6-2001

Genetic algorithms for communications network design - an empirical study of the factors that influence performance

Hsinghua CHOU

G. Premkumar

Chao-Hsien CHU

Singapore Management University, chchu@smu.edu.sg

DOI: https://doi.org/10.1109/4235.930313

Published in IEEE Transactions on Evolutionary Computation, 2001 June, Volume 5, Issue 3, Pages 236-249
https://doi.org/10.1109/4235.930313

Genetic Algorithms for Communications Network Design—An Empirical Study of the Factors that Influence Performance

Hsinghua Chou, G. Premkumar, and Chao-Hsien Chu

***Abstract*—Genetic algorithms (GAs) are being used extensively in optimization problems as an alternative to traditional heuristics. Although the results have been mixed, very limited research has been performed on the impact of various GA factors on performance. We explore the use of GAs for solving a network optimization problem, the degree-constrained minimum spanning tree problem. We also examine the impact of encoding, crossover, and mutation on the performance of the GA. A specialized repair heuristic is used to improve performance. An experimental design with 48 cells and ten data points in each cell is used to examine the impact of two encoding methods (Prüfer and determinant encoding), three crossover methods (one-point, two-point, and uniform), two mutation methods (insert and exchange), and four networks of varying node sizes (20, 40, 60, 80). Two performance measures, solution quality and computation time, are used to evaluate performance. The results indicate that encoding has the greatest effect on solution quality, followed by mutation and crossover. Among the various options, the combination of determinant encoding, exchange mutation, and uniform crossover more often provides better results for solution quality than other combinations. For computation time, the combination of determinant encoding, exchange mutation, and one-point crossover provides better results.**

***Index Terms*—Degree-constrained minimum spanning tree, genetic algorithms, GA operators, network design.**

I. Introduction

GENETIC algorithms (GAs) and other evolutionary algorithms have been used extensively in a wide variety of application domains including engineering, economics, finance, manufacturing, agriculture, and business. Simple GAs seem to perform well in certain problem areas, but not so well in other areas [1]. Since GAs consist of various components and parameters that can be modified, it is important to understand the impact of these factors on GA performance. Many factors such as population size, reproduction operators, fitness function, encoding methods, etc., can have a significant impact [2]–[4]. Researchers have studied the impact of various factors including problem encoding, crossover and mutation operators, population size, crossover and mutation rate, halting criterion, etc. [5]–[7]. For example, researchers have examined the impact of different crossover operators [8]–[15]. Dengiz *et al.* [7] examined the impact of population size, crossover rate, and mutation rate. Abuali *et al.* [16] compared the performance of three different encoding methods—Prüfer, Link-node bias, and determinant encoding. Booker [17] compared one-point and two-point crossover methods for nonorder problems. Most research has focused on only one factor—crossover or mutation operator, population size, or encoding method. Limited research has examined impact of a combination of these factors. The interaction of encoding and reproduction operators is a key to GA's performance [18], [19]. Since these factors are closely interrelated, evaluating the role of these different factors and their interrelationships should help in identifying the important factors and their ideal combinations for effective performance in different settings. This study examines some of these factors in solving communications network design problems.

The minimum spanning tree (MST) problem is one of the best-known network optimization problems used for designing backbone networks. A special case of MST is a degree-constrained minimum spanning tree (DCMST), where additional constraints specify the upper and lower bounds of the number of links to a node. The DCMST problem is NP-hard and traditional heuristics have had only limited success in solving small to midsize problems [20]. The major objectives of the current study are:

1) to develop a GA approach to solve the DCMST problem;
2) to determine the impact of various factors, specifically the impact of encoding, crossover, and mutation, on GA's performance.

The paper is organized as follows. In Section II, we present the details of the DCMST problem. In Section III, we elaborate on the GA approach to solve this problem; specifically, we discuss our encoding strategies, various crossover and mutation operators, initialization and selection strategy, and the repair method. In Section IV, we describe the research hypotheses and experimental design. Finally, in Section V, the results are presented and discussed.

II. DCMST

Many of the network topology design problems start with the MST, which attempts to find a minimum cost tree that connects all the nodes of the network. The links or edges have associated costs that could be based on their distance, capacity, quality of

Manuscript received February 22, 1999; revised July 13, 1999, January 3, 2000, and July 25, 2000.

H. Chou is with the Sprint Corporation, Overland Park, KS 66210 USA.

G. Premkumar is with the College of Business, Iowa State University, Ames, IA 50011 USA (e-mail: prem@iastate.edu).

C.-H. Chu is with the School of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802 USA (e-mail: chu@ist.psu.edu).

Publisher Item Identifier S 1089-778X(01)00002-9.

line, etc. There might be other constraints imposed on the design such as the number of nodes in a subtree, degree constraints on nodes, flow and capacity constraints on any edge or node, and type of services available on the edge or node. The MST problem is found in communication networks, circuit design, transportation, and logistics among others. Several variations to the basic problem have been identified for different design contexts [21], [22]. The complexity of the MST problem increases as the number of nodes increases. Many heuristics have been developed to solve large problems, prominent among them being Kruskal [23] and Prim [24] algorithms.

One of the popular variations of the MST problem is the DCMST. The MST algorithm may occasionally generate a minimal spanning tree where all the links connect to one or two nodes. This solution, although optimal, may be highly vulnerable to failure due to overreliance on a few nodes. Furthermore, the technology to connect many links to a node may not be available or may be too expensive. Hence, it may be necessary to limit the number of links connecting to a node. Alternatively, from a reliability perspective it is desirable to have more than one link connect to a node so that alternative routes can be selected in the case of a node or link failure [25], [26]. Hence, in practice, we may add additional constraints that specify the upper and lower bound of the number of links connecting to a node. DCMST was specifically developed as a special case of MST with additional constraints to improve the reliability of the network and rerouting of traffic in the case of node failures.

The DCMST problem can be stated as follows:. Let $G = (V, E)$ be a complete graph of n nodes in which each edge (i, j) has an associated weight $c(i, j)$. Determine a spanning tree of minimum total edge weight (sum of the weights of the edges is minimum) such that, at each node i, the degree d_i is between a lower bound Ld_i and upper bound Ud_i. The mathematical formulation of the problem is presented in Appendix I.

The problem of constrained trees has been studied for many years [20]. The popular travelling salesman problem (TSP) is a subset of the DCMST, where the degree constraint is two. Johnson [27] describes ten constrained MST problems that are NP-hard. The earliest heuristic algorithm for DCMST was proposed by Obruca [28] as a solution to TSP. Narula and Ho [20] proposed three heuristic algorithms to solve the DCMST problem: primal, dual, and branch and bound. Savelsbergh and Volgenant [29] introduced an "edge exchange" algorithm that provided better performance. Although these methods solve experimental size problems, the computation time increases dramatically when the problem size gets larger. In recent years, researchers have attempted using meta-heuristics such as Tabu search and genetic (evolutionary) algorithms to solve these problems.

III. Applying GAs to Network Design

GAs have been used to solve communications network design problems in a wide variety of contexts [7], [16], [30]–[37]. However, there is very limited research on using GAs for DCMST problems and also examining the impact of GA parameters on its performance. This study attempts to address both these issues. The details of the GA used for this problem are described below.

A. Problem Representation

In a network design problem, the strings used for representation could have information on the link identification, capacity of the network, cost of the link, and whether the link is part of the solution. The encoding should represent all the relevant parameters of the problem and be able to uniformly represent all possible solutions to the problem. The encoding can be of two types: direct or indirect [2], [38]. With direct encoding, the strings can be read directly, while with indirect encoding, a decoding algorithm is used to expand the strings into meaningful information for evaluation.

