem states. Most previous studies have adopted one-dimensional representation. Some real problems are, however, naturally suitable to two-dimensional representation. Therefore, a two-dimensional encoding representation is designed and the traditional genetic algorithm is modified to fit the representation. Particularly, appropriate two-dimensional crossover and mutation operations are proposed to generate candidate chromosomes in the next generations. A two-dimensional repairing mechanism is also developed to adjust infeasible chromosomes to feasible ones. Finally, the proposed approach is used to solve the scheduling problem of assigning aircrafts to a time table in an airline company for demonstrating the effectiveness of the proposed genetic algorithm.

1. Introduction

Genetic algorithms (GAs) [1, 2] have recently been used to solve optimization problems very commonly since they can get nearly optimal solutions in reasonable time. They were first proposed by Holland in 1975 [3] based on Darwin's principle of survival of the fittest. Each possible solution for a problem may be regarded as an individual in a natural population. The next candidate set of solutions are then generated by several operations including crossover, mutation, and reproduction. GAs have been successfully applied to many fields such as optimization [4–6], machine learning [2, 7], neural networks [8, 9], and fuzzy logic controllers [10–13]. The simple genetic algorithm uses a single crossover operator and a single mutation operator throughout the entire genetic process [3]. A representation that describes the possible solutions for a problem must first be defined when applying genetic algorithms to solve a problem. The simple genetic algorithm is described as follows.

The Simple Genetic Algorithm

Step 1. Define a suitable representation of the problem to be solved.

Step 2. Create an initial population of N individuals for evolution.

Step 3. Define a suitable fitness function for evaluating the individuals.

Step 4. Perform genetic operations (crossover and mutation) to generate possible offspring.

Step 5. Evaluate the fitness value of each individual.

Step 6. Select superior N individuals according to their fitness values.

Step 7. If the termination criterion is not satisfied, go to Step 4; otherwise, stop the algorithm.

In addition to the simple genetic algorithm, other variant GAs can also be used. As mentioned above, a chromosome representation must first be defined for GA to proceed. The representation strongly affects the behavior and performance of genetic algorithms. Several chromosome representations have been proposed and commonly used, such as binary strings, real-value vectors, permutations, finite-state representation, and parse-tree representation. Binary strings [2, 3, 14–18] are the standard and the most commonly used representation of solutions for genetic algorithms. They use only the two symbols 0 and 1 to represent a chromosome. Real-valued vectors [2, 14, 19–21] are another popular representation used in GA. Each position in a chromosome is a real value. Real-value vectors are especially useful for solving real-value optimization problems. Permutations are a popular representation for some combinatorial optimization problems [6, 22, 23]. They encode the set of objects into numbers and then arrange them into a chromosome. The finite-state representation [4, 24] first constructs a state transition table according to the given problems and then evolves according to the transitive table. It is used in environments in which sequences of states have some implicit relations and must be generated with the relation. Additionally, the parse-tree representation [25, 26] is often adopted to evolve executable structures, such as programs. Each chromosome is represented by a parse tree.

Most previous representations, such as the bit string, were linear or one-dimensional. However, some real problems are naturally suitable for two-dimensional representation. For example, in a scheduling problem for assigning aircrafts to time slots in an airline company, a two-dimensional array or table is often used to represent the schedule. If the problem is solved by genetic algorithms, each possible solution can be very conveniently and conceptually represented as a two-dimensional table.

In this paper, we focus on the permutation representation. Although a traditional one-dimensional encoding approach can be easily implemented, it has an intrinsic drawback in representing complex structures. That is, any solution to a problem has to be represented by a linear string. This probably causes the loss of some information contained in the problem. A two-dimensional encoding approach can reflect more geographical linkage of genes than one-dimensional encoding approach. For example, Cohoon and Paris [27] proposed a 2D crossover that chose a small rectangle from one parent and copied the genes in the rectangle into the offspring, with the rest of the genes copied from the other parent. Anderson et al. [28] suggested a block-uniform crossover, which tessellated a 2D chromosome into $i \times j$ blocks; the genes in each block were then copied as a group from a uniformly selected parent. Wang and Korfhage [29] used a matrix genome encoding approach to schedule distributed tasks for minimizing the maximum finishing time. They defined a schedulable matrix and an allocation matrix to represent process constraints and to find a complete process schedule. Bui and Moon then proposed a geographic crossover to increase the diversity of offspring [30]. It generalized the conventional block uniform crossover and introduced natural lines. They showed that the entire encoding space could always be divided into two separated regions and that the offspring can be generated by alternately copying the intervals of two parent strings.

Sadrzadeh [31] then presented a genetic algorithm to solve the facility layout problem (FLP) in a manufacturing system. The problem was to design a physical layout with the minimization of the material handling cost as its main objective, and a matrix encoding technique was adopted. The crossover operation selected a rectangular area as the cutting section and then exchanged the sections of a pair of parents to generate new offspring. The mutation operation randomly swapped two genes within a limited boundary. Aiello et al. [14] also proposed a genetic algorithm to solve the facility layout problem. They adapted two kinds of encoding methods in two segments at the same time. The first segment was encoded by numerical values and the second one was by binary variables. The first segment represented a sequence of department placement, and the second showed a type of cutting. The crossover operator, however, basically follows the uniform crossover operation. Chou et al. [32] adopted an inequality-based multiobjective genetic algorithm to solve the aircraft routing problem. They used the concept of two-dimensional encoding and proposed a method of inequality to confine the search of a genetic algorithm for a Pareto optimal set, thus speeding up the evaluation of the fitness functions. However, in their research, the GA operation actually translated the chromosomes to one-dimensional strings in its execution and operated using the conventional Partially Matched Crossover (PMX).

This study proposes a novel genetic algorithm based on two-dimensional encoding. Appropriate two-dimensional crossover and mutation operations are designed to generate the next generations. The proposed crossover operator may adopt either horizontal or vertical combination to generate the offspring chromosomes. A repairing mechanism is also adopted to adjust infeasible chromosomes into feasible ones. Several two dimensional mutation operators, including two-point swapping, string swapping, and substring swapping, are presented. Finally, experiments on assigning aircrafts to time slots in an airline company are performed with different parameter settings to demonstrate the effectiveness of the proposed approach.

