Two-dimensional Encoding Schema and Genetic Operators

Ming-Wen Tsai[1] Tzung-Pei Hong[2] Tung-Kuan Liu[3]

[1]Department of Management Information Systems, National Chengchi University
[2]Department of Computer Science and Information Engineering, National University of Kaohsiung
[3]Department of Mechanical & Automation Engineering, National Kaohsiung First University of Science and Technology

Abstract

In this paper, we propose a new genetic algorithm based on the two-dimensional encoding method. Appropriate two-dimensional crossover and mutation operations are designed based on the two-dimensional representation to generate the next generations. A two-dimensional repairing mechanism is also proposed to adjust infeasible chromosomes into feasible ones. Experiments are finally made to show the effectiveness of the proposed genetic algorithm.

Keywords: genetic algorithm, chromosome, two-dimensional representation, crossover, mutation, repair.

1. Introduction

Genetic algorithms (GAs) [2] have become increasing important for researchers in solving difficult problems since they can provide feasible solutions in a limited amount of time. They were first proposed by Holland in 1975 [5] and have been successfully applied to the fields of optimization [3], machine learning [2], neural networks [4][11], fuzzy logic controllers [7][12], among others [6][10].

When genetic algorithms are used to solve a problem, a representation that describes the problem states must first be defined. In the past, most representations adopted are linear or one-dimensional, such as the bit string. Some real problems are in nature suitable to two-dimensional representation. For example, in a scheduling problem for assigning jobs to staffs in different time intervals, a two-dimensional array or table is often used to represent the assignment. In this paper, we propose a new genetic algorithm based on the two-dimensional encoding method. Appropriate two-dimensional crossover and mutation operations are designed based on the two-dimensional representation to generate the next generations. The proposed crossover operator may adopt either of the horizontal and vertical ways to generate the offspring chromosomes. A repairing mechanism is also proposed to adjust infeasible chromosomes into feasible ones. Several two-dimensional mutation operators, including single-point swapping, string swapping and substring swapping are presented. Experiments are finally made to show the effectiveness of the proposed genetic algorithm.

2. Review of Related Works

When genetic algorithms are used to solve a problem, a representation that describes the problem states must first be defined. A chromosome representation must first be defined for GAs to proceed. It greatly affects the behavior and performance of GAs. 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][5] 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] are another popular representation used in GAs. 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 [8]. They encode the set of objects into numbers and then arrange them into a chromosome. As to the finite-state representation [1], it first constructs a state transition table according to the given problems, and then evolves according to the transitive table. This method is used for the environment in which sequences of states have some implicit relations and must be generated with the relation. In addition, the parse-tree representation [9] is often used for evolving executable structures, such as a program. Each chromosome is represented by a parse tree.

3. Two-dimensional Chromosome Representation

The notation used in this paper is first defined below.

P: a population consisting of two-dimensional chromosomes,
n: the number of chromosomes in P,
c_i: the i-th chromosome in P, $1 \le i \le n$,
$c_i(x, y)$: the gene located at position (x, y) in the i-th 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,
R_c: A random number indicating the column for crossover, $1 \le i \le W$,
R_r: A random number indicating the row for crossover, $1 \le i \le S$.

A chromosome c_i is thus encoded as an $S \times W$ matrix, with each element $c_i(x, y)$ represents the gene value located at (x, y), $1 \le x \le S$ and $1 \le y \le W$. Genetic algorithms require initializing a population of individuals, then gradually update them by the evolution process. The population initiation process for the proposed two-dimensional encoding method is stated as follows.

The population initialization process for the two-dimensional representation:
Input: a number of rows S, a number of columns W, and a set of m objects to be processed;
Output: the i-th two-dimensional initial chromosome;
Step 1: Set $k = 1$, where k is used to represent the number of the object currently being processed;
Step 2: Randomly generate two numbers x and y, $1 \le x \le S$ and $1 \le y \le W$;
Step 3: If the location $c_i(x, y)$ is empty, assign the k-th 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 $k > m$, stop the algorithm; otherwise go to Step 2.

After Step 5, a two-dimensional chromosome is randomly generated.

4. Two-dimensional crossover operations

A crossover operator conventionally exchanges some bits between two chromosomes with probability P_c. Some common crossover operators are multiple-point crossover, uniform crossover, one-point crossover, substring crossover, among others. In this section, a two-dimensional sub-string crossover operator is designed. It is stated as follows.