Spanning trees have been used extensively in a variety of network optimization problems and various encoding methods have been used to represent trees [38]. They can be classified broadly into three categories: edge, node, and edge-node encoding. In *edge encoding* [39], a binary string is used to represent the edges of the spanning tree. Dengiz *et al.* [7] used an integer string to represent a tree in communication networks, where each integer arbitrarily represents an edge. Edge encoding has been found to be a poor representation for MST due to the low probability of obtaining a tree [34], [38]. In *node-* or *vertex*-based encoding the nodes, rather than edges, are represented in the encoding. A popular encoding method for trees is based on Prüfer number [40], which represents a tree of n nodes with $n-2$ digits, where each digit is an integer between one and n. Prüfer encoding is an indirect encoding method that has been used to represent an MST [39]. However, Prüfer encoding has very limited locality as changing any one digit in the Prüfer number can dramatically change the tree [34]. Abuali *et al.* [16] suggested determinant encoding as an alternate node-based encoding for representing spanning trees. *Edge and node* encoding uses information about both nodes and links. An example in this category is link and node biased (LNB) encoding [34], which uses a bias value for each node and link in calculating the cost of the network. This representation has some limitations including long encoding and lack of information about degree on nodes. Abuali *et al.* [16] compared Prüfer, determinant, and LNB encoding methods and found that determinant encoding provided better performance. In this study, we examine the impact of two encoding methods, *Prüfer and determinant encoding*, that are appropriate for the DCMST problem. A brief description of the two algorithms is provided in Appendix II.

B. Population Initialization

There are two parameters to be decided for initialization: the initial population size and the procedure to initialize the population. Initially, researchers thought that the population size needed to increase exponentially with the length of the chromosome string in order to generate good solutions [41]. Recent studies have shown, however, that satisfactory results can be obtained with a much smaller population size [42]. There are two ways to generate the initial population—random initialization and heuristic initialization. We use the random method, where

TABLE I
(a) LINK COSTS FOR A NINE-NODE SPANNING TREE. (b) UPPER AND LOWER DEGREE CONSTRAINTS FOR NODES

	1	2	3	4	5	6	7	8	9
1	*	224	224	361	671	224	539	800	943
2	224	*	200	200	447	283	400	728	762
3	224	200	*	400	556	447	600	922	949
4	361	200	400	*	400	200	200	539	583
5	671	447	566	400	*	600	447	781	510
6	224	283	447	200	447	*	283	500	707
7	539	400	600	200	447	283	*	361	424
8	800	728	922	539	781	500	361	*	500
9	943	762	949	583	510	707	424	500	*

(a)

Node	Lower	Upper
1	1	3
2	1	1
3	2	5
4	1	2
5	4	8
6	1	3
7	1	3
8	1	4
9	1	3

(b)

for each gene we randomly generate an integer from a range of one to the number of nodes (Fig. 2 has nine nodes). The initial chromosomes need not represent a legal or feasible tree.

C. Repair Function

Determinant encoding generates some chromosomes that are illegal (not a spanning tree) and some that are legal, but infeasible due to constraint violations. A combination of repair and penalty functions is used in this study to repair both illegal and infeasible solutions.

1) Repair Function for Illegal Chromosomes: The spanning tree may be illegal due to three reasons: missing node "1," self-loop, or cycles. *Missing node "1"* occurs when a chromosome does not contain any gene that has a value "1." Since the fixed position starts from two, the generated spanning tree will not contain "1" and, therefore, will not span all the nodes. *Self-loop* occurs when the value of a gene is equal to its correspondent node position. For example, in determinant coding, a chromosome (4 4 2 5) cannot construct a five-node spanning tree because the value of the gene is five and its correspondent node position is five, which causes an illegal self-loop connection (5–5). *Cycle* occurs if a subset of links connect in a loop, returning to

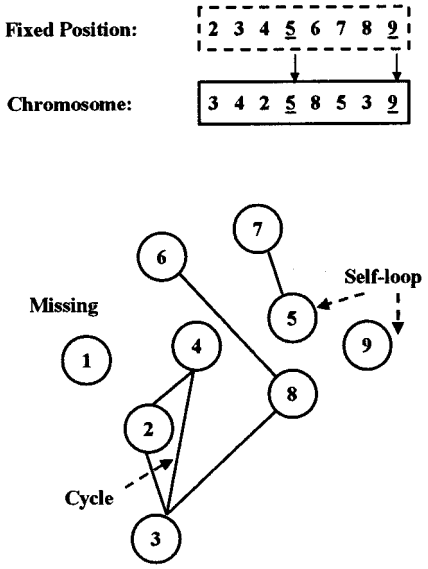

Fig. 1. Problem representation with determinant encoding—illegal tree.

the original node, in which case one of the links may be unnecessary. The algorithm illustrated below describes a strategy to solve all the three situations that lead to illegal spanning trees.

Given a determinant encoding string C with length of $N-1$, where N represents number of nodes. Assume $C(x)$ represents the allele of the xth fixed position in chromosome C, where x starts from two to N. The algorithm is presented below along with an example that illustrates every step of the algorithm. The cost for a nine-node spanning tree problem is shown in Table I(a). Fig. 1 shows the chromosome coded by the determinant encoding method, which contains three kinds of problems: missing node 1, self-loop, and cycles.

S1) Repair missing node 1.

For a given encoding string C, identify x, where $C(x) = 1$. If x exists, go to S2; otherwise, check cost table and pick node x $(x \neq 1)$, where x has the lowest connecting cost to node1. Set $C(x) = 1$. If there is a tie, randomly select a position.

Check if "1" exists in the string. If not, find the lowest-cost node connected to node "1." Based on the cost table, nodes "2," "3," and "6" have the lowest connection cost to node "1." Since they all have the same cost randomly choose one of them, say, fixed position "6." Replace the allele in fixed position "6" with "1." The result is shown in Fig. 2(a).

S2) Repair self-loop.

For a given encoding string C, identify x, where $x = C(x)$. Check the cost of connecting each node $i(i \neq x)$ with node x. Select node n, which has the lowest connecting cost with node x and set $C(x) = n$. If there is a tie, randomly select a position.

Check for self-loop. The fourth and the last position generate a self loop— 5–5 and 9–9. Select a node with lowest cost connection to nodes "5" and "9." Using the cost table, node "4" is paired with node "5" and node "7" is paired with node "9". The result is presented in Fig. 2(b).

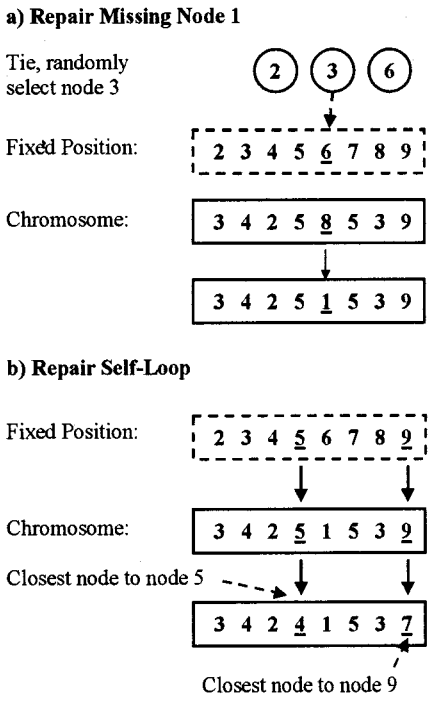

Fig. 2. Repair of illegal tree—missing node and self loop.

S3) Examining cycle.

For a given encoding string C, allocate *vectors* A and B of size N and initialize value as Null. Assume $A(i)$ and $B(i)$ are the ith position in *vectors* A and B.

For a given chromosome C:

```
If A(C(x)) and A(x) are Null,
  select the smaller of C(x) and x.
    Suppose C(x), then set A(C(x)) = C(x)
    and set A(x) = C(x).
Else, if A(C(x)) is Null and A(x) is
  not Null,
    set A(C(x)) = A(x).
Else, if A(C(x)) is not Null and A(x)
  is Null,
     set A(x) = A(C(x)).
Else, if both A(C(x)) and A(x) are
  not Null,
    then if A(C(x)) = A(x), set B(x) = C(x);
    otherwise, select the smaller of
      A(C(x)) and A(x). Suppose A(C(x)),
      then scan A(j) , where 1 ≤ j ≤ N.
      If A(j) = A(x), set A(j) = A(C(x)).
      Set j = 1 and Go to S4.
```

The repair operator initially checks if cycles or subtrees exist by using a grouping algorithm. Nodes in the same subtree are grouped in the same set numbered by the smallest node number. In this example, we find that there is one cycle causing two subtrees. The algorithm maintains two arrays: A and B. Array A contains information on sets where each node belongs to and array B keeps track of information on node pairs that cause cycles. Based on the subsets in A, repair function is used to connect broken subset into one set. The result after examining cycles is presented in Fig. 3.