The rest of the paper is organized as follows. Section 2 describes the adopted two-dimensional encoding scheme. Several two-dimensional crossover operators and a two-dimensional repair mechanism for matching the scheme are proposed in Section 3. Several two-dimensional mutation operators are proposed in Section 4. Section 5 summarizes the experimental results on the performance of the proposed algorithm for aircraft scheduling in an airline company. Conclusions are finally drawn in Section 6, along with recommendations for future research.

2. The Two-Dimensional Chromosome Representation

As mentioned above, an appropriate chromosome representation must be defined for a GA to work. Most previously adopted representations, such as the bit string, are linear or one-dimensional. Some real problems are naturally suitable for two-dimensional representation. If this kind of problems is to be solved by genetic algorithms, then each possible solution can very conveniently and naturally be conceptually represented as a two-dimensional table. Therefore, this paper proposes a two-dimensional genetic algorithm with appropriate operators designed on the two-dimensional representation. The notation used in this work is defined below.

Notation

P: a population consisting of two-dimensional chromosomes;

n: the number of chromosomes in P;

c_i: the ith chromosome in P, $1 \leq i \leq n$;

$c_i(x, y)$: the gene located at position (x, y) in the ith chromosome c_i;

S: the number of rows in the two-dimensional chromosome representation;

W: the number of columns in the two-dimensional chromosome representation;

P_c: the crossover probability;

P_m: the mutation probability;

R: a random number in the range 0 to +1;

R_c: a random number indicating the column for crossover, $1 \leq i \leq W$;

R_r: a random number indicating the row for crossover, $1 \leq i \leq S$.

A chromosome c_i is thus encoded as an $S \times W$ matrix, with each element $c_i(x, y)$ representing the gene value located at (x, y), $1 \leq x \leq S$ and $1 \leq y \leq W$. An example is given below to show the use of the proposed two-dimensional chromosome representation.

Example 1. Consider a scheduling problem with three machines, four time intervals, and eight jobs. The jobs are given as {Job_1, Job_2, Job_3, Job_4, Job_5, Job_6, Job_7, Job_8}. The scheduling goal assigns the jobs to the machines and to the time intervals, such that a job is executed by the specified machine in the specified time. The solutions for the scheduling problems must usually satisfy some constraints or achieve some goals. Since the jobs are assigned to three machines and to four time intervals, each possible scheduling solution can thus be encoded as a 3×4 matrix representation. Table 1 shows an example of a possible schedule, in which the first machine is assigned Job_1, Job_6, and Job_4, respectively, in the first, third, and fourth time intervals, the second machine is assigned Job_5 in the second time interval, and the third machine is assigned Job_2, Job_8, Job_3, and Job_7 in the first to the fourth time intervals, respectively.

Table 1: A possible schedule for the example.

	Time slot 1	Time slot 2	Time slot 3	Time slot 4
Machine 1	Job_1		Job_6	Job_4
Machine 2		Job_5		
Machine 3	Job_2	Job_8	Job_3	Job_7

The possible schedule in Table 1 can be represented as a two-dimensional chromosome, as shown in the following matrix:

$$\begin{bmatrix} 1 & 0 & 6 & 4 \\ 0 & 5 & 0 & 0 \\ 2 & 8 & 3 & 7 \end{bmatrix}. \tag{1}$$

Genetic algorithms require initializing a population of individuals, then gradually updating them by the evolution process. Each individual within the population represents a possible solution state. Not all solutions are feasible, since some violate the problem constraints. Moreover, the offspring generated by the genetic operation may also be infeasible due to the violation of the chromosome representation. The problem can be partially solved by using the fitness function. A penalty value is added to the fitness value of a solution that is infeasible due to some constraints. Repairing mechanisms are then used to convert these infeasible solutions into feasible ones if they are selected. The population initiation process for the proposed two-dimensional encoding method is described as follows.

The Population Initialization Process for the Two-Dimensional Representation

Input. A number S of rows, a number W of columns, and a set of m objects are to be processed.

Output. The output is the ith two-dimensional initial chromosome.

Step 1. Set $k = 1$, where k represents the number of the objects currently being processed.

Step 2. Randomly generate two numbers x and y, $1 \leq x \leq S$ and $1 \leq y \leq W$.

Step 3. If the location $c_i(x, y)$ of the ith chromosome c_i is empty, then assign the kth object at location $c_i(x, y)$; otherwise, repeat Steps 2 and 3 until an empty location is found.

Step 4. Set $k = k + 1$.

Step 5. If $k > m$, then stop the algorithm; otherwise, go to Step 2.

A two-dimensional chromosome is randomly generated after Step 5. For example, in Table 1, the eight jobs are the objects with $m = 8$. In the airline scheduling problem for aircrafts, S represents the number of aircrafts, W represents the number of time slots to depart, and the flights are the objects. The above strategy is efficient when $m \ll S * W$. IF m is nearly equal to $S * W$, then it is better to remove the cells which have already been selected to increase the efficiency of the random process. Other more complicated initialization processes can also be used here.

The design of the genetic operators depends significantly on the encoding method that is used and on the characteristics of the problem to be solved. They are described below.

3. Two-Dimensional Crossover Operations

A crossover operator conventionally exchanges some bits between two chromosomes with probability P_c. Common crossover operators include multiple-point crossover [33], uniform crossover [34], one-point crossover [7], and substring crossover [7]. They are briefly described as follows.

(1) Multipoint Crossover. This method defines a mask to determine the bits to be exchanged between two individuals. Parents exchange bits corresponding to the positions with values of 1 on the mask.

(2) Uniform Crossover Method. This method also defines a mask to determine the bits that should be exchanged between two individuals. However, the bit values of 1 and 0 alternate with each other on the mask.

(3) One-Point Crossover. The mask has only one bit with value 1. That is, the operator randomly selects a single bit within two parents to perform crossover.

(4) Substring Crossover. This method changes arbitrary substrings between two individuals. The lengths and positions of these substrings are chosen at random but are the same for both individuals.

The adopted crossover operator must be appropriately modified for the two-dimensional representation. A two-dimensional substring crossover operator is designed and described as follows.

The Two-Dimensional Substring Crossover

Input. The input is the two chromosomes c_{p1} and c_{p2}.

Output. The output is the two offspring chromosomes, c_{o1} and c_{o2}, which denote the crossover results by c_{p1} and c_{p2}.

