A Multiobjective Hybrid Genetic Algorithm for the Capacitated Multipoint Network Design Problem

Chi-Chun Lo, *Member, IEEE,* and Wei-Hsin Chang, *Member, IEEE*

***Abstract*—The capacitated multipoint network design problem (CMNDP) is NP-complete. In this paper, a hybrid genetic algorithm for CMNDP is proposed. The multiobjective hybrid genetic algorithm (MOHGA) differs from other genetic algorithms (GA's) mainly in its selection procedure. The concept of subpopulation is used in MOHGA. Four subpopulations are generated according to the elitism reservation strategy, the shifting Prüfer vector, the stochastic universal sampling, and the complete random method, respectively. Mixing these four subpopulations produces the next generation population. The MOHGA can effectively search the feasible solution space due to population diversity. The MOHGA has been applied to CMNDP. By examining computational and analytical results, we notice that the MOHGA can find most nondominated solutions and is much more effective and efficient than other multiobjective GA's.**

***Index Terms*—Genetic algorithms, minimal spanning tree, multiobjective function, nondominated solution, subpopulation.**

I. Introduction

THE problem of effectively transmitting data in a network involves the design of communication subnetworks. A critical issue in network design is to find a set of links which connect communication nodes such that the cost (or delay) of the selected paths between pairs of nodes is minimized, and the constraints of network capacity and reliability are met. In the real world, network design has long been recognized as multiobjective in nature. For a centralized multipoint network; i.e., a tree network, the network design problem gives rise to a well-known combinatorial optimization problem, namely, the constrained minimal spanning tree (CMST) problem. The CMST is NP-complete. Many heuristics, e.g., [3], [5], [10], [11], [14], [16], [30], have been proposed. However, their works took account of only cost or delay. In recent years, genetic algorithms (GA's) have been applied to various multiple criteria decision making (MCDM) problems. Fonseca and Fleming [12] explored the fitness assignment method. Horn *et al.* [22] investigated multiobjective problems via test functions. Tamaki *et al.*. [35] studied the multiobjective scheduling problem and the decision tree induction problem. Schaffer [33] proposed the vector evaluated genetic algorithm (VEGA) to solve multiobjective optimization problems. In VEGA, a population is divided into many disjoint subpopulations. For each subpopulation, a different objective function is used to evaluate the fitness of chromosomes (solutions). Ishibuchi and Murata [24] presented the single-objective genetic algorithm (SOGA) that translates multiple objective functions into a single-objective function by using weighting functions.

In this paper, the capacitated multipoint network design problem (CMNDP) is considered. A multiobjective hybrid genetic algorithm (MOHGA) is proposed for CMNDP. The MOHGA differs from other GA's mainly in its selection procedure. The concept of subpopulation is used in MOHGA. Four subpopulations are generated according to the elitism reservation strategy, the shifting Prüfer vector, the stochastic universal sampling, and the complete random method, respectively. Mixing these four subpopulations produces the next generation population. The MOHGA can effectively search the feasible solution space due to population diversity. By applying MOHGA to CMNDP, we notice that the MOHGA can find most nondominated solutions in the feasible solution space and is much more effective and efficient than other multiobjective GA's.

In the next section, a brief introduction to GA's is given. Section III describes the MOHGA. The problem formulation of CMNDP is detailed in Section IV. In Section V, computational experiments are presented. Section VI concludes this paper with possible future research directions.

II. Genetic Algorithms

A. Overview

The concept of a GA, introduced by Holland [21], is based on the mechanics of natural selection and natural genetics. A GA starts with an initial set of random solutions, called a **population**. Each individual in the population is called a **chromosome**, representing a solution to the problem. The initial population evolves through successive iterations, called **generations**. A measure of fitness defines the quality of an individual chromosome. In each generation, chromosomes are evaluated by a fitness function, also called an **evaluation** function. After a number of generations, highly fit individuals, which are analogous to good solutions to a given problem, will emerge. Because of this property, the GA is more robust than existing direct search methods, like the hill climbing method [27].

A GA consists of five components, as described in Davis's book [7]. These five components are as follows:

1) a method for encoding potential solutions into chromosomes;
2) a means of creating the initial population;

Manuscript received February 17, 1999; revised February 29, 2000. This paper was recommended by Associate Editor Y. Y. Haimes.

The authors are with Institute of Information Management, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: cclo@cc.nctu.edu.tw).

Publisher Item Identifier S 1083-4419(00)04124-8.

1083–4419/00$10.00 © 2000 IEEE

3) an evaluation function that can evaluate the fitness of chromosomes;
4) genetic operators that can create the next generation population;
5) a way to set up control parameters; e.g., population size, the probability of applying a genetic operator, etc.

B. Encoding Methods

To solve a problem using GA, the method of encoding its solutions is very important. For a CMST problem, its tree representation is encoded. There are three ways of encoding tree: 1) edge encoding [15], 2) vertex encoding [15], and 3) edge-and-vertex encoding [30].

1) Edge Encoding: Consider an undirected and connected graph $G = (V, E)$, where V is the set of vertices of G, and E is the set of edges of G. For each edge e of E, an index k is assigned; i.e., $E = \{e_k \,|\, k = 1, 2, \ldots, K$, where K is the number of edges of $G\}$. A tree T of G can be represented by edge encoding, $E' = \{e'_k\}$ where $e'_k = 1$ if e_k is an edge of T and 0 otherwise. Edge encoding is an intuitive representation of a tree. In a complete graph of N nodes, the total number of edges is equal to $N(N-1)/2$. There are $2^{(N(N-1)/2)}$ possible values for E'. All trees have exactly $N-1$ edges. If E' contains other than $N-1$ 1s, it is not a tree. However, even if E' contains exactly $N-1$ 1s, it is still unlikely that E' represents a tree. Therefore, edge encoding is not suitable for CMST due to the extremely low probability of using it to obtain a tree.

2) Vertex Encoding: In 1889, Cayley [4] proved the following formula: the number of spanning trees in a complete graph of N nodes is equal to $N^{(N-2)}$. Prüfer [32] presented the simplest proof of Cayley's formula by establishing a one-to-one correspondence between the set of spanning trees and a set of sequences of n integers, with each integer between 1 and n inclusive [15]. The sequence of n integers for encoding a tree is known as Prüfer vector (or Prüfer number). The Prüfer encoding and decoding procedures are explained as follows:

Prüfer encoding procedure

For a tree T, its corresponding Prüfer vector P can be obtained by the following steps:

Step 1) Let vertex i be the lowest labeled leaf node (node of degree 1) of T; let vertex j be incident to vertex i; append j to the end of P (P is constructed from left to right in sequence).
Step 2) Remove vertex i and the edge e_{ij}, which connects vertices i and j.
Step 3) Go back to Step 1 until there is only one edge left in T; P is obtained.