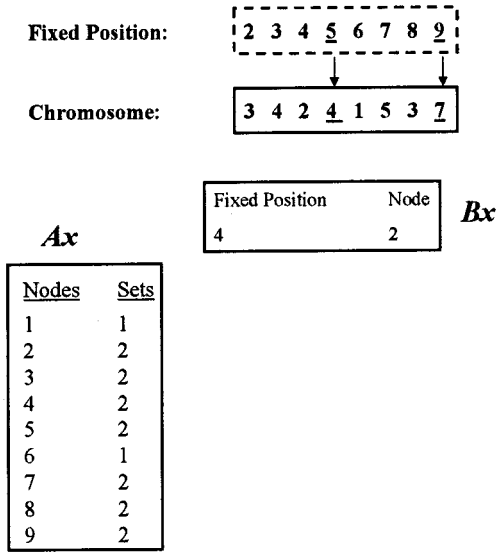

Fig. 3. Repair of illegal tree—cycle.

S4) Repairing cycle.

Exam $B(j)$, if $j > N$, stop; else, if $B(j) =$ Null, set $j = j + 1$ and go back to S4; otherwise, identify i, where $1 \leq i \leq N$, $A(i) \neq A(B(j))$ and has the lowest connection cost t' to node $B(j)$. Identify k, where $1 \leq k \leq N$, $A(k) \neq A(j)$ and has the lowest connection cost t'' to node j. Select the smaller of t' and t''. Assume t' is the smaller, then set $C(B(j)) = i$.. Exam $A(l)$, where $1 \leq l \leq N$, if $A(l) = A(i)$, set $A(l) = A(B(j))$. Set $j = j + 1$ and go to S4.

After examining cycles, the repair function picks the first pair from array B and checks the cost table to locate the node in different set which has the lowest cost to link with the fixed position. In our example, we find out the possible links are $\{4-1\}$ and $\{4-6\}$. After checking the cost table, $\{4-6\}$ is found to be the lower in cost. Thus, we replace the allele in fixed position "4" by node "6." The updated table and chromosome is represented in Fig. 4.

2) Repair Function for Infeasible Chromosomes: Both encoding methods may generate a spanning tree that violates the degree constraints on the nodes. There are two choices to deal with infeasible solutions. One is to repair and the other is to assign a penalty value. The repair option is used extensively, but may be computationally expensive and/or may disturb the certain desirable aspects of the parent solution carried in the children. The penalty option has also been used with considerable success [43]. The design of the penalty function is critical to ensure quick convergence without sacrificing on the search of the entire solution space or expensive computation time.

We use a method that combines both repair and penalty. The repair function has to first check if each node's degree is within the lower and upper degree constraints. If it violates the constraints, the network has to be repaired. The repair algorithm

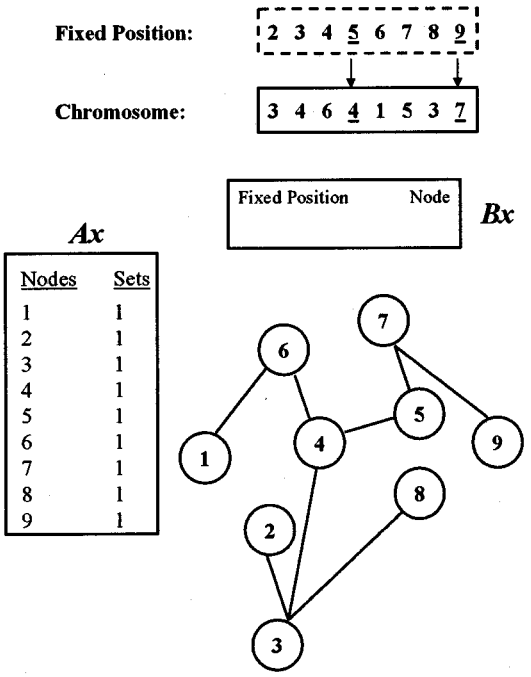

Fig. 4. Legal spanning tree after repair.

replaces a degree-violated node by randomly choosing another node that does not violate the degree constraints.

S1) For a given encoding string C with length of $N-1$, where N represents nodes. A *vector* A is used to store the degree status of all nodes in a graph. Let $A(i)$ represent the value in the ith position in *vector* A. Assume $C(x)$ represents the allele of the x fixed position in chromosome C, where x starts from 2 to N. Set $s=1$ and go to S2.

S2) Check the number of times "s" appears in C. Add the number by one (because there is an extra connection from the fixed position) as its correspondent degree level d_s. Set $A(s)=d_s$. Set $s=s+1$, if $s>N$, set $s=1$ and go to S3; otherwise go back to S2.

S3) Compare $A(s)$ with its upper and lower degree limitation, Ud_s, and Ld_s.

```
If A(s) > Ud_s,
  Randomly select a node n(n ≠ s),
    where A(n) will not be greater
    than Ud_n after adding the extra
    one.
  Randomly select a C(x), where
    C(x) = s and set C(x) = n.
  Update the degree status of n and
    s in vector A such that A(n)
    = A(n) + 1 and A(s) = A(s) − 1;
Else, if A(s) < Ld_s,
  Randomly select a node n(n≠s),
    where d_n will not be less than
    Ld_n after subtracting the extra
    one.
  Randomly select a C(x), where
    C(x) = n and set C(x) = s.
  Update the degree status of n and
    s in vector A such that A(s)
    = A(s) + 1 and A(n) = A(n) − 1;
If A(s) < Ld_s or A(s) > Ud_s, go back to
  S3.
Set s = s + 1, if s > N, stop; other-
  wise, go back to S3.
```

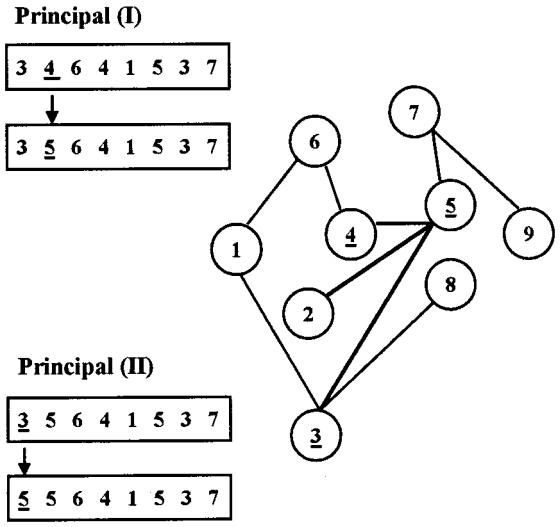

Fig. 5. Random repair for infeasible solutions.

To illustrate, examine the network in Fig. 4. The requirements of degree-constraints are shown in Table I(b). Comparing the degree constraint values in Table I(b) with the given chromosome we observe that node 4 exceeds the upper-level degree constraint by 1 and node 5 is below the lower-level degree constraint by 2. The random repair algorithm, described above, is used to repair the network to satisfy the constraints. Examine the chromosome and find the nodes that violate degree constraints: nodes 4 and 5 in this case. Randomly find a node, say node 5, that does not exceed the upper-level degree constraint after incrementing by one and replace it with node 4. Replace the allele in fixed position 3 (a randomly selected gene contains "4") with node 5. This increases the degree constraint of node 5 by one and decreases node 4 by one. We find from the summary table of degree constraints that node 5 is one degree less than the constraint. Randomly select a fixed position, say "2," so that its correspondent gene's degree will not be less than its lower limitation after decreasing by one. Assign node 5 to fixed position "2" and decrease the degree of node 3 (the original gene in fixed position "2") by one and increase the degree of node 5 by one. The final network layout is presented in Fig. 5.