The two-dimensional sub-string crossover:
Input: two chromosomes c_{p1} and c_{p2};
Output: two chromosomes, c_{o1} and c_{o2}, which are 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 to 1; if $R > 0.5$, execute the two-dimensional horizontal sub-string crossover (Step 3); otherwise, execute the two-dimensional vertical sub-string crossover (Step 4);
Step 3: (Horizontal Crossover) Generate the two chromosomes by the following sub-steps:
Sub-Step 3-1: If $row_i < R_r$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ for $1 \le col_j \le W$, where W is the number of columns;
Sub-Step 3-2: If $row_i = R_r$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ for $1 \le col_j \le R_c$, and copy each gene $c_{p1}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ for $R_c < col_j \le W$;
Sub-Step 3-3: If $row_i > R_r$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ for $1 \le col_j \le W$;
Step 4: (Vertical Crossover) Generate the two chromosomes by the following sub-steps:
Sub-Step 4-1: If $col_i < R_c$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ for $1 \le row_j \le S$, where S is the number of rows;
Sub-Step 4-2: If $col_i = R_c$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ for $1 \le row_j \le R_r$, and copy each gene $c_{p1}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ for $R_r < row_j \le S$;
Sub-Step 4-3: If $col_i > R_c$, copy each gene $c_{p1}(row_i, col_j)$ to $c_{o2}(row_i, col_j)$ and copy $c_{p2}(row_i, col_j)$ to $c_{o1}(row_i, col_j)$ for $1 \le row_j \le S$.

After Step 4, the two offspring chromosomes c_{o1} and c_{o2} can thus be formed. Below, an example is given to illustrate the proposed crossover operation.

Example 1: Suppose a chromosome is encoded as a 3×4 matrix, in which each gene value represents a unique index of a job. Also suppose the two chromosomes at the left side of Figure 1 are selected as the parents for crossover. Assume the crossover point is randomly generated as $R_r = 2$ and $R_c = 2$. The results after the horizontal sub-string crossover

operator is executed on the two parent chromosomes are shown in Figure 1.

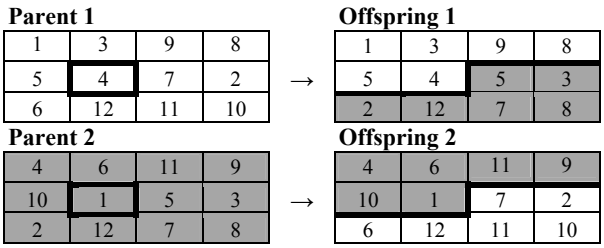

Figure 1. Horizontal sub-string crossover for Example 1

After the crossover operation is executed, the new offspring chromosomes may become infeasible for some problems. This situation usually occurs from permutation representation. Appropriate two-dimensional repairing mechanisms must thus be designed. The repairing algorithm is based on the idea that if two locations have the same value, 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 properties.

The two-dimensional repairing algorithm:

Input: 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: two repaired feasible chromosomes c_{r1} and c_{r2} from c_{o1} and c_{o1};

Step 1: For c_{ok} (k = 1 or 2), do the following steps;

Step 2: Generate a random real number R between 0 to 1;

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

Step 4: (Horizontal Repair) If $0 \le R < 0.5$, repair c_{ok} gene by gene from the point (R_r, R_c) forward to (S, W) in a row-wise way by the following sub-steps:

Sub-Step 4-1: If a gene located at (row_{i1}, col_{j1}) of c_{ok} also exists at the previous location (row_{i2}, col_{j2}) (according to the search direction), replace the gene at (row_{i1}, col_{j1}) of c_{ok} with the one at location (row_{i2}, col_{j2}) of $c_{p(3-k)}$;

Sub-Step 4-2: If the new replaced gene c_{ok}(row_{i1}, col_{j1}) (= $c_{p(3-k)|}$(row_{i2}, col_{j2})) still exists at a certain previous location (row_{i2}, col_{j2}), repeat Sub-Steps 4-1 and 4-2 until the gene at c_{ok}(row_{i1}, col_{j1}) does not appear in the previous locations;

Step 5: If $0.5 \le R \le 1$, repair c_{ok} gene by gene from the point (R_r, R_c) backward to (1, 1) in a row-wise way by the following sub-steps:

Sub-Step 5-1: If a gene located at (row_{i1}, col_{j1}) of c_{ok} also exists at the previous location (row_{i2}, col_{j2}) (according to the search direction), replace the gene at (row_{i1}, col_{j1}) of c_{ok} with the one at location (row_{i2}, col_{j2}) of c_{pk};

Sub-Step 5-2: If the new replaced gene c_{ok}(row_{i1}, col_{j1}) (= c_{pk}(row_{i2}, col_{j2})) still exists at a certain previous location (row_{i2}, col_{j2}), repeat Sub-Steps 5-1 and 5-2 until the gene at c_{ok}(row_{i1}, col_{j1}) does not appear in the previous locations;

Step 6: (Vertical Repair) If $0 \le R < 0.5$, repair c_{ok} gene by gene from the point (R_r, R_c) forward to (S, W) in a column-wise way;

Step 7: If $0.5 \le R \le 1$, repair c_{ok} gene by gene from the point (R_r, R_c) backward to (1, 1) in a column-wise way.

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

Example 2: Continuing Example 1, the resulting offspring chromosomes in Figure 1 need to be repaired. The repairing process for the two possible repairing mechanisms (forward and backward) in a row-wise way is shown in Figure 2.