For example, Fig. 1 depicts a 7-node tree. Vertex 2 is the lowest labeled leaf node and vertex 1 is incident to vertex 2. Vertex 1 becomes the first element of P; then, vertex 2 and edge e_{12} are removed. In the second iteration, vertex 3 is the lowest leaf node and vertex 1 is incident to vertex 3. Append vertex 1 to P, and then remove vertex 3 and edge e_{13}. Repeat the process until only edge e_{47} is left. P is obtained and is equal to $[1, 1, 4, 4, 4]$.

Prüfer decoding procedure

For a Prüfer vector P and the set of its eligible vertices P', a unique tree representation of P, denoted as T, can be obtained by the following steps.

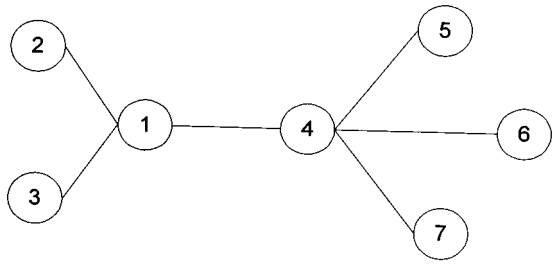

Fig. 1. Seven-node tree and its Prüfer encoding $= [1, 1, 4, 4, 4]$.

Step 1) Let vertex i be the lowest eligible node of P' and vertex j be the first element of P. If $i \neq j$, add the edge from i to j into T, and then remove vertex i from P' and vertex j from P.
Step 2) Repeat Step 1 until no elements are left in P.
Step 3) For the remaining last two vertices u and v of P', add the edge from u to v into T.

3) Edge-and-Vertex Encoding: Palmer and Kershenbaum [30] proposed the link-and-node biased encoding method, also termed as the edge-and-vertex encoding as presented by Gen and Cheng [15]. This encoding does not directly encode a tree, but a modified cost matrix. Based on the modified cost matrix, a tree is generated by Prim's algorithm [31]. For a graph of N nodes, the chromosome of this representation has biases, including node bias b_i and link bias b_{ij}, for the N nodes and each of the $[N(N-1)]/2$ links, for a total of $[N(N+1)]/2$ biases. In this method, two parameters, P_1 and P_2, are used as the multipliers of the maximum link cost, $C_{\max}$. The cost matrix (C_{ij}) is biased by b_i, b_{ij}, P_1, P_2, and $C_{\max}$ using

$$C'_{ij} = C_{ij} + P_1 b_{ij} C_{\max} + P_2 (b_i + b_j) C_{\max}.$$

Palmer claimed that this version of representation could encode any tree, given appropriate values of the b_i, b_j, and b_{ij} [30]. However, as pointed out by Gen and Cheng [15], this encoding has three major disadvantages:

1) it requires a very long encoding (memory cost);
2) it needs a conventional minimum spanning tree algorithm to generate a tree from its encoding (computation cost);
3) it contains no useful information such as degree, connection, etc, about a tree.

III. Multiobjective Hybrid Genetic Algorithm

For a multiobjective optimization problem, its nondominated solutions (The definition is stated in Appendix A.) can be found by GA's. To find good solutions via GA's, the concept of subpopulation proposed by Schaffer [33] is a promising approach.

A. Subpopulation

1) Elitism Reservation Strategy: In traditional GA's, a chromosome in the current generation is selected into the next generation with certain probability. The best chromosomes of the current generation may be lost due to mutation, crossover, or selection during the evolving process, and subsequently causes

difficulty in reaching convergence. In other word, it takes more generations; i.e., running time, to get quality solutions. Tamaki *et al.*. [35] proposed an elitism reservation strategy that permits chromosomes with the best fitness to survive and be carried into the next generation.

2) Shifting Prüfer Vector: The shifting Prüfer vector, introduced in this paper, is a genetic operator. The well-known problem of Prüfer encoding [30] is that it does not preserve locality. Changing one element of a Prüfer vector can change its corresponding tree topology dramatically. To remedy this problem, we introduce a new genetic operator, called the shifting Prüfer vector. This operator maintains maximum locality; i.e., it keeps the similarity between chromosomes. The concept of the shifting Prüfer vector is stated as follows: it replaces the leftmost element of a Prüfer vector by a randomly selected nonleftmost element of the same vector. The new vector differs from the old one only in the leftmost element. Thus, the new topology differs from the old one in at most two edges (The proof of this assertion is presented in Appendix B.) In most cases, the difference is only one edge. The shifting Prüfer vector is a local search method. According to the results obtained by the well-known Add and Drop searching heuristics [26], [28], [31], changing only one element in every iteration of the search process always leads to a globally optimal solution. Thus, the shifting Prüfer vector can significantly improve the quality of newly found chromosomes.

Fig. 2 illustrates the new tree after the shifting Prüfer vector is applied to the 7-node tree shown in Fig. 1. We notice that the new tree and the old one differ in only one edge.

3) Stochastic Universal Sampling: A simple way to perform sampling is to spin a roulette wheel. Unfortunately, this sampling method does not guarantee that any particular sample will actually be chosen in any given generation. This is a well-known problem of the roulette wheel selection method. Baker suggested the stochastic universal sampling method [2]. Baker's algorithm completes the whole sampling in a single pass, and requires only one random number. A wheel spin, whose size is equal to the population size, is divided into a number of equally spaced markers. A single spin is used to generate the random number. The expected value $\mathbf{e_k}$ for chromosome k is expressed as $e_k = \text{pop\_size} * p_k$, where pop_size represents population size and p_k represents selection probability.

4) Complete Random Method: Population is generated according to random number and random position. The major reason for using the complete random method is to maintain the diversity of the population.

B. Mix Method

There are two competing factors in the selection procedure of a genetic search. They are selection pressure and population diversity. An increase of selection pressure decreases the diversity of the population, and vice versa [27]. The stochastic universal sampling method increases selection pressure; however, it may cause the premature convergence of a genetic search. To decrease selection pressure, the complete random method

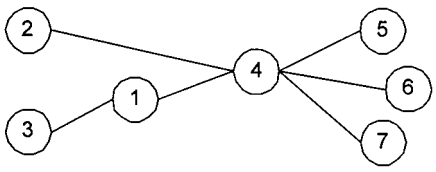

Fig. 2. New tree after applying the shifting Prüfer vector and its Prüfer encoding $= [4, 1, 4, 4, 4]$.

can be used in conjunction with the stochastic universal sampling method. Nevertheless, the best chromosomes of the current generation may be lost due to crossover and mutation. This problem can be overcome by including the elitism reservation strategy during a genetic search. But, globally optimal solutions are rarely obtained. In order to find globally optimal solutions, the shifting Prüfer vector is added to the selection procedure. According to the discussions aforementioned, we propose the mix method as follows: first, four subpopulations are generated according to the elitism reservation strategy [35], the shifting Prüfer vector, the stochastic universal sampling [2], and the complete random method, respectively; then, mixing these four subpopulations produces the next generation population.