3) Penalty Function: Since the network generated after repairing the degree constraints could now be illegal (not a spanning tree), the illegal repair algorithm will be used one more time to ensure that all the repaired chromosomes are legal. After this repair, a few chromosomes may still violate the degree constraints. A function is used to penalize chromosomes that violate the constraints by adding a negative value to the fitness value. Since the cost of network is dependent on the network size and, therefore, the chromosome length, the penalty value has to be

proportional to the cost of the network. The value is computed by providing a weightage for the length of the chromosome. The penalty function is computed as follows:

$$P(x) = \sum_{i \in x} |d_i(x) - D_i(x)|$$
$$g'(x) = g(x) + (P(x) + L(x))^* W$$

where

$g'(x)$	new cost value of chromosome x;
$g(x)$	original cost value of chromosome x;
$P(x)$	penalty value of chromosome x;
$L(x)$	length of chromosome x;
W	weight given by the user;
$d_i(x)$	degree of node i in chromosome x;
$D_i(x)$	lower/upper degree limitation of node i in chromosome x.

As shown above, the penalty value is based on the difference of each allele and the degree constraint of its correspondent node, and the length of the chromosome.

D. Fitness Function

The fitness function will interpret the chromosome in terms of the physical representation (phenotype) and evaluate its fitness based on certain characteristics that are desired in the solution. The definition of the fitness function is very critical because it must accurately measure the desirability of the features described by the chromosome. The function should be efficient in its computation since it is used a large number of times to evaluate each and every solution. The fitness function computes a fitness value for each chromosome. The fitness value is not an absolute value, but a value relative to a given population [41]

$$\begin{aligned} f(x) &= C_{\min} + g(x), && \text{when } g(x) > -C_{\min} \\ &= 0, && \text{otherwise.} \end{aligned}$$

$f(x)$ indicates the new fitness value being scaled and $C_{\min}$ is the smallest objective value in the current generation of the population. The chromosomes are ranked based on the fitness value for selection for the next generation.

E. Selection Methods

There are many methods to select the population. Each has advantages and disadvantages [44]–[46]. Some researchers prefer to use the enlarged sampling approach since it reduces the possibility of duplicate chromosomes entering the population during selection [38]. Typically, there are two enlarged sampling strategies: $(\mu + \lambda)$ and (μ, λ). In $(\mu + \lambda)$ strategy, μ parents and λ offsprings compete for survival and the μ best solutions are selected for the next generation. In (μ, λ) strategy, we select the μ best $(1 < \mu < \lambda)$ solutions from out of λ offspring solutions. We used the stochastic $(\mu + \lambda)$ method.

F. Reproduction Operator—Crossover

Crossover methods such as one-point, two-point, and uniform crossover are extensively used in GA models [10]. Booker [17] compared one and two-point crossover and found the two-point crossover to be better for nonorder problems. Syswerda [10] compared one-point, two-point, and uniform crossover operator on six different types of problems using a steady–state GA with simple mutation and binary encoding. He found uniform crossover to be more effective than two-point crossover and two-point crossover consistently better than one-point crossover. Poon and Carter [15] compared the performance of ten crossover operators over six applications and found that order and union crossover operators performed uniformly well in all applications and position-based and intersection-crossover operators were the worst performers. We evaluate the effectiveness of one-point, two-point, and uniform crossover methods. In *one-point crossover*, a random position is generated for a pair of chromosomes and the alleles of the first chromosome from this fixed position to the end are exchanged with the second chromosome in the same range. In this process, the alleles of the second chromosome are transferred to the respective alleles in the first chromosome. *Two-point crossover* generates two random positions, head and tail. The alleles of the first chromosome from the head position to the tail are exchanged with the second chromosome in the same range. *Uniform crossover* is a dynamic and nondeterministic method where a set of positions, called a mask, is chosen for each of the chromosomes and their alleles are exchanged with each other based on the generated positions. There are two random decisions—the positions to replace and the number of genes to replace.

G. Reproduction Operator—Mutation

Various mutation methods have been examined including inversion, insertion, displacement, reciprocal exchange, and heuristic mutation [38]. Two mutation methods were used here, namely, insert and exchange mutation. *Insert mutation* randomly generates two positions in a given chromosome and inserts the gene from the first position in the second position and shifts all the genes to the right by one position. *Exchange mutation* randomly selects two positions in a given chromosome and exchanges both genes. Insert mutation causes greater changes to the chromosome compared to exchange mutation. This is particularly true in our study since the genes in a chromosome are related to their fixed positions.

H. Control Parameters

An important control parameter is the *halting criterion*. There are several *halting criteria* to choose from including number of generations, computing time, and fitness convergence. Fitness convergence occurs when all the chromosomes in the population have the same fitness value. In this study, fitness convergence is selected as the halting criterion.

The population size, crossover rate, and mutation rate are three other important control parameters for GA. The initial *population size* was fixed at 100 and stochastic $(\mu+\lambda)$ selection method was used. Since researchers suggest a high crossover and mutation rate for enlarged sampling method [38], we chose 1.0 as the *crossover and mutation rate*.

IV. Research Hypotheses and Experimental Design

We use two performance measures to evaluate the results: the value of the objective function (solution quality) and com-

TABLE II
SOLUTION QUALITY(COST) AND COMPUTATION TIME PERFORMANCE FOR COMBINATIONS OF ENCODING, CROSSOVER, MUTATION, AND NETWORK SIZE

Network Size→ / Factors ↓	20		40		60		80	
	Cost	CPU(sec)	Cost	CPU(sec)	Cost	CPU(sec)	Cost	CPU(sec)
E1C1M1	1561.7	4.0	2167.4	29.4	2666.3	118.0	3082.2	396.8
E1C2M1	1562.2	4.0	2173.6	**26.1**	2668.9	99.2	3065.4	342.6
E1C3M1	**1560.8**	**3.8**	**2159.3**	26.5	**2658.7**	**85.4**	**3059.1**	318.0
E1C1M2	1602.5	5.9	2456.6	35.2	3478.4	96.9	4600.2	214.6
E1C2M2	1591.3	6.4	2346.5	30.5	3206.8	96.6	4277.8	**184.8**
E1C3M2	1576.2	6.2	2242.5	38.6	2835	131.0	3403.1	274.4
E2C1M1	1789.9	6.5	3206.8	37.8	4333.8	154.5	5522.5	391.9
E2C2M1	1748.9	7.2	3140.8	47.1	4246.2	156.1	5546.9	429.7
E2C3M1	1749.0	4.8	2780.7	30.0	4064.7	121.6	5361.4	315.
E2C1M2	1838.2	8.6	3241.7	42.9	4670.3	169.3	6239.3	411.1
E2C2M2	1792.5	9.6	3165.8	59.1	4526.3	203.3	6125.6	550.0
E2C3M2	1750.2	17.1	2784.9	127.8	4424.1	687.8	5689.0	1784.9

* E1: Determinant encoding; E2: Prüfer encoding;
* C1: one-point crossover; C2: two-point crossover; C3: uniform crossover;
* M1: exchange mutation; M2: insert mutation

putation time. The factors identified for study are encoding, crossover, mutation, and network size.

The following null hypotheses were formulated.

- H1: The mean value of solution quality is the same for the two encoding methods.
- H2: The mean value of computation time is the same for the two encoding methods.
- H3: The mean value of solution quality is the same for the three crossover methods.
- H4: The mean value of computation time is the same for the three crossover methods.
- H5: The mean value of solution quality is the same for the two mutation methods.
- H6: The mean value of computation time is the same for the two mutation methods.

A. Experimental Design

There are four experimental variables—encoding, crossover, mutation, and network size. We considered two encoding methods (determinant and Prüfer), three crossover methods (one-point, two-point, and uniform), two mutation methods (insert and exchange), and four networks of varying node sizes (20, 40, 60, 80). Hence, an experimental design with 48 cells ($2\times3\times2\times4$) was used to represent the combinations of all the factors. For each cell, ten data sets were generated randomly. In total, there were 480 data points for the experiment, 48 cells with ten data points in each cell. Each network was generated using a special-purpose algorithm [47], where the coordinates of each node in the data set were generated from a $500\times$ 500 graphic plane with each data set using a different random number seed scaled from 0.1 to 0.9. For each cell, ten different networks were created using ten different random number seeds. The cost of the edges was calculated based on the Euclidean distance between the nodes. The experiment controlled for other factors such as population size, selection method, crossover rate, and mutation rate.