Step 1. Generate two random integers R_r and R_c, which represent the two-dimensional crossover point.

Step 2. Generate a random real number R between 0 and 1. If $R > 0.5$, then perform the two-dimensional horizontal substring crossover (Step 3); otherwise, perform the two-dimensional vertical substring crossover (Step 4).

Step 3 (horizontal crossover). Generate the two chromosomes by performing the following substeps.

Substep 3.1. If $\text{row}_i < R_r$, then, for $1 \le \text{col}_j \le W$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$, where W is the number of columns.

Substep 3.2. If $\text{row}_i = R_r$, then, for $1 \le \text{col}_j \le R_c$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$. Additionally, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$ and copy $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ for $R_c < \text{col}_j \le W$.

Substep 3.3. If $\text{row}_i > R_r$, then, for $1 \le \text{col}_j \le W$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$.

Step 4 (vertical crossover). Generate the two chromosomes by the following substeps.

Substep 4.1. If $\text{col}_i < R_c$, then, for $1 \le \text{row}_j \le S$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ and copy $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$, where S is the number of rows.

Substep 4.2. If $\text{col}_i = R_c$, then, for $1 \le \text{row}_j \le R_r$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$. Additionally, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$ for $R_r < \text{row}_j \le S$.

Substep 4.3. If $\text{col}_i > R_c$, then, for $1 \le \text{row}_j \le S$, copy each gene $c_{p1}(\text{row}_i, \text{col}_j)$ to $c_{o2}(\text{row}_i, \text{col}_j)$ and $c_{p2}(\text{row}_i, \text{col}_j)$ to $c_{o1}(\text{row}_i, \text{col}_j)$.

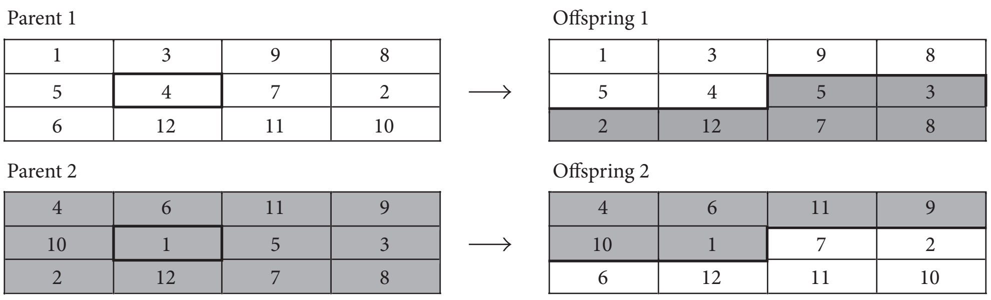
The two offspring chromosomes c_{o1} and c_{o2} are thus formed after the end of Step 4. An example is given below to illustrate the proposed crossover operation.

Example 2. Consider a chromosome encoded as a 3×4 matrix, in which each gene value represents a unique index of a job. Additionally, suppose that the two chromosomes at the left side of Figure 1 are selected as the parents for crossover. Assume that the crossover point is randomly generated as $R_r = 2$ and $R_c = 2$. Figure 1 shows the results after performing the horizontal substring crossover operator on the two parent chromosomes.

Figure 2 shows the results of performing the vertical substring crossover operator.

Note that the two-dimensional substring crossover generates two offspring chromosomes by choosing only one of the two crossover strategies (horizontal or vertical). Alternatively, the two-dimensional crossover operator can be easily modified to generate four offspring chromosomes from a pair of parents by executing the horizontal and the vertical crossovers at the same time.

The new offspring chromosomes that result from executing the crossover operation may become infeasible for some application problems. For instance, the two offspring chromosomes in Example 2 contain identical jobs allocated at different cells, making it apparently unreasonable. This situation typically occurs from permutation representation.



Parent 1

1	3	9	8
5	4	7	2
6	12	11	10

Parent 2

4	6	11	9
10	1	5	3
2	12	7	8

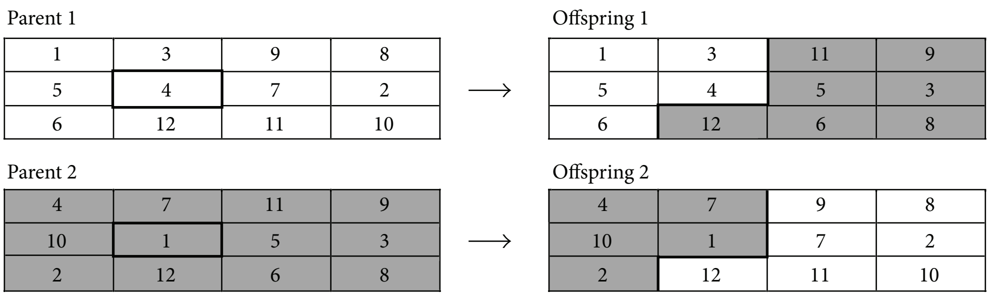
Offspring 1

1	3	9	8
5	4	5	3
2	12	7	8

Offspring 2

4	6	11	9
10	1	7	2
6	12	11	10

FIGURE 1: Horizontal substring crossover for Example 2.



Parent 1

1	3	9	8
5	4	7	2
6	12	11	10

Parent 2

4	7	11	9
10	1	5	3
2	12	6	8

Offspring 1

1	3	11	9
5	4	5	3
6	12	6	8

Offspring 2

4	7	9	8
10	1	7	2
2	12	11	10

FIGURE 2: Vertical substring crossover for Example 2.

Therefore, appropriate two-dimensional repairing mechanisms must be designed to convert infeasible chromosomes into feasible ones.

A two-dimensional repairing algorithm for the permutation representation is proposed below. This algorithm is based on the idea that if two locations have the same value, then the content at one of them can be replaced with the value at the same location of the parents, since genes at the same locations may have similar characteristics. This assumption is quite reasonable, especially for real problems (such as scheduling).

The Two-Dimensional Repairing Algorithm

Input. The input is the two parent chromosomes c_{p1}, c_{p2}, the adopted crossover operation (horizontal or vertical), crossover point (R_r, R_c), and two infeasible offspring chromosomes c_{o1} and c_{o2}.

Output. The output is the two repaired feasible chromosomes c_{r1} and c_{r2} from c_{o1} and c_{o1}.