C. The Multiobjective Hybrid Genetic Algorithm

MOHGA

Step 1) Set the maximum number of generation, $\mathbf{t_{max}}$, and initialize the loop counter, $\mathbf{t}$, to zero.
Step 2) Produce the initial population, $P(\mathbf{t})$, by using the complete random method and Prüfer encoding.
Step 3) Evaluate $P(t)$; exit, if the solutions are found.
Step 4) Generate the subpopulation, $\mathrm{SP}_1(\mathbf{t})$, by using the elitism reservation strategy.
Step 5) Generate the subpopulation, $\mathrm{SP}_2(\mathbf{t})$, by using the shifting Prüfer vector.
Step 6) Generate the subpopulation, $\mathrm{SP}_3(\mathbf{t})$, by using the stochastic universal sampling with crossover probability p_c and mutation probability p_m.
Step 7) Generate the subpopulation, $\mathrm{SP}_4(\mathbf{t})$, by using the complete random method.
Step 8) Form the next generation population, $P(\mathbf{t})$, by mixing $\mathrm{SP}_1(\mathbf{t}), \mathrm{SP}_2(\mathbf{t}), \mathrm{SP}_3(\mathbf{t})$, and $\mathrm{SP}_4(\mathbf{t})$.
Step 9) Increase $\mathbf{t}$ by 1; if $\mathbf{t}$ is less than $\mathbf{t}_{\max}$, then go to Step 3; otherwise, evaluate $\mathbf{P}(\mathbf{t}_{\max})$, exit.

where p_c represents the probability of crossover, p_m the probability of mutation, $\mathbf{P}(\mathbf{t})$ the population of the t-th generation, and $\mathbf{SP}_i(\mathbf{t})$ the i-th subpopulation of the t-th generation.

IV. Capacitated Multipoint Network Design Problem

A. Problem Formulation

The CMNDP can be formulated as follows:

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\min \sum_{\forall(i,j)\in E} d_{ij}x_{ij} \tag{2}$$

subject to

$$\sum_{(i,j)\in T_k} w_i x_{ij} < W \quad \forall k \tag{3}$$

$$\sum_{(i,j)\in S} x_{ij} < |S| \quad \forall S \tag{4}$$

$$\sum_{i,j} x_{ij} = N - 1 \tag{5}$$

$$_{ij} \in 0,1 \quad \forall (i,j) \in E \tag{6}$$

where

- N number of nodes in the network;
- E set of links in the network;
- T_k k-th link; and it may not exist for some k;
- W given weight matrix;
- w_i weight of the i-th node;
- S spanning tree;
- $|S|$ number of edges of S;
- $c_{i,j}$ cost of connecting node i to node j; i.e., the cost of link (i,j); the cost matrix $(c_{i,j})$ is symmetric;
- d_{ij} average delay on link (i,j); the delay matrix $(d_{i,j})$ is symmetric;
- x_{ij} the 0/1 decision variable; 1, if link (i,j) is selected, and 0, otherwise.

Equation (3) guarantees that the total link weight does not exceed the upper limit, (4) guarantees that the set of chosen links does not form any cycle, and (5) guarantees that enough links will be chosen to connect the network.

B. Applying MOHGA to CMNDP

1) Encoding Method: Prüfer encoding provides a one-to-one mapping between the set of spanning trees and the set of sequences of n-integers [15], [30]. Because of this excellent property, we choose Prüfer encoding for encoding chromosomes.

2) Initial Population: Each individual chromosome in the initial population is a solution to the problem. We use the complete random method to generate the initial population.

3) Evaluation Function: We choose both link cost and transmission delay as the evaluation functions for CMNDP.

V. Computational Experiments

A. Test Problems and Results

The simulator is coded in the C++ language and is running on an Intel Pentium-166 MHz PC with 64 MB RAM. In order to evaluate the solutions of CMNDP obtained by MOHGA, we examine a set of problems, with 7, 14, 28, and 56 nodes, respectively. The number of nondominated solutions obtained by MOHGA is affected by control parameters; e.g., mutation probability, crossover probability, and the maximum number of generations. The best solution in terms of cost $(\mathbf{Sol_c})$, the best solution in terms of delay $(\mathbf{Sol_d})$, or other solutions in terms of network designer's preference can be deduced from all nondominated solutions. The maximum number of generations is dependent on problem size. The lager the problem size, the larger the feasible solution space. In other words, more generations are needed to find the solutions.

TABLE I
CPU Time and Number of Solutions of the 7-Node Problem w.r.t. MOHGA, SOGA, and VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained (A)	Ratio (A/ B)	Sol_c	Sol_d
MOHGA	2.278	5.4	0.9	13	77
SOGA	2.16	2	0.333	13	78
VEGA	2.365	1.5	0.25	15	78

where A : The number of non-dominated solutions obtained by the algorithm.
B : The number of all non-dominated solutions, which is 6 in this problem.

Problem 1: For the 7-node network, nodes are randomly distributed. Control parameters are given as follows: mutation probability is set to 0.9, crossover probability is set to 0.5, population size is set to 100, and the maximum number of generations is set to 200. Cost, delay, and constrained weight matrices are given in Appendix C.

All the nondominated solutions; i.e., all possible spanning trees, are enumerated. By doing this, we are able to compare the quality of the solutions obtained by MOHGA directly with the enumerated solutions. The pair (total cost, total delay) represents a solution. By enumerating the solutions, we found all six nondominated solutions of this problem, which are $(13,92),(14,91),(15,85),(16,84),(18,78),$ and $(19,77)$. With ten runs, nondominated solutions obtained by MOHGA can be summarized as follows:

Set 1: $\{(13,92),(14,91),(15,85),(16,84),(18,78),(19,77)\}$;
Set 2: $\{(13,92),(15,85),(16,84),(18,78),(19,77)\}$;
Set 3: $\{(14,91),(15,85),(16,84),(18,78),(19,77)\}$.

Sets 1, 2, and 3 are obtained 4, 4, and 2 times, respectively. Therefore, we can claim that the MOHGA finds 90% of all nondominated solutions.

For comparison, the VEGA and the SOGA are applied to the same test problem. In the weighting function of SOGA, weights $w1$ (cost) and $w2$ (delay) are both set to 0.5 [24]. Note that we only consider the case in which $w1$ and $w2$ are both equal to 0.5, since the largest number of nondominated solutions are obtained by setting $w1 = w2 = 0.5$ (Appendix D.) Other control parameters are set to the same value as those of MOHGA. Computational results are summarized in Table I.

Problem 2: A network of 14 nodes is considered in Problem 2. Network nodes are randomly distributed. Weight, cost, and delay are randomly generated. Control parameters are given as follows: mutation probability is set to 0.9, crossover probability is set to 0.4, population size is set to 100, and the maximum number of generations is set to 1000. Computational results are presented in Table II.