V. COMPUTATIONAL RESULTS

A. Comparison of GA Operators

The results of the experiment are shown in Table II. The column represents the network size and the rows show the various combinations of GA factors. The values in the cell report the average value for the ten data sets in each cell. The highlighted cells represent the best solution.

Table III presents the results of an analysis of variance (ANOVA) for solution quality using four main factors—encoding, crossover, mutation, and network size. The results indicate that all the factors are significant at $P < 0.001$, thereby rejecting the null hypotheses H1, H3, and H5. We also evaluated the two-way interaction among the four factors. All the interactions, except crossover with encoding, are significant. In general, the values of interaction terms for crossover with the other three factors are relatively lower than the other interaction terms.

Table IV presents the results of ANOVA for computation time. The results indicate that all the factors are significant at $P < 0.001$, thereby rejecting the null hypotheses H2, H4, and H6. Among the interaction terms, encoding with mutation and crossover with mutation had high values.

The variation in the absolute value of solution quality for the different combinations are more pronounced for larger networks as compared to smaller networks. While the difference between the lowest and highest cost value is only 278 (18% of absolute value) for the 20-node network it is around 3180 (103% of absolute value) for the 80-node network. Hence, for larger network

TABLE III
ANOVA—SOLUTION QUALITY, CROSSOVER, MUTATION, ENCODING, AND NETWORK SIZE

	Sum of Squares	Df	Mean Square	F Value	Sig.
Main effects					
(Combined)	750000000.0	7	110000000.0	768.59	.0001
Crossover	4660076.0	2	2330038.0	16.675	.0001
Mutation	15000000.0	1	15000000.0	109.069	.0001
Encoding	150000000.0	1	150000000.0	1100.764	.0001
Network Size	580000000.0	3	190000000.0	1379.003	.0001
2-way interactions	87000000.0	17	5108131	36.557	.0001
C x S	4241450.0	6	706908.3	5.059	.0001
C x E	256394.9	2	128197.4	.917	.400
C x M	1152652.0	2	576326.1	4.125	.017
S x E	70000000.0	3	23000000.0	167.285	.0001
S x M	9136805.0	3	3045602.0	21.796	.0001
E x M	1925840.0	1	1925840.0	13.782	.0001
Model	840000000.0	24	350000000.0	250.069	.0001
Residual	64000000.0	455	139731.7		
Total	900000000.0	479	1883507.0		

* C: crossover; M: mutation; E: encoding; S: Network size.
* Sig.: significant level; df: degree of freedom.
* $\alpha = 0.05$

TABLE IV
ANOVA—COMPUTATION TIME, CROSSOVER, MUTATION, ENCODING, AND NETWORK SIZE

	Sum of Squares	Df	Mean Square	F Value	Sig.
Main effects					
(Combined)	20000000.0	7	2865538.0	111.749	.0001
Crossover	1333288.0	2	666644.2	25.998	.0001
Mutation	864047.8	1	864047.8	33.696	.0001
Encoding	2133028.0	1	2133028.00	83.183	.0001
Network Size	16000000.0	3	5242800.0	204.457	.0001
2-way interactions	9575061.0	17	563238.9	21.965	.0001
C x E	1254605.0	2	627302.4	24.463	.0001
C x S	1597472.0	6	266245.4	10.383	.400
C x M	2074420.0	2	1037210.0	40.449	.017
S x E	2371599.0	3	790533.1	30.8295	.0001
S x M	756049.3	3	252016.4	9.828	.0001
E x M	1520915.0	1	1520915.0	59.312	.0001
Model	30000000.0	24	1234743.0	48.152	.0001
Residual	12000000.0	455	25642.59		
Total	41000000.0	479	86223.81		

* C: crossover; M: mutation; E: encoding; S: network size.
* Sig.: significant level; df: degree of freedom.
* $\alpha = 0.05$

sizes, there is a greater possibility of the solution being further away from the minimal solution. A similar pattern is also noticed in the absolute value of computation time. While the difference is 13.3 s for the 20-node network, it is 1466 s for the 80-node network. These results highlight clearly the importance of selecting the right GA parameters. The selection of appropriate parameters becomes more critical as the problem size increases.

TABLE V
EFFECT OF ENCODING, CROSSOVER, AND MUTATION ON SOLUTION QUALITY—PAIRED t-TEST. (a) SOLUTION QUALITY AND ENCODING. (b) SOLUTION QUALITY AND CROSSOVER. (c) SOLUTION QUALITY AND MUTATION

Operator	Mean	SD	t value	P
Determinant	2583.438	889.899	21.840	0.0001
Prüfer	3739.146	1529.513		

(a)

Operators	Mean	SD	t value	P
One point	3278.613	1423.130	3.537 (*a)	0.0001 (*a)
Two Point	3199.094	1385.219	7.022 (*b)	0.0001 (*b)
Uniform	3006.506	1304.137	8.410 (*c)	0.0001 (*c)

*a is t-test of one point and two point, b is t-test of two point and uniform, c is t-test of Uniform and one point

(b)

Operators	Mean	SD	t value	P
Exchange	2994.883	1248.756	10.53	0.0001
Insert	3327.258	1466.88		

(c)

While ANOVA results generally indicate the factors that have the maximum impact on performance, they do not indicate which of the options in each of these factors lead to better performance. An examination of the mean values in Table II indicates that the combination of uniform crossover, exchange mutation, and determinant encoding generates the best solution quality for all network sizes.

To generalize from our experiments, we performed additional data analysis on subgroups. The values for solution quality and computation time are influenced by various options within the four factors: encoding, crossover, mutation, and network size. To compare the options within each factor, we need to control for the effect of all the other factors. For example, if we have to compare the performance of insert and exchange mutation, we need to control for the variations due to encoding, crossover, network size, and problem set. Hence, we recast the data set so that we compared the results of two sets of experiments where all factors, except the one under study, are kept constant. For example, to compare insert and exchange mutation we created pairs of data sets (240 in all), where all the other factors were same, but varied only in the mutation treatment. Paired t-tests were conducted between these pairs to statistically test if the mean values of solution quality and computation time were different for each of the different options. The results for solution quality and computation time are shown in Tables V and VI, respectively. The results for solution quality indicate that: 1) determinant encoding was better than Prüfer encoding; 2) exchange mutation was better than insert mutation; and 3) uniform crossover was better than one- or two-point crossover. The results for computation time indicate that: 1) determinant encoding was better than Prüfer encoding; 2) exchange mutation was better than insert mutation;and 3) one-point crossover was better than two-point or uniform crossover.

VI. DISCUSSION

We found that Prüfer encoding did not provide good results. Since Prüfer encoding does not possess locality, small changes in the chromosome due to mutation or crossover create totally new solutions. Therefore, Prüfer number, although a useful way to represent trees, is not an effective representation for a GA. Determinant encoding generates both illegal and infeasible solutions. The repair of infeasible solutions is a significant computation cost. In our case, the repair of illegal and infeasible solutions could go through an endless cycle, one creating the other—repairing legal infeasible solutions creating illegal solutions, which on repair could become infeasible. We went through two cycles of repair (illegal–infeasible–illegal) and chose a function to penalize illegal solutions that did not get repaired. We chose a simple penalty function based on a distance vector, but better penalty functions can be designed to reduce repair without sacrificing on global search space. Adaptive penalty function, where the function varies between generations, could be used to improve performance [43], [48]–[50]. We could use an adaptive penalty function that uses a low penalty value in the earlier generations to widen the search space, but increases the penalty in later generations to lead to faster convergence.