Step 1. For c_{ok} ($k = 1$ or 2), perform the following steps.

Step 2. Generate a random real number R between 0 and 1.

Step 3. If the crossover is horizontal, then execute Steps 4 and 5; otherwise, execute Steps 6 and 7.

Step 4 (horizontal repair). If $0 \leq R < 0.5$, then repair the genes in the chromosome c_{ok} from point (R_r, R_c) forward to (S, W) in a row-wise manner by performing the following substeps.

Substep 4.1. If a gene located at $(\text{row}_{i1}, \text{col}_{j1})$ of c_{ok} also exists at the previous location $(\text{row}_{i2}, \text{col}_{j2})$ (according to the search direction), replace the gene at $(\text{row}_{i1}, \text{col}_{j1})$ of c_{ok} with the one at location $(\text{row}_{i2}, \text{col}_{j2})$ of $c_{p(3-k)}$.

Substep 4.2. If the new replaced gene $c_{ok}(\text{row}_{i1}, \text{col}_{j1})$ $(= c_{p(3-k)}(\text{row}_{i2}, \text{col}_{j2}))$ still exists at a certain previous location $(\text{row}_{i2}, \text{col}_{j2})$, then repeat Substeps 4.1 and 4.2 until the gene at $c_{ok}(\text{row}_{i1}, \text{col}_{j1})$ no longer appears in the previous locations.

Step 5. If $0.5 \leq R \leq 1$, then repair the genes in the chromosome c_{ok} from the point (R_r, R_c) backward to $(1, 1)$ in a row-wise manner by executing the following substeps.

Substep 5.1. If a gene located at $(\text{row}_{i1}, \text{col}_{j1})$ of c_{ok} also exists at the previous location $(\text{row}_{i2}, \text{col}_{j2})$ (according to the search direction), then replace the gene at $(\text{row}_{i1}, \text{col}_{j1})$ of c_{ok} with the one at location $(\text{row}_{i2}, \text{col}_{j2})$ of c_{pk}.

Substep 5.2. If the new replaced gene $c_{ok}(\text{row}_{i1}, \text{col}_{j1})$ $(= c_{pk}(\text{row}_{i2}, \text{col}_{j2}))$ still exists at a certain previous location $(\text{row}_{i2}, \text{col}_{j2})$, then repeat Substeps 5.1 and 5.2 until the gene at $c_{ok}(\text{row}_{i1}, \text{col}_{j1})$ no longer appears in the previous locations.

Step 6 (vertical repair). If $0 \leq R < 0.5$, repair the genes of c_{ok} from point (R_r, R_c) forward to (S, W) in a column-wise manner.

Step 7. If $0.5 \leq R \leq 1$, then repair the genes of c_{ok} from point (R_r, R_c) backward to $(1, 1)$ in a column-wise manner.

Below, an example is given to illustrate the above repairing algorithm.

Offspring 1:

Repair by moving forward in a row-wise way:

1	3	9	8
5	4	5→10	3→6
2	12	7	8→9→11

Repair by moving backward in a row-wise way:

1	3→2→6	9	8→10
5→7→11	4	5	3
2	12	7	8

Offspring 2:

Repair by moving forward in a row-wise way:

4	6	11	9
10	1	7	2
6→3	12	11→9→8	10→5

Repair by moving backward in a row-wise way:

4	6→2→3	11→7→5	9
10→8	1	7	2
6	12	11	10

Figure 3: The results of row-wise repair of Figure 1.

Offspring 1:

Repair by moving forward in a column-wise way:

1	3	11	9
5	4	5→10	3→7
6	12	6→2	8

Repair by moving backward in a column-wise way:

1	3→2	11	9
5→7	4	5	3
6→11→9→8→10	12	6	8

Offspring 2:

Repair by moving forward in a column-wise way:

4	7	9	8
10	1	7→3	2→6
2	12	11	10→5

Repair by moving backward in a column-wise way:

4	7→5	9	8
10→8→9→11→6	1	7	2
2→3	12	11	10

Figure 4: The results of column-wise repair of Figure 2.

Example 3. Continuing Example 2, the resulting offspring chromosomes in Figure 1 need to be repaired. In Offspring 1, Job 5 is simultaneously assigned to locations (2, 1) and (2, 3), violating the constraint of the chromosome representation. Since the horizontal crossover is performed for the offspring chromosome, the horizontal repairing steps (Steps 4 or 5) are performed. Assuming that the random number generated is below 0.5, Step 4 is then executed. The locations of Job 5 in Offspring 1 are (2, 1) and (2, 3), and the value at (2, 3) needs to be repaired according to Substep 4.1. Job 10 at location (2, 1) in Parent 2 is then assigned to location (2, 3) of Offspring 1. Since Job 10 at location (2, 3) of Offspring 1 does not exist at the previous locations, the repairing process for the location (2, 3) in Offspring 1 is then finished. Similarly, Job 3 at location (2, 4) of Offspring 1 is updated to Job 6. Job 8 at location (3, 4) is first updated to Job 9 and then to Job 11. Figure 3 shows the repair processes for the two possible row-wise repair mechanisms (forward and backward).

Figure 4 shows the repair process for the two possible column-wise repair mechanisms (forward and backward).

4. Two-Dimensional Mutation Operations

Mutation is a genetic operator that is used to maintain the genetic diversity of a population of chromosomes between generations. The conventional mutation operator usually assigns a mutation probability with which an arbitrary bit in a chromosome is changed. A common mutation operator for permutation representation swaps the contents of two arbitrary genes and is appropriately modified here for two-dimensional representation. The proposed two-dimensional mutation operation is described as follows.

The Two-Dimensional Two-Point Swapping Mutation Operation

Input. The input is chromosome c_i and a mutation rate P_m.

Output. The output is the resulting chromosome c_i after it is mutated.

Step 1. Generate a random number R within 0 to 1.

Step 2. If $R > P_m$, then stop the algorithm; otherwise, perform the next step.

Step 3. Generate two random integers, R_r and R_c, where $1 \leq R_r \leq S$ and $1 \leq R_c \leq W$.

Step 4. Generate two random integers, R'_r and R'_c, where $1 \leq R'_r \leq S$ and $1 \leq R'_c \leq W$; if $R_r = R'_r$ and $R_c = R'_c$, then repeat this step to generate another pair of R'_r and R'_c.