Problem 3: A network of 28 nodes is considered in Problem 3. Network nodes are randomly distributed. Weight, cost, and delay are randomly generated. Control parameters are given as follows: mutation probability is set to 0.99, crossover probability is set to 0.4, population size is set to 100, and the maximum number of generations is set to1000. Computational results are shown in Table III.

Problem 4: A network of 56 nodes is considered in Problem 4. Network nodes are randomly distributed. Weight, cost, and delay are randomly generated. Control parameters are given as follows: mutation probability is set to 0.99, crossover probability is set to 0.4, population size is set to 100, and the maximum

TABLE II
CPU TIME AND NUMBER OF SOLUTIONS OF THE 14-NODE PROBLEM w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained	Sol_c	Sol_d
MOHGA	37.90	9.5	63	160
SOGA	28.83	2.5	66	168
VEGA	45.04	2.5	65	162

TABLE III
CPU TIME AND NUMBER OF SOLUTIONS OF THE 28-NODE PROBLEM w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominate solutions obtained	Sol_c	Sol_d
MOHGA	56.792	13	585	628
SOGA	76.192	1.5	866	891
VEGA	83.4	1.5	854	912

TABLE IV
CPU TIME AND NUMBER OF SOLUTIONS OF THE 56-NODE PROBLEM w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained	Sol_c	Sol_d
MOHGA	2167.58	15	1583	1618
SOGA	3780.462	1.3	1678	1620
VEGA	4138.066	1.3	1675	1685

number of generations is set to 10 000. Computation results are listed in Table IV.

Problems 5, 6, and 7: A network of 56 nodes is considered in Problems 5, 6, and 7. Network nodes are not randomly distributed. For Problem 5, most nodes reside at the upper left corner of the network. For Problem 6, node distribution is sparse. For Problem 7, node distribution is dense. Weight, cost, and delay are randomly generated. Control parameters are given as follows: mutation probability is set to 0.99, crossover probability is set to 0.4, population size is set to 100, and the maximum number of generations is set to 10 000. Computation results are reported in Tables V–VII.

Problem 8: Problem 8 is designed to evaluate the correctness of the mix method as described in Section III. Four algorithms are evaluated. In Algorithm 1, only the stochastic universal sampling method is used in the selection procedure. Algorithm 2 is obtained by adding the complete random method to Algorithm 1. Algorithm 3 is obtained by adding the elitism reservation strategy to Algorithm 2. Algorithm 4 (the MOHGA) is obtained by adding the shifting Prüfer vector to Algorithm 3. Network nodes of 7, 14, and 28 are considered. Simulation results are reported in Table VIII. By examining Table VIII, we notice that Algorithm 4 not only finds more nondominated solutions but also obtains them faster than the other three algorithms.

Problem 9: Problem 9 is designed to evaluate the performance of MOHGA with respect to different combinations of crossover probability P_c and mutation probability P_m. A network of 56 nodes is considered in Problem 9. P_c's of values 0.4, 0.5, 0.6, and 0.7 are considered. Each P_c is paired with a P_m, which is of values 0.7, 0.8, 0.9, and 0.99. Average CPU time and average number of nondominated solutions are obtained. Simulation results are shown in Table IX. Results indicate that the MOHGA is not sensitive to the changes of P_c and P_m.

TABLE V
CPU TIME AND NUMBER OF SOLUTIONS OF PROBLEM 5 w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained	Sol_c	Sol_d
MOHGA	2181.68	13	1584	1616
SOGA	3793.26	1.3	1678	1621
VEGA	4147.34	1.3	1677	1662

TABLE VI
CPU TIME AND NUMBER OF SOLUTIONS OF PROBLEM 6 w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained	Sol_c	Sol_d
MOHGA	2270.52	12	1583	1618
SOGA	3650.50	1	1660	1700
VEGA	3869.85	1	1711	1682

TABLE VII
CPU TIME AND NUMBER OF SOLUTIONS OF PROBLEM 7 w.r.t. MOHGA, SOGA, AND VEGA

Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained	Sol_c	Sol_d
MOHGA	2257.46	14	1588	1615
SOGA	3639.25	2	1640	1653
VEGA	4130.50	1.5	1634	1656

TABLE VIII
CPU TIME AND NUMBER OF SOLUTIONS OF PROBLEM 8

Number of Nodes	Algorithm	CPU time (Sec)	Average number of non-dominated solutions obtained
7	1	3.63	1.45
	2	3.18	1.5
	3	3.02	3.2
	4	2.278	5.4
14	1	48.11	1.5
	2	43.11	2
	3	42.15	5.2
	4	37.90	9.5
28	1	96.56	1.5
	2	86.86	2
	3	70.72	6.1
	4	56.792	13

B. Analyses

1) Space Usage Analysis: For a network of N nodes, Prüfer encoding requires $N-2$ elements to encode a chromosome. Assume an element is represented by a two-byte integer and population size is denoted by pop_size. Then, the space usage of Prüfer encoding is equal to $(N-2)*2*$ pop_size (of order $O(N)$). Since the MOHGA, the SOGA, and the VEGA all use Prüfer encoding to encode their chromosomes; therefore, their space usage is the same and of order $O(N)$.

2) Running Time (Complexity) Analysis:

- **Operational Analysis**
  The running time of SOGA, VEGA, and MOHGA is calculated in terms of the number of operations executed. For the three GA's, we have identified eight operations, which are weighted sum, subgroup selection, shuffle,

TABLE IX
CPU TIME AND NUMBER OF SOLUTIONS OF PROBLEM 9

P_c	P_m	CPU time(Sec)	Average number of non-dominated solutions obtained
0.4	{0.7, 0.8, 0.9, 0.99}	2171.8	14
0.5	{0.7, 0.8, 0.9, 0.99}	2168.7	13
0.6	{0.7, 0.8, 0.9, 0.99}	2175.3	14
0.7	{0.7, 0.8, 0.9, 0.99}	2167.8	13

TABLE X
NUMBER OF OPERATIONS OF SOGA, VEGA, AND MOHGA

GAs / Operation	SOGA	VEGA	MOHGA
Weighted Sum (a)	M*pop_size	N/A	N/A
Subgroup Selection (b)	N/A	M*pop_size	N/A
Shuffle (c)	N/A	M*pop_size	N/A
Elitism Reservation (d)	N/A	N/A	M*pop_size/4
Shifting Prüfer Vector (e)	N/A	N/A	M*pop_size/4
Complete Random (f)	N/A	M/A	M*pop_size/4
Crossover & Mutation (g)	M*(N-3)*pop_size +M*pop_size	M*(N-3)*pop_size + M*pop_size	(M*(N-3)*pop_size +M*pop_size)/4
Prüfer Decoding (h)	M*NlogN*pop_size	M*NlogN*pop_size	M*NlogN*pop_size
Total (a)+(b)+(c)+(d)+ (e)+(f)+(g)+(h)	M*(N+NlogN-1)* pop_size	M*(N+NlogN)* pop_size	$M*(\frac{N+1}{4}+NlogN)*$ pop_size

where: M represents the maximum number of generations,
pop_size represents the population size.,
N represents the number of nodes, and
N/A means "Not Applicable"

elitism reservation, shifting Prüfer vector, complete random, crossover and mutation, and Prüfer decoding. We assume that every operation takes the same amount of CPU time. Operational analysis is detailed in Table X. By examining Table X, we notice that the MOHGA requires the least number of operations.