An area for performance improvement lies in the repair algorithm. Researchers have advocated the use of heuristics for improving a GA's performance [18]. We could expect some im-

TABLE VI
EFFECT OF ENCODING, CROSSOVER, AND MUTATION ON COMPUTATION TIME—PAIRED t-TEST. (a) COMPUTATION TIME AND ENCODING. (b) COMPUTATION TIME AND CROSSOVER. (c) COMPUTATION TIME AND MUTATION

Operators	Mean	SD	*t* value	*P*
Determinant	107.287	123.980	6.210	0.0001
Prüfer	240.611	385.373		

(a)

Operators	Mean	SD	*t* value	*P*
One point	132.724	149.077	1.437 (*a)	0.076 (*a)
Two Point	140.785	169.510	3.964 (*b)	0.0001 (*b)
Uniform	248.338	447.674	3.94 (*c)	0.0001 (*c)

*a is t-test of one point and two point, b is t-test of two point and uniform, c is t-test of uniform and one point

(b)

Operators	Mean	SD	*t* value	*P*
Exchange	131.512	149.780	3.951	0.0001
Insert	216.377	383.091		

(c)

provement in performance by using a heuristic for repair rather than the random repair method that is currently being used. With Prüfer encoding, it is difficult to use heuristics for repairing infeasible networks. However, determinant encoding provides an opportunity to use heuristics to repair networks that violate degree constraints. Hence, a heuristic was used and its performance was compared with the random repair method. This heuristic used a "local search hill climber" approach to repair chromosomes and improve performance.

The heuristic repair algorithm selects the longest neighbor node to disconnect and the closest neighbor node to connect so as to meet the degree requirements. The algorithm is as follows.

S1) For the given encoding string C with length of $N-1$, where N represents the number of nodes. A *vector* A is used to store the degree status of all nodes in a graph. Let $A(i)$ represent the value in the ith position in *vector* A. Assume $C(x)$ represents the allele of the x fixed position in chromosome C, where x starts from 2 to N. Set $s=1$ and go to S2.

S2) Check the total number of times "s" appears in C. Add the number by one (because there is an extra connection from the fixed position) as its correspondent degree levels d_s. Set $A(s)=d_s$ and $s=s+1$. If $s>N$, set $s=1$ and go to S3, otherwise go back to S2.

S3) Compare $A(s)$ with its upper and lower degree limitations Ud_s and Ld_s.

If $A(s) > Ud_s$,
  scan string C and identify x, where $C(x)=s$ and node x has the highest connecting cost with s. Scan the cost table and identify node n, $(n \neq s)$, which has the lowest connecting cost with node x and d_n will not be greater than Ud_n, after adding the extra one. Set $C(x)=n$, and update degree status in *vector* A such that $A(n)=A(n)+1$ and $A(s)=A(s)-1$.
Else, if $A(s) < Ld_s$,
  identify $x(x \neq s)$, where $d_{C(x)}$ will not be less than $Ld_{C(x)}$ after subtracting the extra one and node x has the lowest connecting cost with s. Update the degree status such that $A(s)=A(s)+1$ and $A(C(x))=A(C(x))-1$ and set $C(x)=s$.
If $A(s) < Ld_s$ or $A(s) > Ud_s$, go back to S3.
Set $s=s+1$, if $s>N$, stop; otherwise, go back to S3.

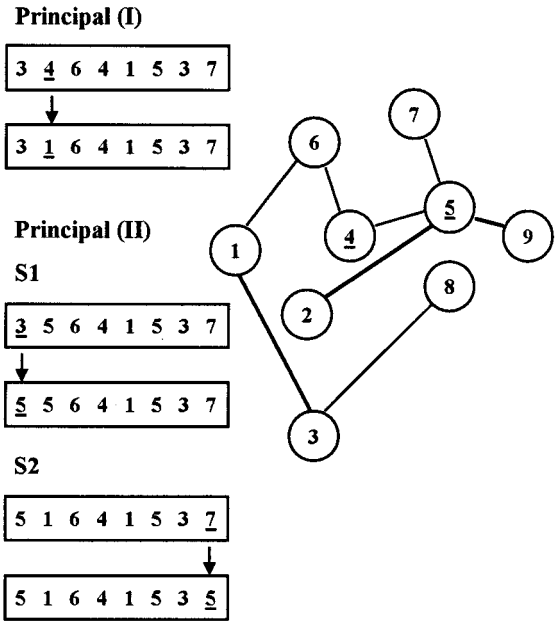

Fig. 6. Heuristic repair for infeasible solutions.

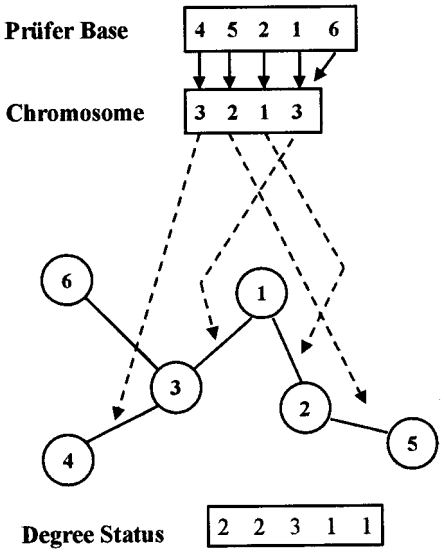

Fig. 7. Tree representation using Prüfer encoding.

Let us examine how the heuristic algorithm repairs the network shown in Fig. 4. It examines the chromosome and find that nodes 4 and 5 violate degree constraints. Identify the two highest cost links (3–4), (5–4) connected to node 4. Randomly break one of them, say link (3–4). Check the cost table and pick the node that has the lowest cost connecting to node 3 and meets the degree constraint requirements after incrementing the degree by one. We notice node 1 is the best fit. Replace the allele in fixed position 3 by node 1 (Fig. 6); increase node 1's degree by one and decrease node 5's degree by one. We find from the degree status table, that node 5 has 2 degrees less than the lower degree limitation. Check the cost table and find out the fixed position node (from 2) that has the lowest connection cost to node 5 and also its correspondent gene's degree does not violate the lower limitation after decreasing by one. Fixed node 2 is the best fit and its correspondent gene 3 does not violate lower limitation after being decreased by one. Replace the allele in fixed position 2 by node 5. However, it is to be noted that it is still one degree less than the lower limitation. Check the table once again and find that the fixed node 9 is the second candidate and its correspondent gene 7 does not violate lower limitation after being decreased by one. Finally, replace the allele in fixed position 9 by 5 and update the final degree status table. The finalized DCMST is as shown in Fig. 6.

The results from random and heuristic repair algorithms are quite different, as is shown in Fig. 5 and 6. In random repair, the convergence of the result will be primarily based on GA evolution. In heuristic repair we specifically identify the closest qualified node, resulting in faster convergence of the result.

An experiment was conducted to compare the performance of random and heuristic repair for various combinations of network sizes, crossover, and mutation operators. Ten samples were run for each combination ($4 \times 3 \times 2$), resulting in 480 data points, 240 for each repair method. A paired t-test was performed to determine if there was any difference in the performance (solution quality and computation time) between the two repair methods after controlling for all the other factors. The results are shown in Table VII. The results indicate the heuristic repair performs better than random repair in both performance measures. The computation time is reduced by 50% by using heuristic repair. The results from the repair method highlight the potential for integrating traditional heuristics with GAs to get improved performance. While GAs are good at finding promising areas of search space but slow to converge to an optimal solution, heuristics are good at converging to optimal solution in a local space, but lack global focus in their search. The combination of these two approaches in a hybrid algorithm provides an algorithm that would be better than the two independently.

VII. Conclusion

We identified the factors that influence the performance of GAs and examined the influence of three critical factors: encoding, crossover, and mutation. The algorithm was used to solve a network optimization problem for DCMST. An experimental design with 48 cells to represent different options within these three factors and ten data points in each cell was used to study the factors and their interactions. The results highlight the importance of choosing the right parameters to get the best performance [19]. We could see a cost reduction of 200% between the best and worst combinations. The combination of determinant encoding, exchange mutation, and uniform crossover provides better results than other combinations most of the time. The study also examined two repair methods, random and heuristic repair, and found that heuristic repair improves performance. This study provides many new opportunities for future research.

First, the study only looked at a few factors. Other possible areas for study are population size, stop criteria, and crossover and mutation rate. There are tradeoffs in each of these and it may be dependent on the problem context. For instance, choosing a smaller population size may reduce the number of computations per generation, but it may take more generations to reach convergence or it may converge in a local optimal area of the search

TABLE VII
EFFECT OF REPAIR STRATEGIES (RANDOM AND HEURISTIC) ON PERFORMANCE—PAIRED t-TEST. (a) SOLUTION QUALITY. (b) COMPUTATION TIME

Operator	Mean	SD	*t* value	*P*
Heuristic	2392.737	596.707	6.557	0.0001
Random	2606.995	889.899		

(a)

Operator	Mean	SD	*t* value	*P*
Heuristic	55.225	65.336	9.403	0.0001
Random	107.287	123.980		

(b)

space providing poor results. Researchers have to examine various population sizes to determine the most optimal size that produces reasonably good results.