Step 5. Interchange $c_i(R_r, R_c)$ with $c_i(R'_r, R'_c)$.

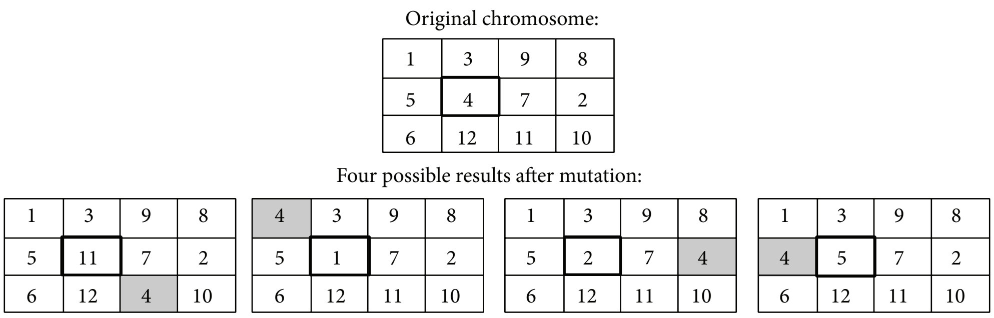


Figure 5: An example for the two-point swapping mutation operator.

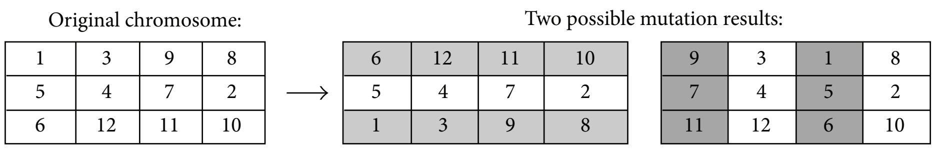


Figure 6: Illustration of the string swapping mutation operator.

An example is given below to demonstrate the proposed mutation operator.

Example 4. Consider the 3×4 two-dimensional chromosome shown at the top of Figure 5. Assume that the content (Job 4) at location (2, 2) is chosen for mutation. It can be swapped with any arbitrary location. The bottom of Figure 5 shows four possible swapping results. The mutation operation can be performed in any direction.

In addition to the two-point swapping mutation, this paper also presents another two-dimensional mutation to exchange two entire rows or columns in a two-dimensional chromosome. This mutation mechanism is described as follows.

The Two-Dimensional String Swapping Mutation

Input. The input is chromosome c_i and a mutation rate P_m.

Output. The output is the resulting chromosome c_i after it is mutated.

Step 1. Generate a random number R between 0 and 1. If $R > 0.5$, then execute the two-dimensional horizontal string mutation (Steps 2 and 3); otherwise, execute the two-dimensional vertical string mutation (Steps 4 and 5).

Step 2 (horizontal string mutation). Generate two random integers, R_{r1} and R_{r2}, where $1 \leq R_{r1}, R_{r2} \leq S$ and $R_{r1} \neq R_{r2}$.

Step 3. Swap $c_i(R_{r1}, \mathrm{Col}_j)$ with $c_i(R_{r2}, \mathrm{Col}_j)$ for $1 \leq \mathrm{Col}_j \leq W$.

Step 4 (vertical string mutation). Generate two random integers, R_{c1} and R_{c2}, where $1 \leq R_{c1}, R_{c2} \leq W$ and $R_{c1} \neq R_{c2}$.

Step 5. Swap $c_i(\mathrm{Row}_i, R_{c1})$ with $c_i(\mathrm{Row}_i, R_{c2})$ for $1 \leq \mathrm{Row}_i \leq S$.

The following example demonstrates the proposed string swapping mutation operator.

Example 5. Consider the original chromosome in Example 3. Figure 6 shows the two possible results after the two-dimensional string swapping mutation operator is performed on the original chromosome. If two numbers, 1 and 3, are randomly generated, the first one swaps rows 1 and 3, while the second one swaps columns 1 and 3.

The string swapping mutation operator can also be extended to swapping two substrings instead of two entire rows or columns.

5. Experiments

This section describes experiments performed to show the performance of the proposed two-dimensional genetic algorithm. They were implemented by Borland C++ Builder on an Intel Core-i7 PC. The proposed algorithm was run with a scheduling problem of assigning aircrafts to a time table in an airline company. The experiments were performed on three data sets, namely, "data_07091," "data_08181," and "data_07092," for scheduling in different months. These data sets contain 88, 78, and 85 jobs, respectively. The parameters were set as S (number of aircrafts) = 10 and W (number of time slots) = 10. Additionally, two basic constraints were set. The first constraint was that an aircraft could only depart at a time slot. The second was that a flight could not be performed twice. In addition, some additional attributes were given for the flights, such as flight locations, connections, and operations. According to the attributes, more constraints were set, including the turnaround-time constraint, the location-connecting constraint, and the operation-cost constraint which considers dining cost and fuel cost. They are described below.

5.1. Turnaround-Time Constraint. The time gap between the arrival and the departure of consecutive flight duties will not be less than the legal turnaround time. This means that an aircraft should be given enough preparation time between two consecutive flights. If (landing time of the previous flight + legal turnaround time − departure time of the next flight) > 0, the penalty value for this constraint may be proportional to the time gap; otherwise, no penalty is caused.

5.2. Connecting-Location Constraint. Since the genetic algorithm is based on a random process (like crossover and mutation operations), it is possible to generate the departure location which mismatches with the previous landing location. In the connection-location constraint, the arrival and the departure locations for a consecutive flight duty should be the same. If two locations are different, an unexecutable plan would be caused, and severe penalty will be given. Otherwise, if the landing location is the same as the departure one, no penalty is caused.

5.3. Dining Cost and Fuel Cost. Dining cost and fuel cost are calculated according to the conditions of airport facility. In other words, the cost of dining cost or fuel cost should depend on different airports and time periods. For example, if the scheduling at ground time is set within the time period from 11:30 to 13:00 and 17:30 to 19:00 local time, the dining cost for the airport should be added. Also, the fuel cost should be charged according to each airport facility status.

