

# An Orthogonal Genetic Algorithm for Multimedia Multicast Routing

Qingfu Zhang and Yiu-Wing Leung, *Senior Member, IEEE*

**Abstract**—Many multimedia communication applications require a source to send multimedia information to multiple destinations through a communication network. To support these applications, it is necessary to determine a multicast tree of minimal cost to connect the source node to the destination nodes subject to delay constraints on multimedia communication. This problem is known as multimedia multicast routing and has been proved to be NP-complete. This paper proposes an orthogonal genetic algorithm for multimedia multicast routing. Its salient feature is to incorporate an experimental design method called orthogonal design into the crossover operation. As a result, it can search the solution space in a statistically sound manner and it is well suited for parallel implementation and execution. We execute the orthogonal genetic algorithm to solve two sets of benchmark test problems. The results indicate that for practical problem sizes, the orthogonal genetic algorithm can find near-optimal solutions within moderate numbers of generations.

**Index Terms**—Genetic algorithm, multimedia multicast routing, NP-complete.

## I. INTRODUCTION

MANY multimedia communication applications require a source to send multimedia information to multiple destinations through a communication network. Several application examples include the following.

- 1) In a multiparty multimedia teleconference, the video and voice captured at each conference site are sent to all the other conference sites so that all the conferees can see and communicate with each other in real time [2]–[3].
- 2) In a remote video lecture for distant education, the video and voice of the instructor are sent to all the students through a communication network [4].
- 3) In video-on-demand systems, the major bottleneck is to retrieve video from the disks [5]. To relieve this bottleneck, we can batch a number of customer requests for the same video object and then use one I/O stream to serve multiple customers [5]. Using this method, multimedia information is sent from the video server to multiple customers.

Manuscript received May 19, 1997; revised May 27, 1998 and November 2, 1998. This work was supported by the HKBU FRG Research Grant/97–98/II-08, the NSF of China, the NSF of Hunan, and the Open Project Foundation of Software Engineering National Key Laboratory of Wuhan University.

Q. Zhang is with GMD—German National Research Center for Information Technology, St. Augustin, 53754 Germany, and with Department of Computer, Changsha Institute of Technology, Changsha, 410073, China.

Y.-W. Leung is with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong (e-mail: y.leung@ieee.org).

Publisher Item Identifier S 1089-778X(99)01663-X.

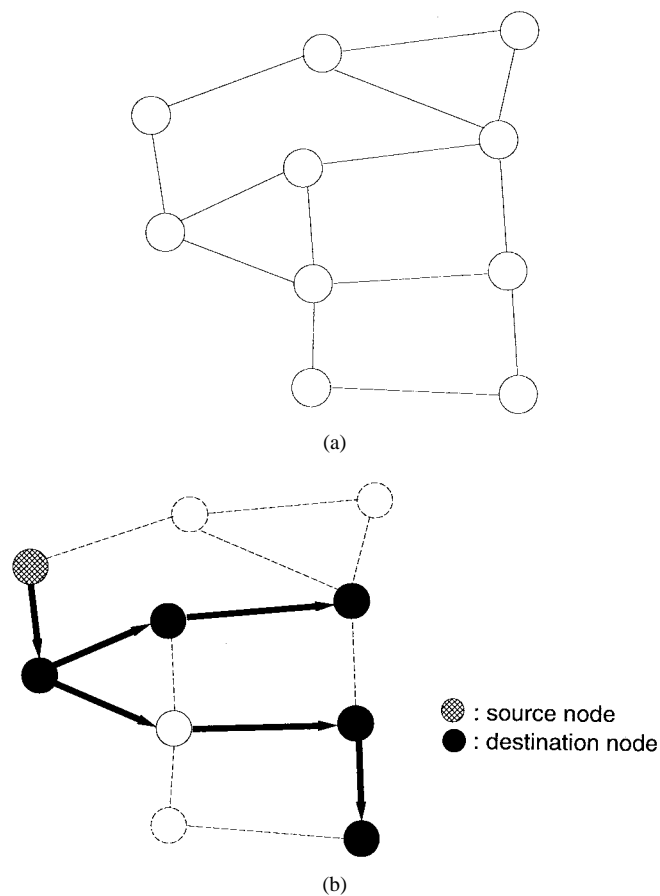


Fig. 1. An example of a multicast tree. (a) A communication network; both the cost and delay of every edge is 1.0. (b) A multicast tree; the delay from the source node to any destination node must be smaller than 5.0.

To support these applications, it is necessary to determine a multicast tree of minimal cost [1], [6] for every communication session. Through this multicast tree, the source node can send multimedia information to all the destination nodes. Since multimedia information cannot be excessively delayed [7], the total delay from the source node to any destination node must be smaller than a given requirement. Fig. 1 shows an example of a multicast tree. The problem of determining multicast trees is known as multimedia multicast routing and has been proved to be NP-complete [1]. Therefore, good heuristic algorithms are of practical interest.

Kompella *et al.* [1] proposed a heuristic algorithm for multimedia multicast routing. In this algorithm, it is necessary to solve the delay constrained shortest path problem  $d(d-1)/2$  times, where  $d$  is the number of source and destination

nodes. This problem is NP-complete [8]–[9], however, and the Kompella’s heuristic algorithm takes a pseudopolynomial time.

Zhu *et al.* [6] proposed another heuristic algorithm for multimedia multicast routing. This algorithm is also required to solve the delay-constrained shortest path problem.

Evolutionary algorithms are promising for solving many hard optimization and search problems [10]–[14]. Two genetic algorithms were proposed to solve the multicast routing problem without delay constraints<sup>1</sup> [16], [17]. In these algorithms, one of the steps is to compute the shortest paths between all pairs of nodes in the network. If we directly modify these algorithms to include the delay constraints, the resulting algorithms must solve the delay-constrained shortest path problem (i.e., solve an NP-complete problem). To the best of our knowledge, there is no existing genetic algorithm for multimedia multicast routing. It is desirable to design a genetic algorithm that is not required to solve any hard subproblem but can give nearly optimal solutions for multimedia multicast routing.

We recently advocated incorporating experimental design methods into the genetic algorithm [18]. Our main idea is based on the observation that some major steps in the genetic algorithm can be considered to be “experiments.” For example, the sampling of genes from the parents for crossover can be considered to be an experiment. If we apply the sophisticated experimental design methods [19] to strengthen these experiments, the resulting genetic algorithms can be statistically sound and have a better performance.