Second, other alternatives for the halting criterion could be examined. If our interest in using GAs is in achieving reasonably good solutions rather than best solutions, we could change the halting criterion from convergence to a single value to a range between a minimum and maximum value (e.g., 1% range for quality and time). A graph plot of solution quality generated for various generations indicates that significant improvements in the objective value happens in the first few generations and there is only marginal improvement in subsequent generations. It takes a significant number of generations to converge to a single value for the total population. We could possibly reduce computation time at the cost of a slight variation in solution quality by using a range approach for the halting criterion.

Third, it may be interesting to examine using heuristics in other areas. While this study integrated heuristics in the repair function, future research could explore integrating it in crossover or mutation operators.

APPENDIX I

The mathematical formulation of the DCMST problem is presented below. The following notation is used in the model.

Indices

i, j Index of nodes $i, j = 1, 2, \ldots, n$.
V Set of nodes in the spanning tree.

Parameters

C_{ij} Cost to link nodes i to j.
Ud_i Upper degree constraint on node i.
Ld_i Lower degree constraint on node i.
$|N|$ Number of nodes in a subset N of nodes in V.
$|V|$ Number of the nodes in V.

Decision Variables

X_{ij} Equals one if the link between nodes i to j exists; zero, otherwise.

Minimize

$$\sum_{\substack{i,\, j \in V \\ i<j}} C_{ij} X_{ij} \tag{1}$$

subject to

$$\sum_{\substack{j \in V \\ i \neq j}} X_{ij} \leq Ud_i \quad \forall i \in V \tag{2}$$

$$\sum_{\substack{j \in V \\ i \neq j}} X_{ij} \geq Ld_i \quad \forall i \in V \tag{3}$$

$$\sum_{\substack{i,\, j \in N \\ i<j}} X_{ij} \leq |N| - 1 \quad \forall N \subset V \tag{4}$$

$$\sum_{\substack{i,\, j \in V \\ i<j}} X_{ij} = |V| - 1 \tag{5}$$

$$X_{ij} = 0 \text{ or } 1 \quad i, j \in V. \tag{6}$$

The objective function (1) seeks to minimize the total connecting cost between nodes. The total cost could be distance cost, material cost, or customers' requirement cost. Constraint (2) and (3) specify the lower and upper bound constraints on the number of edges connecting to a node. Constraint (4) is an anticycle constraint and constraint (5) indicates that the number of edges in a spanning tree is equal to the number of nodes minus one. Constraint (6) expresses the binary requirements of the decision variables. In the formulation, there are $2N + 2^N$ constraints and $N^*(N-1)/4$ binary variables. Constraint (4) increases exponentially with network node size, thereby making it impractical to solve large size problems.

APPENDIX II

A brief description of Prüfer and determinant encoding algorithm is provided below.

A. Prüfer Encoding

Each gene in the chromosome represents a correspondent node in the network. The length of the chromosome in Prüfer encoding is $N-2$, where N is the number of nodes in a given graph G. The procedure to generate a unique tree using Prüfer encoding is as follows.

S1) Let C be the original Prüfer string and C' be the set of all nodes not included in C.

S2) Let i be the eligible nodes with smallest label in C' and j be the leftmost digit of C. Add the edge from i to j into the tree. Remove i from C' and j from C. If j does not occur anymore in C, put j into C'. Repeat the process until no digits remain in C.

S3) After S2 completes, there will be exactly one node r in C' and s in C. Add edge r to s, connecting the tree and forming a spanning tree.

For example, a Prüfer string (3 2 1 3) corresponds to a spanning tree on a six-node graph (Fig. 1). In the construction of the spanning tree, the Prüfer numbers are $C = (3\ 2\ 1\ 3)$ and $C' = (4\ 5\ 6)$. Node 4 is the eligible node with the smallest label and node 3 is the leftmost digit in C. Add edge (3–4) to the tree, remove node 4 from C', and node 3 from C. Node 5 is now eligible with the smallest label and the second node in C is 2. Add edge (2–5) to the tree, remove node 5 from C', and node 2 from C. Since node 2 does not exist anywhere in C, add node 2 to C' and, hence, $C' = (2\ 6)$. Node 2 is eligible with the smallest label and node1 is the leftmost in C. Add edge (2–1) to the tree, remove 2 from C' and 1 from C. Since node 1 does not exist anywhere in C, add node 1 to C' and, hence, $C' = (1\ 6)$. Node 1 is eligible with the smallest label and 3 is the leftmost digit in C. Add edge (1–3) to the tree and remove node 3 from C and 1 from C'. Finally, connect the last digit in C with the last digit in C', which adds edge (3–6) to the tree.

Since the chromosomes coded by Prüfer are all legal spanning trees, there is no need to repair illegal chromosomes. The degree of each node can be easily checked based on the number of each node in a chromosome. The chromosomes, however, may violate the degree constraints. The strategy for repairing infeasible Prüfer coded chromosomes is a difficult task. While it is easy to repair degree-violated genes by random method, it is very difficult to repair genes by heuristic method since we need to decode every chromosome. In addition, due to lack of locality in Prüfer encoding, even a slight change in a gene can cause a completely different spanning tree compared to the original spanning tree [16].

B. Determinant Encoding

Determinant encoding is a simple node-based indirect encoding strategy proposed by Abuali *et al.* [16] to overcome the bottlenecks of Prüfer encoding. The length of the chromosome is $N - 1$, where N is the number of nodes. The decoding algorithm treats each allele of gene to correspond to its position in the chromosome and the position represents its direct connecting node. The first gene is decoded as fixed-position 2, second as fixed-position 3, and so on. The procedure for decoding the algorithm is as follows.

S1) Let C be the given determination string and l be its length. If $C(j)$ is the jth allele in chromosome C and $1 \leq j \leq l$, the number of the nodes in the given graph G is $l + 1$, where a node is denoted as node (x) and $1 \leq x \leq l + 1$.

S2) Set $j = 1$, if $0 < j < l + 1$, go to S3, else Stop.

S3) Connect node $(j+1)$ with node $(C(j))$, Set $j = j+1$, go back to S2.

For example, let Fig. 2 represent a chromosome coded by determinant encoding. Each corresponding fixed position in the figure is equal to the order of the gene plus one. Each allele of chromosome can be connected to its corresponding fixed position as in S3 above. The links in the tree are (2–3), (3–4), (4–2), (5–5), (6–8), (7–5), (8–3), (9–9). The final layout of the tree is shown in Fig. 1. The generated tree may not be legal and need to be repaired by reallocating genes in appropriate positions to generate a legal tree.

Although determinant encoding is an indirect encoding strategy, the decoding algorithm is very simple. We can easily determine degree constraint violations by simply examining the chromosome and counting the number of links starting from a single node. The only disadvantage is that it generates illegal trees that need to be repaired.

REFERENCES

[1] A. Kershenbaum, "When genetic algorithms work best," *INFORMS J. Comput.*, vol. 9, no. 3, pp. 254–255, 1997.

[2] L. Davis, Ed., *Genetic Algorithm and Simulated Annealing*. San Mateo, CA: Morgan Kaufmann, 1987.

[3] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 61–69.

[4] C. Reeves, "Genetic algorithms for the operations researcher," *INFORMS J. Comput.*, vol. 9, no. 3, pp. 231–250, 1997.

[5] K. A. DeJong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1975.

[6] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 122–128, Jan. 1986.

[7] B. Dengiz, F. Altiparmak, and A. E. Smith, "Local search genetic algorithm for optimal design of reliable networks," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 179–188, Sept. 1997.

[8] H. J. Bremermann, M. Rogson, and S. Salaff, "Global properties of evolution processes," in *Natural Automata and Useful Simulations*, H. H. Pattee, E. A. Edlsack, L. Fein, and A. B. Callahan, Eds. Washington, DC: Spartan, 1966, pp. 3–41.