5.4. Problem Objective. There are two types of constraints, hard constraints and soft constraints. For any aircraft schedule which violates the hard constraints, the schedule will not be a feasible one. In our problem, turnaround-time and connecting-location are set as hard constraints. Meanwhile, constraints reflecting operation cost, dining cost, and fuel cost are soft constraints, and the objective is to minimize the summation of costs. Therefore, the problem goal is to find a schedule with feasible and minimized cost.

In the first experiment, the population size was set at 100, the crossover rate was 0.8, and the mutation rate was 0.05. The standard roulette wheel selection method was adopted. Figures 7 to 9 show the relationship between the fitness values (the total costs) and the generations for the three data sets. The total costs consisted of three items, the time cost, the location cost, and the operations cost, which were also shown in Figures 7 to 9.

From these figures, the populations converged along with the increase of generations. Experiments were then performed to show the effect of the population size on the problem. Figure 10 plots the curves of the average performance of the three test data sets, for population sizes of 50, 100, and 200, respectively, with all the other parameters unchanged.

From Figure 10, larger population size decreased the number of generations needed for convergence but also increased the computation time. Therefore, the appropriate population size is a tradeoff between solution quality and execution time. From Figure 10, the solution with a population

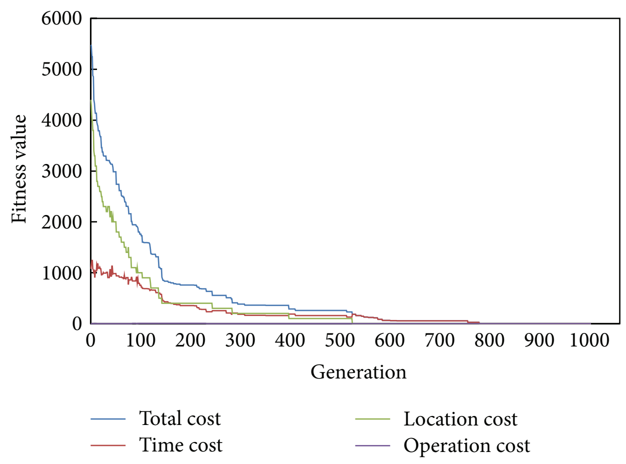


FIGURE 7: The relationship between fitness values and generations for data set data_07091.

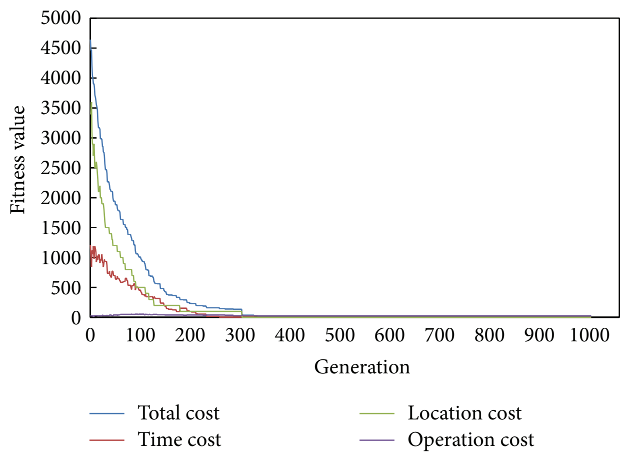


FIGURE 8: The relationship between fitness values and generations for data set data_08181.

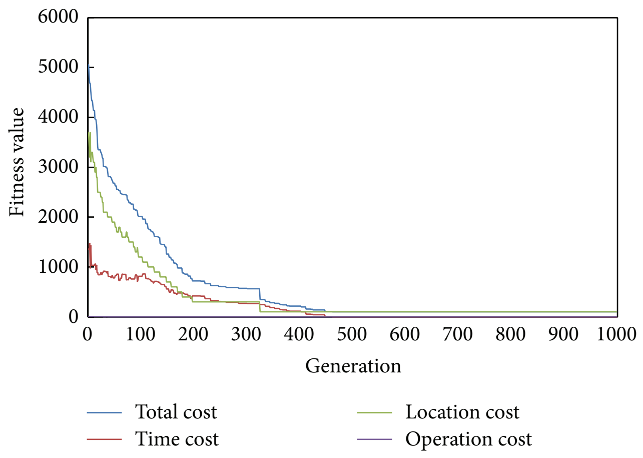


FIGURE 9: The relationship between fitness values and generations for data set data_07092.

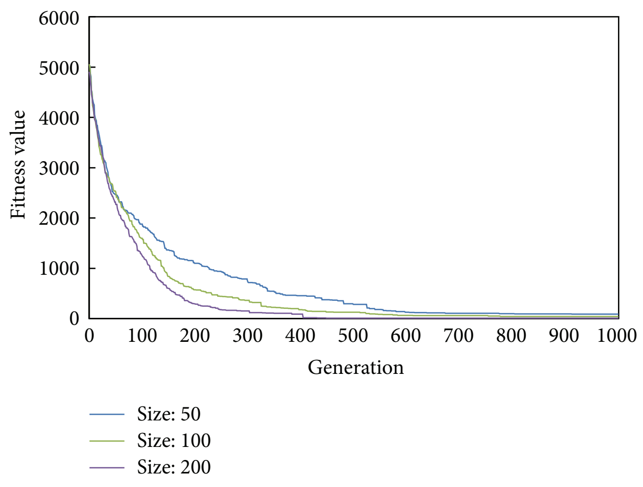


Figure 10: Experimental results for different population sizes.

size of 50 was not good for this problem, since it did not converge quickly in a reasonable period. The solutions with population sizes of 100 and 200 converged around 600 and 500 generations, respectively. Overall, the population size of 100 was found to be most appropriate for this problem.

Many researchers have shown that choosing good genetic parameter values such as crossover rate P_c and mutation rate P_m is not easy [5, 35–37] and depends largely on the characteristics of the problem to be solved [38]. The choice of P_c and P_m is essentially a tradeoff between conservation and exploration. High mutation rates tend to cover the search space well but disrupt partial solutions. On the contrary, low mutation rates tend to keep possible good solutions but do not sufficiently explore the search space. We thus made experiments to show the sensitivity of different crossover and mutation rates on the aircraft scheduling problem. The crossover rate P_c was set from 0.5 to 0.9 in a step of 0.1, and the mutation rate P_m is set at 0.01, 0.05, 0.1, 0.2, and 0.3. Tables 2 to 4 show the effects of P_c and P_m for the three different test data sets, respectively. It can be observed from the experimental results that when the crossover rate P_c was 0.7, 0.6, or 0.5, the performance was not very consistent for the three test data sets. For crossover rate 0.9, it did not perform well for the test data set data_07092. Therefore, $P_c = 0.8$ could be a better choice for the testing problems. In the cases of $P_c = 0.8$, it can be observed that $P_m = 0.05$ can get feasible and minimal cost for the problem. Hence, $P_c = 0.8$ and $P_m = 0.05$ are good choices for the proposed algorithm in the experiments.