Offspring 1:

Repair by moving forward in a row-wise way:				Repair by moving backward in a row-wise way:			
1	3	9	8	1	3→2→6	9	8→10
5	4	5→10	3→6	5→7→11	4	5	3
2	12	7	8→9→11	2	12	7	8

Offspring 2:

Repair by moving forward in a row-wise way:				Repair by moving backward in a row-wise way:			
4	6	11	9	4	6→2→3	11→7→5	9
10	1	7	2	10→8	1	7	2
6→3	12	11→9→8	10→5	6	12	11	10

Figure 2. The repair results of Figure 1 in a row–wise way

5. Two-dimensional mutation operations

Mutation is a genetic operator used to keep genetic diversity of a population of chromosomes from one generation to the next one. The conventional mutation operator usually assigns a mutation probability with which an arbitrary bit in a chromosome will be changed. For permutation representation, a common mutation operator is to swap the contents of two arbitrary genes. It is appropriately modified here for two-dimensional representation. The proposed two-dimensional mutation operation is described as follow.

The two-dimensional single-point swapping mutation operation:

Input: a chromosome c_i and a mutation rate P_m;

Output: the resulting chromosome c_i after it is mutated;

Step 1: Generate a random number R between 0 to 1;
Step 2: If $R > P_m$, stop the algorithm; otherwise do the next step;
Step 3: Generate two random integers, R_r and R_c, where $1 \le R_r \le S$ and $1 \le R_c \le W$;
Step 4: Generate two random integers, R_r' and R_c', where $1 \le R_r' \le S$ and $1 \le R_c' \le W$; if $R_r = R_r'$ and $R_c = R_c'$, repeat this step to generate another pair of R_r' and R_c';
Step 5: Swap $c_i(R_r, R_c)$ with $c_i(R_r', R_c')$.

6. The Experiments

This section reports on experiments made to show the performance of the proposed two-dimensional genetic algorithm. They were implemented by Borland C++ Builder on a Pentium III PC. A scheduling problem of assigning jobs to staffs was used and tested by the proposed algorithm. There are 88 jobs, $S = 10$, $W = 10$, and two constraints might be satisfied in the problem. The first constraint was that a staff could only do one job at a time slot. The second one was a job could not be done twice. There were different costs for a job to be done by different staffs in different time slots. The purpose was to find a good schedule for minimizing the total costs. In all the experiments, the population size was set at 100, the crossover rate was 0.8, and the mutation rate was 0.01. The conventional roulette wheel selection method was used. The relationship between the fitness values and the generations is shown in Figure 3.

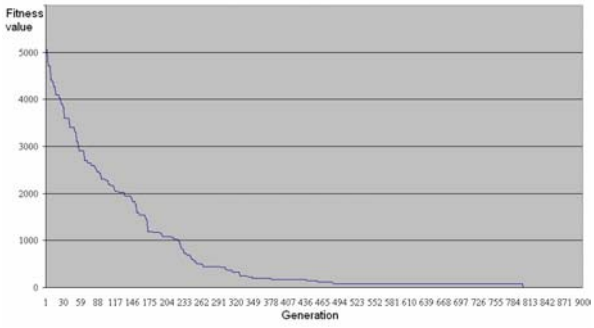

Figure 3. The relationship between the fitness values and the generations

It can be seen from Figure 3 that the population converged after about 500 generations.

7. Conclusion

This paper has presented a two-dimensional encoding schema and appropriate two-dimensional crossover and mutation operators based on the schema. The proposed crossover operator may adopt either of the horizontal and vertical ways to generate the offspring chromosomes. A repairing mechanism is also proposed to adjust infeasible chromosomes into feasible ones. Experiments for solving a two-dimensional scheduling problem has been tested, with the results showing the effectiveness of the proposed genetic algorithm. In the future, we will attempt to extend our approach to solving other problems.

8. References

[1] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, Wiley, 1966.

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

[3] 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.

[4] S. A. Harp, T. Samad, and A. Guha, "Towards the Genetic Synthesis of Neural Networks," *Proceedings of the Third International Conference on Genetic Algorithms*, pp.360-369, 1989.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[6] D. Jong, "Adaptive System Design: A Genetic Approach," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 10, pp. 566-574, 1980.

[7] C. L. Karr, "Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp.450-457, 1991.

[8] R. M. Knuth, *The Art of Computer Programming volume 3: Sorting and Searching*, Addison-Wesley, 1973.

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

[10] L. Kuncheva, "Genetic Algorithm for Feature Selection for Parallel Classifiers," *Information Processing Letters,* 46, pp.163-168, 1993.

[11] G. F. Miller, P. M. Todd, and S. U. Hedge, "Design Neural Networks Using Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms*, pp.379-384, 1989.

[12] P. Thrift, "Fuzzy Logic Synthesis with Genetic Algorithms," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp.509-513, 1991.