In this paper, we design a genetic algorithm using an experimental design method for multimedia multicast routing. Its salient feature is to incorporate the orthogonal design into crossover. As a result, it can search the solution space in a statistically sound manner, and it is well suited for parallel implementation and execution. We execute the proposed algorithm to solve two sets of benchmark test problems to determine its effectiveness.

## II. NETWORK MODEL AND PROBLEM FORMULATION

The communication network is modeled as a graph  $G = (V, E)$ , where  $V$  is a set of nodes and  $E$  is a set of edges. The cost and delay of edge  $e$  are denoted by  $C(e)$  and  $D(e)$ , respectively, where  $C(e)$  and  $D(e)$  can assume any positive real numbers. The cost may measure the monetary utilization cost or the degree of congestion,<sup>2</sup> and the delay measures the time required to transmit a piece of information through this edge. We let the source node be node  $s$  and the set of destination nodes be  $S$ , where  $\{s\} \cup S \subseteq V$ . The delay requirement specifies that the total delay from the source node to any destination node must be smaller than or equal to  $\Delta$ , where  $\Delta$  can be any positive number.

<sup>1</sup>When there is no delay constraint, the problem is also known as the Steiner tree problem in graph theory and it is still NP-complete [15].

<sup>2</sup>For example, when most of the channels in an edge are being occupied, we can choose to assign a larger cost to this edge. This can encourage the new sessions to adopt the other edges, thereby balancing the traffic over the whole network.

TABLE I  
AN EXPERIMENTAL DESIGN PROBLEM WITH THREE  
FACTORS AND THREE LEVELS PER FACTOR

Temperature ( $A$ )	80°C	85°C	90°C
Time ( $B$ )	10 sec	20 sec	30 sec
Amount of Catalyst ( $C$ )	1 mg	2 mg	3 mg

The multimedia multicast routing problem is to determine a multicast tree connecting the source node to every destination node such that the cost of this tree is minimal while the total delay from the source node to any destination node is smaller than  $\Delta$ . Mathematically, the problem is to find a tree  $T = (V_T, E_T)$  (where  $V_T \subseteq V$  and  $E_T \subseteq E$ ) such that the total cost of this tree  $\sum_{e \in E_T} C(e)$  is minimized subject to the following two constraints:

- 1)  $\{s\} \cup S \subseteq V_T$ ;
- 2)  $\sum_{e \in P(s,v)} D(e) \leq \Delta$  for every  $v \in S$ , where  $P(s, v)$  is a set of edges constituting the path from source node  $s$  to destination node  $v$  in the tree  $T$ .

## III. EXPERIMENTAL DESIGN METHODS

In this section, we briefly introduce the concept of experimental design methods. For more details, the readers can refer to [19] and [20].

To aid explanation, we consider the following example. The yield of a chemical product depends on three variables: temperature ( $A$ ), time ( $B$ ), and amount of catalyst ( $C$ ) at the values shown in Table I. The temperature, time, and amount of catalyst are called the factors of the experiment. Each factor has three possible values, and we say that each factor has three levels.

To find the best combination of levels for maximum yield, we can do one experiment for every combination and then select the best one. In the above example, there are  $3 \times 3 \times 3 = 27$  combinations. In general, when there are  $k$  factors at  $n$  levels, the number of combinations is  $n^k$ . When the number of factors and the number of levels are small, it may be feasible to test all the combinations.

Very often, it is not possible or cost effective to test all the combinations. It is desirable to sample a small but representative sample of combinations for testing. The orthogonal design was developed for this purpose [19], [20]. It provides a series of orthogonal arrays for different number of factors and different number of levels. We let  $L_m(n^k)$  denote an orthogonal array for  $k$  factors and  $n$  levels, where “ $L$ ” denotes a Latin square and  $m$  is the number of combinations of levels to be tested. Two orthogonal arrays  $L_4(2^3)$  and  $L_9(3^4)$  are shown in Table II and Table III, respectively. In these tables, each row represents a combination of levels. The orthogonality of an array means that 1) for the factor in any column, every level occurs the same number of times; 2) for the two factors in any two columns, every combination of two levels occurs the same number of times; and 3) the selected combinations are uniformly distributed over the whole space of all the possible combinations. Fig. 2 illustrates the meaning of orthogonality

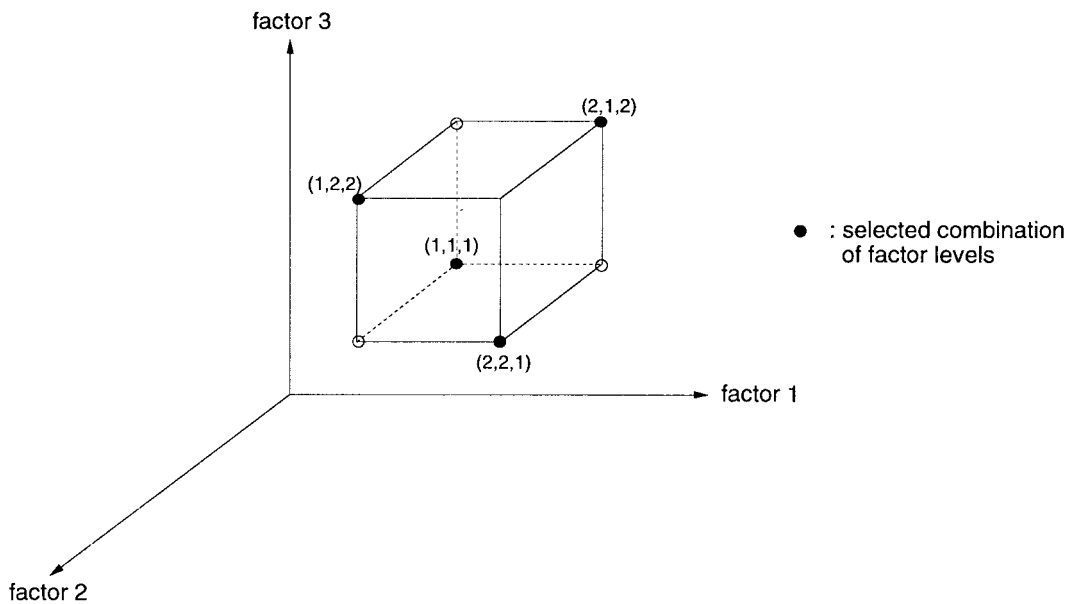


Fig. 2. Orthogonality of the orthogonal array  $L_4(2^3)$ .