- **Simulation Analysis**
  For small size problems; i.e., Problems 1 and 2, the MOHGA takes more time than the SOGA, but less time than the VEGA. For large size problems; i.e., Problems 3 to 7, the MOHGA takes the least time to generate nondominated solutions. The comparison of the running time of MOHGA, SOGA, and VEGA is shown in Fig. 3. Fig. 3 illustrates that the rate of increase of the running time of SOGA, and that of VEGA, is close to an exponential function. For MOHGA, the rate of increase of the running time is only 57.6% of that of SOGA, and that of VEGA. By examining computational results, we have the following observations: the SOGA takes the least time for small size problems, but obtains the worst results; the VEGA takes the most computing time; and the MOHGA is much more efficient than the SOGA and the VEGA.

3) Quality Analysis: For Problem 1, the MOHGA, which finds 90% of all nondominated solutions, performs much better than the SOGA and the VEGA. The same phenomena can be found for Problems 2 through 7, as illustrated in Tables II through VII. The MOHGA always finds more nondominated solutions than the SOGA and the VEGA. Also, by examining Tables I through VII, we notice that the MOHGA always finds the best solution in terms of cost ($\mathbf{Sol_c}$) and the best solution in terms of delay ($\mathbf{Sol_d}$). Therefore, the MOHGA is much more effective than the SOGA and the VEGA.

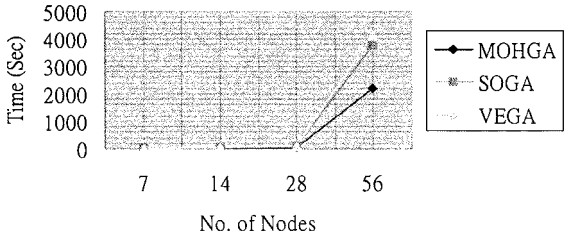

Fig. 3. Running time (complexity) analysis.

C. Discussions

Three observations are worthy of discussing.

1) Both the reasons described in Section III-B. and simulation results obtained from Problem 8 indicate that the mix method is a reasonable approach. Further, the running time analysis and the quality analysis explain why the MOHGA using the mix method is very efficient and effective.
2) The subpopulation generated by using the stochastic universal sampling with crossover probability P_c and mutation probability P_m is one of the four subpopulations considered in the mix method. The influence of P_c and P_m on the total population is reduced by three-fourths. Therefore, the MOHGA is not sensitive to the changes of P_c and P_m. This assertion can also be verified by the simulation results presented in Table IX. Since P_c and P_m can be assigned to any value in a given range, they can be expressed as fuzzy numbers of a fuzzy knowledge-based system. But to define a set of fuzzy rules of a fuzzy knowledge-based system is a difficult task.
3) Cost and delay vary from time to time due to user behavior, increase of bandwidth, business strategy, etc. When designing a network, a network designer usually derives cost and delay by estimation. In CMNDP, cost and delay can be expressed as fuzzy objective functions due to their uncertainty. The CMNDP is transformed into a fuzzy multiobjective decision making problem (FMODMP) by including fuzzy objective functions into its problem formulation. In [36], Buckly proposed a fuzzy genetic algorithm, in which chromosomes are interpreted as fuzzy numbers, to solve single objective fuzzy optimization problems. For solving the FMODMP, the MOHGA can be modified according to Buckly's approach.

VI. CONCLUSION

A. Summary

The MOHGA is based on the subpopulation concept. The elitism reservation strategy, the shifting Prüfer vector, the stochastic universal sampling, and the complete random method are used to produce the next generation population. The

MOHGA has been applied to CMNDP. By examining computational and analytical results, we notice that the MOHGA finds most nondominated solutions and is much more effective and efficient than the SOGA and the VEGA.

B. Future Directions

Here we would like to mention the following areas, which may merit further investigation.

1) Apply the MOHGA to other multiobjective optimization problems, such as the facility layout problem, the wireless channel assignment problem, the resource scheduling problem, etc.
2) Develop an algorithm to determine the maximum (optimal) number of nondominated solutions of a multiobjective optimization problem. In essence, this is a hard problem. For small size problems, enumeration is possible. For large size problems, artificial intelligence techniques; e.g., branch-and-bound, minimax, etc. can be considered.

APPENDIX A
NONDOMINATED SOLUTIONS

A. Multiobjective Optimization Problem

For a multiobjective optimization problem, the set of its feasible decisions, $\mathbf{X}$, is defined as follows:

$$\mathbf{X} = \{\mathbf{x} \,|\, g_j(\mathbf{x}) \le 0, g_j : R^n \to R, j = 1, 2, \ldots, m\},$$

where $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_n)$ is an n-dimension variable;
$g_j(\mathbf{x})$ represents a constraint;
m represents the number of constraints;
R represents the set of real numbers.

The multiobjective optimization problem (MOP) can be stated as follows:

$$\underset{x \in X}{\text{minimize}} \; f_1(x), f_2(x), \ldots, f_k(x) \tag{A.1}$$

where $f_i(x), i = 1, 2, \ldots, k$ is a vector-valued objective function defined on an n-dimension variable $\mathbf{x}$.

B. Nondominated Solutions

A solution $\mathbf{x}^*$ of (A.1) is said to be nondominated if there exist no other feasible solutions $\mathbf{x} \in \mathbf{X} \subseteq R^n$ such that $f_i(\mathbf{x}) \le f_i(\mathbf{x}^*)$, for all $i = 1, \ldots, k$. The $\mathbf{x}^*$ is also called Pareto optimal.

APPENDIX B
SHIFTING PRÜFER VECTOR

A. Shifting Prüfer Vector

The shifting Prüfer vector, introduced in this paper, is a genetic operator. This operator replaces the leftmost element of a Prüfer vector by a randomly selected nonleftmost element of the same vector.

Theorem: If P is a Prüfer vector and P_1 is obtained by the use of the shifting Prüfer vector on P, and Tp and Tp_1 are the corresponding tree representations of P and P_1, respectively, then Tp_1 differs from Tp in at most two edges.