[9] J. Reed, R. Toombs, and N. A. Barricelli, "Simulation of biological evolution and machine learning: I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type, and crossing," *J. Theor. Bio.*, vol. 17, no. 3, pp. 319–342, 1967.

[10] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 2–9.

[11] D. E. Goldberg and Jr. R. Lingle, "Alleles loci and the traveling salesman problem," in *Proceedings of the First International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 154–159.

[12] J. J. Grefenstette and J. M. Fitzpatrick, "Genetic search with approximate function evaluations," in *Proceedings of the First International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 160–168.

[13] D. Smith, "Bin packing with adaptive search," in *Proceedings of the First International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 202–206.

[14] J. J. Grefenstette, "Multilevel credit assignment in a genetic learning system," in *Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 202–209.

[15] P. W. Poon and J. N. Carter, "Genetic algorithm crossover operators for ordering applications," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 135–147, 1995.

[16] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld, "Determinant factorization: Anew encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem," in *Proceedings of The Sixth International Conference on Genetic Algorithms*, L. J. Eshelman, Ed. San Mateo, CA: Morgan Kaufmann, 1995, pp. 470–477.

[17] L. Booker, "Improving search in genetic algorithm," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. San Mateo, CA: Morgan Kaufmann, 1987, pp. 61–73.

[18] R. K. Ahuja and J. B. Orlin, "Developing fitter genetic algorithms," *INFORMS J. Comput.*, vol. 9, no. 5, pp. 251–253, 1997.

[19] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 67–82, Apr. 1997.
[20] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning tree," *Comput. Oper. Res.*, vol. 7, no. 4, pp. 239–249, 1980.
[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[22] R. L. Graham and P. Hell, "On the history of minimum spanning tree problem," *Ann. History Comput.*, vol. 7, no. 1, pp. 43–57, 1985.
[23] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," in *Proc. Amer. Math. Soc.*, vol. 7, 1956, pp. 48–50.
[24] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, 1957.
[25] F. Harary and J. P. Hayes, "Node fault tolerance in graphs," *Networks*, vol. 27, no. 1, pp. 19–23, 1996.
[26] H. K. Ku and J. P. Hayes, "Optimally edge fault-tolerant trees," *Networks*, vol. 27, no. 3, pp. 203–214, 1996.
[27] D. S. Johnson, "The NP-completeness column: An ongoing guide," *J. Algorithms*, vol. 6, no. 1, pp. 145–159, 1985.
[28] A. K. Obruca, "Spanning tree manipulation and the travelling-salesman problem," *Comput. J.*, vol. 10, no. 4, pp. 374–377, 1968.
[29] M. Savelsbergh and T. Volgenant, "Edge exchanges in the degree-constrained spanning tree problem," *Comput. Oper. Res.*, vol. 12, no. 4, pp. 341–348, 1985.
[30] S. Coombs and L. Davis, "Genetic algorithms and communication link speed design: constraints and operators," in *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 257–260.
[31] 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.
[32] M. Cuppini, "A genetic algorithm for channel assignment problems," *Eur. Trans. Telecommun. Related Technol.*, pp. 285–294, 1994.
[33] A. I. Oyman and C. E. Solvinf, "Concentrator location-problems using genetic algorithms," in *Proc. 7th Mediterranean Electrotechnical Conf.*, vol. 3, Antalya, Turkey, 1994, pp. 1341–1344.
[34] C. C. Palmer and A. Kershenbaum, "An approach to a problem in network design using genetic algorithms," *Networks*, vol. 26, no. 3, pp. 151–163, 1995.
[35] H. Esbensen, "Computing near-optimal solutions to the Steiner problem in a graph using genetic algorithm," *Networks*, vol. 26, no. 4, pp. 173–185, 1995.
[36] P. Charddaire, A. Kapsalis, J. W. Mann, V. J. Rayward-Smith, and G. D. Smith, "Applications of genetic algorithms in telecommunications," in *Proceedings of the Second International Workshop on Applications of Neural Networks to Telecommunications*, J. Alspector, R. Goodman, and T. X. Brown, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1995, pp. 290–299.
[37] G. Zhou and M. Gen, "A note in genetic algorithms for degree constrained spanning tree problems," *Networks*, vol. 30, no. 2, pp. 91–97, 1997.
[38] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. New York: Wiley, 1997.
[39] P. Piggott and F. Suraweera, "Encoding graph for genetic algorithms: An investigation using the minimum spanning tree problem," in *Evolutionary Computation: Theory and Applications*, X. Yao, Ed. Singapore: World Scientific, 1996, pp. 305–314.
[40] H. Prüfer , "Neuer beweis eines satzes uber permutation," *Arch. Math. Phys.*, vol. 27, pp. 742–744, 1918.
[41] D. E. Goldberg, *Genetic Algorithms in Search Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[42] C. Reeves, "Genetic algorithms for flow shop sequencing," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 5–13, 1995.
[43] D. W. Coit, A.E. Smith, and D. M. Tate, "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *INFORMS J. Comput.*, vol. 8, no. 2, pp. 173–182, 1996.
[44] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proceedings of the First International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 101–111.
[45] T. Bäck and F. Hoffmeister, "Extended selection mechanism in genetic algorithms," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 92–99.
[46] T. Bäck , D. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Oxford: Oxford Univ. Press, 1996.
[47] J. Xu, S. Y. Chiu, and F. Glover, "Using Tabu search to solve the Steiner tree-star problem in telecommunication networks design," *Telecommun. Syst.*, vol. 6, no. 2, pp. 117–127, 1996.
[48] J. A. Joines and C. R. Houck, "On the use of nonstationery penalty functions to solve nonlinear constrained optimization problems with GAs.," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, D. Fogel, Ed. Piscataway, NJ: IEEE Press, 1994, pp. 579–584.
[49] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evol. Comput.*, vol. 4, no. 1, pp. 1–32, 1996.
[50] V. Petridis, S. Kazarlis, and A. Bakirtzis, "Varying fitness functions in genetic algorithm constrained optimization: The cutting stock and unit commitment problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 629–640, Oct. 1998.

Hsinghua Chou received the undergraduate degree in business administration from Sun Yat-sen University, Kaohsiung, Taiwan, in 1994, and the M.S. degree in business administration sciences from Iowa State University, Ames, IA, in 1998.

He is currently a Software Engineer with the Sprint Corporation, Overland Park, KS. He has been involved in the research of genetic algorithms and its implementation in various business domains including telecommunication network design and operation management. His current research interests include application server development, rule-based virtual machine, and the integration of distributed environment.

G. Premkumar received the B.S. degree in engineering from Regional Engineering College, Trichy, India, the M.B.A. in management from the Indian Institute of Management, Bangalore, India, in 1982, and the Ph.D. degree in information systems from the University of Pittsburgh, Pittsburgh, PA, in 1989.

He is currently a Union Pacific Chair and an Associate Professor of Information Systems in the College of Business at Iowa State University, Ames. He has over nine years of industry experience in information systems and related areas. He has authored or coauthored papers in *Information Systems Research*, *Decision Sciences*, *Journal of Management Information Systems*, *European Journal of Operations Research*, IEEE TRANSACTIONS ON ENGINEERING MANAGEMENT, and other leading journals and conference proceedings. His current research interests include telecommunications, electronic commerce, interorganizational systems/EDI, and adoption and diffusion of information technology.

Chao-Hsien Chu received the undergraduate degree in industrial engineering from Chung Yuan University, Changli, Taiwan, in 1974, the M.B.A. from Tatung Institute of Technology, Taipei, Taiwan, in 1978, and the Ph.D. degree from Pennsylvania State University, University Park, PA, in 1984.

He is currently an Associate Professor of Information Sciences and Technology at Pennsylvania State University, University Park, and is also the Chief Academic Advisor to the School of Management at Hebei University of Sciences and Technology, Hebei, China. He was on the faculty of Baruch College of the City University of New York, Iowa State University, Ames, and the University of Tsukuba, Japan. His research focuses on applying intelligent technologies (expert systems, fuzzy logic, neural networks, and genetic algorithms, etc.) to manufacturing management, communication networks design, and network security.