TABLE II  
ORTHOGONAL ARRAY  $L_4(2^3)$  FOR THREE FACTORS AT TWO LEVELS; THERE ARE FOUR COMBINATIONS OF FACTOR LEVELS

Combination	Factor 1	Factor 2	Factor 3
1st	1	1	1
2nd	1	2	2
3rd	2	1	2
4th	2	2	1

TABLE III  
ORTHOGONAL ARRAY  $L_9(3^4)$  FOR FOUR FACTORS AT THREE LEVELS; THERE ARE NINE COMBINATIONS OF FACTOR LEVELS

Combination	Factor 1	Factor 2	Factor 3	Factor 4
1st	1	1	1	1
2nd	1	2	2	2
3rd	1	3	3	3
4th	2	1	2	3
5th	2	2	3	1
6th	2	3	1	2
7th	3	1	3	2
8th	3	2	1	3
9th	3	3	2	1

for  $L_4(2^3)$ . It has been proved that the orthogonal design is optimal for additive model and quadratic model, and the selected combinations are good representatives for all the possible combinations [21].

In the above example, there are 27 combinations of levels to be tested. We apply the orthogonal design to select nine representative combinations to be tested, and these nine combinations are shown in Table IV.

TABLE IV  
THERE ARE THREE FACTORS. WE REMOVE THE LAST FACTOR OF  $L_9(3^4)$  TO YIELD AN ORTHOGONAL ARRAY WITH THREE FACTORS AND APPLY THIS RESULTING ARRAY TO SELECT NINE REPRESENTATIVE COMBINATIONS

Combination	Factor		
	Temperature (A)	Time (B)	Amount of Catalyst (C)
1st	80°C	10 sec	1 mg
2nd	80°C	20 sec	2 mg
3rd	80°C	30 sec	3 mg
4th	85°C	10 sec	2 mg
5th	85°C	20 sec	3 mg
6th	85°C	30 sec	1 mg
7th	90°C	10 sec	3 mg
8th	90°C	20 sec	1 mg
9th	90°C	30 sec	2 mg

#### IV. ORTHOGONAL GENETIC ALGORITHM FOR MULTIMEDIA MULTICAST ROUTING

##### A. Binary String Representation and Fitness Vector

We number the edges in the graph  $G$  from 1 to  $l$ . We can then represent any multicast tree  $T$  of the graph  $G$  as a binary string  $(e_1, e_2, \dots, e_l)$  where

$$e_i = \begin{cases} 1, & \text{if edge } i \text{ is included in the multicast tree } T \\ 0, & \text{otherwise.} \end{cases}$$

To measure the quality of any multicast tree, we propose the fitness vector  $\mathbf{f} = (f_1, f_2)$  where

$$\begin{cases} f_1 = \sum_{e \in E_T} C(e) \\ f_2 = \sum_{v \in S} \max\{0, \sum_{e \in P(s,v)} D(e) - \Delta\}. \end{cases}$$

In other words,  $f_1$  measures the cost of the multicast tree and  $f_2$  measures how tight the delay constraints are fulfilled.

A multicast tree is good if  $f_1$  is small (i.e., a small cost) and  $f_2$  is zero (i.e., the delay constraints are fulfilled). Given two multicast trees  $T_1$  and  $T_2$  with respective fitness vectors  $(f'_1, f'_2)$  and  $(f''_1, f''_2)$ , we say that  $T_1$  is better than  $T_2$  if and only if

- 1)  $f'_2 < f''_2$  or
- 2)  $f'_2 = f''_2$  and  $f'_1 < f''_1$ .

In this manner, we can minimize the cost of the multicast tree and enforce fulfilling the delay constraints.

### B. Check-and-Repair Operation

In genetic algorithms, crossover and mutation are adopted to search the solution space. After performing these operations, we can induce a subgraph from the resulting binary string. This subgraph, however, may not be a tree connecting the source node to all the destination nodes. To remedy this, we design the check-and-repair operation. This operation applies a simple method called the graph search method [22] to traverse the subgraph. In particular, it applies this method to find a spanning tree from the subgraph and then connect the unconnected destination nodes to the tree. The steps are as follows.

#### Check-and-Repair Operation:

Input: an input binary string  
Output: an output binary string

- Step 1) If the subgraph induced by the input binary string is a tree connecting the source node to all the destination nodes, then stop.
- Step 2) Apply the graph search method to find a spanning tree from the subgraph.
- Step 3) For the destination nodes that are not included in the tree, apply the graph search method to connect them to the tree one after the other. The output binary string represents this tree.

The subgraph is only a part of the entire graph. The check-and-repair operation operates on this subgraph using the simple graph search method, and hence it can be executed quickly. In Section V, we will demonstrate its effectiveness using numerical experiments.

### C. Orthogonal Crossover Based on Orthogonal Array

In this section, we incorporate the orthogonal design [19] into the crossover operation, so that we have a statistically sound method to sample the genes from multiple parents for crossover. The resulting operation is called orthogonal crossover.

There are a very large number of possibilities of sampling the parent strings for crossover (e.g., see [23]–[24]). We apply an orthogonal array to perform a rational and representative sampling. To explain our main idea, we consider the example shown in Fig. 3, in which we use the orthogonal array  $L_4(2^3)$  to sample the genes from two parents for crossover. Since  $L_4(2^3)$  has three factors (see Table II), we divide each parent

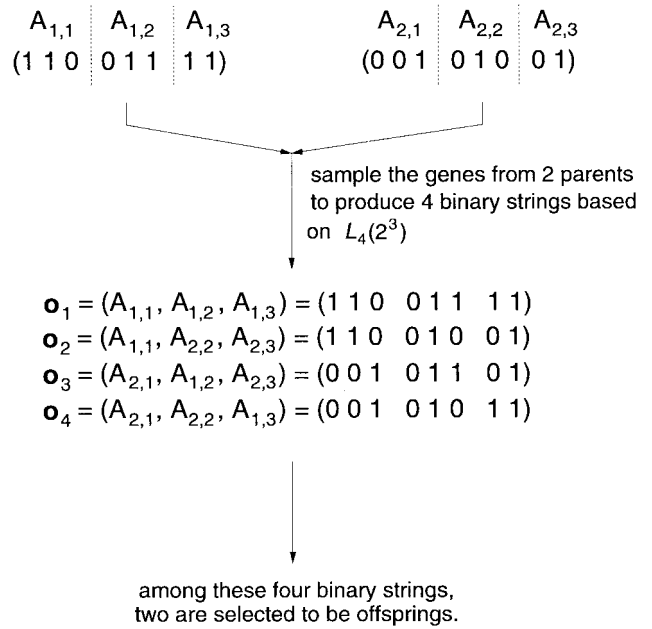


Fig. 3. The orthogonal array  $L_4(2^3)$  is used to sample the genes from two parents for crossover.

string into three parts. We sample these parts from the two parents based on the four combinations of factor levels in  $L_4(2^3)$ , thereby producing four binary strings. Among these four binary strings, we select two of them to be the offspring. In general, for  $L_m(n^k)$  with  $k$  factors and  $m$  combinations of factor levels, we divide each parent string into  $k$  parts, sample these parts from  $n$  parents based on the  $m$  combinations in  $L_m(n^k)$  to produce  $m$  binary strings, and then select  $j$  of them to be the offspring. The details are given as follows.