Proof: Let P be denoted as $[v_1, v_2, v_3, \ldots, v_{n-2}]$, where v_i $(i = 1, 2, \ldots n-2)$ is a vertex and its value, denoted as $|v_i|$, is an integer. Then, P_1 can be expressed as $[v_k, v_2, v_3, \ldots, v_{n-2}]$, where $v_k \in \{v_2, v_3, \ldots, v_{n-2}\}$. To prove the theorem, we will consider the following five cases:

Case 1. $|v_k| = |v_1|$
Because $|v_k| = |v_1|$; therefore, $P_1 = P$. Since, for a given Prüfer vector, the Prüfer decoding procedure always produces a unique tree representation; hence, we can claim that Tp and Tp_1 have the same topology.

Case 2. $|v_k| > |v_1|$ and $|v_1| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$
Let v_{len} and v_{len} be the lowest eligible node of P' and P_1', respectively.

The following proof is derived via the Prüfer decoding procedure.

1) In the first iteration, according to the first step of the procedure, the following statements hold true.
   - Since $|v_1| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$ and $|v_k| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, we have $|v_{\text{len}}| = |v_{\text{len}'}|$. Therefore, after $|v_{\text{len}}|$ and $|v_{\text{len}'}|$ are removed from P' and P_1', respectively, P' and P_1' are reduced to the same set.
   - After v_1 and v_k are removed from P and P_1, respectively, P and P_1 are reduced to the same vector $[v_2, v_3, \ldots, v_{n-2}]$.
   - The first edge of $Tp, (v_{\text{len}}, v_1)$ and that of $Tp_1, (v_{\text{len}'}, v_k)$ is different.
2) In the remaining iterations, since P and P_1 are reduced to the same vector, and P' and P_1' are reduced to the same set, at the end of the first iteration, the decoding of P is the same as that of P_1.

From 1) and 2), we can claim that Tp and Tp_1 are different only in the first edge.

Case 3. $|v_k| < |v_1|$ and $|v_1| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$
The proof is the same as Case 2. We conclude that Tp and Tp_1 are different only in the first edge.

Case 4 $|v_k| > |v_1|$ and $|v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$
Let v_{len} and $v_{\text{len}'}$ be the lowest eligible node of P' and P_1', respectively. Two subcases are considered.

Case 4.1. $|v_{\text{len}}| = |v_{\text{len}'}|$
The proof is similar to Case 2. We conclude that Tp and Tp_1 are different only in the first edge.

Case 4.2. $|v_{\text{len}}| \neq |v_{\text{len}'}|$
The following proof is derived via the Prüfer decoding procedure.

1) In the first iteration, according to the first step of the procedure, the following statements hold true.
   - Since $|v_k| > |v_1|, |v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, and $|v_k| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, we have $|v_{\text{len}}| \ge |v_{\text{len}'}|$. Therefore, after $|v_{\text{len}}|$ and $|v_{\text{len}'}|$ are removed from P' and P_1', respectively, P' is changed to set $\{1, 2, \ldots, n\} - \{|v_{\text{len}}|\}$ and P_1' is changed to set $\{1, 2, \ldots, n\} - \{v_{\text{len}'}|\}$, respectively.
   - After v_1 and v_k are removed from P and P_1, respectively, P and P_1 are reduced to the same vector $[v_2, v_3, \ldots, v_{n-2}]$.
   - The first edge of $Tp, (v_{\text{len}}, v_1)$ and that of $Tp_1, (v_{\text{len}'}, v_k)$ is different.

2) In the second iteration, let v_{slen} and $v_{\text{slen}'}$ be the lowest eligible node of P' and P_1', respectively (Note that v_{slen} and $v_{\text{slen}'}$ are the second lowest eligible nodes of the original P' and P_1', respectively.) According to the first step of the procedure, the following statements hold true.
   - Since $|v_k| > |v_1|, |v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}, |v_k| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, and $|v_{\text{len}}| \geq |v_{\text{len}'}|$ (shown in the first iteration), we have $|v_{\text{slen}}| \leq |v_{\text{slen}'}|, |v_{\text{slen}}| = |v_{\text{len}'}|$, and $|v_{\text{slen}'}| = |v_{\text{len}}|$. Therefore, after $|v_{\text{slen}}|$ and $|v_{\text{slen}'}|$ are removed from P' and P_1', respectively, P' and P_1' are reduced to the same set.
   - After v_2 are removed from both P and P_1, P and P_1 are reduced to the same vector $[v_3, \ldots, v_{n-2}]$;
   - The second edge of $Tp, (v_{\text{slen}}, v_2)$, and that of $Tp_1, (v_{\text{slen}'}, v_2)$, is different.
3) In the remaining iterations, since P and P_1 are reduced to the same vector, and P' and P_1' are reduced to the same set, at the end of the second iteration, the decoding of P is the same as that of P_1.

From 1), 2), and 3), we can claim that Tp and Tp_1 are different only in the first edge and the second edge.

Case 5 $|v_k| > |v_1|$ and $|v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$

Let v_{len} and $v_{\text{len}'}$ be the lowest eligible node of P' and P_1', respectively. Two subcases are considered.

Case 5.1. $|v_{\text{len}}| = |v_{\text{len}'}|$

The proof is similar to Case 2. We conclude that Tp and Tp_1 are different only in the first edge.

Case 5.2. $|v_{\text{len}}| \neq |v_{\text{len}'}|$

The following proof is derived via the Prüfer decoding procedure.

1) In the first iteration, according to the first step of the procedure, the following statements hold true.
   - Since $|v_k| < |v_1|, |v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, and $|v_k| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, we have $|v_{\text{len}}| \geq |v_{\text{len}'}|$. Therefore, after $|v_{\text{len}}|$ and $|v_{\text{len}'}|$ are removed from P' and P_1', respectively, P' is changed to set $\{1, 2, \ldots, n\} - \{|v_{\text{len}}|\}$ and P_1' is changed to set $\{1, 2, \ldots, n\} - \{|v_{\text{len}'}|\}$, respectively.
   - After v_1 and v_k are removed from P and P_1, respectively, P and P_1 are reduced to the same vector $[v_2, v_3, \ldots, v_{n-2}]$.
   - The first edge of $Tp, (v_{\text{len}}, v_1)$ and that of $Tp_1, (v_{\text{len}'}, v_k)$ is different.
2) In the second iteration, let v_{slen} and $v_{\text{slen}'}$ be the lowest eligible node of P' and P_1', respectively (Note that v_{slen} and $v_{\text{slen}'}$ are the second lowest eligible nodes of the original P' and P_1', respectively.) According to the first step of the procedure, the following statements hold true.
   - Since $|v_k| < |v_1|, |v_1| \notin \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}, |v_k| \in \{|v_2|, |v_3|, \ldots, |v_{n-2}|\}$, and $|v_{\text{len}}| \geq |v_{\text{len}'}|$ (shown in the first iteration), we have $|v_{\text{slen}}| \leq |v_{\text{slen}'}|, |v_{\text{slen}}| = |v_{\text{len}'}|$, and $|v_{\text{slen}'}| = |v_{\text{len}}|$. Therefore, after $|v_{\text{slen}}|$ and $|v_{\text{slen}'}|$ are removed from P' and P_1', respectively, P' and P_1' are reduced to the same set.
   - After v_2 are removed from both P and P_1, P and P_1 are reduced to the same vector $[v_3, \ldots, v_{n-2}]$.
   - The second edge of $Tp, (v_{\text{slen}}, v_2)$ and that of $Tp_1, (v_{\text{slen}'}, v_2)$ is different.
3) In the remaining iterations, since P and P_1 are reduced to the same vector, and P' and P_1' are reduced to the same set, at the end of the second iteration, the decoding of P is the same as that of P_1.