Airline scheduling problem is in general NP-hard problem [39], and thus many researchers handle it by approaching a near-optimal solution in reasonable time. In our problem, the minimum value of the objective function is zero. It happens when no soft constraints are violated. According to Tables 2–4, when the generations reached 5000, the proposed approach got the objective value of zero in most cases. In some cases, the approach converged to zero even much earlier. For $P_c = 0.8$ and $P_m = 0.05$, no or only relatively low penalty is obtained in the three test data sets. The proposed approach was thus suitable to applications of aircraft scheduling.

Table 2: Effect of crossover and mutation rates for data set data_07091.

GA parameters		Objective function value in different generations				
P_c	P_m	1000	2000	3000	4000	5000
0.9	0.01	1340	400	280	200	200
	0.05	0	0	0	0	0
	0.1	90	0	0	0	0
	0.2	0	0	0	0	0
	0.3	110	0	0	0	0
0.8	0.01	770	200	200	150	0
	0.05	200	0	0	0	0
	0.1	30	0	0	0	0
	0.2	0	0	0	0	0
	0.3	0	0	0	0	0
0.7	0.01	900	500	500	230	230
	0.05	0	0	0	0	0
	0.1	0	0	0	0	0
	0.2	600	600	600	600	600
	0.3	0	0	0	0	0
0.6	0.01	800	290	30	0	0
	0.05	400	400	200	200	200
	0.1	270	200	200	200	200
	0.2	270	0	0	0	0
	0.3	210	200	200	200	200
0.5	0.01	1750	1220	780	330	330
	0.05	250	250	200	200	0
	0.1	0	0	0	0	0
	0.2	0	0	0	0	0
	0.3	110	0	0	0	0

Figure 11 shows the comparison between the Partially Matched Crossover (PMX) approach [2] and the proposed method. PMX is one-dimensional representation. Its two crossover points are selected randomly from the parent's chromosomes to generate offspring. We thus transformed our approach into one-dimensional representation in a row-wise way for evaluating the performance of the PMX approach. In this experiment, the crossover rate and the mutation rate were set to 0.8 and 0.05, respectively. The lines labeled "PMX" and "proposed method" represent the average fitness values (total costs) obtained for the three data sets by the two methods.

PMX performed better than the proposed method at the beginning, but the proposed method converged more quickly and had better fitness values than PMX after 200 generations. Thus, the proposed two-dimensional operations could cause good effects.

6. Conclusion

Genetic algorithms have become increasingly important for researchers in solving difficult problems since they can provide nearly optimal solutions in a limited amount of

Table 3: Effect of crossover and mutation rates for data set data_08181.

GA parameters		Objective function value in different generations				
P_c	P_m	1000	2000	3000	4000	5000
0.9	0.01	70	50	50	50	50
	0.05	40	40	40	40	40
	0.1	250	250	250	250	250
	0.2	60	60	60	60	60
	0.3	390	390	390	390	390
0.8	0.01	600	440	240	40	40
	0.05	50	40	40	40	40
	0.1	90	90	90	80	80
	0.2	90	90	90	90	80
	0.3	160	60	60	60	60
0.7	0.01	1070	570	50	50	50
	0.05	260	260	260	260	60
	0.1	60	50	50	50	50
	0.2	290	280	260	250	250
	0.3	60	60	60	60	60
0.6	0.01	1470	320	250	250	250
	0.05	60	50	50	50	50
	0.1	260	260	260	260	250
	0.2	40	40	40	40	40
	0.3	110	110	100	100	100
0.5	0.01	1320	790	690	690	690
	0.05	50	50	40	40	40
	0.1	50	50	50	50	50
	0.2	340	260	260	260	260
	0.3	330	80	80	50	50

Table 4: Effect of crossover and mutation rates for data set data_07092.

GA parameters		Objective function value in different generations				
P_c	P_m	1000	2000	3000	4000	5000
0.9	0.01	140	80	10	10	10
	0.05	270	200	200	200	200
	0.1	200	200	200	200	200
	0.2	0	0	0	0	0
	0.3	470	400	400	400	400
0.8	0.01	690	400	400	400	400
	0.05	0	0	0	0	0
	0.1	200	200	200	200	200
	0.2	210	210	200	200	200
	0.3	0	0	0	0	0
0.7	0.01	690	0	0	0	0
	0.05	290	0	0	0	0
	0.1	0	0	0	0	0
	0.2	70	70	70	70	70
	0.3	80	80	80	80	80
0.6	0.01	1260	610	300	200	200
	0.05	200	200	200	200	200
	0.1	120	0	0	0	0
	0.2	0	0	0	0	0
	0.3	270	200	0	0	0
0.5	0.01	920	30	0	0	0
	0.05	210	200	200	200	200
	0.1	200	200	200	200	200
	0.2	140	70	0	0	0
	0.3	280	210	210	210	210

time. Some real problems are in nature suitable for two-dimensional representation. This paper has thus presented a two-dimensional encoding schema and appropriate two-dimensional crossover and mutation operators to solve this kind of problems. The proposed two-dimensional crossover operators have been designed to generate offspring chromosomes with either the horizontal or the vertical approach. A repair mechanism is also proposed to adjust infeasible chromosomes into feasible ones. Several two-dimensional mutation operators, such as two-point swapping, string swapping, and substring swapping, have also been presented. Experiments on an aircraft scheduling problem in an airline company have also been performed to show the two-dimensional effects of the proposed approach. The experimental results show that the proposed two-dimensional genetic algorithm is effective. In the future, we will attempt to extend the proposed approach to solving other problems, which have two-dimensional property in nature.