#### $n$ -to- $j$ Orthogonal Crossover and Mutation:

Inputs:  $n$  parent strings  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{il})$   
for  $1 \leq i \leq n$   
Outputs:  $j$  offspring strings  $\bar{\mathbf{o}}_i = (\bar{o}_{i1}, \bar{o}_{i2}, \dots, \bar{o}_{il})$   
for  $1 \leq i \leq j$

- Step 1) Randomly and independently generate  $p(i) \in \{1, 2, \dots, k\}$  for all  $1 \leq i \leq l$ .
- Step 2) Based on the  $i$ th combination of factor levels  $(a_1(i), a_2(i), \dots, a_k(i))$  in the orthogonal array  $L_m(n^k)$ , produce the binary string  $\mathbf{o}_i = (x_{a_{p(1)}(i),1}, x_{a_{p(2)}(i),2}, \dots, x_{a_{p(l)}(i),l})$  for all  $1 \leq i \leq m$ .
- Step 3) Each  $\mathbf{o}_i$  undergoes mutation with a small probability  $p_m$ . To perform mutation on a binary string, flip every bit in this string with a small probability  $p_f$ .
- Step 4) Execute the check-and-repair operation on  $\mathbf{o}_i$  for all  $1 \leq i \leq m$ .
- Step 5) Evaluate the fitness vectors of  $\mathbf{o}_1, \mathbf{o}_2, \dots$  and  $\mathbf{o}_m$ , and then select  $j$  of them to be the offspring (these offspring are denoted by  $\bar{\mathbf{o}}_1, \bar{\mathbf{o}}_2, \dots$  and  $\bar{\mathbf{o}}_j$ ).

In Step 1), we generate the indexes  $p(i)$  ( $1 \leq i \leq l$ ) to partition each parent string into  $k$  parts. In Step 2), we sample these parts from  $n$  parents based on  $L_m(n^k)$  to produce

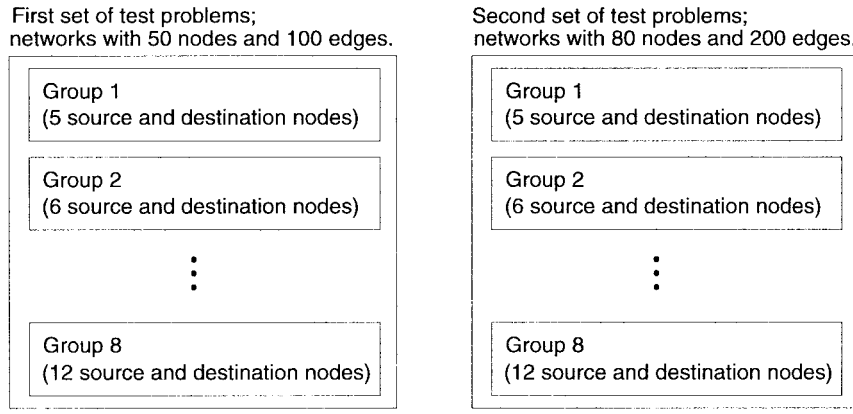


Fig. 4. Two sets of test problems. Every group consists of 100 test problems. There are a total of 1600 test problems.

$m$  binary strings. In Step 3), we perform mutation with a small probability. In Step 4), we execute the check-and-repair operation on every string. In Step 5), we evaluate the fitness vector of every string and then select  $j$  of them to be the offspring.

*Example:* Consider a three-to-two orthogonal crossover and mutation operation based on  $L_9(3^4)$ . Let the parent strings be

$$\begin{cases} \mathbf{x}_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{1,5}) \\ \mathbf{x}_2 = (x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4}, x_{2,5}) \\ \mathbf{x}_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}) \end{cases}$$

In Step 1), we can choose  $p(1) = 1$ ,  $p(2) = 2$ ,  $p(3) = 1$ ,  $p(4) = 3$ , and  $p(5) = 4$ . In Step 2), we produce nine binary strings  $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_9$  based on the nine combinations in the orthogonal array  $L_9(3^4)$  (see Table III)

$$\begin{cases} \mathbf{o}_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{1,5}) \\ \mathbf{o}_2 = (x_{1,1}, x_{2,2}, x_{1,3}, x_{2,4}, x_{2,5}) \\ \mathbf{o}_3 = (x_{1,1}, x_{3,2}, x_{1,3}, x_{3,4}, x_{3,5}) \\ \mathbf{o}_4 = (x_{2,1}, x_{1,2}, x_{2,3}, x_{2,4}, x_{3,5}) \\ \mathbf{o}_5 = (x_{2,1}, x_{2,2}, x_{2,3}, x_{3,4}, x_{1,5}) \\ \mathbf{o}_6 = (x_{2,1}, x_{3,2}, x_{2,3}, x_{1,4}, x_{2,5}) \\ \mathbf{o}_7 = (x_{3,1}, x_{1,2}, x_{3,3}, x_{3,4}, x_{2,5}) \\ \mathbf{o}_8 = (x_{3,1}, x_{2,2}, x_{3,3}, x_{1,4}, x_{3,5}) \\ \mathbf{o}_9 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{2,4}, x_{1,5}) \end{cases}$$

Since  $\mathbf{o}_1$  is equal to the first parent string  $\mathbf{x}_1$ , we execute the check-and-repair operation on  $\mathbf{o}_2, \mathbf{o}_3, \dots$  and  $\mathbf{o}_9$ . In Step 3), we perform mutation with a small probability. In Step 4), we perform the check-and-repair operation. In Step 5), we evaluate their fitness vectors and then select two of them to be the offspring.

After executing Step 2), a few binary strings may be equal to the parent strings. In the above example,  $\mathbf{o}_1$  is equal to the first parent string, and it competes with its immediate offspring. This direct competition is similar to but different from that studied in [25]–[26].