From 1), 2), and 3), we can claim that Tp and Tp_1 are different only in the first edge and the second edge.

From the conclusions obtained from the above five cases, we have proved that Tp_1 differs from Tp in at most two edges. □

B. Examples

Example 1: This example is used to illustrate the proof of Case 4.2. For a 7-node network, assume that $P = [1, 2, 3, 5, 3]$ and $P_1 = [3, 2, 3, 5, 3]$. P' and P_1' can be expressed as sets $\{1, 2, 3, 4, 5, 6, 7\}$ and $\{1, 2, 3, 4, 5, 6, 7\}$, respectively. Let v_{len} and $v_{\text{len}'}$ be the lowest eligible node of P' and P_1', respectively.

1) In the first iteration of the Prüfer decoding procedure, since $|v_1| = 1$ and $|v_k| = 3$; therefore, $|v_{\text{len}}| = 4$ and $|v_{\text{len}'}| = 1$. After node 4 and node 1 are removed from P' and P_1', respectively, P' is changed to set $\{1, 2, 3, 5, 6, 7\}$ and P_1' is changed to set $\{2, 3, 4, 5, 6, 7\}$, respectively. After element 1 and element 3 are removed from P and P_1, respectively, P and P_1 are reduced to the same vector $[2, 3, 5, 3]$. The first edge of $Tp, (4, 1)$, and that of $Tp_1, (1, 3)$, is different.

   In the second iteration of the Prüfer decoding procedure, let v_{slen} and $v_{\text{slen}'}$ be the lowest eligible node of P' and P_1', respectively. Since P and P_1 are equal to the same vector $[2, 3, 5, 3], P' = \{1, 2, 3, 5, 6, 7\}$, and $P_1' = \{2, 3, 4, 5, 6, 7\}$; therefore, $|v_{\text{slen}}| = 1$ and $|v_{\text{slen}'}| = 4$. After node 1 and node 4 are removed from P' and P_1', respectively, P' and P_1' are both changed to set $\{2, 3, 5, 6, 7\}$. After element 2 are removed from both P and P_1, P and P_1 are both reduced to the same vector $[3, 5, 3]$. The second edge of $Tp, (1, 2)$, and that of $Tp_1, (4, 2)$, is different.
2) In the remaining iterations, since P and P_1 are reduced to the same vector, and P' and P_1' are reduced to the same set, at the end of the second iteration, the decoding of P is the same as that of P_1.

From 1), 2), and 3), we notice that Tp and Tp_1 are different only in the first edge and the second edge.

Example 2: This example is used to illustrate the proof of Case 5.2. For a 7-node network, assume that $P = [3, 1, 2, 4, 5]$ and $P_1 = [1, 1, 2, 4, 5]$. P' and P_1' can be expressed as sets $\{1, 2, 3, 4, 5, 6, 7\}$ and $\{1, 2, 3, 4, 5, 6, 7\}$, respectively. Let v_{len} and $v_{\text{len}'}$ be the lowest eligible node of P' and P_1', respectively.

1) In the first iteration of the Prüfer decoding procedure, since $|v_1| = 3$ $|v_k| = 1$; therefore, $|v_{\text{len}}| = 6$ and $|v_{\text{len}'}| = 3$. After node 6 and node 3 are removed from P' and P_1', respectively, P' is changed to set $\{1, 2, 3, 4, 5, 7\}$

TABLE XI
CPU TIME AND NUMBER OF SOLUTIONS w.r.t DIFFERENT $(w1, w2)$ PAIRS

(w1, w2)	CPU time (Sec)	Average number of non-dominated solutions obtained
(0.3, 0.7)	3788.52	1
(0.4, 0.6)	3787.63	1.2
(0.5, 0.5)	3793.26	1.3
(0.6, 0.4)	3792.35	1.15
(0.7, 0.3)	3797.11	1

and P_1' is changed to set $\{1,2,4,5,6,7\}$, respectively. After element 3 and element 1 are removed from P and P_1, respectively, P and P_1 are reduced to the same vector $[1,2,4,5]$. The first edge of $Tp,(6,3)$, and that of $Tp_1,(3,1)$, is different.

2) In the second iteration of the Prüfer decoding procedure, let v_{slen} and $v_{\text{slen}'}$ be the lowest eligible node of P' and P_1', respectively. Since P and P_1 are equal to the same vector $[1,2,4,5], P' = \{1,2,3,4,5,7\}$, and $P_1' = \{1,2,4,5,6,7\}$; therefore, $|v_{\text{slen}}| = 3$ and $|v_{\text{slen}'}| = 6$. After node 3 and node 6 are removed from P' and P_1', respectively, P' and P_1' are both changed to set $\{1,2,4,5,7\}$. After element 1 are removed from both P and P_1, P and P_1 are both reduced to the same vector $[2,4,5]$. The second edge of $Tp,(3,1)$, and that of $Tp_1,(6,1)$, is different.

3) In the remaining iterations, since P and P_1 are reduced to the same vector, and P' and P_1' are reduced to the same set, at the end of the second iteration, the decoding of P is the same as that of P_1.

From 1), 2), and 3), we notice that Tp and Tp_1 are different only in the first edge and the second edge.

APPENDIX C
MATRICES OF TEST PROBLEM 1

Cost matrix:

$$\begin{matrix}
\cdot & & & & & & \\
4 & \cdot & & & & & \\
2 & 3 & \cdot & & & & \\
\cdot & \cdot & 4 & \cdot & & & \\
\cdot & 1 & 5 & 2 & \cdot & & \\
4 & 2 & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & 3 & 4 & \cdot
\end{matrix}$$

Delay matrix:

$$\begin{matrix}
\cdot & & & & & & \\
12 & \cdot & & & & & \\
12 & 13 & \cdot & & & & \\
\cdot & \cdot & 14 & \cdot & & & \\
\cdot & 21 & 15 & 12 & \cdot & & \\
14 & 21 & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & 13 & 14 & \cdot
\end{matrix}$$

Note that "$\cdot$" represents infinity.

Constrained weight matrix:

$$\begin{matrix}
- & & & & & & \\
3 & - & & & & & \\
3 & 3 & - & & & & \\
- & - & 3 & - & & & \\
- & 3 & 3 & 3 & - & & \\
3 & 3 & - & - & - & - & \\
- & - & - & - & 3 & 3 & -
\end{matrix}$$

Note that "$-$" represents "0."