Conflict of Interests

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

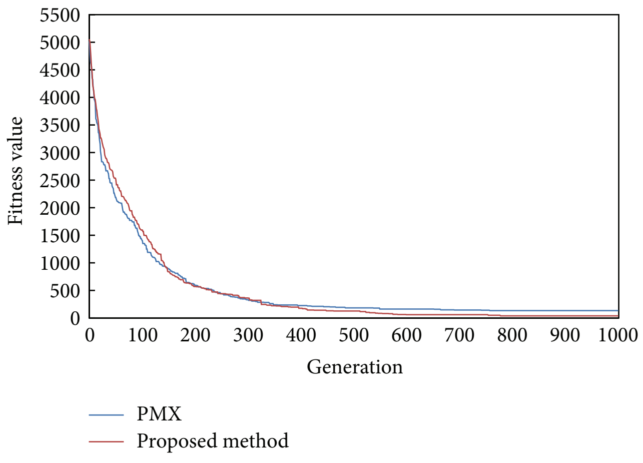


Figure 11: Comparison between PMX and the proposed method.

References

[1] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[3] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[4] B. Filipic and D. Juricic, "An interactive genetic algorithm for controller parameter optimization," in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp. 458–462, Innsbruck, Austria, 1993.

[5] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

[6] J. D. Schaffer, "A study of control parameters affecting on-line performance of genetic algorithms for function optimization," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 675–682, 1989.

[7] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," Tech. Rep. 8, University of Michigan, 1987.

[8] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 789–795, 1990.

[9] G. F. Miller, P. M. Todd, and S. U. Hedge, "Design neural networks using genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 379–384, 1989.

[10] C. L. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 450–457, 1991.

[11] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 509–513, 1991.

[12] K. A. de Jong, "Adaptive system design: a genetic approach," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, no. 9, pp. 566–574, 1980.

[13] M. A. Lee and H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 76–83, 1993.

[14] G. Aiello, G. La Scalia, and M. Enea, "A multi objective genetic algorithm for the facility layout problem based upon slicing structure encoding," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10352–10358, 2012.

[15] K. A. de Jong, *An analysis of the behavior of a class of genetic adaptive systems [Ph.D. thesis]*, University of Michigan, 1975.

[16] M. A. Duarte-Villaseñor, E. Tlelo-Cuautle, and L. G. de la Fraga, "Binary genetic encoding for the synthesis of mixed-mode circuit topologies," *Circuits, Systems, and Signal Processing*, vol. 31, no. 3, pp. 849–863, 2012.

[17] K. T. Lin and P. H. Lin, "Information hiding based on binary encoding methods and crossover mechanism of genetic algorithms," in *Genetic and Evolutionary Computing*, vol. 238 of *Advances in Intelligent Systems and Computing*, pp. 203–212, Springer, 2014.

[18] E. Tlelo-Cuautle, I. Guerra-Gómez, M. A. Duarte-Villaseñor et al., "Applications of evolutionary algorithms in the design automation of analog integrated circuits," *Journal of Applied Sciences*, vol. 10, no. 17, pp. 1859–1872, 2010.

[19] R. Hwang and H. Katayama, "Uniform workload assignments for assembly line by GA-based amelioration approach," *International Journal of Production Research*, vol. 48, no. 7, pp. 1857–1871, 2010.

[20] W. H. Ip, D. Wang, and V. Cho, "Aircraft ground service scheduling problems and their genetic algorithm with hybrid assignment and sequence encoding scheme," *IEEE Systems Journal*, vol. 7, no. 4, pp. 649–657, 2013.

[21] W. Yang, F. T. S. Chan, and V. Kumar, "Optimizing replenishment polices using Genetic Algorithm for single-warehouse multi-retailer system," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3081–3086, 2012.

[22] D. Datta, A. R. S. Amaral, and J. R. Figueira, "Single row facility layout problem using a permutation-based genetic algorithm," *European Journal of Operational Research*, vol. 213, no. 2, pp. 388–394, 2011.

[23] C. Shaefer, "The ARGOT strategy: adaptive representation genetic optimizer technique," in *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Application*, pp. 50–58, 1987.

[24] P. J. Angeline and J. B. Pollack, "Evolutionary module acquisition," in *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pp. 154–163, 1993.

[25] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 183–187, 1985.

[26] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[27] J. P. Cohoon and W. Paris, "Genetic placement," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 422–425, 1986.

[28] C. A. Anderson, K. F. Jones, and J. Ryan, "A two-dimensional genetic algorithm for the Ising problem," *Complex System*, vol. 5, pp. 327–333, 1991.

[29] P.-C. Wang and W. Korfhage, "Process scheduling using genetic algorithms," in *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, pp. 638–641, IEEE, San Antonio, Tex, USA, October 1995.

[30] T. N. Bui and B. R. Moon, "On multi-dimensional encoding/crossover," in *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 49–56, Pittsburgh, Pa, USA, 1995.

[31] A. Sadrzadeh, "A genetic algorithm with the heuristic procedure to solve the multi-line layout problem," *Computers & Industrial Engineering*, vol. 62, no. 4, pp. 1055–1064, 2012.

[32] T.-Y. Chou, T.-K. Liu, C.-N. Lee, and C.-R. Jeng, "Method of inequality-based multiobjective genetic algorithm for domestic daily aircraft routing," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 38, no. 2, pp. 299–308, 2008.

[33] W. M. Spears and K. A. de Jong, "An analysis of multipoint crossover," in *Proceedings of the Workshop of the Foundations of Genetic Algorithms*, pp. 301–315, 1991.

[34] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9, Fairfax, Va, USA, June 1989.

[35] K. A. de Jong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, vol. 10, pp. 38–47, 1990.

[36] R. Myers and E. R. Hancock, "Genetic algorithm parameter sets for line labelling," *Pattern Recognition Letters*, vol. 18, no. 11–13, pp. 1363–1371, 1997.

[37] C. Srinivas, B. R. Reddy, K. Ramji, and R. Naveen, “Sensitivity analysis to determine the parameters of genetic algorithm for machine layout,” in *Proceedings of the 3rd International Conference on Materials Processing and Characterisation*, pp. 866–876, 2014.

[38] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, Wiley-Interscience, New York, NY, USA, 1997.

[39] D. Levine, “Application of a hybrid genetic algorithm to airline crew scheduling,” *Computers & Operations Research*, vol. 23, no. 6, pp. 547–558, 1996.