In orthogonal crossover, one of the main design choices is the orthogonal array. In addition to  $L_4(2^3)$  and  $L_9(3^4)$

shown in Tables II and III, there are orthogonal arrays having larger size. In general, a larger orthogonal array provides more combinations of factor levels, and hence it can perform a better sampling for crossover. The resulting computational complexity, however, is also larger. As we shall demonstrate in Section V,  $L_9(3^4)$  is already a good choice for multimedia multicast routing with practical problem sizes.

In Step 5) of the orthogonal crossover operation, we select  $j$  offspring among the  $m$  binary strings produced in Step 2). There are many possible offspring selection schemes. For example, to ensure a monotonic crossover operation (i.e., the best offspring is not poorer than the best parent), we can simply select the best parent to be an offspring and the best one among  $\mathbf{o}_1, \mathbf{o}_2, \dots$  and  $\mathbf{o}_m$  to be another offspring. As another example, we can also introduce randomness into the selection scheme (say,  $\mathbf{o}_i$  with a lower fitness can be chosen to be an offspring with a small probability). In our experiments (see Section V), the best  $j$  offspring among  $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_m$  are selected.

#### D. Orthogonal Genetic Algorithm

Initially, we randomly generate a population of  $N$  binary strings, where the number of bits in every string is equal to the number of edges in the network. We then apply the  $n$ -to- $j$  orthogonal crossover and mutation operation to evolve and improve the population iteratively. Since  $n$  can be different from  $j$ , we perform this operation  $\frac{N}{j}$  times in each generation to maintain a constant population size. The details are as follows.

*Orthogonal Genetic Algorithm:*

Inputs: network  $G = (V, E)$ , edge cost  $C(e)$  and edge delay  $D(e)$  for all  $e \in E$ ; delay requirement  $\Delta$ ; source node  $s$  and set of destination nodes  $S$

Output: binary string representing the multicast tree

##### Step 1) Initialization

Randomly create an initial generation of  $N$  binary strings  $P_0 = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ , and initialize the generation number  $gen$  to 0.

##### Step 2) Population Evolution

WHILE (stopping condition is not met) DO  
BEGIN

Step 2.1) ***n*-to-*j* Orthogonal Crossover and Mutation**  
 DO  $\frac{N}{j}$  times

Randomly select  $n$  parent strings from  $P_{gen}$ , and perform *n*-to-*j* orthogonal crossover and mutation on them to generate  $j$  offspring  $\bar{o}_1, \bar{o}_2, \dots, \bar{o}_j$  for the next generation  $P_{gen+1}$ .

END

Step 2.2) Increment the generation number *gen* by 1.

END

In Step 2), the population is evolved and improved iteratively until a stopping condition is met. One possible stopping criterion is to halt when the number of generations *gen* is equal to a given maximum value.

In many traditional genetic algorithms, global selection of parents is needed, and this is a serious bottleneck in parallel implementation. In the orthogonal genetic algorithm, selection is performed only within the orthogonal crossover operation but global selection is not needed. In each generation, all the  $N/j$  orthogonal crossover and mutation operations can be executed in parallel. Therefore, the orthogonal genetic algorithm is well suited for parallel implementation and execution.

To create a new generation, orthogonal crossover is executed  $N/j$  times and mutation is executed  $p_m N$  times on average. In every execution of orthogonal crossover and mutation,  $m$  and 1 new binary strings are produced, respectively. Therefore, the algorithm has to handle an average of  $((mN/j) + p_m N)$  new binary strings in every generation. For the special cases  $L_4(2^3)$  and  $L_9(3^4)$ , the factor levels of the first combination are all equal to one, and hence the first binary string  $\mathbf{o}_1$  is equal to the first parent string  $\mathbf{x}_1$ . In these special cases, three and eight new binary strings, respectively, are produced in every execution of orthogonal crossover.

## V. NUMERICAL RESULTS

### A. Test Problems

A test problem generator was proposed in [27] to generate test problems for multicast routing without any delay constraint. We input the number of nodes, the number of links, and the number of source/destination nodes. This generator will then give a test problem, which specifies a network topology, the link costs, a set of source and destination nodes, and an optimal multicast tree  $\Gamma$  with cost  $C^*$ . We now enhance this generator, so that it can generate test problems for multimedia multicast routing with a delay constraint. The details are as follows.

- Step 1) Execute the problem generator proposed in [27] to generate a test problem. Record the network, the set of source and destination nodes, the optimal multicast tree  $\Gamma$ , and its cost  $C^*$ .
- Step 2) Randomly assign a delay value to every link in this network.
- Step 3) Compute the delay  $\Delta_v$  from the source node to destination node  $v$  in the multicast tree  $\Gamma$  for all  $v \in S$ . The delay requirement  $\Delta$  is chosen to be  $\max_{v \in S} \{\Delta_v\}$ .

TABLE V

TRADITIONAL GENETIC ALGORITHM; FIRST SET OF TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c) COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

1st Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	0%	0%	0%	0%	0%
Group 2	0%	0%	0%	0%	0%
Group 3	1%	0%	0%	0%	0%
Group 4	0%	0%	0%	0%	0%
Group 5	0%	0%	0%	0%	0%
Group 6	0%	0%	0%	0%	0%
Group 7	0%	0%	0%	0%	0%
Group 8	0%	0%	0%	0%	0%

(a)

1st Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	(NA)	(NA)	(NA)	(NA)	(NA)
Group 2	(NA)	(NA)	(NA)	(NA)	(NA)
Group 3	6.00	(NA)	(NA)	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	(NA)	(NA)	(NA)	(NA)	(NA)
Group 6	(NA)	(NA)	(NA)	(NA)	(NA)
Group 7	(NA)	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(b)

1st Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	(NA)	(NA)	(NA)	(NA)	(NA)
Group 2	(NA)	(NA)	(NA)	(NA)	(NA)
Group 3	0.25	(NA)	(NA)	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	(NA)	(NA)	(NA)	(NA)	(NA)
Group 6	(NA)	(NA)	(NA)	(NA)	(NA)
Group 7	(NA)	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(c)

In Step 1), we generate a test problem without any delay constraint. In Step 2), we randomly assign a delay value to every link in the network. In Step 3), we choose the delay requirement  $\Delta$  such that the delay constraint can always be fulfilled. In this manner, the optimal multicast tree is still  $\Gamma$  with cost  $C^*$ . By executing the above three steps once, we get a test problem for multimedia multicast routing.