APPENDIX D
WEIGHTING FUNCTION OF SOGA

In order to evaluate the effect of the weighting function of SOGA, we have simulated the SOGA with respect to different combinations of cost weight $(w1)$ and delay weight $(w2)$. Five different pairs of $(w1, w2)$ are assumed. A network of 56 nodes is considered. Simulation results are presented in Table XI. By examining Table XI, we have the following observations: for all five test cases, their running time is very close; with $w1 = w2 = 0.5$, the SOGA finds the largest number of nondominated solutions. Table XI indicates that the SOGA has the best performance when there is no bias between cost and delay.

REFERENCES

[1] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld, "Determinant factorization: A new encoding scheme for spanning tree applied to the probability minimum spanning tree problem," in *Proc. 6th ICGA*, 1995, pp. 470–477.

[2] J. Baker, "Adaptive selection methods for genetic algorithms," in *Proc. 2nd ICGA*, 1987, pp. 100–111.

[3] R. R. Boorstyn and H. Frank, "Large scale network topological optimization," *IEEE Trans. Commun.*, vol. COM-25, pp. 29–47, Jan. 1977.

[4] A. Cayley, "A theorem on tree," *Quart. J. Math.*, vol. 23, pp. 376–378, 1889.

[5] L. W. Clarke and G. Anandalingam, "An integrated system for designing minimum cost survivable telecommunications networks," *IEEE Trans. Syst., Man, Cybern. A*, vol. 26, pp. 856–862, Nov. 1996.

[6] *Handbook of Genetic Algorithms*, L. Davis, Ed., Van Reinhold, New York, 1991, pp. 124–143.

[7] L. Davis, *Genetic Algorithms and Simulated Annealing*. Los Altos, CA: Morgan Kaufmann, 1987. Research Notes in Artificial Intelligence.

[8] ——, "A genetic algorithm for survivable network design," in *Proc. 5th ICGA*, 1993, pp. 408–415.

[9] K. A. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, 1975.

[10] R. Elbaum and M. Sidi, "Topological design of local-area networks using genetic algorithms," *IEEE/ACM Trans. Networking*, vol. 4, no. 5, pp. 766–778, Oct. 1996.

[11] C. Ersoy and S. S. Panwar, "Topological design of interconnected LAN/WAN networks," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 1172–1182, Oct. 1993.

[12] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proc. 5th ICGA*, 1993, pp. 416–423.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[14] B. Gavish, "Topological design of telecommunication networks-local access design methods," *Ann. Oper. Res.*, vol. 33, pp. 17–71, 1991.

[15] M. Gen and R. Cheng, *Genetic Algorithms & Engineering Design*. New York: Wiley, 1997.

[16] M. Gerla and L. Kleinrock, "On the topological design of distributed computer networks," *IEEE Trans. Commun.*, vol. COM-25, pp. 48–60, Jan. 1977.

[17] A. Gibbons, *Algorithmic Graph Theory*. Cambridge, U.K.: Cambridge Univ., 1985.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[19] J. J. Grenfenstette, "Optimized control of parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 122–128, 1986.
[20] ——, "Incorporation problem specific knowledge into genetic algorithms," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. Los Altos, CA: Morgan Kaufmann, 1987.
[21] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michgan Press, 1992. Reprint by MIT Press, Cambridge, MA.
[22] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proc. 1st ICEC*, 1994, pp. 82–87.
[23] T. C. Hu, "Optimum communication spanning trees," *SIAM J. Comput.*, no. 3, pp. 188–195, 1994.
[24] H. Ishibuchi and T. Murata, "Multi-objective genetic local search algorithm," in *Proc. 3rd ICEC*, 1996, pp. 119–124.
[25] B. A. Julstrom, "A genetic algorithm for rectilinear Steiner problem," in *Proc. 5th ICGA*, 1993, pp. 474–479.
[26] A. Kershenbaum, *Telecommunication Network Design Algorithms*. New York: McGraw-Hill, 1993, pp. 225–236.
[27] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*. New York: Springer-Verlag, 1996.
[28] B. M. E. Moret and H. D. Shapiro, *Algorithms from P to NP: Design and Efficiency*. San Francisco, CA: Benjamin/Cummings, 1991, vol. I, pp. 254–292.
[29] T. Murata and H. Ishibuchi, "MOGA: Multi-objective genetic algorithms," in *Proc. 2nd ICEC*, 1995, pp. 289–294.
[30] C. C. Palmer and A. Kershenbaum, "An approach to a problem in network design using genetic algorithms," *Networks*, vol. 26, pp. 151–163, 1995.
[31] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, 1957.
[32] H. Prüfer, "Neuer bewis eines Satzes über Permutationnen," *Arch. Math. Phys.*, vol. 27, pp. 742–744, 1918.
[33] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proc. 1st ICGA*, 1985, pp. 93–100.
[34] J. D. Schaffer, R. A. Carunam, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proc. 3rd ICGA*, J. D. Schaffer, Ed., 1989, pp. 51–60.
[35] H. Tamaki, H. Kita, and S. Kobayahi, "Multi-objective optimization by genetic algoritms: A review," in *Proc. 3rd ICEC*, 1996, pp. 517–522.
[36] J. J. Buckly and Y. Hayashi, "Fuzzy genetic algorithm and applications," *Fuzzy Sets and Systems*, vol. 61, pp. 129–136, 1994.

Chi-Chun Lo (M'88) was born in Taipei, Taiwan, R.O.C., on August 22, 1951. He received the B.S. degree in mathematics from the National Central University, Chungli, Taiwan, in 1974, the M.S. degree in computer science from the Memphis State University, Memphis, TN, in 1978, and the Ph.D. degree in computer science from the Polytechnic University, Brooklyn, NY, in 1987.

From 1981 to 1986, he was with the AT&T Bell Laboratories, Holmdel, NJ, as Member of Technical Staff. From 1986 to 1990, he worked for the Bell Communications Research as Member of Technical Staff. Since 1990, he has been an Associate Professor in the Institute of Information Management, National Chaio-Tung University, Hsinchu, Taiwan. Currently, he is the Director of the institute. His major current research interests include network design algorithm, network management, network security, network architecture, and multimedia system.

Wei-Hsin Chang (M'94) was born in Hsinchu, Taiwan, R.O.C., on June 5, 1965. He received the M.S. and Ph.D. degrees in information management from National Chiao Tung University, Hsinchu, Taiwan, in 1992 and 2000, respectively.

From 1997 to 1999, he was with the Prime View International, Inc., Hsinchu, as a Network Section Manager in the information center. Since 2000, he has been a Supervisor of the MIS Department in the AMBIT Microsystems Company, Hsinchu. His major research interests include network design algorithm, network management, and network architecture.