A real-world multimedia wide area network usually has several tens of nodes with a moderate connectivity. For example, the Arpanet has 20 nodes [28] and the Arpa2 has 21 nodes [29]. We generate two sets of test problems with two respective network sizes: 1) 50 nodes with 100 edges and 2) 80 nodes with 200 edges. In these networks, the number of possible multicast trees is already very large. Every set is further divided into eight groups of test problems, where every group consists of 100 different problem instances and different groups adopt different number of destination nodes. Therefore, there are a total of 1600 different test problems. Fig. 4 shows the test problems.

### B. Experiments and Performance Measures

We test two versions of the orthogonal genetic algorithm. The first version adopts two-to-two orthogonal crossover based

TABLE VI  
ORTHOGONAL GENETIC ALGORITHM USING  $L_4(2^3)$ ; FIRST SET OF TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c) COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

1st Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	33%	16%	10%	6%	6%
Group 2	56%	35%	30%	20%	20%
Group 3	10%	4%	0%	0%	0%
Group 4	8%	1%	1%	1%	1%
Group 5	8%	1%	1%	1%	1%
Group 6	3%	1%	0%	0%	0%
Group 7	3%	0%	0%	0%	0%
Group 8	0%	0%	0%	0%	0%

(a)

1st Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	92.60	72.87	78.00	86.83	86.83
Group 2	97.50	114.91	122.06	124.15	124.15
Group 3	99.50	107.50	(NA)	(NA)	(NA)
Group 4	113.75	46.00	46.00	46.00	46.00
Group 5	125.00	139.00	139.00	139.00	139.00
Group 6	142.33	34.00	(NA)	(NA)	(NA)
Group 7	139.66	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(b)

1st Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	11.14	8.58	9.08	10.13	10.13
Group 2	15.21	18.05	19.22	20.62	20.63
Group 3	12.11	13.49	(NA)	(NA)	(NA)
Group 4	16.08	7.81	7.81	7.81	7.81
Group 5	20.55	26.67	26.67	26.67	26.67
Group 6	20.12	4.82	(NA)	(NA)	(NA)
Group 7	20.49	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(c)

on  $L_4(2^3)$ , and the second version adopts three-to-two orthogonal crossover based on  $L_9(3^4)$ . These two versions are referred to as OGA<sub>4</sub> and OGA<sub>9</sub>, respectively. For comparison, we also consider the following traditional genetic algorithm (TGA).

Step 1) **Initialization**

Randomly create an initial generation of  $N$  binary strings and initialize the generation number  $gen$  to 0.

Step 2) **Population Evolution**

WHILE (stopping condition is not met) DO  
BEGIN

Step 2.1) **Crossover**

DO  $\frac{N}{2}$  times

Randomly select four parent strings from the current generation. Perform single-point crossover on the two best strings to generate two offspring for the next generation.

END

Step 2.2) **Mutation**

Each string in the new generation undergoes mutation with a small probability  $p_m$ . To

TABLE VII  
ORTHOGONAL GENETIC ALGORITHM USING  $L_9(3^4)$ ; FIRST SET OF TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c) COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

1st Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	100%	100%	100%	95%	95%
Group 2	100%	100%	99%	99%	99%
Group 3	100%	99%	97%	82%	82%
Group 4	100%	95%	92%	89%	89%
Group 5	100%	98%	97%	95%	81%
Group 6	100%	100%	97%	91%	91%
Group 7	100%	100%	97%	90%	83%
Group 8	100%	100%	96%	78%	65%

(a)

1st Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	14.56	21.15	22.28	24.76	24.76
Group 2	11.06	14.43	17.73	21.24	21.24
Group 3	15.81	20.86	27.60	39.01	39.01
Group 4	19.08	25.14	33.01	31.35	31.35
Group 5	17.43	23.66	25.76	26.89	33.85
Group 6	15.67	22.20	32.07	34.93	36.35
Group 7	15.15	21.62	30.78	33.64	35.45
Group 8	15.12	23.99	33.58	39.24	41.32

(b)

1st Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	3.18	4.56	4.82	5.36	5.39
Group 2	2.46	3.18	3.88	4.62	4.62
Group 3	3.52	4.60	6.03	8.41	8.41
Group 4	4.33	5.65	7.35	7.02	7.02
Group 5	3.96	5.33	5.80	6.06	7.56
Group 6	3.80	5.33	7.60	8.27	8.61
Group 7	3.67	5.17	7.27	7.94	8.36
Group 8	3.70	5.78	7.99	9.32	9.81

(c)

perform mutation on a binary string, flip every bit in this string with a small probability  $p_f$ .

Step 2.3) Execute the check-and-repair operation for the strings in the new generation.

Step 2.4) Increment the generation number  $gen$  by 1.

END

We adopt the following parameters or schemes for the experiments.

- *Population Size*: We adopt the same population size for the TGA, OGA<sub>4</sub>, and OGA<sub>9</sub>, so that they have the same space complexity. In particular, the population sizes are experimentally chosen to be 30 and 80 for the first set and the second set of test problems, respectively.
- *Stopping Condition*: We note that routing must be done within a specified interval, so that the users can start a multimedia session without waiting indefinitely. Therefore, it is desirable to find a feasible and good multicast tree within a specified number of generations. To study the effectiveness of an algorithm, we evaluate the solution quality after it has been executed for a given number of generations. Since the ratios of the computational complexity per generation for the TGA, OGA<sub>4</sub> and OGA<sub>9</sub> are 2:3:8, we choose the maximum number of

TABLE VIII  
TRADITIONAL GENETIC ALGORITHM; SECOND SET OF TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c) COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

2nd Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	3%	0%	0%	0%	0%
Group 2	6%	4%	0%	0%	0%
Group 3	0%	0%	0%	0%	0%
Group 4	0%	0%	0%	0%	0%
Group 5	2%	2%	0%	0%	0%
Group 6	8%	3%	0%	0%	0%
Group 7	18%	2%	2%	0%	0%
Group 8	0%	0%	0%	0%	0%

(a)

2nd Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	9.30	(NA)	(NA)	(NA)	(NA)
Group 2	9.00	9.25	(NA)	(NA)	(NA)
Group 3	(NA)	(NA)	(NA)	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	8.00	8.00	(NA)	(NA)	(NA)
Group 6	10.00	9.00	(NA)	(NA)	(NA)
Group 7	1.027	10.00	10.00	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(b)

2nd Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	1.29	(NA)	(NA)	(NA)	(NA)
Group 2	1.28	1.30	(NA)	(NA)	(NA)
Group 3	(NA)	(NA)	(NA)	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	1.29	1.29	(NA)	(NA)	(NA)
Group 6	2.00	1.58	(NA)	(NA)	(NA)
Group 7	1.89	1.77	1.77	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(c)

generations for the TGA, OGA<sub>4</sub>, and OGA<sub>9</sub> to be 400, 300, and 100, respectively. Since this stopping condition is based on the computational complexity, it implicitly takes the number of evaluations of the cost into account.

- *Selection*: In the orthogonal crossover operation, among the binary strings produced, we select the best two to be the offspring.
- *Mutation Probability*: We choose  $p_m = p_f = 0.1$ , such that every bit undergoes mutation with probability 0.01.

To measure the solution quality, we record the percentage of runs in which the costs of the feasible solutions are within  $x\%$  from the optimal one. To measure the computational complexity, we record two quantities for finding a feasible solution whose cost is within  $x\%$  from the optimal one: 1) the mean number of generations required and 2) the mean execution time on a Sun Sparc 4 workstation.

### C. Results

The results for the first set of test problems are shown in Tables V–VII. In particular, Tables V(a), VI(a), and VII(a) show the solution quality when the algorithms have been executed for the given numbers of generations. We see that the TGA can only find one feasible solution out of 800 test

TABLE IX  
ORTHOGONAL GENETIC ALGORITHM USING  $L_4(2^3)$ ; SECOND SET OF TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c) COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

2nd Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	18%	3%	2%	2%	2%
Group 2	15%	9%	9%	1%	1%
Group 3	1%	1%	1%	0%	0%
Group 4	0%	0%	0%	0%	0%
Group 5	1%	0%	0%	0%	0%
Group 6	0%	0%	0%	0%	0%
Group 7	0%	0%	0%	0%	0%
Group 8	0%	0%	0%	0%	0%

(a)

2nd Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	116.66	106.00	94.50	94.50	94.50
Group 2	108.00	117.11	117.11	134.00	134.00
Group 3	75.00	75.00	75.00	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	214.00	(NA)	(NA)	(NA)	(NA)
Group 6	(NA)	(NA)	(NA)	(NA)	(NA)
Group 7	(NA)	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(b)

2nd Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	69.73	59.16	53.83	53.84	53.87
Group 2	58.51	63.11	63.12	68.31	68.33
Group 3	63.20	63.20	63.20	(NA)	(NA)
Group 4	(NA)	(NA)	(NA)	(NA)	(NA)
Group 5	138.89	(NA)	(NA)	(NA)	(NA)
Group 6	(NA)	(NA)	(NA)	(NA)	(NA)
Group 7	(NA)	(NA)	(NA)	(NA)	(NA)
Group 8	(NA)	(NA)	(NA)	(NA)	(NA)

(c)

problems, OGA<sub>4</sub> can find feasible solutions for many test problems, and OGA<sub>9</sub> can find feasible solutions for all the test problems. In addition, OGA<sub>9</sub> can give the closest-to-optimal solutions. For example, 95.63% of the solutions are within 5% of the optimum, and 89.75% of the solutions are within 2% of the optimum. This indicates the effectiveness of experimental design methods: if we use a larger orthogonal array for a more representative sampling of genes for crossover, we can get better solutions. Nevertheless, Table VII(b)–(c) shows that OGA<sub>9</sub> only has a moderate computational complexity. For group 1 test problems, the mean number of generations required is 24.76 (equivalently, the mean execution time on a Sun Sparc 4 workstation is 4.16 s).

The results for the second set of test problems are shown in Tables VIII–X. The network size is now larger. We observe that OGA<sub>9</sub> is still better than OGA<sub>4</sub>, which is in turn better than TGA. In addition, the OGA<sub>9</sub> can find close-to-optimal solutions within moderate numbers of generations.

From the above results for these test problems, we can conclude two points.

- OGA<sub>9</sub> is better than OGA<sub>4</sub>, which is better than TGA. This indicates that experimental design methods can effectively improve the genetic algorithms.

TABLE X  
 ORTHOGONAL GENETIC ALGORITHM USING  $L_9(3^4)$ ; SECOND SET OF  
 TEST PROBLEMS. (a) SOLUTION QUALITY; (b) COMPUTATIONAL  
 COMPLEXITY: MEAN NUMBER OF GENERATIONS REQUIRED; (c)  
 COMPUTATIONAL COMPLEXITY: MEAN EXECUTION TIME REQUIRED

2nd Set of Test Problems	Percentage of runs in which the costs are within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	100%	99%	99%	99%	99%
Group 2	100%	100%	100%	92%	92%
Group 3	100%	99%	93%	92%	92%
Group 4	100%	100%	100%	95%	95%
Group 5	100%	100%	99%	88%	88%
Group 6	100%	100%	100%	99%	46%
Group 7	100%	100%	100%	93%	67%
Group 8	100%	100%	86%	77%	65%

(a)

2nd Set of Test Problems	Mean number of generations to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	13.97	20.03	20.81	20.81	20.81
Group 2	15.56	18.87	19.95	28.08	28.08
Group 3	19.27	22.40	25.74	27.80	27.80
Group 4	19.60	22.83	23.74	27.32	27.32
Group 5	21.08	24.49	26.46	28.85	28.85
Group 6	14.97	19.34	22.80	24.43	40.65
Group 7	16.16	20.93	25.27	32.02	38.56
Group 8	21.80	26.76	31.73	34.06	35.36

(b)

2nd Set of Test Problems	Mean execution time required (seconds) to find a solution whose cost is within $x\%$ from optimal				
	$x = 20$	$x = 10$	$x = 5$	$x = 2$	$x = 0$
Group 1	13.26	18.53	19.27	19.27	19.27
Group 2	14.50	17.36	18.31	25.24	25.24
Group 3	18.24	21.01	23.95	25.76	25.76
Group 4	18.81	21.70	22.52	25.71	25.71
Group 5	20.33	23.43	25.22	27.40	27.40
Group 6	14.69	18.68	21.82	23.30	37.86
Group 7	16.39	20.88	24.94	31.16	37.31
Group 8	22.05	26.77	31.43	33.70	34.96

(c)

- OGA<sub>9</sub> can find close-to-optimal solutions within moderate numbers of generations. This indicates that the orthogonal array  $L_9(3^4)$  is a good choice for multimedia multicast routing with practical problem sizes.

VI. SUMMARY

The multimedia multicast routing problem arises in many multimedia communication applications, and this problem has been proved to be NP-complete. In this paper, we proposed an orthogonal genetic algorithm for this routing problem. Its salient feature is to incorporate an experimental design method called orthogonal array into the crossover operation, so that the resulting operation can sample the genes from  $n$  parents in a statistically sound manner to produce  $j$  offspring. As a result, the orthogonal genetic algorithm can search the solution space effectively and it is well suited for parallel implementation and execution. In general, a larger orthogonal array gives a better solution quality but has a higher computational complexity. We executed the orthogonal genetic algorithm to solve two sets of benchmark test problems. The experimental results demonstrate that, for practical problem sizes, the orthogonal genetic algorithm using the orthogonal array  $L_9(3^4)$  can find close-to-optimal solutions within a moderate number of

generations. One possible extension of this research is to incorporate experimental design methods into steady state genetic algorithms.

ACKNOWLEDGMENT

The authors sincerely thank the anonymous reviewers for their constructive comments, D. B. Fogel for his suggestions on improving the presentation, and Prof. P. M. Pardalos for sending them the source code of his problem generator proposed in [27].

REFERENCES

- [1] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 286–292, June 1993.
- [2] Y. W. Leung and T. S. Yum, "Connection optimization for two types of videoconferences," *IEE Proc. Commun.*, vol. 143, no. 3, pp. 133–140, June 1996.
- [3] T. S. Yum, M. S. Chen, and Y. W. Leung, "Video bandwidth allocation for multimedia teleconferences," *IEEE Trans. Commun.*, vol. 43, no. 2, pp. 457–465, Feb. 1995.
- [4] *Special Issue on Educational System Using Multimedia and Communication Technology*, *IEICE Trans. Info. Syst.*, vol. E80-D, no. 2, Feb. 1997.
- [5] D. P. Anderson, "Metascheduling for continuous media," *ACM Trans. Computer Syst.*, vol. 11, no. 3, pp. 226–252, Aug. 1993.
- [6] Q. Zhu, M. Parsa, and J. J. G. L. Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *Proc. IEEE INFOCOM*, 1995, pp. 377–385.
- [7] F. Fluckiger, *Understanding Networked Multimedia: Applications and Technology*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [8] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, pp. 293–310, 1980.
- [9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
- [10] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. New York: Springer-Verlag, 1994.
- [11] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. London: Oxford Univ. Press, 1996.
- [12] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. of Michigan, 1975.
- [13] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.
- [14] B. A. Julstrom, "A genetic algorithm for the rectilinear Steiner problem," in *Proc. ICGA*, S. Forrest, Ed. San Mateo, CA: Morgan Kaufmann, 1993, pp. 474–480.
- [15] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*. Amsterdam, the Netherlands: Elsevier Science, 1992.
- [16] H. Esbensen, "Computing near-optimal solutions to the Steiner problem in a graph," *Networks*, vol. 26, pp. 173–185, 1995.
- [17] A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith, "Solving the graphical Steiner tree problem using genetic algorithms," *J. Oper. Res. Soc.*, vol. 44, no. 4, pp. 397–406, 1993.
- [18] Y. W. Leung and Q. Zhang, "Evolutionary algorithms + experimental design methods: A new optimization approach," Dept. of Computer Science, Hong Kong Baptist University, Research grant proposal, 1997.
- [19] D. C. Montgomery, *Design and Analysis of Experiments*, 3rd ed. New York: Wiley, 1991.
- [20] K. T. Fang and Y. Wang, *Number-Theoretic Methods in Statistics*. New York: Chapman & Hall, 1994.
- [21] Q. Wu, "On the optimality of orthogonal experimental design," *Acta Mathematicae Applicatae Sinica*, vol. 1, no. 4, pp. 283–299, 1978.
- [22] C. H. Papadimitriou, and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
- [23] Z. Ruttkay, A. E. Eiben, and P. E. Rane, "Genetic algorithms with multiparent recombination," in *Proc. 3rd Conf. Parallel Problem Solving from Nature*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Berlin, Germany: Springer, 1994, pp. 78–87.
- [24] J. N. Kok, A. E. Eiben, and C. H. M. van Kemenade, "Orgy in the computer: Multiparent recombination in genetic algorithms," in *Proc. 3rd Int. Conf. Artificial Life*, 1995, pp. 934–945.
- [25] S. W. Mahfoud, and D. E. Goldberg, "A genetic algorithm for parallel simulated annealing," in *Proc. Conf. Parallel Problem Solving from Na-*

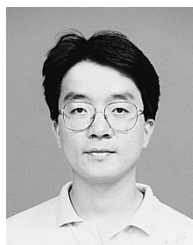
ture, R. Männer and B. Manderick, Eds. Amsterdam: North Holland, 1992, pp. 301–310.

- [26] D. Thierens, and D. E. Goldberg, "Elitist recombination: An integrated selected recombination genetic algorithm," in *Proc. 1st IEEE Int. Conf. Evol. Comput.*, Florida, 1994, pp. 508–512.
- [27] B. N. Khoury, P. M. Pardalos, and D. Z. Du, "A test problem generator for the Steiner problem in graphs," *ACM Trans. Math. Soft.*, vol. 19, no. 4, pp. 509–522, Dec. 1993.
- [28] R. Ramaswami and K. N. Sivarajan, "Routing and wavelength assignment in all-optical networks," *IEEE/ACM Trans. Network.*, vol. 3, no. 5, pp. 489–500, Oct. 1995.
- [29] K. C. Lee and V. O. K. Li, "A wavelength-convertible optical networks," *J. Lightwave Technol.*, vol. 11, no. 5, pp. 962–970, May 1993.



**Qingfu Zhang** was born in Tianshui, China, in 1965. He received the B.S. degree in mathematics from Shanxi University, China, in 1984 and the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from Xidian University, China, in 1991 and 1994, respectively.

In 1994, he joined the National Key Laboratory for Parallel Computing at Changsha Institute of Technology, China. Since July 1997, he has been a Research Fellow in German National Research Center for Information Technology (GMD), St Augustin Germany. His research interests include mathematical programming, neural networks, evolutionary computation, and signal processing.



**Yiu-Wing Leung** (S'91–M'92–SM'96) received the B.Sc. degree in electronics and computer science in 1989 and the Ph.D. degree in information engineering in 1992, both from the Chinese University of Hong Kong, Hong Kong.

He is currently an Associate Professor in the Department of Computer Science of the Hong Kong Baptist University, Hong Kong. His research interests include information networks, multimedia systems, computational intelligence, and algorithmic and heuristic techniques.